CONTENTS

UNIT I

UNIT II

UNIT III

UNIT I

EXPLORATORY DATA ANALYSIS

SYLLABUS

EDA fundamentals - Understanding data science - Significance of EDA - Making sense of data - Comparing EDA with classical and Bayesian analysis - Software tools for EDA - Visual Aids for EDA - Data transformation techniques - Merging database, reshaping and pivoting, Transformation techniques - Grouping Datasets - Data aggregation - Pivot tables and cross-tabulations.

- EDA Fundamentals
- Understanding Data Science
- Significance of EDA
- Making Sense of Data
- Comparing EDA with Classical and Bayesian Analysis
- Software Tools for EDA
- Visual Aids for EDA
- Data Transformation Techniques
- Merging Database, Reshaping and Pivoting
- Transformation Techniques
- Grouping Datasets
- Data Aggregation
- Pivot Tables and Cross -Tabulations

UNIT I

EXPLORATORY DATA ANALYSIS

1.1. EXPLORATORY DATA ANALYSIS FUNDAMENTALS

EDA is a process of examining the available dataset to discover patterns, spot anomalies, test hypotheses, and check assumptions using statistical measures. In this chapter, we are going to discuss the steps involved in performing top-notch exploratory data analysis and get our hands dirty using some open source databases. As mentioned here and in several studies, the primary aim of EDA is to examine what data can tell us before actually going through formal modeling or hypothesis formulation.

1.2. UNDERSTANDING DATA SCIENCE

The main takeaway here is the stages of EDA. Let's understand in brief what these stages are:

Data requirements: There can be various sources of data for an organization. It is important to comprehend what type of data is required for the organization to be collected, curated, and stored.

Data collection: Data collected from several sources must be stored in the correct format and transferred to the right information technology personnel within a company. As mentioned previously, data can be collected from several objects on several events using different types of sensors and storage tools.

Data processing: Preprocessing involves the process of pre-curating the dataset before actual analysis. Common tasks involve correctly exporting the dataset, placing them under the right tables, structuring them, and exporting them in the correct format.

Data cleaning: Preprocessed data must be correctly transformed for an incompleteness check, duplicates check, error check, and missing value check. These tasks are performed in the data cleaning stage, which involves matching the correct record, finding inaccuracies in the dataset, understanding the overall data quality, removing duplicate items, and filling in the missing values.

Modeling and algorithm: From a data science perspective, generalized models or mathematical formulas can represent or exhibit relationships among different variables, such as correlation or causation. These models or equations involve one or more variables that depend on other variables to cause an event.

Data Product: Any computer software that uses data as inputs, produces outputs, and provides feedback based on the output to control the environment is referred to as a data product. A data product is generally based on a model developed during data analysis, for example, a recommendation model that inputs user purchase history and recommends a related item that the user is highly likely to buy.

Communication: This stage deals with disseminating the results to end stakeholders to use the result for *business intelligence*. One of the most notable steps in this stage is data visualization. Visualization deals with information relay techniques such as tables, charts, summary diagrams, and bar charts to show the analyzed result.

1.3. THE SIGNIFICANCE OF EDA

Exploratory data analysis is key, and usually the first exercise in data mining. It allows us to visualize data to understand it as well as to create hypotheses for further analysis. The exploratory analysis centers around creating a synopsis of data or insights for the next steps in a data mining project.

Steps in EDA

Problem definition: The problem definition works as the driving force for a data analysis plan execution. The main tasks involved in problem definition are defining the main objective of the analysis, defining the main deliverables, outlining the main roles and responsibilities, obtaining the current status of the data, defining the timetable, and performing cost/benefit analysis.

Data preparation: This step involves methods for preparing the dataset before actual analysis. In this step, we define the sources of data, define data schemas and tables, understand the main characteristics of the data, clean the dataset, delete non-relevant datasets, transform the data, and divide the data into required chunks for analysis.

Data analysis: This is one of the most crucial steps that deals with descriptive statistics and analysis of the data. The main tasks involve summarizing the data, finding the hidden correlation and relationships among the data, developing predictive models, evaluating the models,and calculating the accuracies. Some of the techniques used for data summarization are summary tables, graphs, descriptive statistics, inferential statistics, correlation statistics, searching and grouping.

Development and representation of the results: This step involves presenting the dataset to the target audience in the form of graphs, summary tables, maps, and diagrams. This is also an essential step asthe result analyzed from the dataset should be interpretable by the business stakeholders, which is one of the major goals of EDA.

1.4. MAKING SENSE OF DATA

A dataset contains many observations about a particular object. For instance, a dataset about patients in a hospital can contain many observations. A patient can be described by a patient identifier (ID), name, address, weight, date of birth, address, email, and gender. Each of these features that describes a patient is a variable. Each observation can have a specific value for each of these variables.

Numerical Data

This data has a sense of measurement involved in it; for example, a person's age, height, weight, blood pressure, heart rate, temperature, number of teeth, number of bones, and the number of family members. This data is often referred to as **quantitative data** in statistics. The numerical dataset can be either discrete or continuous types.

Discrete Data

This is data that is countable and its values can be listed out. For example, if we flip a coin, the number of heads in 200 coin flips can take values from 0 to 200 (finite) cases. A variable that represents a discrete dataset is referred to as a discrete variable. The discrete variable takes a fixed number of distinct values. For example, the Country variable can have values such as Nepal, India, Norway, and Japan. It is fixed. The Rank variable of a student in a classroom can take values from 1, 2, 3, 4, 5, and so on.

Continuous Data

A variable that can have an infinite number of numerical values within aspecific range is classified as continuous data. A variable describing continuous data is a continuous variable.

Categorical Data

This type of data represents the characteristics of an object; for example, gender, marital status, type of address, or categories of the movies. This data is often referred to as **qualitative datasets** in statistics.

A variable describing categorical data is referred to as a **categorical variable**. These types of variables can have one of a limited number of values. There are different types of categorical variables:

A binary categorical variable can take exactly two values and is also referred to as a **dichotomous variable**. For example, when you create an experiment, the result is either success or failure. Hence, results can be understood as a **binary categorical variable**.

Polytomous variables are categorical variables that can take more than two possible values. For example, marital status can have severalvalues, such as annulled, divorced, interlocutory, legally separated, married, polygamous, never married, domestic partners, unmarried, widowed, domestic partner, and unknown. Since marital status cantake more than two possible values, it is a **polytomous variable.**

1.5. COMPARING EDA WITH CLASSICAL AND BAYESIAN ANALYSIS

There are several approaches to data analysis.

Classical data analysis: For the classical data analysis approach, the problem definition and data collection step are followed by model development, which is followed by analysis and result communication. **Exploratory data analysis approach**: For the EDA approach, it follows the same approach as classical data analysis except the model imposition and the data analysis steps are swapped. The main focus is on the data, its structure, outliers, models, and visualizations.

Bayesian data analysis approach: The Bayesian approach incorporates prior probability distribution knowledge into the analysis steps as shown in the following diagram. Well, simply put, prior probability distribution of any quantity expresses the belief about that particular quantity before considering some evidence.

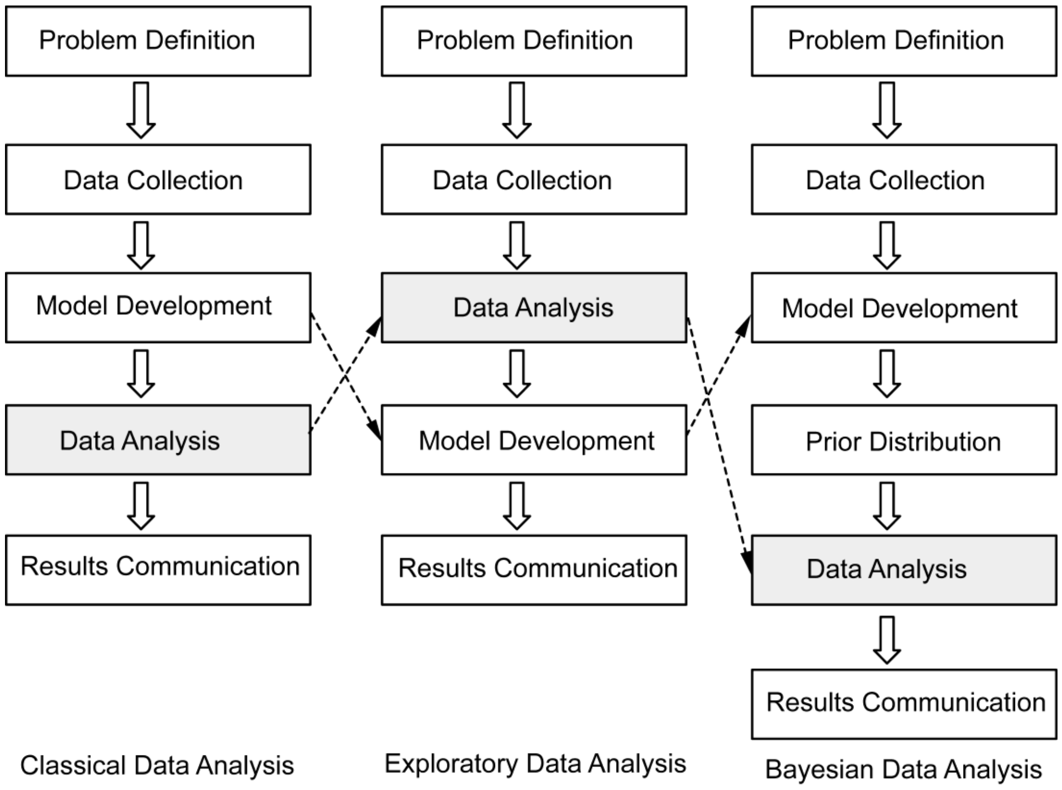


Fig. 1.1.

1.6. SOFTWARE TOOLS AVAILABLE FOR EDA

There are several software tools that are available to facilitate EDA.

NumPy

For importing numpy, we will use the following code:

```
import numpy as np
```

For NumPy arrays and file operations, we will use the following code:

```
# Save a numpy array into filex = np.arange(0.0,50.0,1.0)
np.savetxt('data.out', x, delimiter=',')
# Loading numpy array from text
z = np.loadtxt('data.out', unpack=True)print(z)
# Loading numpy array using genfromtxt method my_array2 = np.genfromtxt('data.out',
    skip_header=1, filling_values=-999) print(my_array2)
```

For inspecting NumPy arrays, we will use the following code:

```
# Print the number of `my2DArray`'s dimensionsprint(my2DArray.ndim)
# Print the number of `my2DArray`'s elementsprint(my2DArray.size)
# Print information about `my2DArray`'s memory layoutprint(my2DArray.flags)
# Print the length of one array element in bytesprint(my2DArray.itemsize)
# Print the total consumed bytes by `my2DArray`'s elements print(my2DArray.nbytes)
```

Pandas

1. Use the following to set default parameters:

```
import numpy as npimport pandas as pd
print("Pandas Version:", pd._version_)
pd.set_option('display.max_columns', 500)
```

pd.set_option('display.max_rows', 500)

2. In pandas, we can create data structures in two ways: series and dataframes. The following code snippet shows how we can create a dataframe from a series:

```
series = pd.Series([2, 3, 7, 11, 13, 17, 19, 23])print(series)

# Creating dataframe from Seriesseries_df = pd.DataFrame({

'A': range(1, 5),

'B': pd.Timestamp('20190526'),

'C': pd.Series(5, index=list(range(4)), dtype='float64'),'D': np.array([3] * 4,
dtype='int64'),

'E': pd.Categorical(["Depression", "Social Anxiety", "Bipolar Disorder","Eating
Disorder"]),

'F': 'Mental health','G': 'is challenging'

})

print(series_df)
```

3. Now, let's load a dataset from an external source into a pandas DataFrame. Afterthat, let's see the first 10 entries:

```
columns = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
'marital_status', 'occupation', 'relationship', 'ethnicity',

'gender','capital_gain','capital_loss','hours_per_week','country_of_origin','inco me']

df=pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data',names=columns)

df.head(10)
```

If you run the preceding cell, you should get an output similar to the following screenshot:

	age	***work class***	***fnlwgt***	***education***	***education-num***	***marital-status***	***Occupation***	***relationship***	***Ethinicity***	***Gender***	***Capital_gain***	***Capital_loss***	***Hours_per_week***
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-speciality	Wife	Black	Female	0	0	40
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in family	Black	Female	0	0	16
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45
8	31	Private	45781	Masters	14	Never-married	Prof-speciality	Not-in family	White	Female	14084	0	50
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	40

4. The following code displays the rows, columns, data types, and memory usedby the dataframe:

df.info()

The output of the preceding code snippet should be similar to thefollowing:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 32561 entries, 0 to 32560 Data columns (total 15 columns):age 32561 non-null int64 workclass 32561 non-null object fnlwgt 32561 non-null int64 education 32561 non-null object

education_num 32561 non-null int64 marital_status 32561 non-null object occupation 32561 non-null object relationship 32561 non-null object ethnicity 32561 non-null object gender 32561 non-null object capital_gain 32561 non-null int64 capital_loss 32561 non-null int64 hours_per_week 32561 non-null int64

country_of_origin 32561 non-null objectincome 32561 non-null object

dtypes: int64(6), object(9)memory usage: 3.7 + MB

5. Let's now see how we can select rows and columns in any dataframe:

# Selects a rowdf.iloc[10]

# Selects 10 rowsdf.iloc[0:10]

# Selects a range of rowsdf.iloc[10:15]

# Selects the last 2 rowsdf.iloc[-2:]

# Selects every other row in columns 3-5df.iloc[::2, 3:5].head()

6. Let's combine NumPy and pandas to create a dataframe as follows:

import pandas as pdimport numpy as np

np.random.seed(24)

dFrame = pd.DataFrame({'F': np.linspace(1, 10, 10)})

dFrame = pd.concat([df, pd.DataFrame(np.random.randn(10, 5), columns=list('EDCBA'))],

axis=1) dFrame.iloc[0, 2] = np.nandFrame

7. Let's style this table using a custom rule. If the values are greater than zero, we change the color to black (the default color); if the value is less

than zero,we change the color to red; and finally, everything else would be colored green. Let's define a Python function to accomplish that:

```
# Define a function that should color the values that are less than 0 def
colorNegativeValueToRed(value):
if value < 0: color = 'red'
elif value > 0: color = 'black'
else:
color = 'green'
return 'color: %s' % color
```

8. Now, let's pass this function to the dataframe. We can do this by using the style method provided by pandas inside the dataframe:

```
s = df.style.applymap(colorNegativeValueToRed, subset=['A','B','C','D','E'])s
```

It should display a colored dataframe as shown in the followingscreenshot:

	F	E	D	C	B	A
0	1	1.32921	nan	**–0.31628**	**–0.99081**	**–1.07082**
1	2	**–1.43871**	0.564417	0.295722	**–1.6264**	0.219565
2	3	0.678805	1.88927	0.961538	0.104011	**–0.481165**
3	4	0.850229	1.45342	1.05774	0.165562	0.515018
4	5	**–1.33694**	0.562861	1.39285	**– 0.063328**	0.121668
5	6	1.2076	**– 0.00204021**	1.6278	0.354493	1.03753
6	7	**–0.385684**	0.519818	1.68658	**–1.32596**	1.42898
7	8	**–2.08935**	**– 0.12982**	0.631523	**–0.586538**	0.29072
8	9	1.2641	0.290035	**–1.97029**	0.803906	1.03055
9	10	0.118098	**– 0.0218533**	0.0468407	**–1.62875**	**–0.392361**

It should be noted that the apply map and apply methods are computationally expensive as they apply to each value inside the dataframe. Hence, it will take some time to execute. Have patience and await execution.

9. Now, let's go one step deeper. We want to scan each column and highlight themaximum value and the minimum value in that column:

```
def highlightMax(s): isMax = s == s.max()
return ['background-color: orange' if v else '' for v in isMax]
def highlightMin(s): isMin = s == s.min()
return ['background-color: green' if v else '' for v in isMin]
```

We apply these two functions to the dataframe as follows:

```
df.style.apply(highlightMax).apply(highlightMin).highlight_null(null_color='red)
```

The output should be similar to the following screenshot:

	F	E	D	C	B	A
0	**1**	**1.32921**	**nan**	–0.31628	–0.99081	**–1.07082**
1	2	–1.43871	0.564417	0.295722	–1.6264	0.219565
2	3	0.678805	**1.88927**	0.961538	0.104011	–0.481165
3	4	0.850229	1.45342	1.05774	0.165562	0.515018
4	5	–1.33694	0.562861	1.39285	– 0.063328	0.121668
5	6	1.2076	– 0.00204021	1.6278	0.354493	1.03753
6	7	–0.385684	0.519818	**1.68658**	–1.32596	**1.42898**
7	8	**–2.08935**	**– 0.12982**	0.631523	–0.586538	0.29072
8	9	1.2641	0.290035	**–1.97029**	**0.803906**	1.03055
9	**10**	0.118098	– 0.0218533	0.0468407	**–1.62875**	–0.392361

SciPy

SciPy is a scientific library for Python and is open source. We are going touse this library in the upcoming chapters. This library depends on the NumPy library, which provides an efficient n-dimensional array manipulation function. If you wantto get started early, check for scipy.statsfrom the SciPy library.

Matplotlib

Matplotlib provides a huge library of customizable plots, along with a comprehensive set of backends. It can be utilized to create professional reporting applications, interactive analytical applications, complex dashboard applications, web/GUI applications, and embedded views.

1.7. VISUAL AIDS FOR EDA

As data scientists, two important goals in our work would be to extract knowledge from the data and to present the data to stakeholders. Presenting results to stakeholders is very complex in the sense that our audience may not have enough technical know-how to understand programming jargon and other technicalities. Hence, visual aids are very useful tools.

Line Chart

We have created a function using the faker Python library to generate the dataset. It is the simplest possible dataset you can imagine, with just two columns. The first column is Date and the second column is Price,

My generate Data function is defined here:

```
import datetimeimport math
import pandas as pdimport random
import radar
from faker import Fakerfake = Faker()
def generateData(n):listdata = []
start = datetime.datetime(2019, 8, 1)
end = datetime.datetime(2019, 8, 30)delta = end - start
for _ in range(n):
date = radar.random_datetime(start='2019-08-1', stop='2019-08-30').strftime("%Y-
%m-%d")
```

price = round(random.uniform(900, 1000), 4)listdata.append([date, price])

df = pd.DataFrame(listdata, columns = ['Date', 'Price']) df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')df = df.groupby(by='Date').mean()

return df

Having defined the method to generate data, let's get the data into a pandasdataframe and check the first 10 entries:

df = generateData(50)df.head(10)

The output of the preceding code is shown in the following screenshot:

Date	Price
2019-08-01	999.598900
2019-08-02	957.870150
2019-08-04	978.674200
2019-08-05	963.380375
2019-08-06	978.092900
2019-08-07	987.847700
2019-08-08	952.669900
2019-08-10	973.929400
2019-08-13	971.485600
2019-08-14	977.036200

Steps involved

Let's look at the process of creating the line chart:

1. Load and prepare the dataset.
2. Import the matplotlib library. It can be done with this command:

import matplotlib.pyplot as plt

3. Plot the graph:

```
plt.plot(df)
```

4. Display it on the screen:

```
plt.show()
```

Here is the code if we put it all together:

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (14, 10)plt.plot(df)
```

And the plotted graph looks something like this:

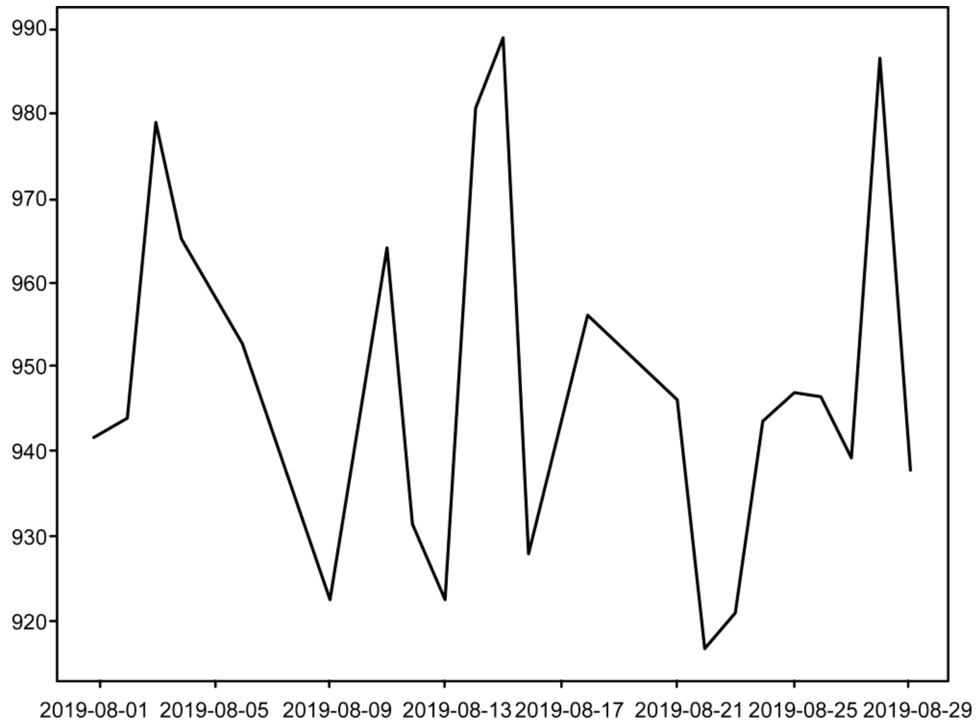


Fig. 1.2.

Bar Charts

This is one of the most common types of visualization that almost everyonemust have encountered. Bars can be drawn horizontally or vertically to represent **categorical variables**.

Bar charts are frequently used to distinguish objects between distinct collections in order to track variations over time. In most cases, bar charts are very convenient when the changes are large. In order to learn about bar charts, let's assume a pharmacy in Norway keeps track of the amount of **Zoloft** sold every month. Zoloft is a medicine prescribed to patients suffering from depression. We can use the calendar

1. Let's import the required libraries:

```
import numpy as npimport calendar
import matplotlib.pyplot as plt
```

2. Set up the data. Remember, the range stopping parameter is exclusive, meaning if you generate range from (1, 13), the last item, 13, is not included:

```
months = list(range(1, 13))
sold_quantity = [round(random.uniform(100, 200)) for x in range(1, 13)]
```

3. Specify the layout of the figure and allocate space:

```
figure, axis = plt.subplots()
```

4. In the *x* axis, we would like to display the names of the months:

```
plt.xticks(months, calendar.month_name[1:13], rotation=20)
```

5. Plot the graph:

```
plot = axis.bar(months, sold_quantity)
```

6. This step is optional depending upon whether you are interested in displaying the data value on the head of the bar. It visually gives more meaning to show an actual number of sold items on the bar itself:

```
for rectangle in plot:
height = rectangle.get_height()
axis.text(rectangle.get_x() + rectangle.get_width() /2., 1.002 * height, '%d' % int(height), ha='center', va = 'bottom')
```

7. Display the graph on the screen:

```
plt.show()
```

The bar chart is as follows:

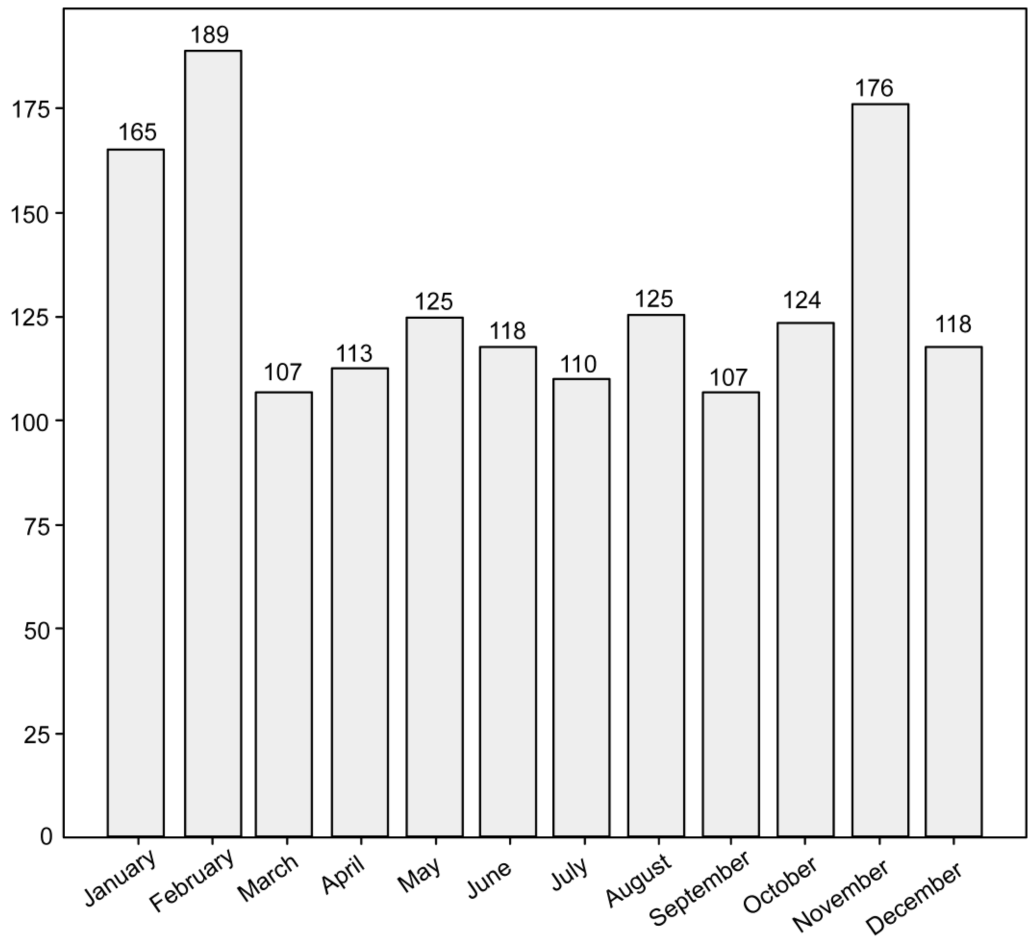


Fig. 1.3.

Scatter Plot

Scatter plots are also called scatter graphs, scatter charts, scattergrams, and scatter diagrams. They use a **Cartesian coordinates system** to display values of typically two variables for a set of data.

When should we use a scatter plot? Scatter plots can be constructed in the following two situations:

When one continuous variable is dependent on another variable, which is under the control of the observer

When both continuous variables are independent

There are two important concepts - **independent variable** and **dependent variable**. In statistical modeling or mathematical modeling, the values of dependent variables rely on the values of independent variables. The dependent variable is the outcome variable being studied. The independent variables are also referred to as **regressors**.

Here, we are using seaborn to load the dataset:

1. Import seaborn and set some default parameters of matplotlib: import seaborn as sns

```
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (8, 6)

plt.rcParams['figure.dpi'] = 150
```

2. Use style from seaborn. Try to comment on the next line and see the difference in the graph:

```
sns.set()
```

3. Load the Iris dataset:

```
df = sns.load_dataset('iris')


df['species'] = df['species'].map({'setosa': 0, "versicolor": 1,

"virginica": 2})
```

4. Create a regular scatter plot:

```
plt.scatter(x=df["sepal_length"], y=df["sepal_width"], c = df.species)
```

5. Create the labels for the axes:

```
plt.xlabel('Septal Length')plt.ylabel('Petal length')
```

6. Display the plot on the screen:

```
plt.show()
```

The scatter plot generated by the preceding code is as follows:

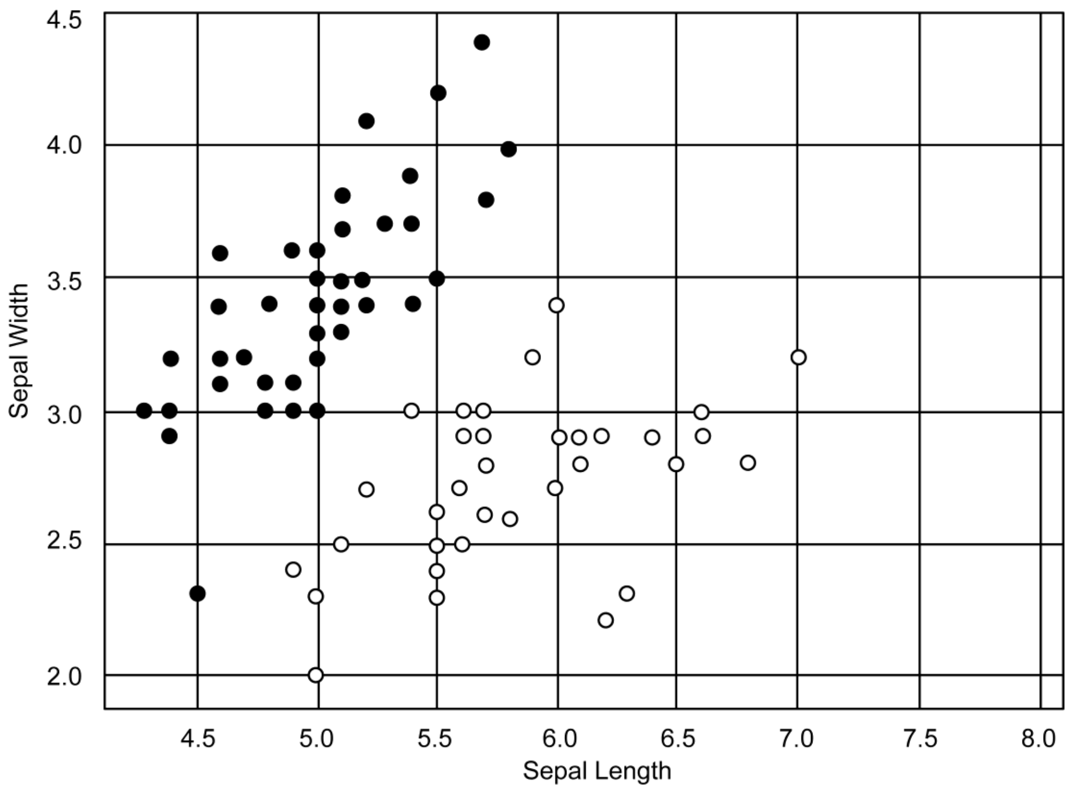


Fig. 1.4.

Pie Chart

This is one of the more interesting types of data visualization graphs.

There are two ways in which you can load the data: first, directly from the GitHub URL; or you can download the dataset from the GitHub and reference it from your local machine by providing the correct path. In either case, you can use the read_csv method from the pandas library. Check out the followingsnippet:

# Create URL to JSON file (alternatively this can be a filepath)

url='https://raw.githubusercontent.com/hmcuesta/PDA_Book/master/Chapter3/pokemonByType

.csv'

# Load the first sheet of the JSON file into a data framepokemon = pd.read_csv(url, index_col='type')pokemon

The preceding code snippet should display the dataframe as follows:

Type	**Amount**
Bug	45
Dark	16
Dragon	12
Electric	7
Fighting	3
Fire	14
Ghost	10
Grass	31
Ground	17
Ice	11
Normal	29
Poison	11
Psychic	9
Rock	24
Steel	13
Water	45

Next, we attempt to plot the pie chart:

```
import matplotlib.pyplot as plt
plt.pie(pokemon['amount'],labels=pokemon.index,shadow=False,startangle=90,autop
ct='%1.1f%%',)
plt.axis('equal')plt.show()
```

We should get the following pie chart from the preceding code:

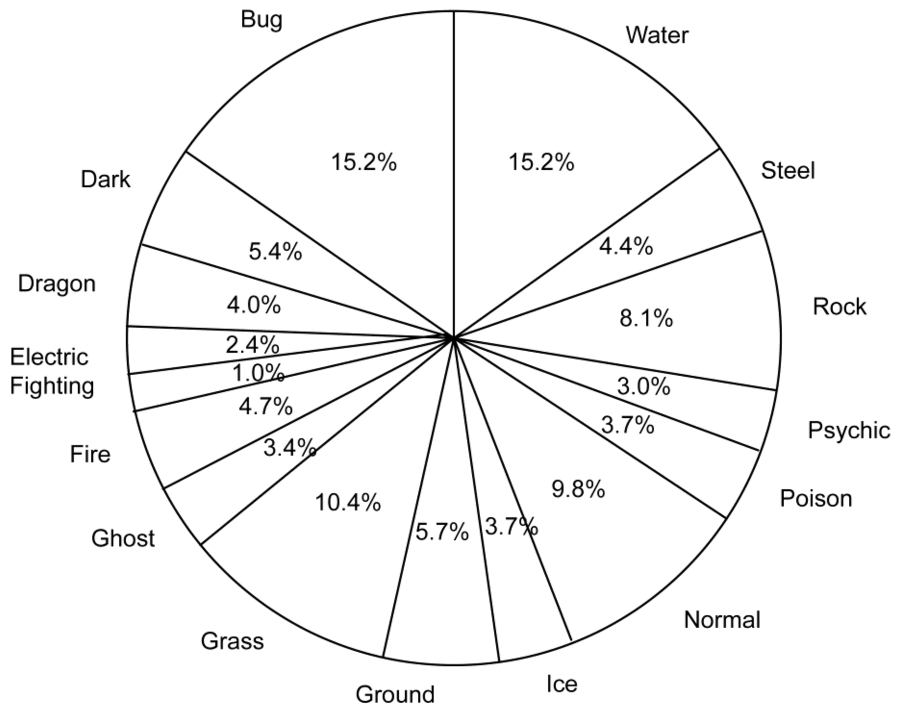


Fig. 1.5.

Histogram

Histogram plots are used to depict the distribution of any continuous variable. These types of plots are very popular in statistical analysis.

Consider the following use cases. A survey created in vocational training sessions of developers had 100 participants. They had several years of Python programming experience ranging from 0 to 20.

Let's import the required libraries and create the dataset:

```
import numpy as np
import matplotlib.pyplot as plt
#Create data set
yearsOfExperience = np.array([10, 16, 14, 5, 10, 11, 16, 14, 3, 14, 13, 19, 2, 5,
7, 3, 20,11, 11, 14, 2, 20, 15, 11, 1, 15, 15, 15, 2, 9, 18, 1, 17, 18,
13, 9, 20, 13, 17, 13, 15, 17, 10, 2, 11, 8, 5, 19, 2, 4, 9,
17, 16, 13, 18, 5, 7, 18, 15, 20, 2, 7, 0, 4, 14, 1, 14, 18,
8, 11, 12, 2, 9, 7, 11, 2, 6, 15, 2, 14, 13, 4, 6, 15, 3,
6, 10, 2, 11, 0, 18, 0, 13, 16, 18, 5, 14, 7, 14, 18])
```

yearsOfExperience

In order to plot the histogram chart, execute the following steps:

1. Plot the distribution of group experience:

```
nbins = 20n, bins, patches = plt.hist(yearsOfExperience, bins=nbins)
```

2. Add labels to the axes and a title:

```
plt.xlabel("Years of experience with Python Programming")plt.ylabel("Frequency")
plt.title("Distribution of Python programming experience in the vocational training session")
```

3. Draw a green vertical line in the graph at the average experience:

```
plt.axvline(x=yearsOfExperience.mean(), linewidth=3, color = 'g')
```

4. Display the plot:

```
plt.show()
```

The preceding code generates the following histogram:

Distribution of Python Programming Experience in the Vocational Training Session

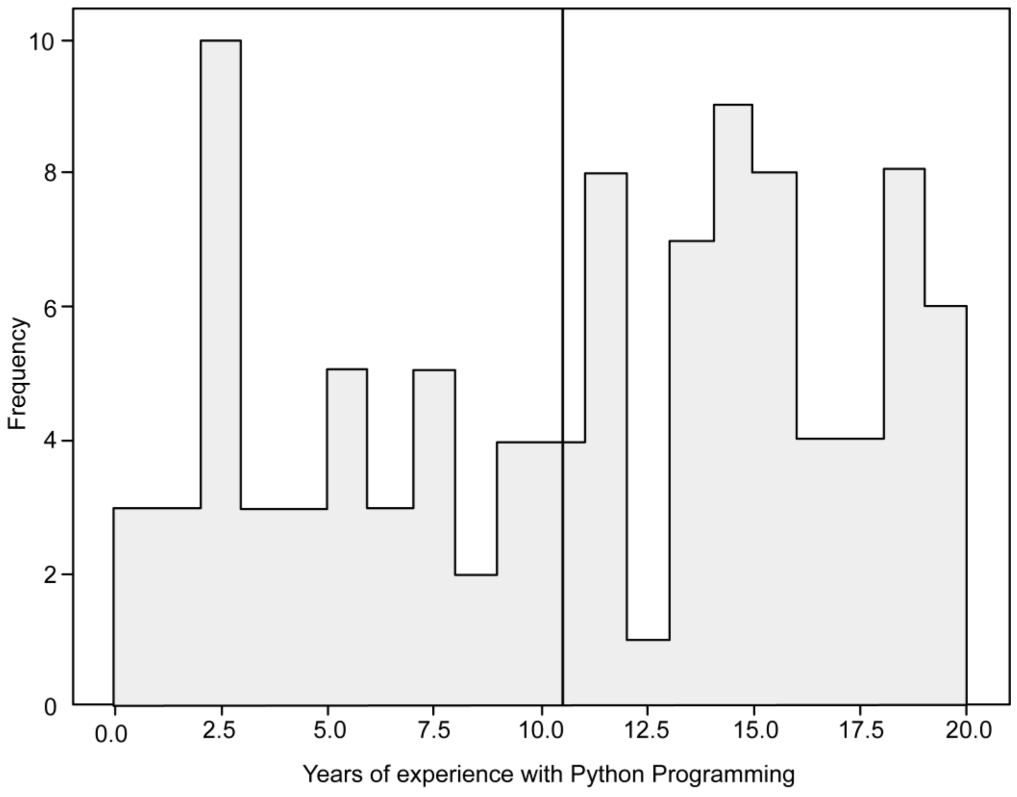


Fig. 1.6.

EDA with Personal Email

The exploration of useful insights from a dataset requires a great deal of thought and a high level of experience and practice.

Loading the Dataset

1. Let's load the required libraries:

```
import numpy as np import pandas as pd
import matplotlib.pyplot as plt
```

2. When you have loaded the libraries, load the dataset:

```
import mailbox
mboxfile = "PATH TO DOWNLOADED MBOX FIL"
mbox = mailbox.mbox(mboxfile)mbox
```

Note that it is essential that you replace the mbox file path with yourown path.

The output of the preceding code is as follows:

```
<mailbox.mbox at 0x7f124763f5c0>
```

3. Next, let's see the list of available keys:

```
for key in mbox[0].keys():print(key)
```

1.8. DATA TRANSFORMATION

Although there are a lot of objects returned by the extracted data, we do not need all the items. We will only extract the required fields. Data cleansing is one of the essential steps in the data analysis phase. For our analysis, all we need is data for the following: *subject, from, date, to, label,* and *thread*.

Data Ccleansing

Let's create a CSV file with only the required fields. Let's start with the following steps:

1. Import the csv package:

```
import csv
```

2. Create a CSV file with only the required attributes:

```
with open('mailbox.csv', 'w') as outputfile:
```

```
writer = csv.writer(outputfile) writer. write row (['subject', 'from', 'date', 'to', 'label', 'thread'])
for message in mbox:writer.writerow([
message['subject'],message['from'],
message['date'],
message['to'],
message['X-Gmail-Labels'],message['X-GM-THRID']
]
```

Loading the CSV File

We will load the CSV file. Refer to the following code block:

```
dfs = pd.read_csv('mailbox.csv', names=['subject', 'from', 'date', 'to', 'label','thread'])
```

The preceding code will generate a pandas dataframe with only the required fields stored in the CSV file.

Converting the Date

Next, we will convert the date.

Check the datatypes of each column as shown here:

```
dfs.dtypes
```

The output of the preceding code is as follows:

```
subject objectfrom object date object
to object label object
thread float64dtype: object
```

Note that a date field is an object. So, we need to convert it into a DateTime argument. In the next step, we are going to convert the date field into an actual DateTime argument. We can do this by using the pandas to_datetime() method. See the following code:

```
dfs['date'] = dfs['date'].apply(lambda x: pd.to_datetime(x, errors='coerce',utc=True))
```

Removing NaN Values

Next, we are going to remove NaN values from the field. We can do this as follows:

```
dfs = dfs[dfs['date'].notna()]
```

Next, it is good to save the preprocessed file into a separate CSV file in case we need it again. We can save the dataframe into a separate CSV fileas follows:

```
dfs.to_csv('gmail.csv')
```

Data Refactoring

We noticed that the from field contains more information than we need. We just need to extract an email address from that field. Let's do some refactoring:

1. First of all, import the regular expression package:

```
import re
```

2. Next, let's create a function that takes an entire string from any columnand extracts an email address:

```
def extract_email_ID(string):
email = re.findall(r'<(.+?)>', string)if not email:
email = list(filter(lambda y: '@' in y, string.split())) return email[0] if email else np.nan
```

3. Next, let's apply the function to the from column:

```
dfs['from'] = dfs['from'].apply(lambda x: extract_email_ID(x))
```

We used the lambda function to apply the function to each and every value in the column.

1. Next, we are going to refactor the label field. The logic is simple. If an email is from your email address, then it is the *sent* email. Otherwise, it is a received email, that is, an inbox email:

```
myemail = 'itsmeskm99@gmail.com'
dfs['label'] = dfs['from'].apply(lambda x: 'sent' if x==myemail else'inbox')
```

Dropping Columns

Let's drop a column:

1. Note that the to column only contains your own email. So, we can drop this irrelevant column:

```
dfs.drop(columns='to', inplace=True)
```

2. This drops the to column from the dataframe. Let's display the first 10 entries now:

```
dfs.head(10)
```

The output of the preceding code is as follows:

	Subject	From	Date	label	thread
0	New Books: The Python Journey man + understandi…	james@sitepoint.com	2019-09-20 14.07.05 + 00:00	Inbox	1.645217e+18
1	iphone 11 Pro og iphone 11 er her	News_Europe@InsideApple.Apple.com	2019-09-20 10:33:27 + 00:00	Inbox	1.645190e+18
2	=?utf-8?Q?Save=20on=20Burlap=20Bags=20Today=21…	support@totebagfactory.com	2019-09-20 15:32:31 +00:00	Inbox	1.645210e+18
3	Hi there, looking for the best Dashain deals?…	info@email.daraz.com.np	2019-09-17 06:19:10 +00:00	Inbox	1.644916e+18
4	The file=?UTF-8?B?JOJyYW5kXOJvb2sgdGVzdC5wZGY…	noreply@box.com	2019-09-20 19:04:16 +00:00	Inbox	1.645222e+18
5	We miss you on Google Maps	noreply-local-guides@google.com	2019-09-20 11:19:56 +00:00	Inbox	1.645193e+18
6	=?utf-8?B?VGFrZSB5b3VyIHNraWxscyBObyBOaGUgbmV4…	news@edx.org	2019-09-17 13:32:49 +00:00	Inbox	1.644930e+18
7	Freelancing 101: How to market a small Business	partners@email.shopify.com	2019-09-17 14:10:12 +00:00	Inbox	1.644932e+18
8	Suresh KUMAR, your profile is getting hits	linkedin@e.linkedin.com	2019-09-17 17:29:38 +00:00	Inbox	1.644956e+18
9	=?UTF-8?Q?Forget_FOMO_=E2=80=93_you_can_go_bac…	info@flatironschool.com	2019-09-19 18:01:13 +00:00	Inbox	1.645128e+18

Data Analysis

This is the most important part of EDA. This is the part where we gain insights from the data that we have.

Let's answer the following questions one by one:

1. How many emails did I send during a given timeframe?
2. At what times of the day do I send and receive emails with Gmail?
3. What is the average number of emails per day?
4. What is the average number of emails per hour?
5. What am I mostly emailing about?

Number of emails

The answer to the first question, "How many emails did I send during a given timeframe?", can be answered as shown here:

```
print(dfs.index.min().strftime('%a, %d %b %Y %I:%M %p'))print (dfs.index. max().
strftime('%a, %d %b %Y %I:%M %p'))
print(dfs['label'].value_counts())
```

The output of the preceding code is given here:

```
Tue, 24 May 2011 11:04 AM
Fri, 20 Sep 2019 03:04 PM
inbox 32952
sent 4602
Name: label, dtype: int64
```

Average emails per day and hour

Let's answer the rest of the questions, taking a look at the average number of emails per day and per hour:

1. To do so, we will create two functions, one that counts the total number of emails per day and one that plots the average number of emails per hour:

```
def plot_number_perday_per_year(df, ax, label=None, dt=0.3, **plot_kwargs): year =
df[df['year'].notna()]['year'].values
T = year.max() - year.min()bins = int(T / dt)
weights = 1 / (np.ones_like(year) * dt * 365.25)
ax.hist(year, bins=bins, weights=weights, label=label, **plot_kwargs); ax.grid(ls=':',
color='k')
```

The preceding code creates a function that plots the average number of emails per day.

```
def plot_number_perdhour_per_year(df, ax, label=None, dt=1, smooth=False,
weight_fun=None, **plot_kwargs):
tod = df[df['timeofday'].notna()]['timeofday'].values year = df [df ['year']. notna()]
['year'].values
Ty = year.max() - year.min()T = tod.max() - tod.min() bins = int(T / dt)
if weight_fun is None:
weights = 1 / (np.ones_like(tod) * Ty * 365.25 / dt)else:
weights = weight_fun(df)if smooth:
hst, xedges = np.histogram(tod, bins=bins, weights=weights);x = np.delete(xedges, -1)
+ 0.5*(xedges[1] - xedges[0])
hst = ndimage.gaussian_filter(hst, sigma=0.75)f = interp1d(x, hst, kind='cubic')
x = np.linspace(x.min(), x.max(), 10000)hst = f(x)
ax.plot(x, hst, label=label, **plot_kwargs)else:
ax.hist(tod, bins=bins, weights=weights, label=label,
**plot_kwargs);
ax.grid(ls=':', color='k')
orientation = plot_kwargs.get('orientation')
if orientation is None or orientation == 'vertical':
ax.set_xlim(0, 24) ax.xaxis.set_major_locator(MaxNLocator(8))
ax.set_xticklabels([datetime.datetime.strptime(str(int(np.mod(ts,          24))),
"%H").strftime("%I %p")
for ts in ax.get_xticks()]);elif orientation == 'horizontal':
ax.set_ylim(0, 24) ax.yaxis.set_major_locator(MaxNLocator(8))
ax.set_yticklabels([datetime.datetime.strptime(str(int(np.mod(ts,          24))),
"%H").strftime("%I %p")
for ts in ax.get_yticks()]);
```

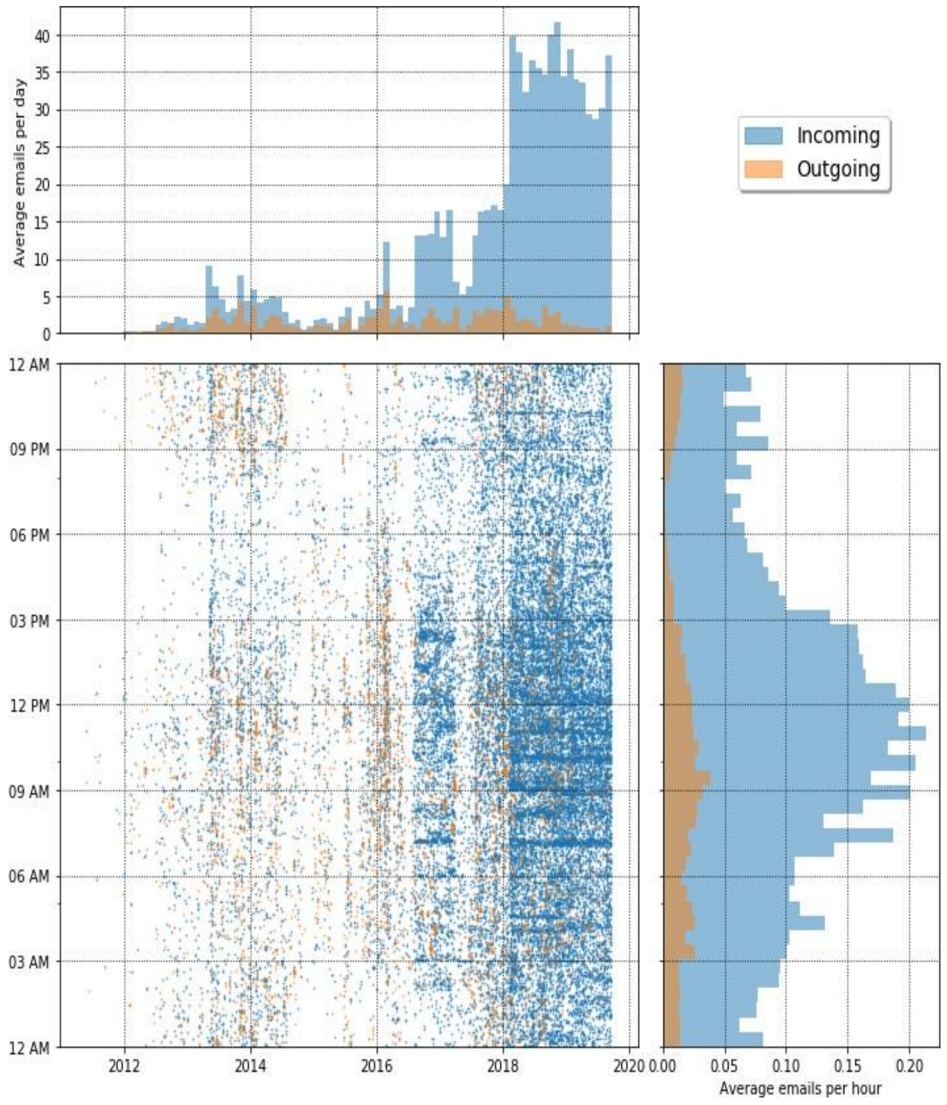


Fig. 1.7.

Number of emails per day

Let's find the busiest day of the week in terms of emails:

```
counts = dfs.dayofweek.value_counts(sort=False)counts.plot(kind='bar')
```

The output of the preceding code is as follows:

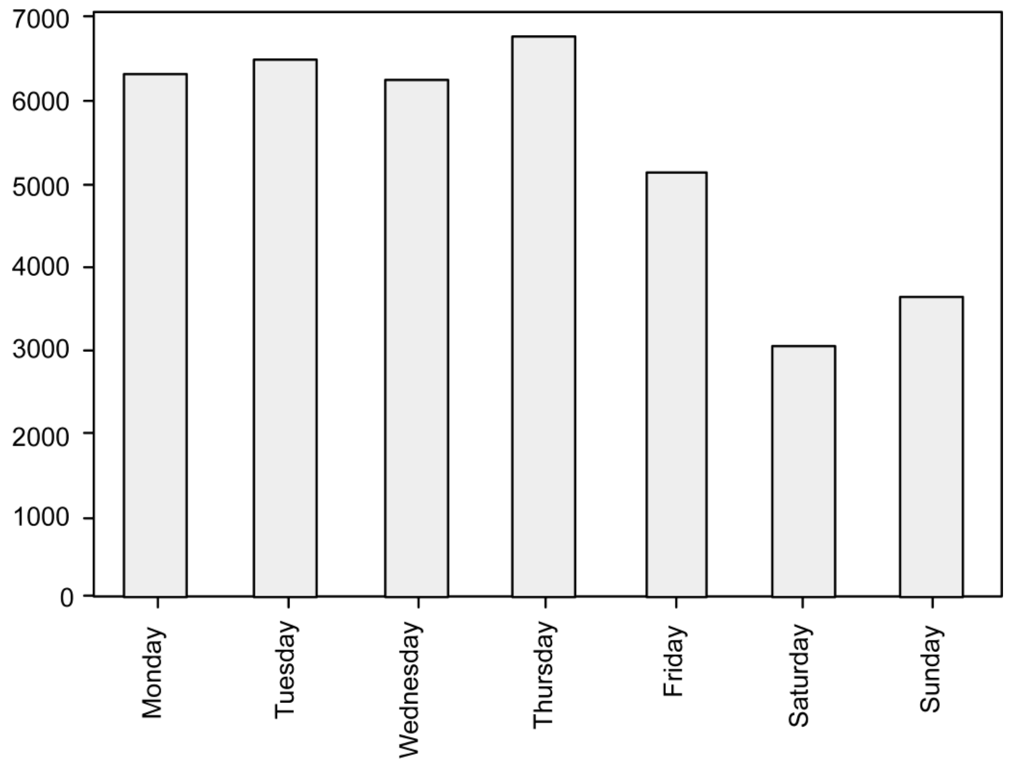


Fig. 1.8.

The preceding output shows that my busiest day is Thursday. I receive most of my emails on Thursdays. Let's go one step further and see the most active days for receiving and sending emails separately:

```
sdw = sent.groupby('dayofweek').size() / len(sent)
rdw = received.groupby('dayofweek').size() / len(received)
df_tmp = pd.DataFrame(data={'Outgoing Email': sdw, 'Incoming Email':rdw}) df_tmp.plot(kind='bar', rot=45, figsize=(8,5), alpha=0.5)
plt.xlabel('');
plt.ylabel('Fraction of weekly emails');plt.grid(ls=':', color='k', alpha=0.5)
```

The output of the preceding code is as follows:

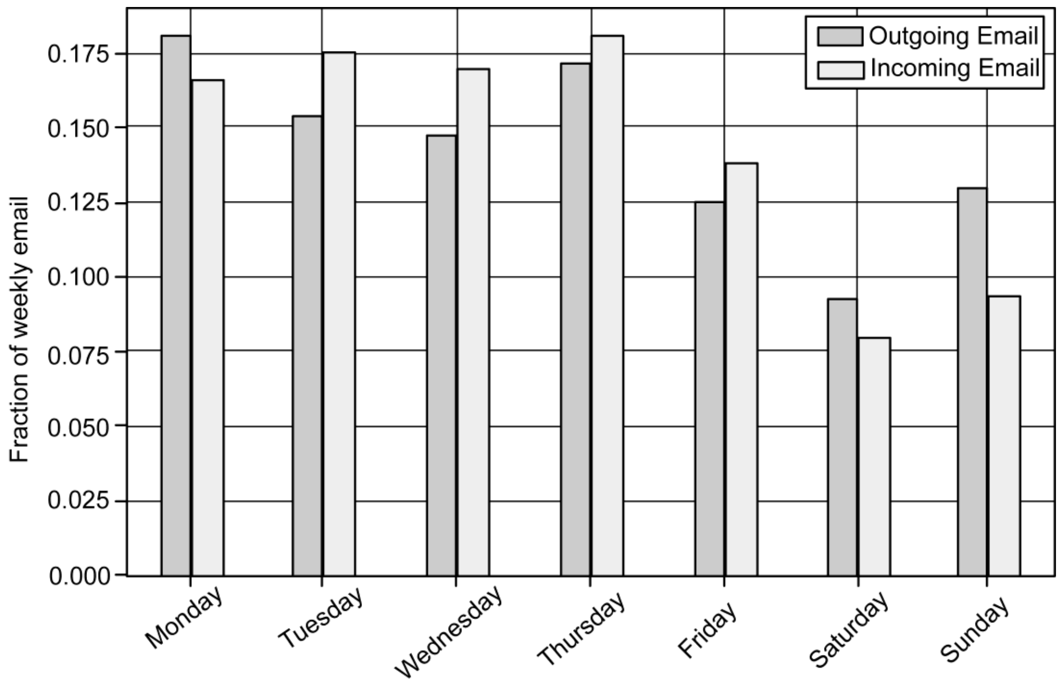


Fig. 1.9.

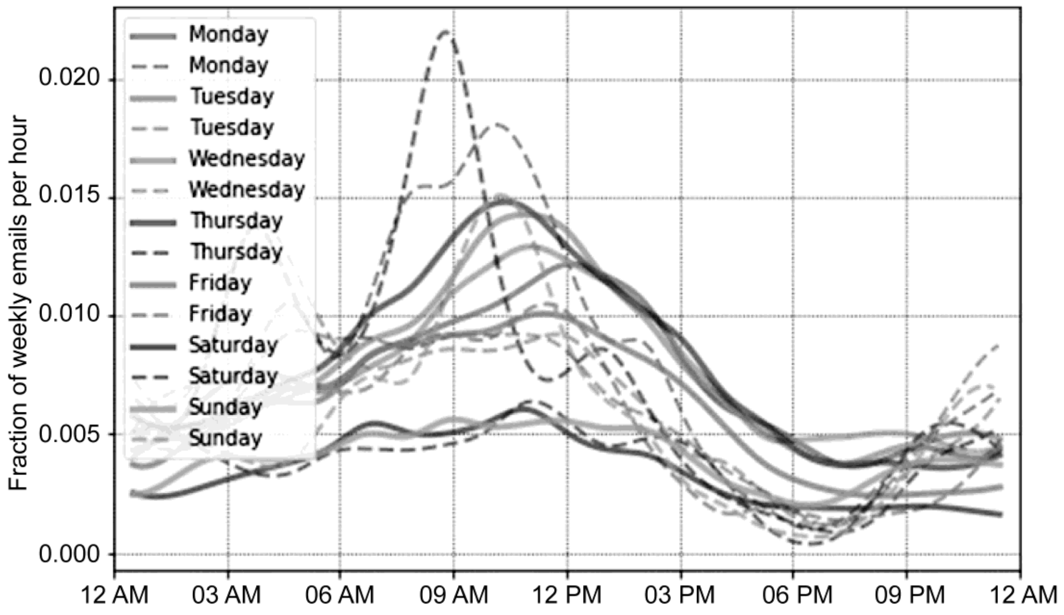


Fig. 1.10.

StudentID	ScoreSE		StudentID	ScoreSE
1	89		2	98
3	39		4	93
5	50		6	44
7	97		8	77
9	20		10	69
…	…		…	…
…	…		…	…
27	73		28	56
29	92		30	27

1.9. MERGING DATABASE

We can concatenate the data using the pandas concat() method:

dataframe = pd.concat([dataFrame1, dataFrame2], ignore_index=True)dataframe

See the difference using the following code:

pd.concat([dataFrame1, dataFrame2], axis=1)

The output of the preceding code is shown in the following screenshot:

	StudentID	Score	StudentID	Score
0	1	89	2	98
1	3	39	4	93
2	5	50	6	44
3	7	97	8	77
4	9	22	10	69
5	11	66	12	56
6	13	31	14	31
7	15	51	16	53
8	17	71	18	78

	StudentID	**Score**	**StudentID**	**Score**
9	19	91	20	93
10	21	56	22	56
11	23	32	24	77
12	25	52	26	33
13	27	73	28	56
14	29	92	30	27

Check the following dataframes:

StudentID	**ScoreSE**		**StudentID**	**ScoreSE**
9	22		2	98
11	66		4	93
13	31		6	44
15	51		8	77
17	71		10	69
…	…		…	…
…	…		…	…
27	73		28	56
29	92		30	27
StudentID	**ScoreSE**		**StudentID**	**ScoreSE**
1	39		2	98
3	49		4	93
5	55		6	44
7	77		8	77
9	52		10	69
…	…		…	…
…	…		…	…
27	23		28	56
29	49		30	27

Using the pd.merge() method witha left join

The third option is to use the pd.merge() method with the left join technique. By now, you should have understood the concept of a merge. The argument of the pd.merge() method allows us to use different types of joins.

These are the following types of joins:

The inner join takes the intersection from two or more dataframes. It corresponds to the INNER JOIN in **Structured Query Language** (**SQL**).The outer join takes the union from two or more dataframes. It corresponds to the FULL OUTER JOIN in SQL.

The left join uses the keys from the left-hand dataframe only. Itcorresponds to the LEFT OUTER JOIN in SQL.

The right join uses the keys from the right-hand dataframe only. Itcorresponds to the RIGHT OUTER JOIN in SQL.

Let's see how we can use the left outer join:

dfSE = pd.concat([df1SE, df2SE], ignore_index=True) dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='left')df

The output of the preceding code is as follows:

	StudentID	ScoreSE	ScoreML
0	9	22	52.0
1	11	66	86.0
2	13	31	41.0
3	15	51	77.0
4	17	71	73.0
5	19	91	51.0
6	21	56	86.0
7	23	32	82.0
8	25	52	92.0
9	27	73	23.0
10	29	92	49.0
11	2	98	93.0
12	4	93	44.0

	StudentID	ScoreSE	ScoreML
13	6	44	78.0
14	8	77	97.0
15	10	69	87.0
16	12	56	89.0
17	14	31	39.0
18	16	53	43.0
19	18	78	88.0
20	20	93	78.0
21	22	56	NaN
22	24	77	NaN
23	26	33	NaN
24	28	56	NaN
25	30	27	NaN

Using the pd.merge() method witha right join

This is the fourth option. Similarly to those options we've already looked at, we can use the right join to get a list of all the students who appeared in the *Machine Learning* course.

The code for doing it is as follows:

dfSE = pd.concat([df1SE, df2SE], ignore_index=True) dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='right')df

The output of this snippet is left as part of an exercise for you to complete. Check which columns have NaN values.

Using pd.merge() methods withouter join

This is the fifth option. Finally, we want to know the total number of students appearing for at least one course. This can be done using an outerjoin:

dfSE = pd.concat([df1SE, df2SE], ignore_index=True)dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='outer')df

Check the output and compare the differences with the previous output.

Merging on Index

Sometimes the keys for merging dataframes are located in the dataframesindex. In such a situation, we can pass left_index=True or right_index=True toindicate that the index should be accepted as the merge key.

	Key Value		**Group_Val**	
0	apple	0	apple	33.4
1	ball	1	ball	5.0
2	apple	2		
3	apple	3		
4	ball	4		
5	cat	5		

Merging on index is done in the following steps:

1. Consider the following two dataframes:

left1 = pd.DataFrame({'key': ['apple','ball','apple', 'apple', 'ball', 'cat'], 'value': range(6)})

right1 = pd.DataFrame({'group_val': [33.4, 5]}, index=['apple', 'ball'])

If you print these two dataframes, the output looks like thefollowing screenshot:

2. Now, let's consider two different cases. Firstly, let's try merging using an inner join, which is the default type of merge. In this case, the default merge is the intersection of the keys. Check the followingexample code:

df = pd.merge(left1, right1, left_on='key', right_index=True)df

The output of the preceding code is as follows:

	Key value	**group_val**	
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0

3. Secondly, let's try merging using an outer join, as follows:

```
df = pd.merge(left1, right1, left_on='key', right_index=True, how='outer')df
```

The output of the preceding code is as follows:

	Key value	group_val	
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0
5	cat	5	NaN

1.10. RESHAPING AND PIVOTING

This can be done with hierarchical indexing using twoactions:

Stacking: Stack rotates from any particular column in the data to therows.

Unstacking: Unstack rotates from the rows into the column.

We will look at the following example:

1. Let's create a dataframe that records the rainfall, humidity, and wind conditions of five different counties in Norway:

```
data = np.arange(15).reshape((3,5))
indexers = ['Rainfall', 'Humidity', 'Wind']
dframe1 = pd.DataFrame(data, index=indexers, columns=['Bergen', 'Oslo', 'Trondheim', 'Stavanger', 'Kristiansand'])
dframe1
```

The output of the preceding snippet is as follows:

	Bergen	Oslo	Trondheim	Stavanger	Kristiansand
Rainfall	0	1	2	3	4
Humidity	5	6	7	8	9
Wind	10	11	12	13	14

2. Now, using the stack() method on the preceding dframe1, we can pivot thecolumns into rows to produce a series:

stacked = dframe1.stack()stacked

The output of this stacking is as follows:

```
Rainfall   Bergen           0
           Oslo             1
           Trondheim        2
           Stavanger        3
           Kristiansand     4
Humidity   Bergen           5
           Oslo             6
           Trondheim        7
           Stavanger        8
           Kristiansand     9
Wind       Bergen          10
           Oslo            11
           Trondheim       12
           Stavanger       13
           Kristiansand    14
dtype: int64
```

3. The preceding series stored unstacked in the variable can be rearranged into a dataframe using the unstack() method:

stacked.unstack()

4. Now, let's unstack the concatenated frame:

series1 = pd.Series([000, 111, 222, 333], index=['zeros','ones', 'twos','threes'])

series2 = pd.Series([444, 555, 666], index=['fours', 'fives', 'sixes'])

frame2 = pd.concat([series1, series2], keys=['Number1', 'Number2'])frame2.unstack()

The output of the preceding unstacking is shown in the followingscreenshot:

	Fives	Fours	Ones	Sixs	Threes	Twos	Zeros
Number1	NaN	NaN	111.0	NaN	333.0	222.0	0.0
Number2	555.0	444.0	NaN	666.0	NaN	NaN	NaN

1.11. TRANSFORMATION TECHNIQUES

Let's dive more into how we can perform other types of data transformations including cleaning,filtering, deduplication, and others.

Performing Data Deduplication

It is very likely that your dataframe contains duplicate rows. Removing them is essential to enhance the quality of the dataset. This can be done with the following steps:

1. Let's consider a simple dataframe, as follows:

```
frame3 = pd.DataFrame({'column 1': ['Looping'] * 3 + ['Functions'] * 4,'column 2':
[10, 10, 22, 23, 23, 24, 24]})
```

The preceding code creates a simple dataframe with two columns.You can clearly see from the following screenshot that in both columns, there are some duplicate entries:

	Column 1	Column 2
0	Looping	10
1	Looping	10
2	Looping	22
3	Functions	23
4	Functions	23
5	Functions	24
6	Functions	24

2. The pandas dataframe comes with a duplicated() method that returns a Boolean series stating which of the rows are duplicates:

frame3.duplicated()

The output of the preceding code is pretty easy to interpret:

	0	False
	1	True
	2	False
	3	False
	4	True
	5	False
	6	True
	dtype: bool	

3. Now, we can drop these duplicates using the drop_duplicates() method:

frame4 = frame3.drop_duplicates()frame4

The output of the preceding code is as follows:

		Column 1	**Column 2**
	0	Looping	10
	2	Looping	22
	3	Functions	23
	5	Functions	24

Note that rows 1, 4, and 6 are removed. Basically, both the duplicated() and drop_duplicates() methods consider all of the columns for comparison. Instead of all the columns, we could specify any subset of the columns to detect duplicated items.

4. Let's add a new column and try to find duplicated items based on the second column:

frame3['column 3'] = range(7)

frame5 = frame3.drop_duplicates(['column 2'])frame5

The output of the preceding snippet is as follows:

	Column 1	Column 2	Column 3
0	Looping	10	0
2	Looping	22	2
3	Functions	23	3
5	Functions	24	5

Note that both the duplicated and drop_duplicates methods keep the first observed value during the duplication removal process. If we pass the take_last=True argument, the methods return the last one.

Replacing Values

Often, it is essential to find and replace some values inside a dataframe. This can be done with the following steps:

1. We can use the replace method in such cases:

```
import numpy as np
replaceFrame = pd.DataFrame({'column 1': [200., 3000., -786., 3000., 234.,
444., -786., 332., 3332. ], 'column 2': range(9)})
replaceFrame.replace(to_replace =-786, value= np.nan)
```

The output of the preceding code is as follows:

	Column 1	Column 2
0	200.0	0
1	3000.0	1
2	NaN	2
3	3000.0	3
4	234.0	4
5	444.0	5
6	NaN	6
7	332.0	7
8	3332.0	8

Note that we just replaced one value with the other values. We can also replace multiple values at once.

2. In order to do so, we display them using a list:

```
replaceFrame = pd.DataFrame({'column 1': [200., 3000., -786., 3000., 234.,
444., -786., 332., 3332. ], 'column 2': range(9)})
replaceFrame.replace(to_replace =[-786, 0], value= [np.nan, 2])
```

Handling Missing Data

Whenever there are missing values, a NaN value is used, which indicates that there is no value specified for that particular index. There could be several reasons why a value could be NaN:

It can happen when data is retrieved from an external source and there are some incomplete values in the dataset.

It can also happen when we join two different datasets and some values are not matched.

Missing values due to data collection errors.

When the shape of data changes, there are new additional rows or columns that are not determined.

Reindexing of data can result in incomplete data.

Let's see how we can work with the missing data:

1. Let's assume we have a dataframe as shown here:

```
data = np.arange(15, 30).reshape(5, 3)
dfx = pd.DataFrame(data, index=['apple', 'banana', 'kiwi', 'grapes', 'mango'], columns=['store1', 'store2', 'store3'])
dfx
```

And the output of the preceding code is as follows:

⇥		**Store 1**	**Store 2**	**Store 3**
	apple	15	16	17
	banana	18	19	20
	kiwi	21	22	23
	grapes	24	25	26
	mango	27	28	29

2. Let's add some missing values to our dataframe:

dfx['store4'] = np.nan dfx.loc['watermelon'] = np.arange(15, 19) dfx.loc['oranges'] = np.nan

dfx['store5'] = np.nan dfx['store4']['apple'] = 20.dfx

And the output will now look like the following screenshot:

	store 1	store 2	store 3	store 4	store 5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	NaN	NaN
kiwi	21.0	22.0	23.0	NaN	NaN
grapes	24.0	25.0	26.0	NaN	NaN
mango	27.0	28.0	29.0	NaN	NaN
watermelon	15.0	16.0	17.0	18.0	NaN
oranges	NaN	NaN	NaN	NaN	NaN

Note that we've added two more stores, store4 and store5, and two more types of fruits, watermelon and oranges. Assume that we know how many kilosof apples and watermelons were sold from store4, but we have not collected any data from store5. Moreover, none of the stores reported sales of oranges. We are quite a huge fruit dealer, aren't we?

NaN values in pandas objects

We can use the is null() function from the pandas library to identify NaN values:

1. Check the following example:

dfx.isnull()

The output of the preceding code is as follows:

	store 1	store 2	store 3	store 4	store 5
apple	False	False	False	False	True
banana	False	False	False	True	True
kiwi	False	False	False	True	True
grapes	False	False	False	True	True
mango	False	False	False	True	True
watermelon	False	False	False	False	True
Oranges	True	True	True	True	True

Note that the True values indicate the values that are NaN. Pretty obvious, right? Alternatively, we can also use the notnull() method to do the same thing. The only difference would be that the function will indicate True for the values which are not null.

2. Check it out in action:

```
dfx.notnull()
```

And the output of this is as follows:

	store 1	**store 2**	**store 3**	**store 4**	**store 5**
apple	True	True	True	True	False
banana	True	True	True	False	False
kiwi	True	True	True	False	False
grapes	True	True	True	False	False
mango	True	True	True	False	False
watermelon	True	True	True	True	False
oranges	False	False	False	False	False

Compare these two tables. These two functions, notnull() andisnull(), are the complement to each other.

3. We can use the sum() method to count the number of NaN values in each store. How does this work, you ask? Check the following code:

```
dfx.isnull().sum()
```

And the output of the preceding code is as follows:

```
store1 1

store2 1

store3 1

store4 5

store5 7 dtype: int64
```

The fact that *True* is 1 and *False* is 0 is the main logic for summing. The preceding results show that one value was not reported by store1, store2, and store3. Five values were not reported by store4 andseven values were not reported by store5.

4. We can go one level deeper to find the total number of missing values:

```
dfx.isnull().sum().sum()
```

And the output of the preceding code is as follows:

```
15
```

This indicates 15 missing values in our stores. We can use an alternative way to find how many values were actually reported.

5. So, instead of counting the number of missing values, we can count the number of reported values:

```
dfx.count()
```

And the output of the preceding code is as follows:

```
store1 6
store2 6
store3 6
store4 2
store5 0 dtype: int64
```

Pretty elegant, right? We now know two different ways to find the missingvalues, and also how to count the missing values.

Dropping Missing Values

One of the ways to handle missing values is to simply remove them from our dataset. We have seen that we can use the isnull() and notnull() functions from the pandas library to determine null values:

```
dfx.store4[dfx.store4.notnull()]
```

The output of the preceding code is as follows:

```
apple 20.0
watermelon 18.0
```

Name: store4, dtype: float64

Dropping by Rows

We can also drop rows that have NaN values. To do so, we can use the how=all argument to drop only those rows entire values are entirely NaN:

dfx.dropna(how='all')

The output of the preceding code is as follows:

	store 1	store 2	store 3	store 4	store 5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	NaN	NaN
kiwi	21.0	22.0	23.0	NaN	NaN
grapes	24.0	25.0	26.0	NaN	NaN
mango	27.0	28.0	29.0	NaN	NaN
watermelon	15.0	16.0	17.0	18.0	NaN

Note that only the orange rows are removed because those entire rows contained NaN values.

Dropping by Columns

Furthermore, we can also pass axis=1 to indicate a check for NaN bycolumns.

dfx.dropna(how='all', axis=1)

And the output of the preceding code is as follows:

	store 1	store 2	store 3	store 4
apple	15.0	16.0	17.0	20.0
banana	18.0	19.0	20.0	NaN
kiwi	21.0	22.0	23.0	NaN
grapes	24.0	25.0	26.0	NaN
mango	27.0	28.0	29.0	NaN
watermelon	15.0	16.0	17.0	18.0
oranges	NaN	NaN	NaN	NaN

Note that store5 is dropped from the dataframe. By passing in axis=1, we are instructing pandas to drop columns if all the values in the column are NaN. Furthermore, we can also pass another argument, thresh, to specify a minimum number of NaNs that must exist before the column should be dropped:

dfx.dropna(thresh=5, axis=1)

And the output of the preceding code is as follows:

	store 1	store 2	store 3
apple	15.0	16.0	17.0
banana	18.0	19.0	20.0
kiwi	21.0	22.0	23.0
grapes	24.0	25.0	26.0
mango	27.0	28.0	29.0
watermelon	15.0	16.0	17.0
oranges	NaN	NaN	NaN

Compared to the preceding, note that even the store4 column is now dropped because it has more than five NaN values.

Filling Missing Values

We can use the fillna() method to replace NaN values with any particular values.

Check the following example:

filledDf = dfx.fillna(0)filledDf

The output of the preceding code is shown in the following screenshot:

	store 1	store 2	store 3	store 4	store 5
apple	15.0	16.0	17.0	20.0	0.0
banana	18.0	19.0	20.0	0.0	0.0
kiwi	21.0	22.0	23.0	0.0	0.0
grapes	24.0	25.0	26.0	0.0	0.0
mango	27.0	28.0	29.0	0.0	0.0
watermelon	15.0	16.0	17.0	18.0	0.0
oranges	0.0	0.0	0.0	0.0	0.0

Note that in the preceding dataframe, all the NaN values are replaced by 0. Replacing the values with 0 will affect several statistics including mean, sum, and median.

Check the difference in the following two examples:

```
dfx.mean()
```

And the output of the preceding code is as follows:

```
store1 20.0
store2 21.0
store3 22.0
store4 19.0
store5 NaN dtype: float64
```

And the output we get is as follows:

```
store1 17.142857
store2 18.000000
store3 18.857143
store4 5.428571
store5 0.000000dtype: float64
```

Benefits of Data Transformation

Let's try to list these benefits:

Data transformation promotes interoperability between several applications. The main reason for creating a similar format and structure in the dataset is that it becomes compatible with other systems.

Comprehensibility for both humans and computers is improved when using better-organized data compared to messier data.

Data transformation ensures a higher degree of data quality and protects applications from several computational challenges such as null values, unexpected duplicates, and incorrect indexings, as well as incompatible structures or formats.

Data transformation ensures higher performance and scalability for modern analytical databases and dataframes.

Mean/average

The mean, or average, is a number around which the observed continuous variables are distributed. This number estimates the value of the entire dataset. Mathematically, it is the result of the division of the sum of numbers by the number of integers in the dataset.

Let x be a set of integers:

$$x = (12, 2, 3, 5, 8, 9, 6, 4, 2)$$

Hence, the mean value of x can be calculated as follows:

$$\text{Mean } (x) = \frac{12 + 2 + 3 + 5 + 8 + 9 + 6 + 4 + 2}{9} = 5.66$$

Median

Given a dataset that is sorted either in ascending or descending order, the median divides the data into two parts. The general formula for calculating the median is as follows:

$$\text{Median position} = \frac{(n + 1)}{2} \text{th observation}$$

Here, n is the number of items in the data. The steps involved in calculating the median are as follows:

1. Sort the numbers in either ascending or descending order.
2. If n is odd, find the $(n + 1) / 2$th term. The value corresponding to this termis the median.
3. If n is even, find the $(n + 1) / 2$th term. The median value is the average of numbers on either side of the median position.

For a set of integers such as x, we must arrange them in ascending order and then select the middle integer.

In ascending order = (2, 2, 3, 4, 5, 6, 8, 9, 12).Here, the median is 5.

Mode

The mode is the integer that appears the maximum number of times in the dataset. It happens to be the value with the highest frequency in the dataset. In the x dataset in the median example, the mode is 2 because it occurs twice in the set.

Python provides different libraries for operating descriptive statistics in the dataset. Commonly used libraries are pandas, numpy, and scipy. These measures of central tendency can simply be calculated by the numpy and pandas functionalities.

Here is a dataset of automobiles that enlists different features and attributes of cars, such as symboling, normalized losses, aspiration, and many others, an analysis of which will provide some valuable insight and findings in relation to automobiles in this dataset.

Let's begin by importing the datasets and the Python libraries required:

```
import pandas as pdimport numpy as np
```

Now, let's load the automobile database:

```
df = pd.read_csv("data.csv")df.head()
```

The output of the code is given here:

Symboling	**normalized losses**	**make**		**aspiration**	**number of doors**	**body style**	**drive-wheels**	**engine location**	**wheel-base**	**length**	**width**	**height**	**curb weight**	**engine-type**	**Num of cylinders**	**engine -size**
0	3	122	alfa-	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130
1	3	122	romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130
2	1	122	alfa-	std	two	hatchback	rwd	front	94.5	0.822681	0.909722	52,4	2823	ohcv	six	152
3	2	164	romero	std	four	sedan	fwd	front	99.8	0.848630	0.919444	54.3	2337	ohc	four	109
4	2	164	alfa	std	four	sedan	4wd	front	99.4	0.848630	0.922222	54.3	2824	ohc	five	136
			romero													
			audi													
			audi													

Standard Deviation

Different Python libraries have functions to get the standard deviation of the dataset. The NumPy library has the numpy.std(dataset) function. The statistics library has the statistics.stdev(dataset). function. Using the pandas library, we calculate the standard deviation in our df data frame using the df.std() function:

```
#standard variance of dataset using std() functionstd_dev =df.std()
print(std_dev)
# standard variance of the specific column sv_height=df.loc[:,"height"].std()
print(sv_height)
```

The output of the preceding code is as follows:

symboling normalized-losses wheel-base length width height curb-weight engine-size bore stroke compression-ratio horsepower peak-rpm city-mpg highway-mpg price city-L/100 km diesel gas dtype: float64 2.44782216129631	1.254802 31.996250 6.066366 0.059213 0.029187 2.447822 517.296727 41.546834 0.268072 0.319256 4.004965 37.365700 478.113805 6.423220 6.815150 7947.066342 2.534599 0.300083 0.300083

Variance

Variance is the square of the average/mean of the difference between each value in the dataset with its average/mean; that is, it is the square of standard deviation.

Different Python libraries have functions to obtain the variance of the dataset. The NumPy library has the numpy.var(dataset) function. The statistics library has the statistics.variance(dataset) function. Using the pandas library, we calculate the variance in our df data frame using the df.var() function:

```
# variance of dataset using var() functionvariance=df.var()
print(variance)
# variance of the specific column var_height=df.loc[:,"height"].var()print(var_height)
```

The output of the preceding code is as follows:

```
symboling              1.574527e+00
normalized-losses      1.023760e+03
wheel-base             3.680079e+01
length                 3.506151e–03
width                  8.518865e–04
height                 5.991833e+00
curb-weight            2.675959e+05
engine-size            1.726139e+03
bore                   7.186252e–02
stroke                 1.019245e–01
compression-ratio      1.603975e+01
horsepower             1.396195e+03
peak-rpm               2.285928e+05
city-mpg               4.125776e+01
highway-mpg            4.644627e+01
price                  6.315586e+07
city-L/100 km          6.424193e+00
diesel                 9.004975e–02
gas                    9.004975e–02
dtype: float64
5.991833333333338
```

Skewness

In probability theory and statistics, skewness is a measure of the asymmetryof the variable in the dataset about its mean. The skewness value can be positive or negative, or undefined. The skewness value tells us whether the data is skewed or symmetric. Here's an illustration of a positively skewed dataset, symmetrical data, and some negatively skewed data:

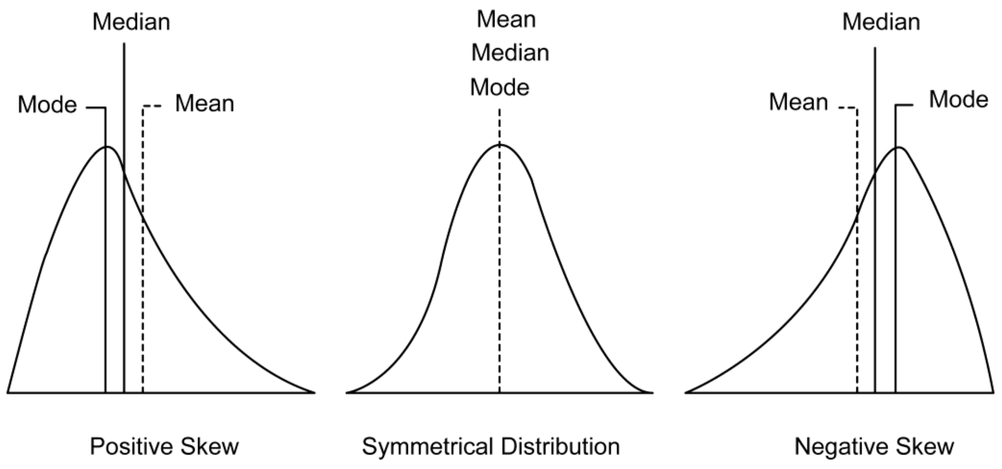


Fig. 1.11.

Note the following observations from the preceding diagram:

The graph on the right-hand side has a tail that is longer than the tail on the right-hand side. This indicates that the distribution of the data is skewed to the left. If you select any point in the left-hand longer tail, the mean is less than the mode. This condition is referred to as **negativeskewness**.

The graph on the left-hand side has a tail that is longer on the right- hand side. If you select any point on the right-hand tail, the mean valueis greater than the mode. This condition is referred to as

Positive Skewness.

The graph in the middle has a right-hand tail that is the same as the left-hand tail. This condition is referred to as a **symmetrical condition**.

Different Python libraries have functions to get the skewness of the dataset. The SciPy library has a scipy.stats.skew(dataset) function. Using the pandas library, we can calculate the skewness in our df data frame using the df.skew()function.

Here, in our data frame of automobiles, let's get the skewness using the df.skew() function:

df.skew()

The output of the preceding code is as follows:

```
symboling              0.204275
normalized-losses      0.209007
wheel-base             1.041170
length                 0.154086
width                  0.900685
height                 0.064134
curb-weight            0.668942
engine-size            1.934993
bore                   0.013419
stroke                –0.669515
compression-ratio      2.682640
horsepower             9.985047
peak-rpm               0.073094
city-mpg               0.673533
highway-mpg            0.549104
price                  1.812335
dtype: float64
```

Kurtosis

Basically, kurtosis is a statistical measure that illustrates how heavily the tails of distribution differ from those of a normal distribution. This technique can identify whether a given distribution contains extreme values.

Kurtosis, unlike skewness, is not about the peakedness or flatness. It is the measure of outlier presence in a given distribution. Both high and low kurtosis are an indicator that data needs further investigation. The higher the kurtosis, the higher the outliers.

Types of Kurtosis

There are three types of kurtosis - mesokurtic, leptokurtic, and platykurtic. Let's look at these one by one:

Mesokurtic: If any dataset follows a normal distribution, it follows a mesokurtic distribution. It has kurtosis around 0.

Leptokurtic: In this case, the distribution has kurtosis greater than 3 and the fat tails indicate that the distribution produces more outliers. **Platykurtic:** In this case, the distribution has negative kurtosis and the tails are very thin compared to the normal distribution.

All three types of kurtosis are shown in the following diagram:

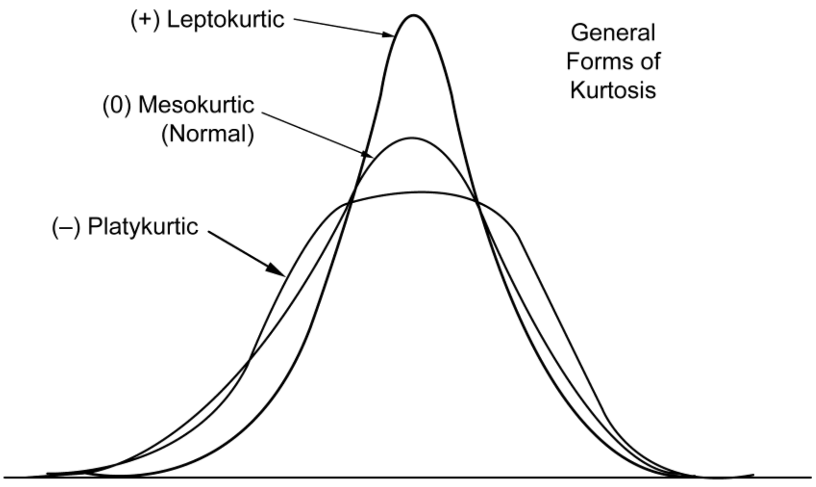


Fig. 1.12.

Calculating Percentiles

Percentiles measure the percentage of values in any dataset that lie below a certain value. In order to calculate percentiles, we need to make sure our list is sorted. An example would be if you were to say that the 80th percentile of data is 130: then what does that mean? Well, it simply means that 80% of the values lie below 130. Pretty easy, right? We will use the following formula for this:

$$\left.\begin{array}{r}\text{The formula for calculating}\\ \text{percentile of X}\end{array}\right\} = \frac{\text{Number of values less than X}}{\text{Total number of observations}} \times 100$$

Suppose we have the given data: 1, 2, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10. Then the percentile value of 4 = (4 / 12) * 100 = 33.33%.

This simply means that 33.33% of the data is less than 4.

Quartiles

Given a dataset sorted in ascending order, quartiles are the values that split the given dataset into quarters. Quartiles refer to the three data points that divide the given dataset into four equal parts, such that each split makes 25% of the dataset. In terms of percentiles, the 25th percentile is referred to as the first quartile (Q1), the 50th percentile is referred to as the second quartile (Q2), and the 75th percentile is referred to as the third quartile (Q3).

Based on the quartile, there is another measure called inter-quartile rangethat also measures the variability in the dataset. It is defined as follows:

$$\text{IQR} = Q_3 - Q_1$$

IQR is not affected by the presence of outliers. Let's get the IQR for the price column from the same dataframe we have been using so far:

rice = df.price.sort_values() Q1 = np.percentile(price, 25) Q2 = np.percentile(price, 50) Q3 = np.percentile(price, 75)

IQR = Q3 - Q1IQR

The output of the preceding snippet is as follows:

8718.5

1.12. GROUPING DATASETS

Understanding groupby()

During the data analysis phase, categorizing a dataset into multiple categories or groups is often essential. We can do such categorization usingthe pandas library. The pandas groupby function is one of the most efficient and time-saving features for doing this. Groupby provides functionalities that allow us to split-apply-combine throughout the dataframe; that is, this function can be used for splitting, applying, and combining dataframes.

Group by mechanics

To work with groupby functionalities, we need a dataset that has multiple numerical as well as categorical records in it so that we can group by different categories and ranges.

Let's take a look at a dataset of automobiles that enlists the different features and attributes of cars, such as symbolling, normalized-losses, make, aspiration, body- style, drive-wheels, engine-location, and many others. Let's get started:

1. Let's start by importing the required Python libraries and datasets:

```
import pandas as pd
df = pd.read_csv("/content/automobileEDA.csv")df.head()
```

The output of the preceding code is as follows:

Symboling	normalized losses	make		aspiration	number of doors	body style	drive-wheels	engine location	wheel-base	length	width	height	curb weight	engine type	sum of cylinders	engine size
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130
1	3	122	alfa romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130
2	1	122	alfa romero	std	two	hatchback	rwd	front	94.5	0.822681	0.909722	52,4	2823	ohcv	six	152
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	0.919444	54.3	2337	ohc	four	109
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	0.922222	54.3	2824	ohc	five	136

As you can see, there are multiple columns with categorical variables.

2. Using the groupby() function lets us group this dataset on the basis of the body-style column:

```
df.groupby('body-style').groups.keys()
```

The output of the preceding code is as follows:

```
dict_keys(['convertible', 'hardtop', 'hatchback', 'sedan', 'wagon'])
```

From the preceding output, we know that the body-style column has five unique values, including convertible, hardtop, hatchback, sedan, and wagon.

3. Now, we can group the data based on the body-style column. Next, let's print the values contained in that group that have the body-style value ofconvertible. This can be done using the following code:

```
# Group the dataset by the column body-stylestyle = df.groupby('body-style')
# Get values items from group with value convertiblestyle.get_group("convertible")
```

The output of the preceding code is as follows:

Symboling	normalized losses	make		aspiration	number of doors	body style	drive-wheels	engine location	wheel-base	length	width	height	curb weight	engine type	sum of cylinders	engine size
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130
1	3	122	alfa romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130
69	3	142	Mercedes	std	two	convertible	rwd	front	96.6	0.866410	0.979167	50.8	3685	ohcv	eight	234
125	3	122	benz	std	two	convertible	rwd	rear	89.5	0.811629	0.902778	– 51.6	2800	ohcf	six	194
168	2	134	porsche	std	two	convertible	rwd	front	98.4	0.846708	0.911111	53.0	2975	ohc	four	146
185	2	122	Toyota	std	two	convertible	fwd	front	94.5	0.765497	0.891667	55.6	2254	ohc	four	109
			volkswagen													

Max and Min

Let's compute the maximum and minimum entry for each group. Here, we will find the maximum and minimum for the normalized-losses column:

```
# max() will print the maximum entry of each groupstyle['normalized-losses'].max()
# min() will print the minimum entry of each groupstyle['normalized-losses'].min()
```

The output of the preceding code is as follows:

```
body-style convertible 122
hardtop 93
hatchback 65
sedan 65
wagon 74
Name: normalized-losses, dtype: int64
```

1.13. DATA AGGREGATION

Aggregation is the process of implementing any mathematical operation on a dataset or a subset of it. Aggregation is one of the many techniques in pandas that's used to manipulate the data in the dataframe for data analysis.

The Dataframe.aggregate() function is used to apply aggregation across one or more columns. Some of the most frequently used aggregations are as follows:

sum: Returns the sum of the values for the requested axis

min: Returns the minimum of the values for the requested axis

- **max:** Returns the maximum of the values for the requested axis We can apply aggregation in a DataFrame, df, as df.aggregate() or df.agg().

Since aggregation only works with numeric type columns, let's take some of the numeric columns from the dataset and apply some aggregation functions to them:

```
# new dataframe that consist length,width,height,curb-weight and price
new_dataset = df.filter(["length","width","height","curb-weight","price"],axis=1)
new_dataset
```

The output of the preceding code snippet is as follows:

	Length	**Width**	**Height**	**Curb-weight**	**Price**
0	0.811148	0.890278	48.8	2548	13495.0
1	0.811148	0.890278	48.8	2548	16500.0
2	0.822681	0.909722	52.4	2823	16500.0
3	0.848630	0.919444	54.3	2337	13950.0
4	0.848630	0.922222	54.3	2824	17450.0
…					
196	0.907256	0.956944	55.5	2952	16845.0
197	0.907256	0.955556	55.5	3049	19045.0
198	0.907256	0.956944	55.5	3012	21485.0
199	0.907256	0.956944	55.5	3217	22470.0
200	0.907256	0.956944	55.5	3062	22625.0
201 rows × 5 columns					

Next, let's apply a single aggregation to get the mean of the columns. To do this, we can use the agg() method, as shown in the following code:

```
# applying single aggregation for mean over the columns new_dataset.agg("mean", axis="rows")
```

The output of the preceding code is as follows:

```
length 0.837102

width 0.915126

height 53.766667

curb-weight 2555.666667
```

```
price 13207.129353
dtype: float64
```

	Length	Width	Height	Curb-weight
Max	NaN	1.0000	NaN	NaN
Min	0.678039	0.8375	47.8	NaN
Sum	168.257568	NaN	10807.1	513689.0

Group-wise operations

The most important operations groupBy implements are aggregate, filter, transform, and apply. An efficient way of implementing aggregation functions in the dataset is by doing so after grouping the required columns. The aggregated function will return a single aggregated value for each group.

Let's group the DataFrame, df, by body-style and drive-wheels and extract stats from each group by passing a dictionary of aggregation functions:

```
# Group the data frame df by body-style and drive-wheels and extract stats fromeach group
df.groupby(
["body-style","drive-wheels"]
).agg(
{
'height':min, # minimum height of car in each group 'length': max, # maximum length of car in each group 'price': 'mean', # average price of car in each group
}
)
```

The output of the preceding code is as follows:

Body-style	**Drive wheels**	**Height**	**Length**	**Price**
Convertible	**fwd**	55.6	0.765497	11595.000000
	rwd	48.8	0.866410	23949.600000
Hardtop	**fwd**	53.3	0.780394	8249.000000
	rwd	51.6	0.957232	24202.714286
Hatchback	**4wd**	55.7	0.755887	7603.000000
	fwd	49.4	0.896684	8396.387755
	rwd	49.6	0.881788	14337.777778
Sedan	**4wd**	54.3	0.848630	12647.333333
	fwd	50.6	0.925997	9811.800000
	rwd	47.8	1.000000	21711.833333
Wagon	**4wd**	54.9	0.834214	9095.750000
	fwd	53.0	0.925997	9997.333333
	rwd	54.1	0.955790	16994.222222

The preceding code groups the dataframe according to body-style and then driver-wheels. Then, the aggregate functions are applied to the height, length, and price columns, which return the minimum height, maximum length, and average price in the respective groups.

```
# create dictionary of aggregationsaggregations=(
{
'height':min, # minimum height of car in each group 'length': max, # maximum length of car in each group 'price': 'mean', # average price of car in each group
}
)
```

```
# implementing aggregations in groupsdf.groupby(
["body-style","drive-wheels"]
).agg(aggregations)
```

The output of the preceding code is as follows:

Body-style	Drive wheels	Sum	Mean	Std
Convertible	**fwd**	11595.0	11595.000000	NaN
	rwd	119748.0	23949.600000	11165.099700
Hardtop	**fwd**	8249.0	8249.000000	NaN
	rwd	169419.0	24202.714286	14493.311190
Hatchback	**4wd**	7603.0	7603.000000	NaN
	fwd	411423.0	8396.387755	3004.675695
	rwd	258080.0	14337.777778	3831.795195
Sedan	**4wd**	37942.0	12647.333333	4280.814681
	fwd	539649.0	9811.800000	3519.517598
	rwd	781626.0	21711.833333	9194.820239
Wagon	**4wd**	36383.0	9095.750000	1775.652063
	fwd	119968.0	9997.333333	3584.185551
	rwd	152948.0	16994.222222	4686.703313

Group-wise transformations

Working with groupby() and aggregation, you must have thought, why can't we group data, apply aggregation, and append the result into the dataframe directly? Is it possible to do all this in a single step? Yes, it is.

Performing a transformation on a group or a column returns an object that is indexed by the same axis length as itself. It is an operation that's used in conjunction with groupby(). The aggregation operation has to return a reduced version of the data, whereas the transformation operation can return a transformed version of the full data. Let's take a look:

1. Let's begin by using a simple transformation function to increase the price of each car by 10% using the lambda function:

```
df["price"]=df["price"].transform(lambda x:x + x/10)df.loc[:,'price']
```

The output of the preceding code is as follows:

```
0 14844.5
1 18150.0
2 18150.0
3 15345.0
4 19195.0
...196 18529.5
197 20949.5
198 23633.5
199 24717.0
200 24887.5
Name: price, Length: 201, dtype: float64
```

2. Let's observe the average price of cars for each grouping by body-style and drive-wheels:

```
df.groupby(["body-style","drive-wheels"])["price"].transform('mean')
```

The output of the preceding code is as follows:

```
0 26344.560000
1 26344.560000
 15771.555556
3 10792.980000
4 13912.066667
...
```

```
196 23883.016667
197 23883.016667
198 23883.016667
199 23883.016667
```

3. Now, create a new column for an average price in the originaldataframe:

```
df["average-price"]=df.groupby(["body-style","drive-wheels"])
["price"].transform('mean')
# selecting columns body-style,drive-wheels,price and average-price df.loc[:,["body-style","drive-wheels","price","average-price"]]
```

The output of the preceding code is as follows:

	body-style	**drive wheels**	**price**	**average-price**
0	convertible	rwd	14844.5	26344.560000
1	convertible	rwd	18150.0	26344.560000
2	hatchback	rwd	18150.0	15771.555556
3	sedan	fwd	15345.0	10792.980000
4	sedan	4wd	19195.0	13912.066667
…	…	…	…	…
196	sedan	rwd	18529.5	23883.016667
197	sedan	rwd	20949.5	23883.016667
198	sedan	rwd	23633.5	23883.016667
199	sedan	rwd	24717.0	23883.016667
200	sedan	rwd	24887.5	23883.016667
201 rows × 4 columns				

1.14. PIVOT TABLES

The pandas.pivot_table() function creates a spreadsheet-style pivot table as a dataframe. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the resulting dataframe.

The simplest pivot tables must have a dataframe and an index/list of the index. Let's take a look at how to do this:

1. Let's make a pivot table of a new dataframe that consists of the body-style, drive-wheels, length, width, height, curb-weight, and price columns:

new_dataset1 = df.filter(["body-style","drive-wheels",

"length","width","height","curb-

weight","price"],axis=1)

#simplest pivot table with dataframe df and index body-style table = pd.pivot_table(new_dataset1, index =["body-style"])table

The output of the preceding code is as follows:

Body-style	**Curb-weight**	**Height**	**Length**	**Price**	**Width**
Convertible	2801.666667	51.433333	0.818757	24079.550000	0.910880
Hardtop	2810.625000	52.850000	0.850252	24429.350000	0.925174
Hatchback	2322.852941	52.133824	0.799078	10953.185294	0.904228
Sedan	2625.893617	54.387234	0.855583	15905.730851	0.921070
Wagon	2784.240000	56.728000	0.871235	13609.156000	0.920222

The output table is similar to how we group a dataframe with respect to body-style. The values in the preceding table are the mean of the values in the corresponding category.

2. Now, design a pivot table with the new_dataset1 dataframe and make body- style and drive-wheels as an index. Note that providing multiple indexes will make a grouping of the dataframe first and then summarize the data:

```
#pivot table with dataframe df and index body-style and drive-wheels
table = pd.pivot_table(new_dataset1, index =["body-style","drive-wheels"])table
```

The output of the preceding code is as follows:

Body-style	drive wheels	Curb-weight	Height	Length	Price	Width
Convertible	fwd	2254.000000	55.600000	0.765497	12754.500000	0.891667
	rwd	2911.200000	50.600000	0.829409	26344.560000	0.914722
Hardtop	fwd	2008.000000	53.300000	0.780394	9073.900000	0.886111
	rwd	2925.285714	52.785714	0.860232	26622.985714	0.930754
Hatchback	4wd	2240.000000	55.700000	0.755887	8363.300000	0.886111
	fwd	2181.551020	52.442857	0.787818	9236.026531	0.898214
	rwd	2712.11111	51.094444	0.832132	15771.555556	0.921605
Sedan	4wd	2573.000000	54.300000	0.833894	13912.066667	0.921963
	fwd	2313.018182	53.956364	0.828404	10792.980000	0.908182
	rwd	3108.305556	55.052778	0.898913	23883.016667	0.941435
Wagon	4wd	2617.500000	57.000000	0.824844	1005.325000	0.895833
	fwd	2464.333333	56.008333	0.843064	10997.066667	0.910185
	rwd	3284.888889	57.566667	0.929414	18693.644444	0.944444

3. We can also apply a different aggregation function to different columns:

```
table = pd.pivot_table(new_dataset1, values=['price','height','width'],
index =["body-style","drive-wheels"], aggfunc={'price': np.mean,'height': [min,
max],'width': [min, max]},
fill_value=0)
table
```

The output of the preceding code is as follows:

body-style	**drive-wheels**	**Height**		**Price**	**Width**	
		max.	**min**	**mean**	**max**	**min**
Convertible	**fwd**	55.6	55.6	12754.500000	0.891667	0.891667
	rwd	53.0	48.8	26344.560000	0.890278	0.890278
Hardtop	**fwd**	53.3	53.3	9073.900000	0.886111	0.886111
	rwd	55.4	51.6	26622.985714	1.000000	0.902778
Hatchback	**4wd**	55.7	55.7	8363.300000	0.886111	0.886111
	fwd	56.1	49.4	9236.026531	0.925000	0.837500
	rwd	54.8	49.6	15771.555556	0.948611	0.888889
Sedan	**4wd**	54.3	54.3	13912.066667	0.922222	0.908333
	fwd	56.1	50.6	10792.980000	0.991667	0.868056
	rwd	56.7	47.8	23883.016667	0.995933	0.858333
Wagon	**4wd**	59.1	54.9	10005.325000	0.908333	0.883333
	fwd	59.8	53.0	10997.066667	0.991667	0.883333
	rwd	58.7	54.1	18693.644444	0.976389	0.923611

This pivot table represents the maximum and minimum of the height and width and the average price of cars in the respective categories mentioned in the index.

1.15. CROSS-TABULATIONS

We can customize the pandas dataframe with another technique called cross-tabulation. This allows us to cope with groupby and aggregation for better data analysis. Pandas has the crosstab function, which helps when it comes to building a cross-tabulation table. The cross-tabulation table shows the frequency with which certain groups of data appear. Let's take a look:

1. Let's use pd.crosstab() to look at how many different body styles cars are made by different makers:

pd.crosstab(df["make"], df["body-style"])

The output of the preceding code is as follows:

Body-style make	Convertible	Hardtop	Hatchback	Sedan	Wagon
alfa-romero	2	0	1	0	0
audi	0	0	0	5	1
bmw	0	0	0	8	0
Chevrolet	0	0	2	1	0
Dodge	0	0	5	3	1
Honda	0	0	7	5	1
Isuzu	0	0	1	1	0
Jaguar	0	0	0	3	0
Mazda	0	0	10	7	0
Mercedes-benz	1	2	0	4	1
Mercury	0	0	1	0	0

Let's apply margins and the margins_name attribute to display the row- wise and column-wise sum of the cross tables, as shown in the following code:

# apply margins and margins_name attribute to displays the row wise # and column wise sum of the cross table

pd.crosstab(df["make"], df["body-style"],margins=True,margins_name="Total Made")

The output of the preceding code is as follows:

Body-style make	**Convertible**	**Hardtop**	**Hatchback**	**Sedan**	**Wagon**	**Total Made**
alfa-romero	2	0	1	0	0	3
audi	0	0	0	5	1	6
bmw	0	0	0	8	0	8
Chevrolet	0	0	2	1	0	3
Dodge	0	0	5	3	1	9
Honda	0	0	7	5	1	13
Isuzu	0	0	1	1	0	2
Jaguar	0	0	0	3	0	3
Mazda	0	0	10	7	0	17
Mercedes-benz	1	2	0	4	1	8
Mercury	0	0	1	0	0	1
Mitsubishi	0	0	9	4	0	13
Nissan	0	1	5	9	3	18
Peugot	0	0	0	7	4	11

2. Let's see how the data is distributed by the body-type and drive_wheels columns within the maker of car and their door type in a crosstab:

```
pd.crosstab([df["make"],df["num-of-doors"]], [df["body-style"],df["drive- wheels"]],
margins=True,margins_name="Total Made")
```

The output of the preceding code is as follows:

Make	**body-style drive wheels num-of-wheels**	**Convertible**		**Hardtop**		**Hatchback**			**Sedan**			**Wagon**			**Total made**
		fwd	**rwd**	**fwd**	**rwd**	**4wd**	**fwd**	**rwd**	**4wd**	**fwd**	**rwd**	**4wd**	**fwd**	**rwd**	
alfa-romero	**two**	0	2	0	0	0	0	1	0	0	0	0	0	0	3
audi	**four**	0	0	0	0	0	0	0	1	3	0	0	1	0	5
	two	0	0	0	0	0	0	0	0	1	0	0	0	0	1
bmw	**four**	0	0	0	0	0	0	0	0	0	5	0	0	0	5
	two	0	0	0	0	0	0	0	0	0	3	0	0	0	3
Chevrolet	**four**	0	0	0	0	0	0	0	0	1	0	0	0	0	1
	two	0	0	0	0	0	2	0	0	0	0	0	0	0	2
Dodge	**four**	0	0	0	0	0	1	0	0	3	0	0	1	0	5
	two	0	0	0	0	0	4	0	0	0	0	0	0	0	4
Honda	**four**	0	0	0	0	0	0	0	0	4	0	0	1	0	5
	two	0	0	0	0	0	7	0	0	1	0	0	0	0	8
Isuzu	**four**	0	0	0	0	0	0	0	0	0	1	0	0	0	1

The pivot table syntax of pd.crosstab also takes some arguments, such as dataframe columns, values, normalize, and the aggregation function. We can apply the aggregation function to a cross table at the same time. Passing the aggregation function and values, which are the columns that aggregation will be applied to, gives us a cross table of a summarized subset of the dataframe.

3. First, let's look at the average curb-weight of cars made by different makers with respect to their body-style by applying the mean() aggregationfunction to the crosstable:

# values are the column in which aggregation function is to be applied # aggfunc is the aggregation function to be applied

# round() to round the output

pd.crosstab(df["make"], df["body-style"],values=df["curb-weight"], aggfunc= 'mean').round(0)

The output of the preceding code is as follows:

Body-style **make**	**Convertible**	**Hardtop**	**Hatchback**	**Sedan**	**Wagon**
alfa-romero	2548.0	NaN	2823.0	NaN	NaN
audi	NaN	NaN	NaN	2720.0	2954.0
bmw	NaN	NaN	NaN	2929.0	NaN
Chevrolet	NaN	NaN	1681.0	1909.0	NaN
Dodge	NaN	NaN	2132.0	2056.0	2535.0
Honda	NaN	NaN	1970.0	2289.0	2024.0
Isuzu	NaN	NaN	2734.0	237.0	NaN
Jaguar	NaN	NaN	NaN	4027.0	NaN
Mazda	NaN	NaN	2254.0	2361.0	NaN
Mercedes-benz	3685.0	3605.0	NaN	3731.0	3750.0
Mercury	NaN	NaN	2910.0	NaN	NaN
Mitsubishi	NaN	NaN	2377.0	2394.0	NaN
Nissan	NaN	2008.0	2740.0	2238.0	2452.0
Peugot	NaN	NaN	NaN	NaN	3358.0

TWO MARKS QUESTION AND ANSWERS - (PART-A)

1. ***Define EDA.***

   EDA is a process of examining the available dataset to discover patterns, spot anomalies, test hypotheses, and check assumptions using statistical measures. In this chapter, we are going to discuss the steps involved in performing top-notch exploratory data analysis and get our hands dirty using some open source databases.

2. ***What is data processing?***

   Preprocessing involves the process of pre-curating the dataset before actual analysis. Common tasks involve correctly exporting the dataset, placing them under the right tables, structuringthem, and exporting them in the correct format.

3. ***What do you understand from data cleaning?***

   Preprocessed data must be correctly transformed for an incompleteness check, duplicates check, error check, and missing value check. These tasks are performed in the data cleaning stage, which involves matching the correct record, finding inaccuracies in the dataset, understanding the overall data quality, removing duplicate items, and filling in the missing values.

4. ***List down the steps in EDA***

   - Problem definition
   - Data preparation
   - Data analysis
   - Development and representation of the results

5. ***What are different categories of data available in EDA?***

   - Numerical data
   - Discrete data
   - Continuous data
   - Categorical data

6. ***Brief the term Bayesian analysis?***

   The Bayesian approach incorporates prior probability distribution knowledge into the analysis steps as shown in the following diagram. Well, simply put, prior probability distribution of any quantity expresses the belief about that particular quantity before considering some evidence.

7. ***List the Software tools available for EDA.***

- ❖ NumPy
- ❖ Pandas
- ❖ Sea born
- ❖ Sci py
- ❖ Matplotlib

8. ***Define matplotlib***

Matplotlib provides a huge library of customizable plots, along with a comprehensive set of back ends. It can be utilized to create professional reporting applications, interactive analytical applications, complex dashboard applications, web/GUI applications, embedded views, and manymore. What are the visual aids for EDA?

- ❖ Line chart
- ❖ Bar chart
- ❖ Scatter plot
- ❖ Pie chart
- ❖ Histogram

9. ***What is the purpose of bar chart?***

This is one of the most common types of visualization that almost everyone must have encountered. Bars can be drawn horizontally or vertically to represent categorical variables. Bar charts are frequently used to distinguish objects between distinct collections in order to track variations over time. In most cases, bar charts are very convenient when the changes are large.

10. ***What is a scatter plot?***

Scatter plots are also called scatter graphs, scatter charts, scattergrams, and scatter diagrams. They use a Cartesian coordinates system to display values of typically two variables for a set of data.

Scatter plots can be constructed in thefollowing two situations:

- ❖ When one continuous variable is dependent on another variable, which is under the control of the observer
- ❖ When both continuous variables are independent

11. ***Differentiate dependent and independent variable***

There are two important concepts—independent variable and dependent variable. In statistical modeling or mathematical modeling, the values of dependent variables rely on the values of independent variables. The dependent variable is the outcome variable being studied. The independentvariables are also referred to as regressors.

12. ***What are the various steps in data transformation?***

- ❖ Data cleansing
- ❖ Data refactoring
- ❖ Data analysis

13. ***List the different methods in merging a database.***

- ❖ pd.merge() method with a left join
- ❖ pd.merge() method with a right join
- ❖ pd.merge() method with a inner join
- ❖ pd.merge() method with a outer join

14. ***Distinguish stacking and unstacking.***

Pivoting can be done with hierarchical indexing using two actions:

- ❖ Stacking: Stack rotates from any particular column in the data to the rows.
- ❖ Unstacking: Unstack rotates from the rows into the column.

15. ***What are the benefits of data transformation***

- ❖ Data transformation promotes interoperability between several applications. The main reason for creating a similar format and structure in the dataset is that it becomes compatible with other systems.
- ❖ Data transformation ensures a higher degree of data quality and protects applications from several computational challenges such as null values, unexpected duplicates, and incorrect indexings, as well as incompatible structures or formats.

- ❖ Data transformation ensures higher performance and scalability for modern analytical databases and dataframes.

16. ***Define mean, median, mode.***

Mean/average

The mean, or average, is a number around which the observed continuous variables are distributed. This number estimates the value of the entire dataset. Mathematically, it is the result of the division of the sum of numbers by the number of integers in the dataset.

Median

Given a dataset that is sorted either in ascending or descending order, the median divides the data into two parts. The general formula for calculating the median is as follows:

$$\text{Median position} = \frac{(n+1)}{2} \text{th observation}$$

Mode

The mode is the integer that appears the maximum number of times in the dataset. It happens to be the value with the highest frequency in the dataset. In the *x* dataset in the median example, the mode is 2 because it occurs twice in the set.

17. ***Define standard deviation and variance.***

Standard deviation

Different Python libraries have functions to get the standard deviation of the dataset. The NumPy library has the numpy.std(dataset) function. The statistics library has the statistics.stdev(dataset). function. Using the pandas library, we calculate the standard deviation in our df data frame using the df.std() function

Variance

Variance is the square of the average/mean of the difference between each value in the dataset with its average/mean; that is, it is the square of standard deviation.

18. ***What is skewness?***

Skewness is a measure of the asymmetryof the variable in the dataset about its mean. The skewness value can be positive or negative, or undefined. The skewness value tells us whether the data is skewed or symmetric.

19. *Define kurtosis*

Basically, kurtosis is a statistical measure that illustrates how heavily the tails of distribution differ from those of a normal distribution. This technique can identify whether a given distribution contains extreme values.

20. *What are the types of kurtosis?*

❖ Mesokurtic: If any dataset follows a normal distribution, it follows a mesokurtic distribution. It has kurtosis around 0.

❖ Leptokurtic: In this case, the distribution has kurtosis greater than 3 and the fat tails indicate that the distribution produces more outliers.

❖ Platykurtic: In this case, the distribution has negative kurtosis and thetails are very thin compared to the normal distribution.

21. *Define percentile*

Percentiles measure the percentage of values in any dataset that lie below a certain value. In order to calculate percentiles, we need to make sure our list is sorted. We will use the following formula for this:

$$\left.\begin{array}{r}\text{The formula for calculating} \\ \text{percentile of X}\end{array}\right\} = \frac{\text{Number of values less than X}}{\text{Total number of observations}} \times 100$$

22. *Define Quartile*

Quartiles refer to the three data points that divide the given dataset into four equal parts, such that each split makes 25% of the dataset. In terms of percentiles, the 25th percentile is referred to as the first quartile (Q1), the 50th percentile is referred to as the second quartile (Q2), and the 75th percentile is referred to as the third quartile (Q3).

$$\text{IQR} = Q_3 - Q_1$$

23. *What is data aggregation?*

Aggregation is the process of implementing any mathematical operation ona dataset or a subset of it. Aggregation is one of the many techniques in pandas that's used to manipulate the data in the dataframe for data analysis.

24. *What are the group-wise operations?*

The most important operations groupBy implements are aggregate, filter, transform, and apply. An efficient way of implementing aggregation functions in the dataset is by doing so after grouping the required columns.

25. *Define group-wise transformation*

Performing a transformation on a group or a column returns an object that is indexed by the same axis length as itself. It is an operation that's used in conjunction with groupby(). The aggregation operation has to return a reduced version of the data, whereas the transformation operation can return a transformed version of the full data

26. *What is a pivot table?*

The pandas.pivot_table() function creates a spreadsheet-style pivot table as a dataframe. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the resulting dataframe.The simplest pivot tables must have a dataframe and an index/list of theindex.

27. *What is a cross tabulation?*

We can customize the pandas dataframe with another technique called cross-tabulation. This allows us to cope with groupby and aggregation for better data analysis. pandas has the crosstab function, which helps when it comes to building a cross-tabulation table.

28. *What are the transformation techniques?*

- ❖ Performing data deduplication
- ❖ Replacing values
- ❖ Handling missing data
- ❖ Filling missing value**s**

PART-B & C

1. Explain the various stages in EDA.
2. Write down the steps in EDA, Explain.
3. Explain the software tools available for EDA
4. Elaborate in detail the visual aids for EDA
5. Elucidate the different transformation techniques in EDA
6. Define pivot table and cross tabulation, Explain.

****

UNIT II

VISUALIZING USING MATPLOTLIB

SYLLABUS

Importing Matplotlib – Simple line plots – Simple scatter plots – visualizing errors – density and contour plots – Histograms – legends – colors – subplots – text and annotation – customization – three dimensional plotting - Geographic Data with Basemap - Visualization with Seaborn.

- ❖ Importing Matplotlib
- ❖ Simple Line Plots
- ❖ Simple Scatter Plots
- ❖ Visualizing Errors
- ❖ Density and Contour Plots
- ❖ Histograms
- ❖ Legends
- ❖ Colors
- ❖ Subplots
- ❖ Text and Annotation
- ❖ Customization
- ❖ Three Dimensional Plotting
- ❖ Geographic Data with Basemap
- ❖ Visualization with Seaborn

UNIT II

VISUALIZING USING MATPLOTLIB

2.1. IMPORTING MATPLOTLIB

Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility. Matplotlib is a Python library that helps to plot graphs. It is used in data visualization and graphical plotting. To use matplotlib, we need to install it.

Step 1 – Make sure Python and pip is preinstalled on your system

Type the following commands in the command prompt to check is python and pip is installed on your system.

To check Python

```
python --version
```

If python is successfully installed, the version of python installed on your system will be displayed.

To check pip

```
pip -V
```

The version of pip will be displayed, if it is successfully installed on your system.

Step 2 – Install Matplotlib

Matplotlib can be installed using pip. The following command is run in the command prompt to install Matplotlib.

```
pip install matplotlib
```

This command will start downloading and installing packages related to the matplotlib library. Once done, the message of successful installation will be displayed.

Step 3 – Check if it is installed successfully

To verify that matplotlib is successfully installed on your system, execute the following command in the command prompt. If matplotlib is successfully installed, the version of matplotlib installed will be displayed.

```
import matplotlib
matplotlib.__version__
```

Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

```
import matplotlib.pyplot as plt
```

Now the Pyplot package can be referred to as plt.

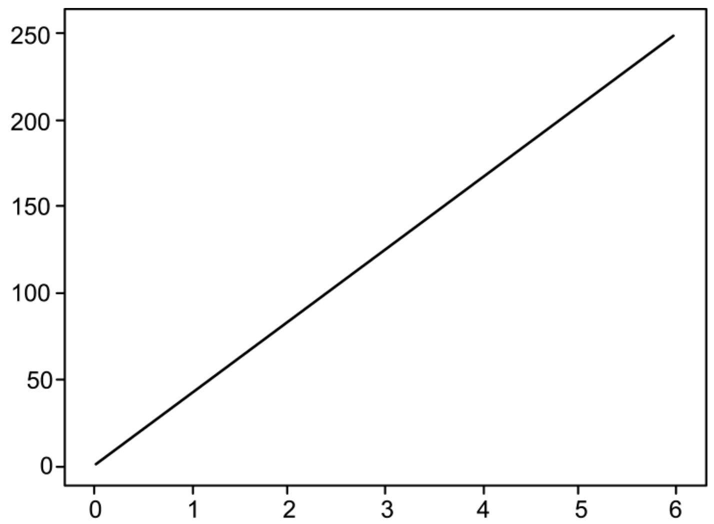


Fig. 2.1.

Example

Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints=np.array([0, 6])
ypoints=np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

2.2. SIMPLE LINE PLOTS

Perhaps the simplest of all plots is the visualization of a single function y=f(x)y=f(x). Here we will take a first look at creating a simple plot of this type. As with all the following sections, we'll start by setting up the notebook for plotting and importing the packages we will use:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

For all Matplotlib plots, we start by creating a figure and an axes. In their simplest form, a figure and axes can be created as follows:

```
fig = plt.figure()
ax = plt.axes()
```

In Matplotlib, the *figure* (an instance of the class plt.Figure) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels. The *axes* (an instance of the class plt.Axes) is what we see above: a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization.

Once we have created an axes, we can use the ax.plot function to plot some data. Let's start with a simple sinusoid:

```
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
```

```
ax.plot(x, np.sin(x));
```

Alternatively, we can use the pylab interface and let the figure and axes be created for us in the background,

```
plt.plot(x, np.sin(x));
```

If we want to create a single figure with multiple lines, we can simply call the plot function multiple times:

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
```

Adjusting the Plot: Line Colors and Styles

The first adjustment you might wish to make to a plot is to control the line colors and styles. The plt.plot() function takes additional arguments that can be used to specify these. To adjust the color, you can use the color keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways:

```
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines.

Similarly, the line style can be adjusted using the linestyle keyword:

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');


# For short, you can use the following codes:
```

```
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```

If you would like to be extremely terse, these linestyle and color codes can be combined into a single non-keyword argument to the plt.plot() function:

```
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```

Adjusting the Plot: Axes Limits

Matplotlib does a decent job of choosing default axes limits for your plot, but sometimes it's nice to have finer control. The most basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods:

```
plt.plot(x, np.sin(x))
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```

If for some reason you'd like either axis to be displayed in reverse, you can simply reverse the order of the arguments:

```
plt.plot(x, np.sin(x))
plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```

A useful related method is plt.axis() (note here the potential confusion between axes with an e, and axis with an i). The plt.axis() method allows you to set the x and y limits with a single call, by passing a list which specifies [xmin, xmax, ymin, ymax]:

```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```

The plt.axis() method goes even beyond this, allowing you to do things like automatically tighten the bounds around the current plot:

```
plt.plot(x, np.sin(x))
plt.axis('tight');
```

It allows even higher-level specifications, such as ensuring an equal aspect ratio so that on your screen, one unit in x is equal to one unit in y:

```
plt.plot(x, np.sin(x))
plt.axis('equal');
```

Labeling Plots

As the last piece of this section, we'll briefly look at the labeling of plots: titles, axis labels, and simple legends.

Titles and axis labels are the simplest such labels—there are methods that can be used to quickly set them:

```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```

The position, size, and style of these labels can be adjusted using optional arguments to the function. For more information, see the Matplotlib documentation and the docstrings of each of these functions.

When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type. Again, Matplotlib has a built-in way of quickly creating such a legend. It is done via the (you guessed it)plt.legend() method.

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')
```

plt.legend();

As you can see, the plt.legend() function keeps track of the line style and color, and matches these with the correct label. More information on specifying and formatting plot legends can be found in the plt.legend docstring

2.3. SIMPLE SCATTER PLOTS

Another commonly used plot type is the simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape. We’ll start by setting up the notebook for plotting and importing the functions we will use:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

Scatter Plots with plt.plot

In the previous section we looked at plt.plot/ax.plot to produce line plots. It turns out that this same function can produce scatter plots as well:

```
x = np.linspace(0, 10, 30)
y = np.sin(x)
plt.plot(x, y, 'o', color='black');
```

The third argument in the function call is a character that represents the type of symbol used for the plotting. Just as you can specify options such as '-', '--' to control the line style, the marker style has its own set of short string codes. The full list of available symbols can be seen in the documentation of plt.plot, or in Matplotlib's online documentation. Most of the possibilities are fairly intuitive, and we'll show a number of the more common ones here:

```
rng = np.random.RandomState(0)
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(5), rng.rand(5), marker,
             label="marker='{0}'".format(marker))
```

```
plt.legend(numpoints=1)
plt.xlim(0, 1.8);
```

Additional keyword arguments to plt.plot specify a wide range of properties of the lines and markers:

```
plt.plot(x, y, '-p', color='gray',
         markersize=15, linewidth=4,
         markerfacecolor='white',
         markeredgecolor='gray',
         markeredgewidth=2)
plt.ylim(-1.2, 1.2);
```

Scatter Plots with plt.scatter

A second, more powerful method of creating scatter plots is the plt.scatter function, which can be used very similarly to the plt.plot function:

```
plt.scatter(x, y, marker='o');
```

The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.

Let's show this by creating a random scatter plot with points of many colors and sizes. In order to better see the overlapping results, we'll also use the alpha keyword to adjust the transparency level:

```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,cmap='viridis')
plt.colorbar();  # show color scale
```

Notice that the color argument is automatically mapped to a color scale (shown here by the colorbar() command), and that the size argument is given in pixels. In this way, the color and size of points can be used to convey information in the visualization, in order to visualize multidimensional data.

For example, we might use the Iris data from Scikit-Learn, where each sample is one of three types of flowers that has had the size of its petals and sepals carefully measured:

```
import load_iris
iris = load_iris()
features = iris.data.T
plt.scatter(features[0],features[1],alpha=0.2,s=100*features[3],c=iris.target,
cmap='viridis')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```

We can see that this scatter plot has given us the ability to simultaneously explore four different dimensions of the data: the (x, y) location of each point corresponds to the sepal length and width, the size of the point is related to the petal width, and the color is related to the particular species of flower. Multicolor and multifeature scatter plots like this can be useful for both exploration and presentation of data.

Plot Versus Scatter:

Aside from the different features available in plt.plot and plt.scatter, why might you choose to use one over the other? While it doesn't matter as much for small amounts of data, as datasets get larger than a few thousand points, plt.plot can be noticeably more efficient than plt.scatter. The reason is that plt.scatter has the capability to render a different size and/or color for each point, so the renderer must do the extra work of constructing each point individually. In plt.plot, on the other hand, the points are always essentially clones of each other, so the work of determining the appearance of the points is done only once for the entire set of data. For large datasets, the difference between these two can lead to vastly different performance, and for this reason, plt.plot should be preferred over plt.scatter for large datasets.

2.4. VISUALIZING ERRORS

For any scientific measurement, accurate accounting for errors is nearly as important, if not more important, than accurate reporting of the number itself. In visualization of data and results, showing these errors effectively can make a plot convey much more complete information.

Basic Errorbars

A basic errorbar can be created with a single Matplotlib function call:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50)
plt.errorbar(x, y, yerr=dy, fmt='.k');
```

Here the fmt is a format code controlling the appearance of lines and points, and has the same syntax as the shorthand used in plt.plot, outlined in Simple Line Plots and Simple Scatter Plots.

In addition to these basic options, the errorbar function has many options to fine-tune the outputs. Using these additional options you can easily customize the aesthetics of your errorbar plot.

```
plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
    ecolor='lightgray', elinewidth=3, capsize=0);
```

In addition to these options, you can also specify horizontal errorbars (xerr), one-sided errorbars, and many other variants. For more information on the options available, refer to the docstring of plt.errorbar.

Continuous Errors

In some situations it is desirable to show errorbars on continuous quantities. Though Matplotlib does not have a built-in convenience routine for this type of application, it's relatively easy to combine primitives like plt.plot and plt.fill_between for a useful result.

Here we'll perform a simple *Gaussian process regression*, using the Scikit-Learn API This is a method of fitting a very flexible non-parametric function to data with a continuous measure of the uncertainty. We won't delve into the details of Gaussian process regression at this point, but will focus instead on how you might visualize such a continuous error measurement:

Import Gaussian Process

```
# define the model and draw some data
model = lambda x: x * np.sin(x)
xdata = np.array([1, 3, 5, 6, 8])
ydata = model(xdata)
# Compute the Gaussian process fit
gp = GaussianProcess(corr='cubic', theta0=1e-2, thetaL=1e-4, thetaU=1E-1,random_start=100)
gp.fit(xdata[:, np.newaxis], ydata)
xfit = np.linspace(0, 10, 1000)
yfit, MSE = gp.predict(xfit[:, np.newaxis], eval_MSE=True)
dyfit = 2 * np.sqrt(MSE)  # 2*sigma ~ 95% confidence region
```

We now have xfit, yfit, and dyfit, which sample the continuous fit to our data. We could pass these to the plt.errorbar function as above, but we don't really want to plot 1,000 points with 1,000 errorbars. Instead, we can use the plt.fill_between function with a light color to visualize this continuous error:

```
# Visualize the result
plt.plot(xdata, ydata, 'or')
plt.plot(xfit, yfit, '-', color='gray')
plt.fill_between(xfit, yfit - dyfit, yfit + dyfit,
                 color='gray', alpha=0.2)
plt.xlim(0, 10);
```

Note what we've done here with the fill_between function: we pass an x value, then the lower y-bound, then the upper y-bound, and the result is that the area between these regions is filled. The resulting figure gives a very intuitive view into what the Gaussian process regression algorithm is doing: in regions near a measured data point, the model is strongly constrained and this is reflected in the small model errors. In regions far from a measured data point, the model is not strongly constrained, and the model errors increase.

2.5. DENSITY AND CONTOUR PLOTS

Sometimes it is useful to display three-dimensional data in two dimensions using contours or color-coded regions. There are three Matplotlib functions that can be helpful for this task: plt.contour for contour plots, plt.contourf for filled contour plots, and plt.imshow for showing images. This section looks at several examples of using these. We'll start by setting up the notebook for plotting and importing the functions we will use:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np
```

Visualizing a Three-Dimensional Function

We'll start by demonstrating a contour plot using a function z=f(x,y)z=f(x,y), using the following particular choice for ff (we've seen this before in Computation on Arrays: Broadcasting, when we used it as a motivating example for array broadcasting):

```
def f(x, y):
return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

A contour plot can be created with the plt.contour function. It takes three arguments: a grid of *x* values, a grid of *y* values, and a grid of *z* values. The *x* and *y* values represent positions on the plot, and the *z* values will be represented by the contour levels. Perhaps the most straightforward way to prepare such data is to use the np.meshgrid function, which builds two-dimensional grids from one-dimensional arrays:

```
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

Now let's look at this with a standard line-only contour plot:

```
plt.contour(X, Y, Z, colors='black');
```

Notice that by default when a single color is used, negative values are represented by dashed lines, and positive values by solid lines. Alternatively, the lines can be color-coded by specifying a colormap with the cmap argument. Here, we'll also specify that we want more lines to be drawn—20 equally spaced intervals within the data range:

```
plt.contour(X, Y, Z, 20, cmap='RdGy');
```

Here we chose the RdGy (short for *Red-Gray*) colormap, which is a good choice for centered data. Matplotlib has a wide range of colormaps available, which you can easily browse in IPython by doing a tab completion on the plt.cm module:

```
plt.cm.<TAB>
```

Our plot is looking nicer, but the spaces between the lines may be a bit distracting. We can change this by switching to a filled contour plot using the plt.contourf() function (notice the f at the end), which uses largely the same syntax as plt.contour().Additionally, we'll add a plt.colorbar() command, which automatically creates an additional axis with labeled color information for the plot:

```
plt.contourf(X, Y, Z, 20, cmap='RdGy')
plt.colorbar();
```

The colorbar makes it clear that the black regions are "peaks," while the red regions are "valleys."

One potential issue with this plot is that it is a bit "splotchy." That is, the color steps are discrete rather than continuous, which is not always what is desired. This could be remedied by setting the number of contours to a very high number, but this results in a rather inefficient plot: Matplotlib must render a new polygon for each step in the level. A better way to handle this is to use the plt.imshow() function, which interprets a two-dimensional grid of data as an image.

The following code shows this:

```
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',cmap='RdGy')
plt.colorbar()
plt.axis(aspect='image');
```

There are a few potential gotchas with imshow(), however:

- ❖ plt.imshow() doesn't accept an *x* and *y* grid, so you must manually specify the *extent* [*xmin*, *xmax*, *ymin*, *ymax*] of the image on the plot.
- ❖ plt.imshow() by default follows the standard image array definition where the origin is in the upper left, not in the lower left as in most contour plots. This must be changed when showing gridded data.
- ❖ plt.imshow() will automatically adjust the axis aspect ratio to match the input data; this can be changed by setting, for example, plt.axis(aspect='image') to make *x* and *y* units match.
- ❖ Finally, it can sometimes be useful to combine contour plots and image plots. For example, here we'll use a partially transparent background image (with transparency set via the alpha parameter) and overplot contours with labels on the contours themselves (using the plt.clabel() function):

```
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',cmap='RdGy', alpha=0.5)
plt.colorbar();
```

The combination of these three functions—plt.contour, plt.contourf, and plt.imshow—gives nearly limitless possibilities for displaying this sort of three-dimensional data within a two-dimensional plot.

2.6. PLOTTING HISTOGRAM IN PYTHON USING MATPLOTLIB

A histogram is basically used to represent data provided in a form of some groups. It is accurate method for the graphical representation of numerical data distribution. It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

Creating a Histogram

To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

The following table shows the parameters accepted by matplotlib.pyplot.hist() function :

Attribute	Parameter
x	array or sequence of array
bins	optional parameter contains integer or sequence or strings
density	optional parameter contains boolean values
range	optional parameter represents upper and lower range of bins
histtype	optional parameter used to create type of histogram [bar, barstacked, step, stepfilled], default is “bar”
align	optional parameter controls the plotting of histogram [left, right, mid]
weights	optional parameter contains array of weights having same dimensions as x
bottom	location of the baseline of each bin
rwidth	optional parameter which is relative width of the bars with respect to bin width

Attribute	Parameter
color	optional parameter used to set color or sequence of color specs
label	optional parameter string or sequence of string to match with multiple datasets
log	optional parameter used to set histogram axis on log scale

Let's create a basic histogram of some random values. Below code creates a simple histogram of some random values:

```
import numpy as np
 # Creating dataset
a = np.array([22, 87, 5, 43, 56,73, 55, 54, 11,20, 51, 5, 79, 31,27])
 # Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])
 # Show plot   plt.show()
```

Output :

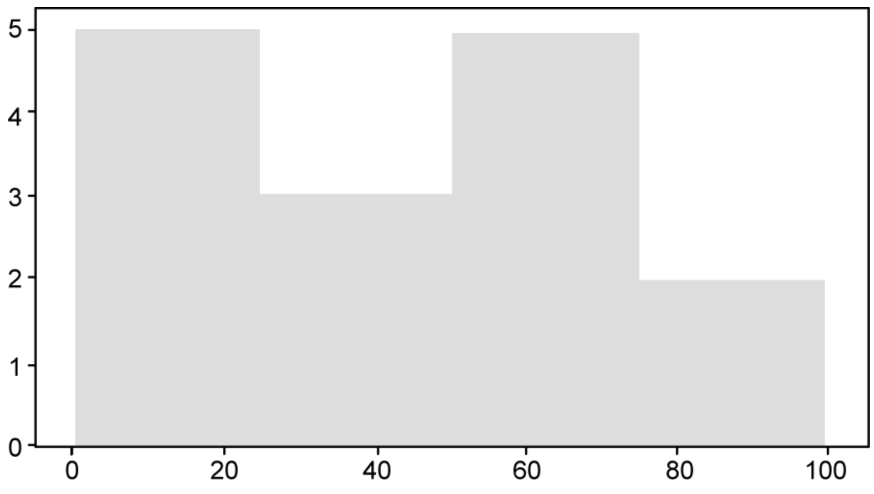


Fig. 2.2.

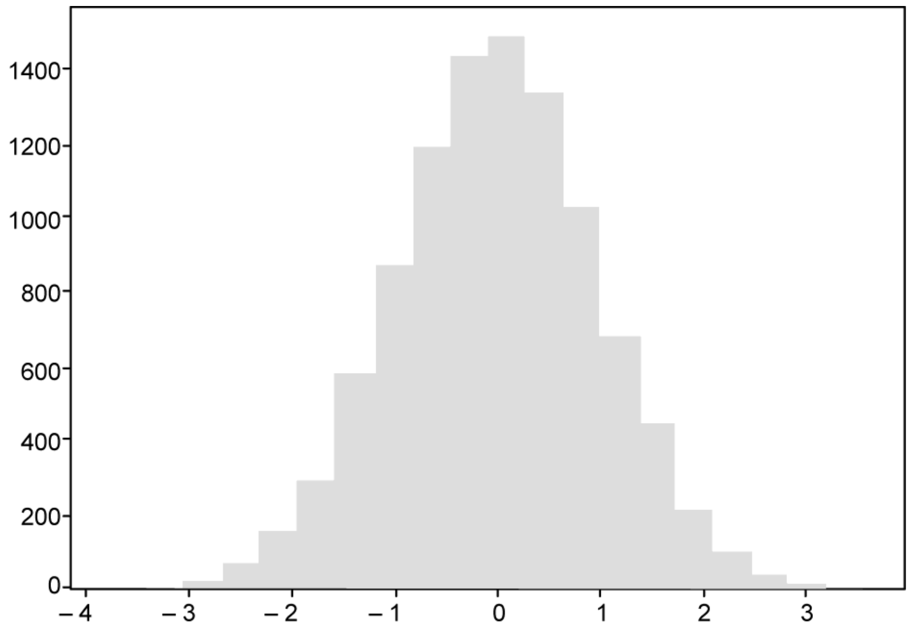
Customization of Histogram

Matplotlib provides a range of different methods to customize histogram. matplotlib.pyplot.hist() function itself provides many attributes with the help of which we can modify a histogram. The hist() function provide a patches object which gives access to the properties of the created objects, using this we can modify the plot according to our will.

Example 1:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
 # Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20
 # Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25
 # Creating histogram
fig, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)
 axs.hist(x, bins = n_bins)
 # Show plot
plt.show()
```

Output :



Example 2:

The code below modifies the above histogram for a better view and accurate readings.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
 # Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20
 # Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25
legend = ['distribution']
```

```
 # Creating histogram
fig, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)
# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    axs.spines[s].set_visible(False)
 # Remove x, y ticks
axs.xaxis.set_ticks_position('none')
axs.yaxis.set_ticks_position('none')
  # Add padding between axes and labels
axs.xaxis.set_tick_params(pad = 5)
axs.yaxis.set_tick_params(pad = 10)
 # Add x, y gridlines
        axs.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.6)
# Add Text watermark
fig.text(0.9, 0.15, 'Jeeteshgavande30',
         fontsize = 12,
         color ='red',
         ha ='right',
         va ='bottom',
         alpha = 0.7)
 # Creating histogram
N, bins, patches = axs.hist(x, bins = n_bins)
 # Setting color
fracs = ((N**(1 / 5)) / N.max())
```

```
norm = colors.Normalize(fracs.min(), fracs.max())
 for thisfrac, thispatch in zip(fracs, patches):
    color = plt.cm.viridis(norm(thisfrac))
    thispatch.set_facecolor(color)
 # Adding extra features
plt.xlabel("X-axis")
plt.ylabel("y-axis")
plt.legend(legend)
plt.title('Customized histogram')
 # Show plot
plt.show()
```

Output :

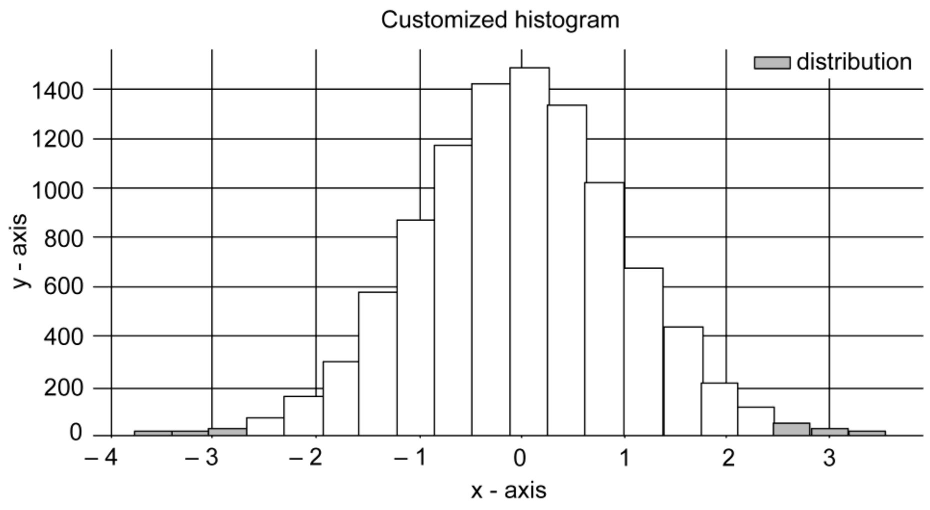


Fig. 2.3.

2.7. HOW TO MANUALLY ADD A LEGEND WITH A COLOR BOX ON A MATPLOTLIB FIGURE ?

A legend is basically an area in the plot which describes the elements present in the graph. Matplotlib provides an inbuilt method named legend() for this purpose. The syntax of the method is below :

Example: Adding Simple legend

```
# Import libraries
import matplotlib.pyplot as plt
 # Creating plot
  plt.plot([1, 2, 3, 4])
  plt.title('simple legend example ')
  # Creating legend
plt.legend(['simple legend example'])
 # Show plot
plt.show()
```

Output:

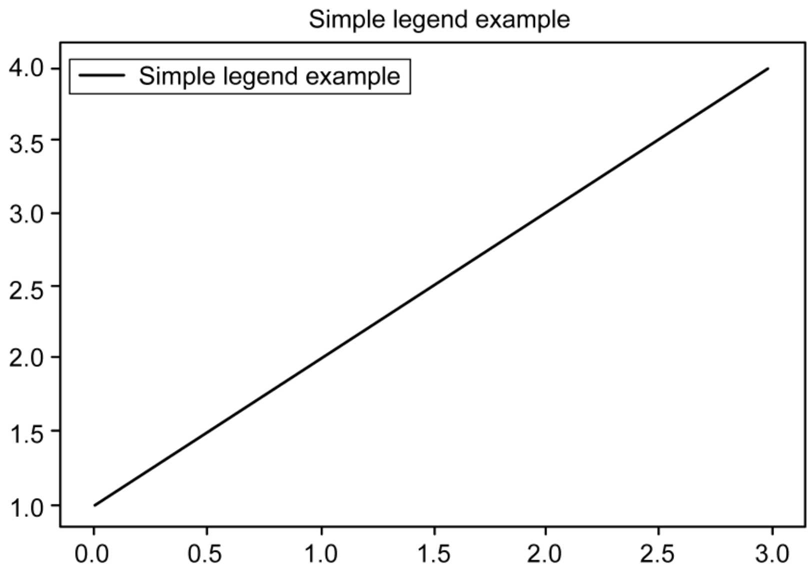


Fig. 2.4.

2.8. CREATING LEGEND WITH COLOR BOX

To create a legend with a color box, patches are used provided by the matplotlib.patches module. A patch nothing but a 2D artist with face color and edge color. Below is a simple example of this:

Example 1:

```
# Import libraries
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
  # Creating plot
plt.plot([1, 2, 3, 4], color='blue')
  plt.title('simple legend example ')
  # Creating legend with color box
blue_patch = mpatches.Patch(color='blue', label='blue legend')
plt.legend(handles=[blue_patch])
# Show plot
plt.show()
```

Output:

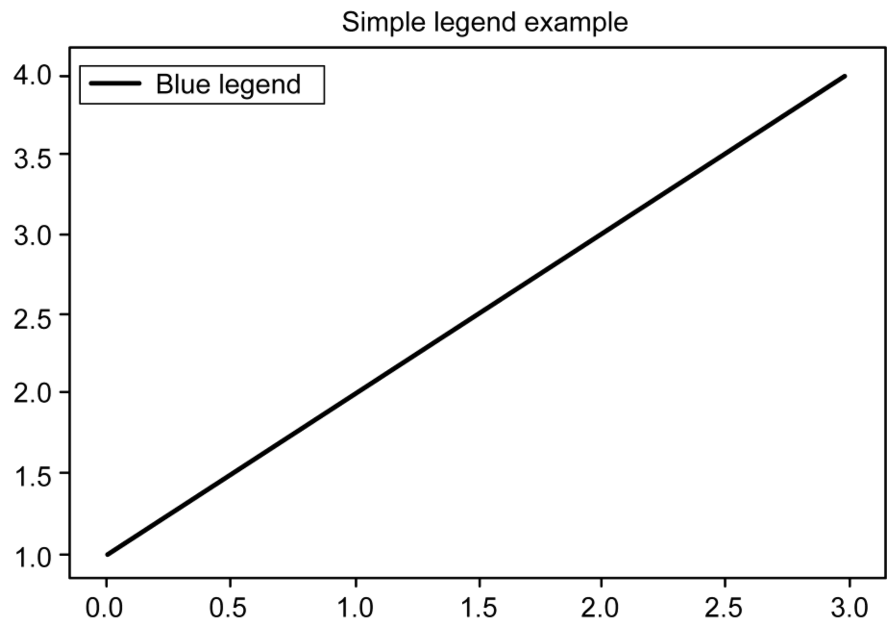


Fig. 2.5.

2.9. SUBPLOTS

What is a Subplot?

There are many cases where you will want to generate a plot that contains several smaller plots within it. That is exactly what a subplot is! A common version of the subplot is the 4x4 subplot. An example of the 4x4 subplot is below:

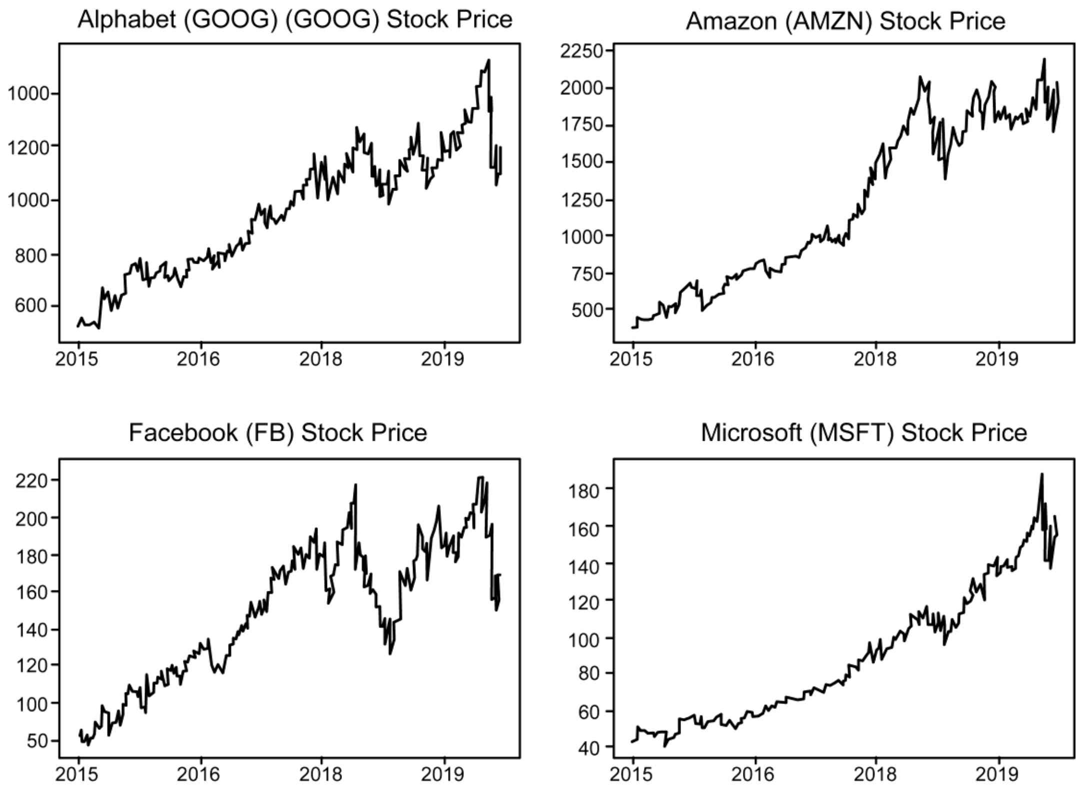


Fig. 2.6.

Subplots can be very complicated to create when done properly. As an example, consider the code that I used to create the above 4 × 4 subplot:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import datetime
```

```
tech_stocks_data = pd.read_csv('https://raw.githubusercontent.com/nicholasmccullum/python-visualization/master/tech_stocks/GOOG_MSFT_FB_AMZN_data.csv')
tech_stocks_data.sort_values('Period', ascending = True, inplace = True)
google = tech_stocks_data['Alphabet Inc Price']
amazon = tech_stocks_data['Amazon.com Inc Price']
facebook = tech_stocks_data['Facebook Inc Price']
microsoft = tech_stocks_data['Microsoft Corp Price']
dates = tech_stocks_data['Period']
x = []
for date in tech_stocks_data['Period']:
    x.append(datetime.strptime(date, '%Y-%m-%d %H:%M:%S').year)
plt.figure(figsize=(16,12))

#Plot 1
plt.subplot(2,2,1)
plt.xticks(np.arange(0, len(x) + 1)[::365], x[::365])
plt.plot(dates, google)
plt.title('Alphabet (GOOG) (GOOGL) Stock Price')

#Plot 2
plt.subplot(2,2,2)
plt.xticks(np.arange(0, len(x) + 1)[::365], x[::365])
plt.plot(dates, amazon)
plt.title('Amazon (AMZN)) Stock Price')

#Plot 3
plt.subplot(2,2,3)
plt.xticks(np.arange(0, len(x) + 1)[::365], x[::365])
plt.plot(dates, facebook)
plt.title('Facebook (FB) Stock Price')


#Plot 4
plt.subplot(2,2,4)
plt.xticks(np.arange(0, len(x) + 1)[::365], x[::365])
plt.plot(dates, microsoft)
plt.title('Microsoft (MSFT) Stock Price')
```

How To Create Subplots in Python Using Matplotlib?

We can create subplots in Python using matplotlib with the subplot method, which takes three arguments:

nrows: The number of rows of subplots in the plot grid.

ncols: The number of columns of subplots in the plot grid.

index: The plot that you have currently selected.

The nrows and ncols arguments are relatively straightforward, but the index argument may require some explanation. It starts at 1 and moves through each row of the plot grid one-by-one. When it reaches the end of a row, it will move down to the first entry of the next row.

A few examples of selecting specific subplots within a plot grid are shown below:

```
plt.subplot(3,3,5)
#Selects the middle entry of the second row in the 3x3 subplot grid
plt.subplot(1,2,2)
#Selects the second entry in a 1x2 subplot grid
plt.subplot(4,4,16)
#Selects the last entry in a 4x4 subplot grid
```

We will work through two examples of how to create subplot grids before concluding this lesson.

Example #1: A 2×2 Subplot Grid

First, let's import the Iris data set:

```
iris_data = pd.read_json('https://raw.githubusercontent.com/nicholasmccullum/python-visualization/master/iris/iris.json')
```

Using the Iris data set, let's create a 2 × 2 subplot with a subplot for each of the following variables (in the order they're listed):

sepalLength

sepalWidth

petalLength

petalWidth

Make each subplot a histogram with X bins. Make sure to give each subplot a reasonable title so that an outside reader could understand the data.

Once you have attempted this on your own, you can view the code below for a full solution:

```
plt.subplot(2,2,1)
plt.hist(iris_data['sepalLength'], bins = 15)
plt.title('A Histogram of Sepal Lengths from the Iris Data Set')
plt.subplot(2,2,2)
plt.hist(iris_data['sepalWidth'], bins = 15)
plt.title('A Histogram of Sepal Widths from the Iris Data Set')
plt.subplot(2,2,3)
plt.hist(iris_data['petalLength'], bins = 15)
plt.title('A Histogram of Petal Lengths from the Iris Data Set')
plt.subplot(2,2,4)
plt.hist(iris_data['petalWidth'], bins = 15)
plt.title('A Histogram of Petal Widths from the Iris Data Set')
```

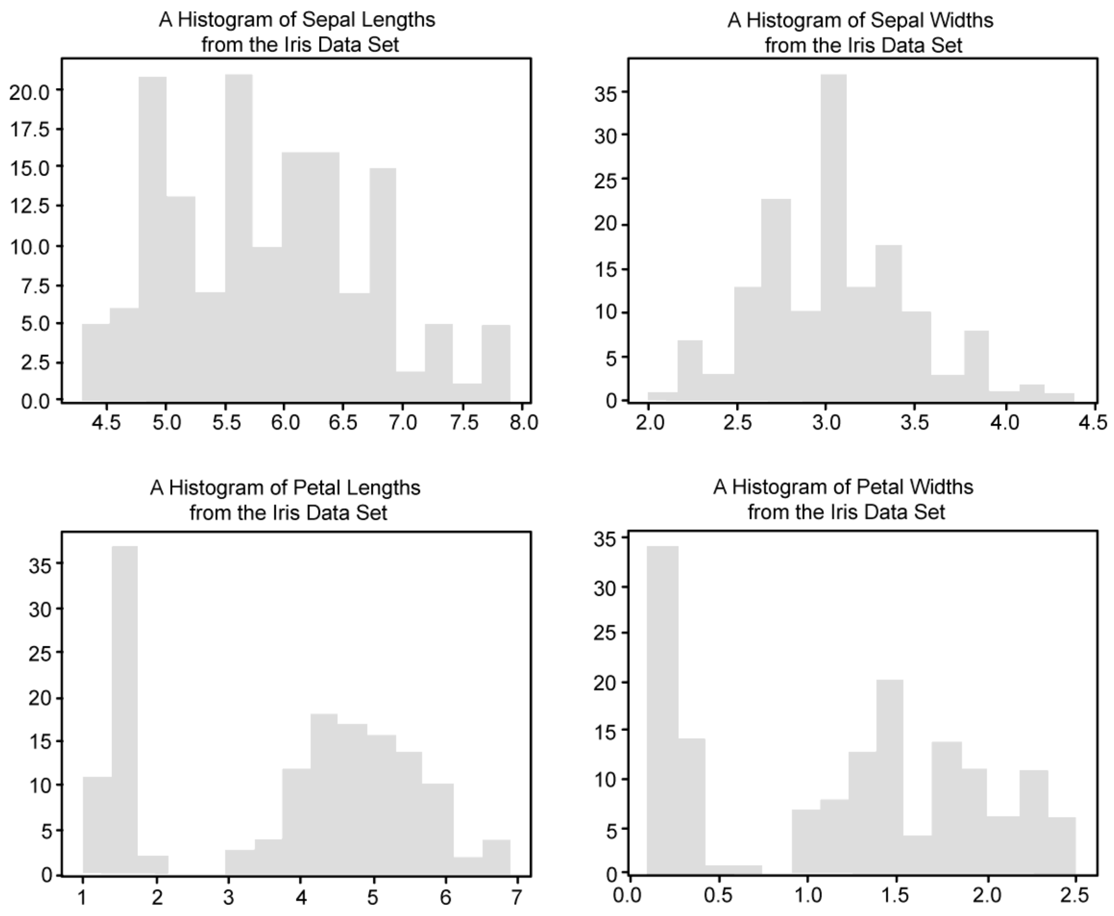


Fig. 2.7.

Example #2: A 2 × 3 Subplot Grid

Let's create a 2 × 3 subplot with the following plots (in the order they're listed):

chlorides

quality

alcohol

density

total sulfur dioxide

citric acid

Let's make each subplot a scatterplot, with the x-variable for each scatterplot being fixed acidity. Name each plot with an appropriate title for an outside reader to understand it.

Give this a try yourself before proceeding!

Once you have attempted this on your own, you can view the code below for a full solution:

```
x = wine_data['fixed acidity']
plt.subplot(2,3,1)
plt.scatter(x, wine_data['chlorides'])
plt.title('Chlorides plotted against Fixed Acidity')
plt.subplot(2,3,2)
plt.scatter(x, wine_data['quality'])
plt.title('Quality plotted against Fixed Acidity')
plt.subplot(2,3,3)
plt.scatter(x, wine_data['alcohol'])
plt.title('Alcohol plotted against Fixed Acidity')
plt.subplot(2,3,4)
plt.scatter(x, wine_data['density'])
plt.title('Density plotted against Fixed Acidity')
plt.subplot(2,3,5)
plt.scatter(x, wine_data['total sulfur dioxide'])
plt.title('Total Sulfur Dioxide plotted against Fixed Acidity')
plt.subplot(2,3,6)
plt.scatter(x, wine_data['citric acid'])
plt.title('Citric Acid plotted against Fixed Acidity')
```

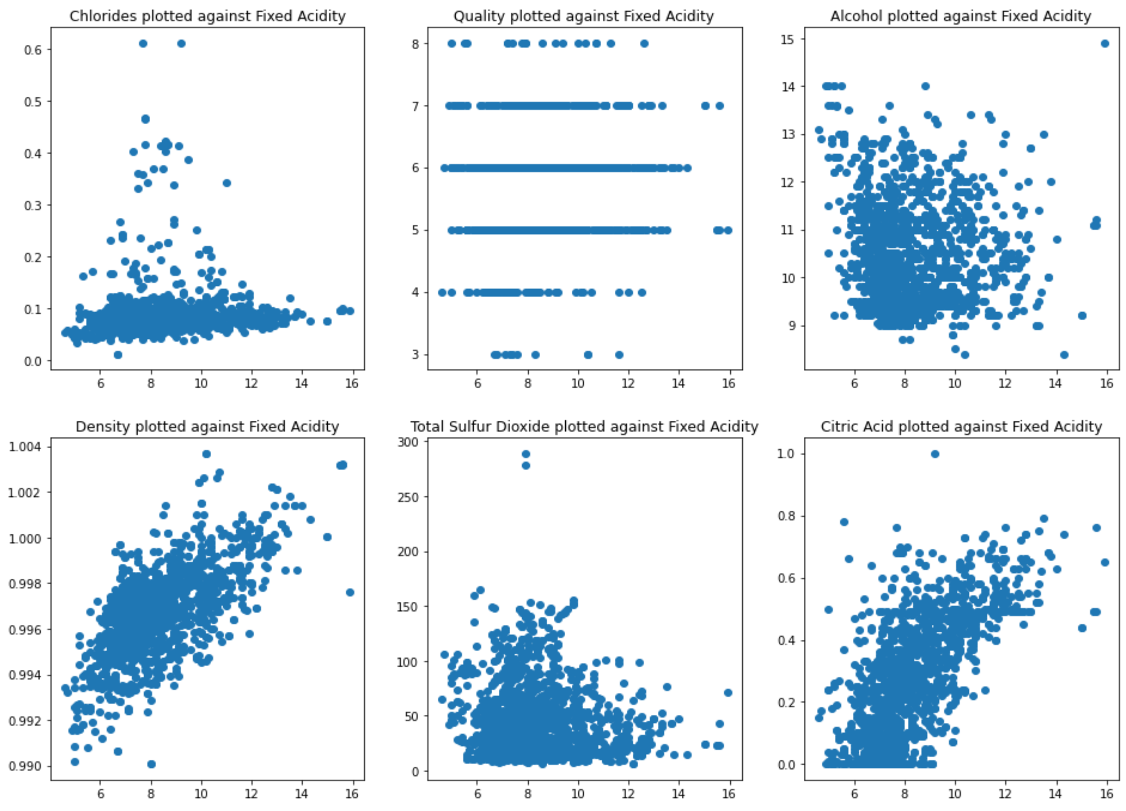


Fig. 2.8.

2.10. TEXT AND ANNOTATION

Annotate using text

Matplotlib offers the ability to place text within a chart. The only condition is it requires the positioning co-ordinate of the x and y-axis to place the text.

1. Annotate graph: plt.annotate()

To input text using matplotlib's "plt.annotate()" we need to declare two things, which is the "xy" coordinates which tells matplotlib where we want to input our text and the "s" attribute.

There is also an added attribute aswell, this is called the "arrowprops" attribute, which basically allows us to input an arrow pointing towards a specific point in our graph.

If arrows and texts are used within the “plt.annotate()” function, you can also use two xy coordinates, one for the arrow and the other for the text. This can be declared via “xy()” and “xytext()” respectively.

These are the following parameters used:

s : The text of the annotation

xy : The point (x,y) to annotate

xytext : The position (x,y) to place the text at (If None, defaults to xy)

arrowprops : The properties used to draw an arrow between the positions xy and xytext

#input annotation

plt.annotate(

# Label and coordinate

'My Money Goal Has been Reached!', xy=(2003, 14000), xytext=(2002, 20000),

#Arrow Pointer

arrowprops=dict(facecolor='red'))

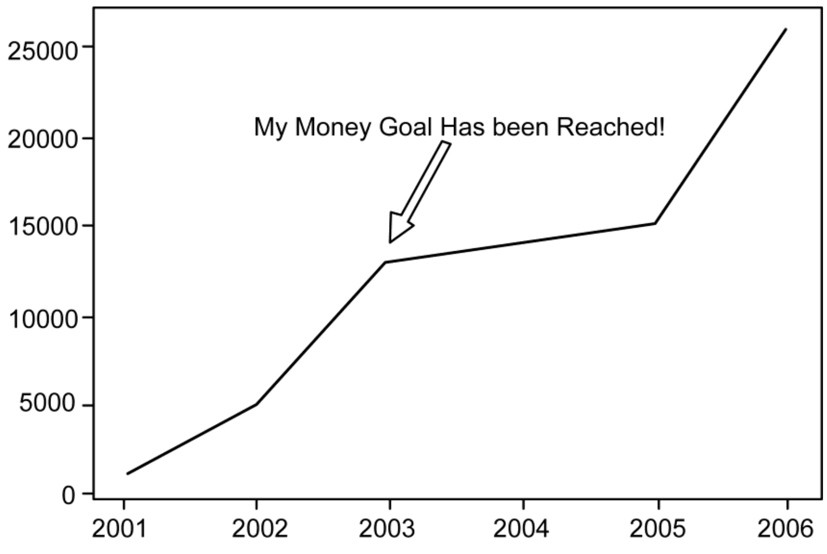


Fig. 2.9.

!pip install matplotlib # install matplotlib

import matplotlib.pyplot as plt #import matplotlib

```
#Create our x, y values and variance
year = [2001, 2002, 2003, 2004, 2005, 2006]
income = [1000, 5000, 13000, 14000. 15000, 26000]
#plot our graph
ax = plt.plot(year, income)

#input annotation
plt.annotate(
# Label and coordinate
‘My Money Goal Has been Reached!’, xy=(2003, 14000), xytext=(2002, 20000),
#Arrow Pointer
arrowprops=dict(facecolor=’red’))

#output chart
plt.show( )
```

2. Annotate graph: ax.text() or plt.text()

Another way to annotate text is to use the function “ax.text()” which is also known as Axes.text(). This function is a much simpler way to input text, as ax.text() only adds the text to the Axes at locations x, y in data coordinates. This exact function can also be called using plt.text() as well.

x : The x axis position to place text

y : The y axis position to place text

s : String to input text

fontsize: change the size of the font

ha: horizontal alignment

va: vertical alignment

```
ax.text(0.5, 0.5, 'Hello World!', size=24, ha='center', va='center')
```

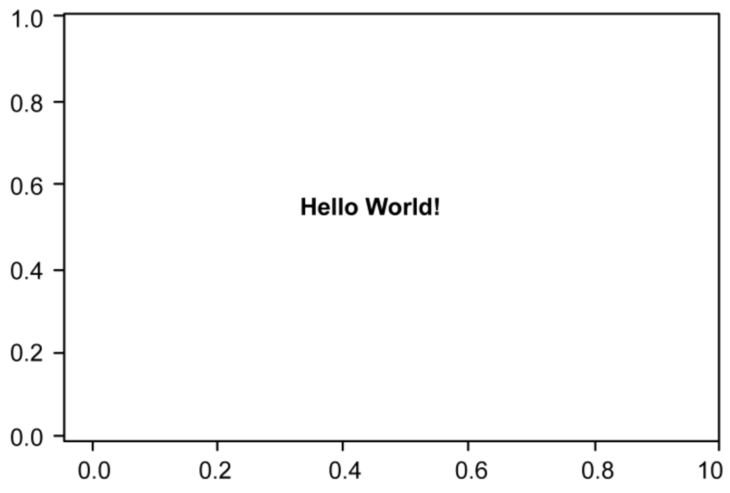


Fig. 2.10.

```
!pip install matplotlib # install matplotlib
import matplotlib.pyplot as plt #import matplotlib

#create our graph
fig, ax = plt.subplots( )

#create our annotations
ax.text(0.5, 0.5, ‘Hello World!’, size=24, ha=’center’, va=’center’)
```

3. Placing a text box: ax.text()

Similarly, you can also place text boxes with our “ax.text()” function, the text box is created through the use of the “bbox” function. The only difference between our previous step and this current one is we require to create a new variable called “textbox” which helps create our text box.

Below is the exact function to use for the current text box used.

```
# Create our text box
textbox = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
# place a text box in upper left in axes coords input our text box
ax.text(0.05, 0.95, 'This is a histogram', transform=ax.transAxes, fontsize=14,
verticalalignment='top', bbox=textbox)
```

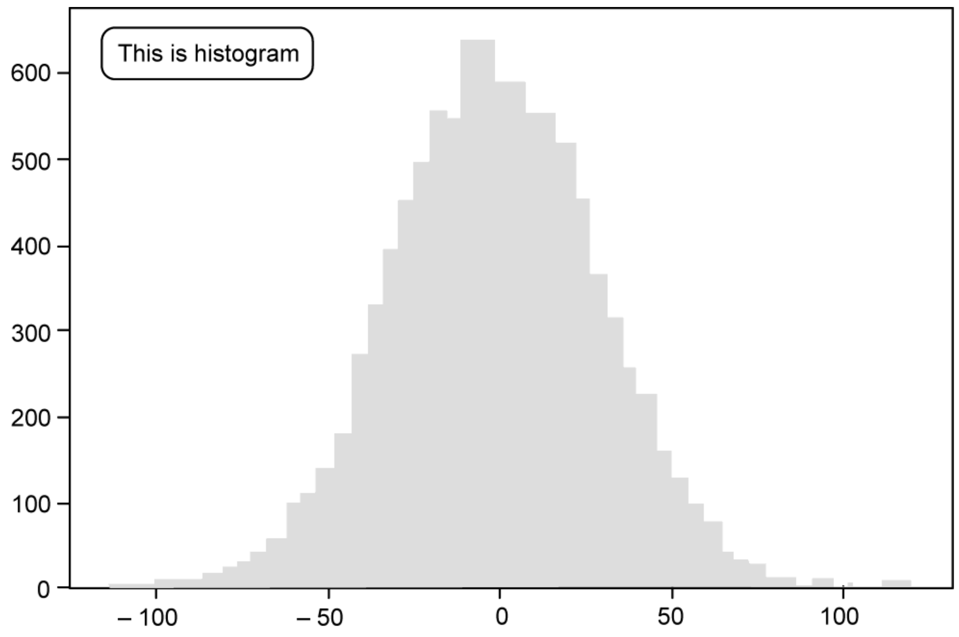


Fig. 2.11.

```
!pip install matplotlib #download matplotlib
import numpy as np
import matplotlib.pyplot as plt #import relevant packages
#create out graph
np.random.seed(19680801)
fig, ax = plt.subplots ( )
ax.hist(x, 50)
# Create out text box
textbox = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
#place a text box in upper left in axes cords input out text box
ax.text(0.05, 0.95, 'This is a histogram', transform=ax.transAxes, fontsize = 14,
        verticalalignment='top', bbox=textbox)
plt.show( )
```

Annotate using shapes

Another way to annotate your graph is to input shapes, this can include things such as a vertical line, a rectangle and many more objects which highlight a spot within a specific graph. Combining shapes with text annotations will definitely create a better solution to annotating graphs.

4. Annotate using a vertical line: plt.axvline()

Matplotlib allows the ability to input a vertical line to highlight a specific spot of a graph, by doing so we can identify key statistics such as in our current graph where I used a line annotation and also a text annotation to visualize the significance of the 2008 Great financial crash.To create a vertical line annotation we can use the function plt.axvline() this basically creates a vertical line from a specified x-axis spot.

x : The x axis position to place vertical line

color : Color of the line graph

linestyle : Line graph style

plt.axvline(2007, color='r', linestyle='dashed')

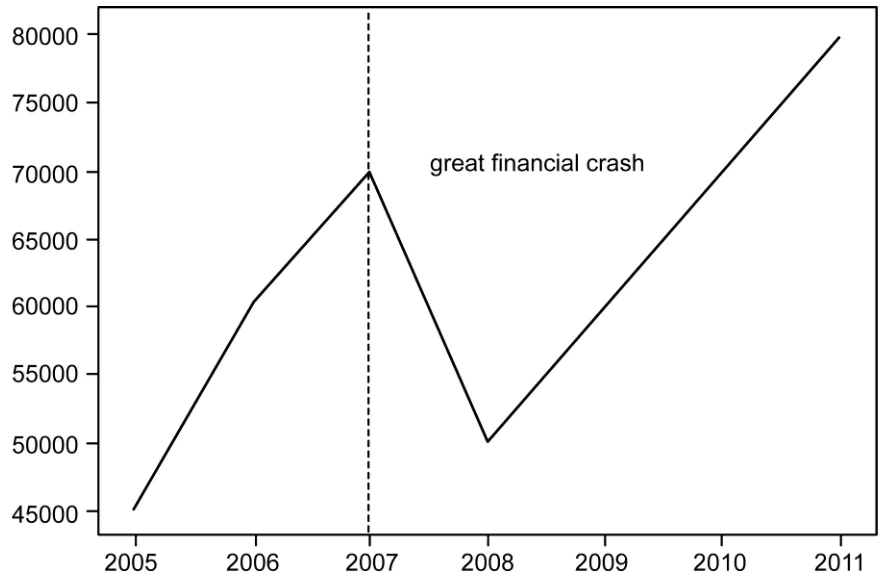


Fig. 2.12.

!pip install matplotlib #download matplotlib import matplotlib.pyplot as plt #import relevant packages

#create a graph

year=[2005,2006, 2007, 2008, 2009, 2010, 2011]

income=[45000, 60000,70000,50000,60000,70000,80000]

plt.plot(year, income)

#Create our line graph

plt.axvline(2007, color=’r’, linestyle=’dashed’)

#input some text

plt.text(2007.5, 70000, s=’great financial crash)

5. Annotate using a horizontal line: plt.axhline()

Similarly, you can also annotate using a horizontal line, this can simply be created using the function plt.axhline(). The only difference in the parameters between this function and the previous plt.axvline() is the requirement of the y axis position.

y : The y axis position to place vertical line

color : Color of the line graph

linestyle : Line graph style

plt.axhline(60000, color='r', linestyle='dashed')

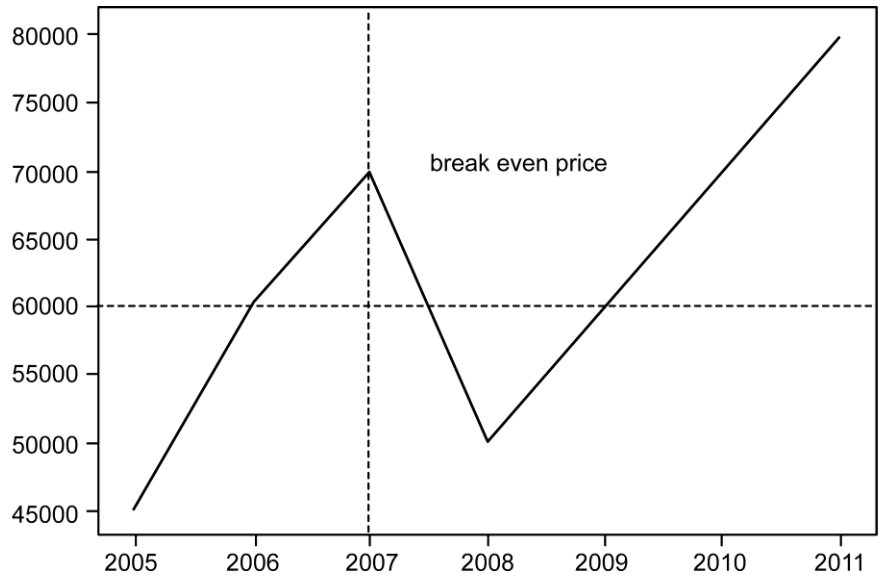


Fig. 2.13.

```
!pip install matplotlib #download matplotlib
import matplotlib.pyplot as plt #import relevant packages

#create a graph
year=[2005,2006, 2007, 2008, 2009, 2010, 2011]
income=[45000, 60000,70000,50000,60000,70000,80000]
plt.plot(year, income)

#Create our line graph
plt.axhline(60000, color='r', linestyle='dashed')
#input some text
plt.text(2007.5, 70000, s=break even price')
```

6. Highlight using a rectangle: ax.add_patch(Rectangle())

To input a rectangle in matplotlib we need to import our rectangle package, this can be declared using the function "from matplotlib.patches import Rectangle". Afterwards, we can now use our function "ax.add_patch(Rectangle())".

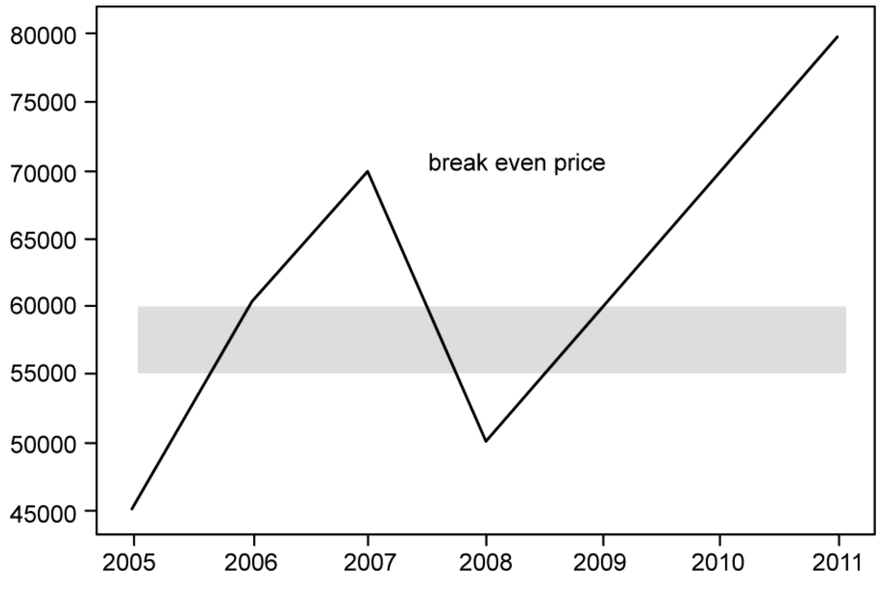


Fig. 2.14.

These are the following parameters used in the function

xy : The xy axis starting point for the lower left side of the rectangle

width: the width span of the rectangle

height: the height span of the rectangle

color: the color of the rectangle

```
ax.add_patch(Rectangle((2005, 55000), 6, 5000, color="red"))

!pip install matplotlib #download matplotlib
import matplotlib.pyplot as plt #import relevant packages
from matplotlib.patches import Rectangle

#create a graph
year=[2005,2006, 2007, 2008, 2009, 2010, 2011]
income=[45000, 60000,70000,50000,60000,70000,80000]
fig, ax = plt.subplots( )
ax.plot(year, income)

#input our rectangle
ax.add_patch(Rectangle((2005, 55000), 6, 5000, color=”red”))
```

7. Highlight using a circle: patches.Circle()

We can also create a ‘looking glass’ effect using matplotlib. To do this we need to import the package “matplotlib.patches as patches” this library stores our circle function (patches.Circle()).

These are the following parameters of our function

xy : The xy axis of the center of the circle

radius: the size of the circle

fc: the color of the rectangle

alpha: transparency of the circle

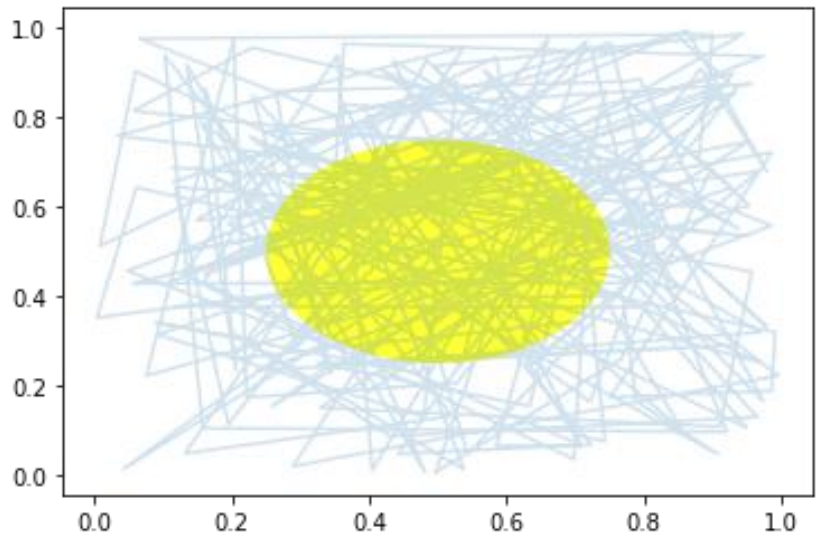


Fig. 2.15.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
# Creating out data
np.random.seed(19680801)
x, y = np.random.rand(2, 200)
#plotting our graph
fig, ax = plt.subplots( )
ax.plot(x, y, alpha=0.2)
#creating our circle
circle = patches.Circle(0.5, 0.5), 0.25, alpha=0.8, fc=’yellow’)
ax.add_patch(circle)
```

2.11. CUSTOMIZATION

Customizing Individual Plots

First import Matplotlib’s pyplot module, as well as NumPy to generate the sample data for the figures.

```
from matplotlib import pyplot as plt
import numpy as np
```

Setting Tick Marks and Tick Labels

For the first plot, I’ll use several trigonometric functions as sample data.

```
π = np.pi
x = 2 * π * np.linspace(-1, 1, 1000)_, ax = plt.subplots()
ax.plot(x, np.cos(x))
ax.plot(x, np.sin(x))
ax.plot(x, np.cos(x - π))
```

Using the defaults from the current version of Matplotlib (v. 3.0.2 at the time of this writing) yields the following figure:

Since there is more than one line, let’s add a legend. Here, I’m using Matplotlib’s included miniature TeX distribution for the mathematical symbols. (Matplotlib can also render text elements with an external LaTeX distribution; more on LaTeX below.)

```
ax.plot(x, np.cos(x), label=r"$ \cos \left( x \right) $")
ax.plot(x, np.sin(x), label=r"$ \sin \left( x \right) $")
ax.plot(x, np.cos(x - π), label=r"$ \cos \left( x - \pi \right) $")
ax.legend(loc="upper right")
```

It’s often helpful to set the ticks and labels in terms of ππ when dealing with trigonometric functions. (You can define a custom tick formatter, but for this plot it’s simpler and more readable to list the ticks and labels explicitly.)

```
xticks = π * np.arange(-2, 3, 1)
ax.set_xticks(xticks)
xlabels = [r"$-2 \pi$", r"$- \pi$", "0", r"$\pi$", r"$2 \pi$"]
ax.set_xticklabels(xlabels)
ax.set_yticks([-1, 0, 1])
```

Adding Fills

Fills are useful for (among other things) visualizing where multiple distributions overlap.

Let's start with several Gaussian distributions for illustration. The probability density for a Gaussian distribution is given by

p(x)=1√2πσ2e−(x−μ)22σ2p(x)=12πσ2e−(x−μ)22σ2

where μμ is the mean, and σσ is the standard deviation.

```
def gaussian(x, μ=0, σ=1, normalized=True):
    u = (x - μ) / σ
    g = np.exp(-u**2 / 2)
    if normalized:
        g /= np.sqrt(2 * π * σ**2)
    return g
```

Let's generate 3 Gaussian distributions for the plot.

```
z = np.linspace(-10, 10, 1000)
μ0, μ1, μ2 = -4, 0, 2
y0 = gaussian(z, μ=μ0, σ=1.25)
y1 = gaussian(z, μ=μ1, σ=1.0)
y2 = gaussian(z, μ=μ2, σ=1.5)
```

You can specify colors from Matplotlib's color cycler with a "CN" color specification. Since this plot only has a few lines, it's simpler to explicitly match the fill color to the color of the associated line. (If you have more than a few lines in your plot, iterate over the lines and use line.get_color() to set the fill color.)_, ax = plt.subplots()

```
ax.plot(z, y0, label=r"$G_0$", color="C0")
ax.plot(z, y1, label=r"$G_1$", color="C1")
ax.plot(z, y2, label=r"$G_2$", color="C2")
ax.legend(loc="upper right")
```

Next, place tick marks at the mean of each distribution.

```
ax.set_xticks([μ0, μ1, μ2])
ax.set_xticklabels([r"$\mu_0$", r"$\mu_1$", r"$\mu_2$"])
ax.set_yticks([])
```

Now shade the area between each distribution and the x axis.

```
ax.fill_between(z, y0, 0, color="C0", alpha=0.2)
ax.fill_between(z, y1, 0, color="C1", alpha=0.2)
ax.fill_between(z, y2, 0, color="C2", alpha=0.2)
```

Finally, remove unnecessary axis spines.

```
ax.spines["top"].set_visible(False)
ax.spines["left"].set_visible(False)
ax.spines["right"].set_visible(False)
```

Writing Your Own Style Sheets

Recall that styles are invoked with plt.style.use(<stylename>).

Style files in Matplotlib have the form <style_name>.mplstyle. If you place your style files in the <mpl_configdir>/stylelib directory, Matplotlib will load them at runtime. (You can also pass the full file path or URL to the style sheet.)

Style sheets can be chained together, e.g.

```
plt.style.use([style1, style2])
```

This means you can have one style file to set the margins, another to define line properties, etc.

Rendering All Plot Elements with LaTeX

Let's create an example style sheet. The following configuration will render all figure text (legend, axes labels, tick marks, etc.) with LaTeX's Computer Modern font. This is useful for producing publication-quality figures.

```
# Use LaTeX's Computer Modern font for everything
font.family : 'serif'
font.serif : 'Computer Modern'
text.usetex : True
```

If you place the above code block in a file called LaTeX_everywhere.mplstyle (or similar) in the stylelib directory (see above), you can then invoke it with

with plt.style.context("LaTeX_everywhere"):

Rendering the trigonometric plot from the first example with our custom LaTeX style produces

The matplotlibrc File

Matplotlib takes the first matplotlibrc file it finds. To display the path to the current matplotlibrc file, use

matplotlib.matplotlib_fname()

Matplotlib also provides a template matplotlibrc file. This is incredibly useful, not just for writing your own style sheets or matplotlibrc file, but also for understanding what customizations are available for individual plots.

Meta example: the matplotlibrc file I used to render the figures in this post was

savefig.format: svg

savefig.transparent: True

Then to render each figure as a transparent SVG, I could just use

plt.savefig("assets/figure1") rather than

plt.savefig("assets/figure1.svg", transparent=True) for each figure.

2.12. THREE-DIMENSIONAL PLOTTING USING MATPLOTLIB

Matplotlib was introduced keeping in mind, only two-dimensional plotting. But at the time when the release of 1.0 occurred, the 3d utilities were developed upon the 2d and thus, we have 3d implementation of data available today! The 3d plots are enabled by importing the mplot3d toolkit. In this article, we will deal with the 3d plots using matplotlib.

Example:

import numpy as np

import matplotlib.pyplot as plt

```
 fig = plt.figure()
ax = plt.axes(projection ='3d')
```

Output:

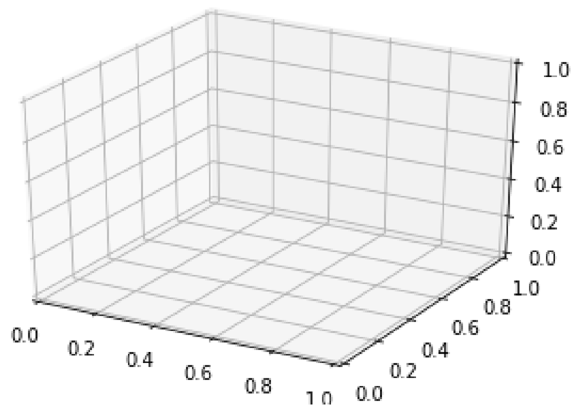


Fig. 2.16.

With the above syntax three -dimensional axes are enabled and data can be plotted in 3 dimensions. 3 dimension graph gives a dynamic approach and makes data more interactive. Like 2-D graphs, we can use different ways to represent 3-D graph. We can make a scatter plot, contour plot, surface plot, etc. Let's have a look at different 3-D plots.

Plotting 3-D Lines and Points

Graph with lines and point are the simplest 3 dimensional graph. ax.plot3d and ax.scatter are the function to plot line and point graph respectively.

Example 1: 3 dimensional line graph

```
# importing mplot3d toolkits, numpy and matplotlib
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
 fig = plt.figure()
 # syntax for 3-D projection
ax = plt.axes(projection ='3d')
```

```
 # defining all 3 axes
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
 # plotting
ax.plot3D(x, y, z, 'green')
ax.set_title('3D line plot geeks for geeks')
plt.show()
```

Output:

```
Text(0.5, 0.92, '3D line plot geeks for geeks')
```

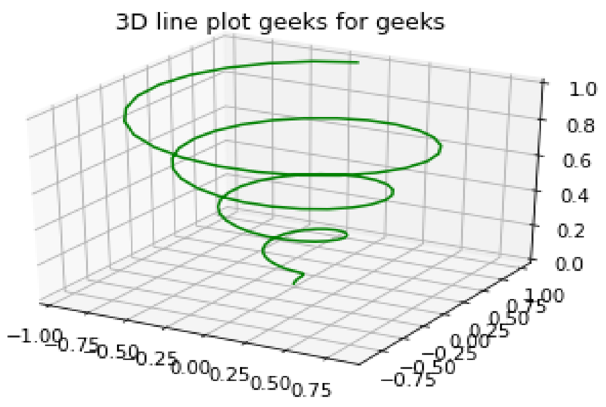


Fig. 2.17.

Example 2: 3 dimensional scattered graph

```
 # importing mplot3d toolkits
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
 fig = plt.figure()
 # syntax for 3-D projection
ax = plt.axes(projection ='3d')
```

```
 # defining axes
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
c = x + y
ax.scatter(x, y, z, c = c)
 # syntax for plotting
ax.set_title('3d Scatter plot geeks for geeks')
plt.show()
```

Output:

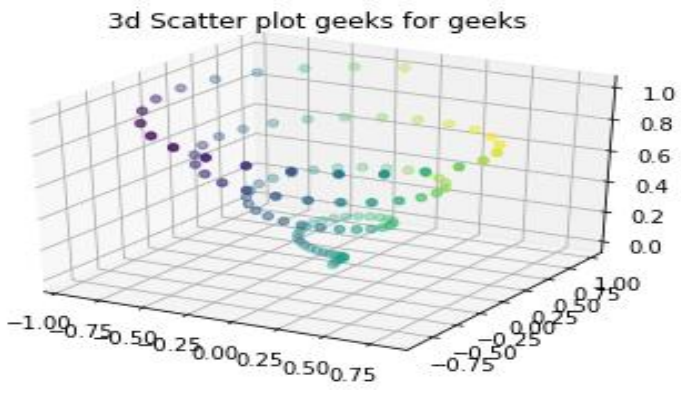


Fig. 2.18.

2.13. GEOGRAPHIC DATA WITH BASEMAP

One common type of visualization in data science is that of geographic data. Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits which lives under the mpl_toolkits namespace. Admittedly, Basemap feels a bit clunky to use, and often even simple visualizations take much longer to render than you might hope. More modern solutions such as leaflet or the Google Maps API may be a better choice for more intensive map visualizations. Still, Basemap is a useful tool for Python users to have in their virtual toolbelts. In this section, we'll show several examples of the type of map visualization that is possible with this toolkit.

Installation of Basemap is straightforward; if you're using conda you can type this and the package will be downloaded:

```
$ conda install basemap
```

We add just a single new import to our standard boilerplate:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
```

Once you have the Basemap toolkit installed and imported, geographic plots are just a few lines away (the graphics in the following also requires the PIL package in Python 2, or the pillow package in Python 3):

```
plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```

The meaning of the arguments to Basemap will be discussed momentarily.

The useful thing is that the globe shown here is not a mere image; it is a fully-functioning Matplotlib axes that understands spherical coordinates and which allows us to easily overplot data on the map! For example, we can use a different map projection, zoom-in to North America and plot the location of Seattle. We'll use an etopo image (which shows topographical features both on land and under the ocean) as the map background:

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
            width=8E6, height=8E6,
            lat_0=45, lon_0=-100,)
m.etopo(scale=0.5, alpha=0.5)
# Map (long, lat) to (x, y) for plotting
```

```
x, y = m(-122.3, 47.6)
plt.plot(x, y, 'ok', markersize=5)
plt.text(x, y, ' Seattle', fontsize=12);
```

This gives you a brief glimpse into the sort of geographic visualizations that are possible with just a few lines of Python. We'll now discuss the features of Basemap in more depth, and provide several examples of visualizing map data. Using these brief examples as building blocks, you should be able to create nearly any map visualization that you desire.

Map Projections

The first thing to decide when using maps is what projection to use. You're probably familiar with the fact that it is impossible to project a spherical map, such as that of the Earth, onto a flat surface without somehow distorting it or breaking its continuity. These projections have been developed over the course of human history, and there are a lot of choices! Depending on the intended use of the map projection, there are certain map features (e.g., direction, area, distance, shape, or other considerations) that are useful to maintain.

The Basemap package implements several dozen such projections, all referenced by a short format code. Here we'll briefly demonstrate some of the more common ones.

```
from itertools import chain
def draw_map(m, scale=0.2):
   # draw a shaded-relief image
   m.shadedrelief(scale=scale)
   # lats and longs are returned as a dictionary
   lats = m.drawparallels(np.linspace(-90, 90, 13))
   lons = m.drawmeridians(np.linspace(-180, 180, 13))
   # keys contain the plt.Line2D instances
   lat_lines = chain(*(tup[1][0] for tup in lats.items()))
   lon_lines = chain(*(tup[1][0] for tup in lons.items()))
   all_lines = chain(lat_lines, lon_lines)
```

```
    # cycle through these lines and set the desired style
  for line in all_lines:
    line.set(linestyle='-', alpha=0.3, color='w')
```

Cylindrical Projections

The simplest of map projections are cylindrical projections, in which lines of constant latitude and longitude are mapped to horizontal and vertical lines, respectively. This type of mapping represents equatorial regions quite well, but results in extreme distortions near the poles. The spacing of latitude lines varies between different cylindrical projections, leading to different conservation properties, and different distortion near the poles. In the following figure we show an example of the equidistant cylindrical projection, which chooses a latitude scaling that preserves distances along meridians. Other cylindrical projections are the Mercator (projection='merc') and the cylindrical equal area (projection='cea') projections.

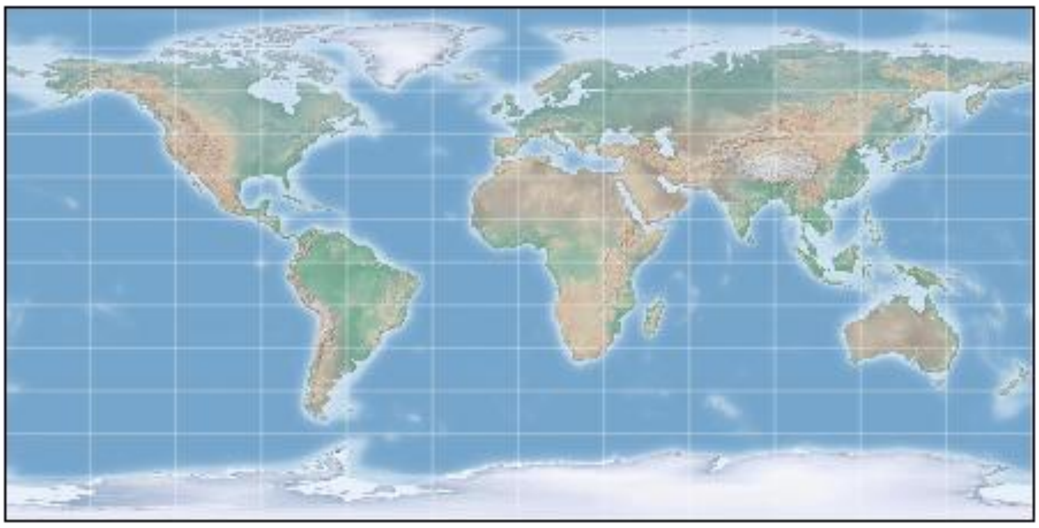


Fig. 2.19.

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='cyl', resolution=None,
        llcrnrlat=-90, urcrnrlat=90,
        llcrnrlon=-180, urcrnrlon=180, )
draw_map(m)
```

Pseudo-cylindrical Projections

Pseudo-cylindrical projections relax the requirement that meridians (lines of constant longitude) remain vertical; this can give better properties near the poles of the projection. The Mollweide projection (projection='moll') is one common example of this, in which all meridians are elliptical arcs. It is constructed so as to preserve area across the map: though there are distortions near the poles, the area of small patches reflects the true area. Other pseudo-cylindrical projections are the sinusoidal (projection='sinu') and Robinson (projection='robin') projections.

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='moll', resolution=None,
            lat_0=0, lon_0=0)
draw_map(m)
```

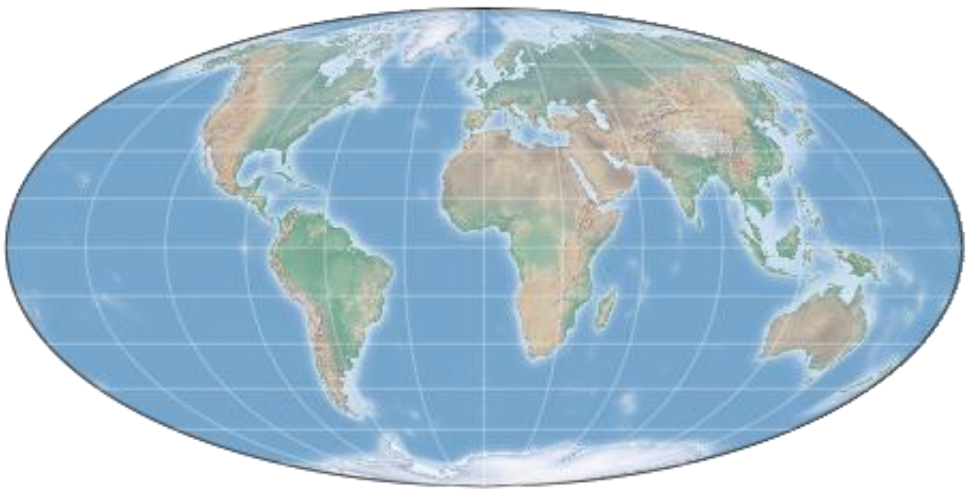


Fig. 2.20.

The extra arguments to Basemap here refer to the central latitude (lat_0) and longitude (lon_0) for the desired map.

Perspective Projections

Perspective projections are constructed using a particular choice of perspective point, similar to if you photographed the Earth from a particular point in space (a point which, for some projections, technically lies within the Earth!). One common example is the orthographic projection (projection='ortho'), which shows one side of the globe as seen from a viewer at a very long distance. As such, it can show only half the globe at a time. Other perspective-based projections include the gnomonic projection (projection='gnom') and stereographic projection (projection='stere'). These are often the most useful for showing small portions of the map.

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None,
            lat_0=50, lon_0=0)
draw_map(m);
```

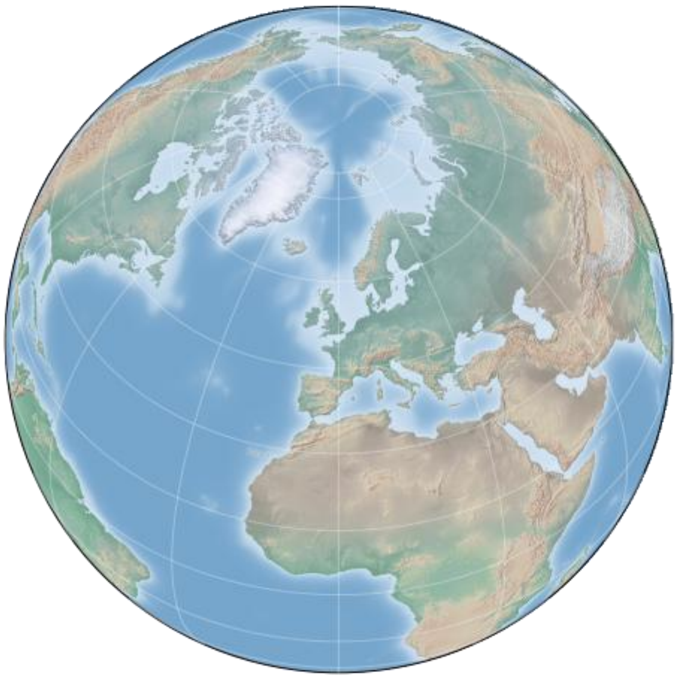


Fig. 2.21.

Conic Projections

A Conic projection projects the map onto a single cone, which is then unrolled. This can lead to very good local properties, but regions far from the focus point of the cone may become very distorted. One example of this is the Lambert Conformal Conic projection (projection='lcc'), which we saw earlier in the map of North America. It projects the map onto a cone arranged in such a way that two standard parallels (specified in Basemap by lat_1 and lat_2) have well-represented distances, with scale decreasing between them and increasing outside of them. Other useful conic projections are the equidistant conic projection (projection='eqdc') and the

Albers equal-area projection (projection='aea'). Conic projections, like perspective projections, tend to be good choices for representing small to medium patches of the globe.

```
In [8]:
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
            lon_0=0, lat_0=50, lat_1=45, lat_2=55,
            width=1.6E7, height=1.2E7)
draw_map(m)
```

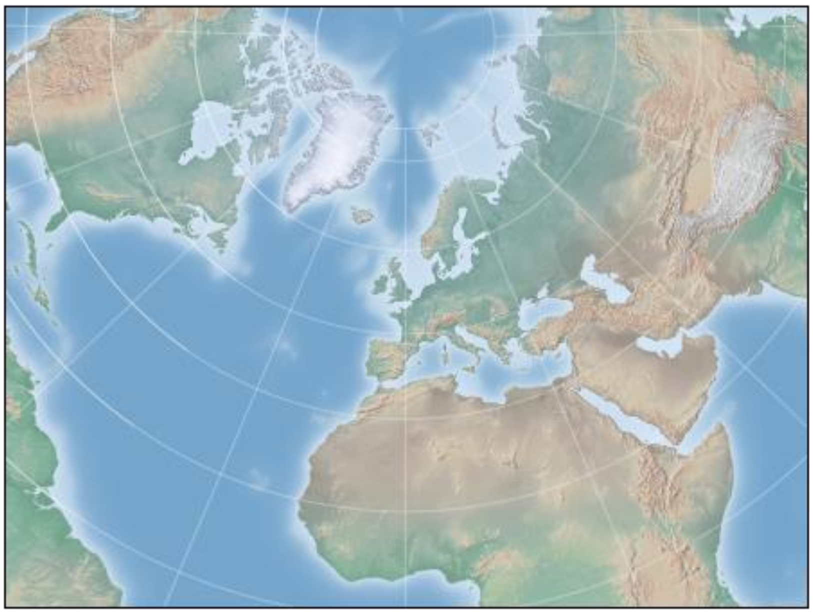


Fig. 2.22.

Drawing a Map Background

Earlier we saw the bluemarble() and shadedrelief() methods for projecting global images on the map, as well as the drawparallels() and drawmeridians() methods for drawing lines of constant latitude and longitude. The Basemap package contains a range of useful functions for drawing borders of physical features like continents, oceans, lakes, and rivers, as well as political boundaries such as countries and US states and counties. The following are some of the available drawing functions that you may wish to explore using IPython's help features:

Physical boundaries and bodies of water

drawcoastlines(): Draw continental coast lines

drawlsmask(): Draw a mask between the land and sea, for use with projecting images on one or the other

drawmapboundary(): Draw the map boundary, including the fill color for oceans.

drawrivers(): Draw rivers on the map

fillcontinents(): Fill the continents with a given color; optionally fill lakes with another color

Political boundaries

drawcountries(): Draw country boundaries

drawstates(): Draw US state boundaries

drawcounties(): Draw US county boundaries

Map features

drawgreatcircle(): Draw a great circle between two points

drawparallels(): Draw lines of constant latitude

drawmeridians(): Draw lines of constant longitude

drawmapscale(): Draw a linear scale on the map

Whole-globe images

bluemarble(): Project NASA's blue marble image onto the map

shadedrelief(): Project a shaded relief image onto the map

etopo(): Draw an etopo relief image onto the map

warpimage(): Project a user-provided image onto the map

Plotting Data on Maps

Perhaps the most useful piece of the Basemap toolkit is the ability to over-plot a variety of data onto a map background. For simple plotting and text, any plt function works on the map; you can use the Basemap instance to project latitude and longitude coordinates to (x, y) coordinates for plotting with plt, as we saw earlier in the Seattle example.

In addition to this, there are many map-specific functions available as methods of the Basemap instance. These work very similarly to their standard Matplotlib counterparts, but have an additional Boolean argument latlon, which if set to True allows you to pass raw latitudes and longitudes to the method, rather than projected (x, y) coordinates.

Some of these map-specific methods are:

contour()/contourf() : Draw contour lines or filled contours

imshow(): Draw an image

pcolor()/pcolormesh() : Draw a pseudocolor plot for irregular/regular meshes

plot(): Draw lines and/or markers.

scatter(): Draw points with markers.

quiver(): Draw vectors.

barbs(): Draw wind barbs.

drawgreatcircle(): Draw a great circle.

Example: California Cities

Recall that in Customizing Plot Legends, we demonstrated the use of size and color in a scatter plot to convey information about the location, size, and population of California cities. Here, we'll create this plot again, but using Basemap to put the data in context.

We start with loading the data, as we did before:

```
import pandas as pd
cities = pd.read_csv('data/california_cities.csv')
# Extract the data we're interested in
lat = cities['latd'].values
```

```
lon = cities['longd'].values
population = cities['population_total'].values
area = cities['area_total_km2'].values
```

Next, we set up the map projection, scatter the data, and then create a colorbar and legend:

```
# 1. Draw the map background
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution='h',
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
m.drawstates(color='gray')
# 2. scatter city data, with color reflecting population
# and size reflecting area
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap='Reds', alpha=0.5)
# 3.make legend with dummy points
for a in [100, 300, 500]:
    plt.scatter([], [], c='k', alpha=0.5, s=a,
                label=str(a) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc='lower left');
```

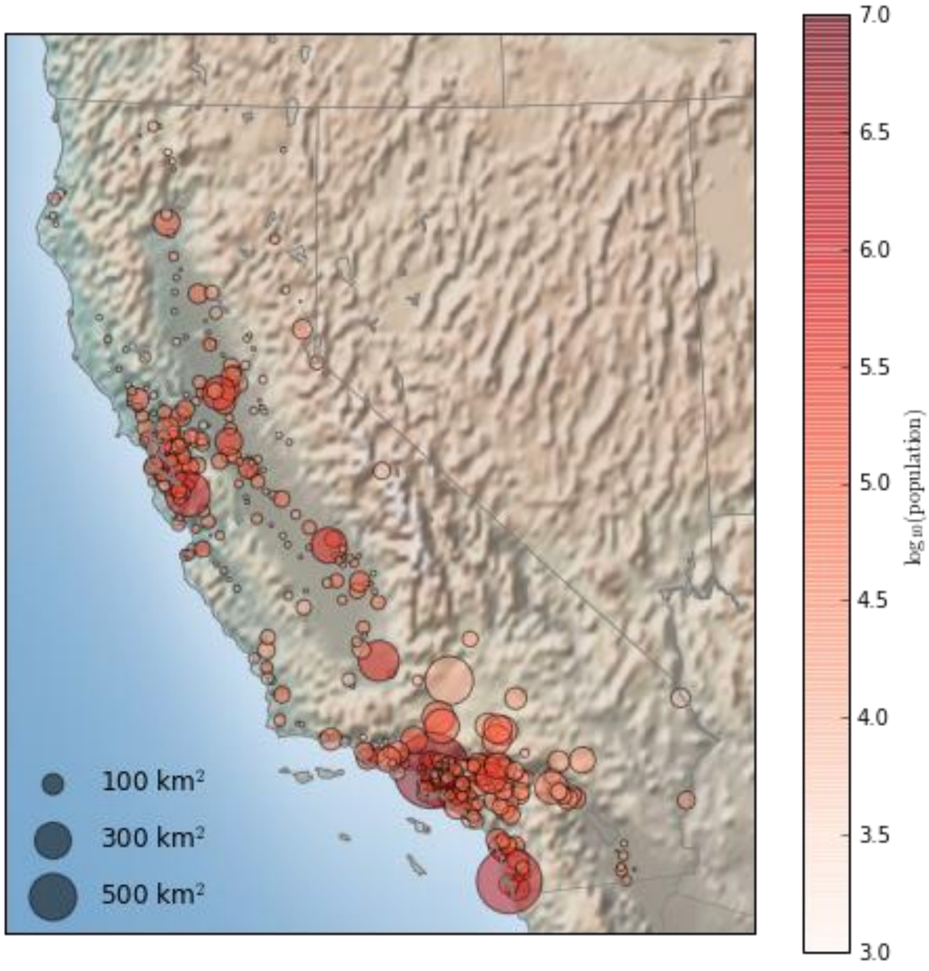


Fig. 2.23.

This shows us roughly where larger populations of people have settled in California: they are clustered near the coast in the Los Angeles and San Francisco areas, stretched along the highways in the flat central valley, and avoiding almost completely the mountainous regions along the borders of the state.

2.14. VISUALIZATION WITH SEABORN

Data Visualization is the presentation of data in pictorial format. It is extremely important for Data Analysis, primarily because of the fantastic ecosystem of data-centric Python packages. And it helps to understand the data, however, complex it is, the significance of data by summarizing and presenting a huge amount of data in a simple and easy-to-understand format and helps communicate information clearly and effectively.

Pandas

Pandas offer tools for cleaning and process your data. It is the most popular Python library that is used for data analysis. In pandas, a data table is called a dataframe.

Example 1:

```
# Python code demonstrate creating
  import pandas as pd
  # initialise data of lists.
data = {'Name':[ 'Mohe' , 'Karnal' , 'Yrik' , 'jack' ],
        'Age':[ 30 , 21 , 29 , 28 ]}
  # Create DataFrame
df = pd.DataFrame( data )
  # Print the output.
df
```

Output:

	Name	**Age**
0	Mohe	30
1	Karnal	21
2	Yrik	29
3	Jack	28

Example 2:

Load the CSV data from the system and display it through pandas.

```
# import module
import pandas
 # load the csv
```

```
data = pandas.read_csv("nba.csv")
 # show first 5 column
data.head()
```

Output:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

Seaborn

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It is built on the top of matplotlib library and also closely integrated into the data structures from pandas.

Installation

For python environment :

```
pip install seaborn
```

Let’s create Some basic plots using seaborn:

```
# Importing libraries
import numpy as np
import seaborn as sns
  # Selecting style as white,
# dark, whitegrid, darkgrid
# or ticks
sns.set( style = "white" )
 # Generate a random univariate
# dataset
```

```
rs = np.random.RandomState( 10 )
d = rs.normal( size = 50 )
  # Plot a simple histogram and kde
# with binsize determined automatically
sns.distplot(d, kde = True, color = "g")
```

Output:

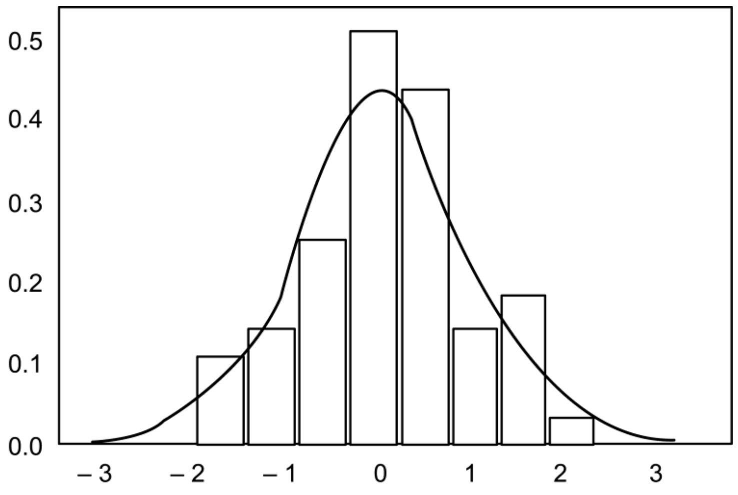


Fig. 2.24.

Seaborn: statistical data visualization

Seaborn helps to visualize the statistical relationships, To understand how variables in a dataset are related to one another and how that relationship is dependent on other variables, we perform statistical analysis. This Statistical analysis helps to visualize the trends and identify various patterns in the dataset.

These are the plot will help to visualize:

Line Plot

Scatter Plot

Box plot

Bar plot

KDE Plot

Line plot:

Lineplot Is the most popular plot to draw a relationship between x and y with the possibility of several semantic groupings.

Syntax : sns.lineplot(x=None, y=None)

Parameters:

x, y: Input data variables; must be numeric. Can pass data directly or reference columns in data.

Let’s visualize the data with a line plot and pandas:

Example 1:

```
# import module
import seaborn as sns
import pandas
 # loading csv
data = pandas.read_csv("nba.csv")
# plotting lineplot
sns.lineplot( data['Age'], data['Weight'])
```

Output:

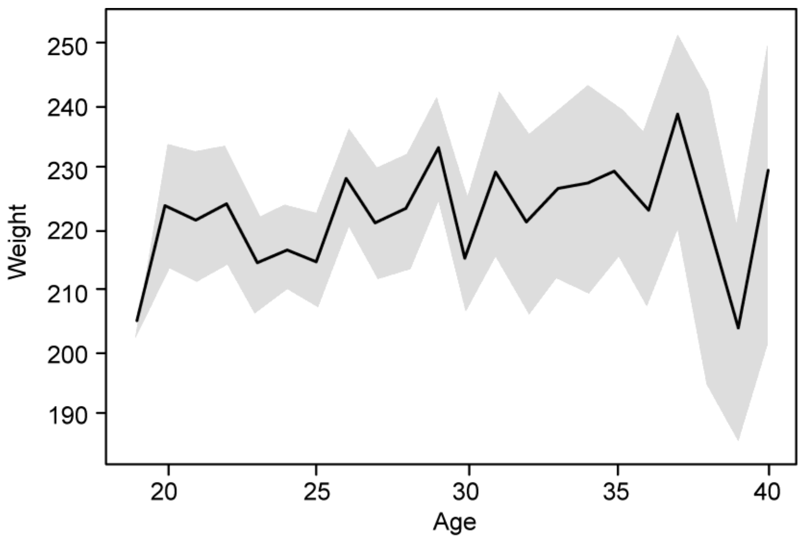


Fig. 2.25.

Scatter Plot:

Scatterplot Can be used with several semantic groupings which can help to understand well in a graph against continuous/categorical data. It can draw a two-dimensional graph.

Syntax: seaborn.scatterplot(x=None, y=None)

Parameters:

y: Input data variables that should be numeric.

Returns: This method returns the Axes object with the plot drawn onto it.

Example 1:

```
# import module
import seaborn
import pandas
 # load csv
data = pandas.read_csv("nba.csv")
 # plotting
seaborn.scatterplot(data['Age'],data['Weight'])
```

Output:

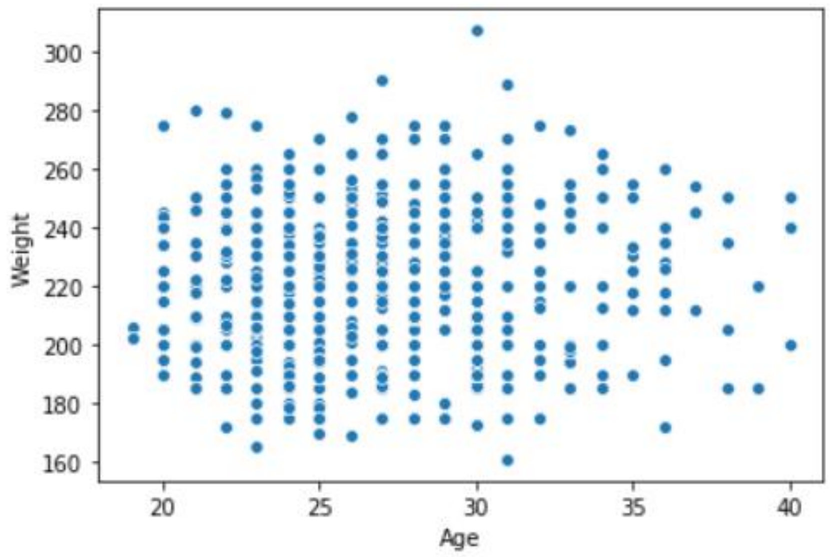


Fig. 2.26.

Box Plot:

A box plot (or box-and-whisker plot) s is the visual representation of the depicting groups of numerical data through their quartiles against continuous/categorical data.

A box plot consists of 5 things.

Minimum

First Quartile or 25%

Median (Second Quartile) or 50%

Third Quartile or 75%

Maximum

Syntax:

seaborn.boxplot(x=None, y=None, hue=None, data=None)

Parameters:

x, y, hue: Inputs for plotting long-form data.

data: Dataset for plotting. If x and y are absent, this is interpreted as wide-form.

Returns: It returns the Axes object with the plot drawn onto it.

Draw the box plot with Pandas:

Example 1:

Python3

```
# import module
import seaborn as sns
import pandas
 # read csv and plotting
data = pandas.read_csv( "nba.csv" )
sns.boxplot( data['Age'] )
```

Output:

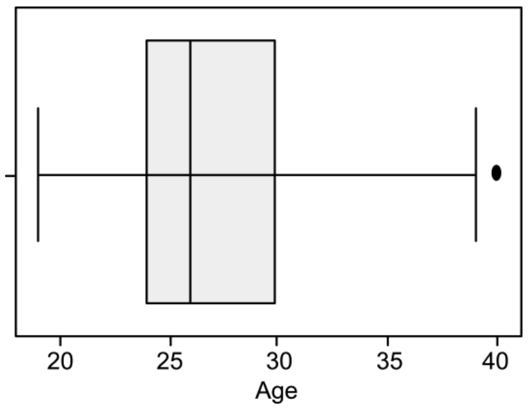


Fig. 2.27.

Bar Plot:

Barplot represents an estimate of central tendency for a numeric variable with the height of each rectangle and provides some indication of the uncertainty around that estimate using error bars.

Syntax : seaborn.barplot(x=None, y=None, hue=None, data=None)

Parameters :

x, y : This parameter take names of variables in data or vector data, Inputs for plotting long-form data.

hue : (optional) This parameter take column name for colour encoding.

data : (optional) This parameter take DataFrame, array, or list of arrays, Dataset for plotting. If x and y are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.

Returns : Returns the Axes object with the plot drawn onto it.

Example 1:

```
# import module
import seaborn
 seaborn.set(style = 'whitegrid')
 # read csv and plot
data = pandas.read_csv("nba.csv")
seaborn.barplot(x =data["Age"])
```

Output:

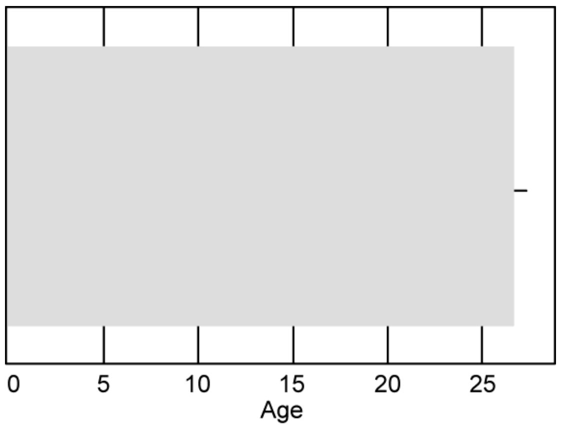


Fig. 2.28.

KDE Plot:

KDE Plot described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

Syntax: seaborn.kdeplot(x=None, *, y=None, vertical=False, palette=None, **kwargs)

Parameters:

x, y : vectors or keys in data

vertical : boolean (True or False)

data : pandas.DataFrame, numpy.ndarray, mapping, or sequence

Draw the KDE plot with Pandas:

Example 1:

```
# importing the required libraries
from sklearn import datasets
import pandas as pd
import seaborn as sns
  # Setting up the Data Frame
```

```
iris = datasets.load_iris()
  iris_df = pd.DataFrame(iris.data, columns=['Sepal_Length',
               'Sepal_Width', 'Patal_Length', 'Petal_Width'])
  iris_df['Target'] = iris.target
 iris_df['Target'].replace([0], 'Iris_Setosa', inplace=True)
iris_df['Target'].replace([1], 'Iris_Vercicolor', inplace=True)
iris_df['Target'].replace([2], 'Iris_Virginica', inplace=True)

# Plotting the KDE Plot
sns.kdeplot(iris_df.loc[(iris_df['Target'] =='Iris_Virginica'),
         'Sepal_Length'], color = 'b', shade = True, Label ='Iris_Virginica')
```

Output:

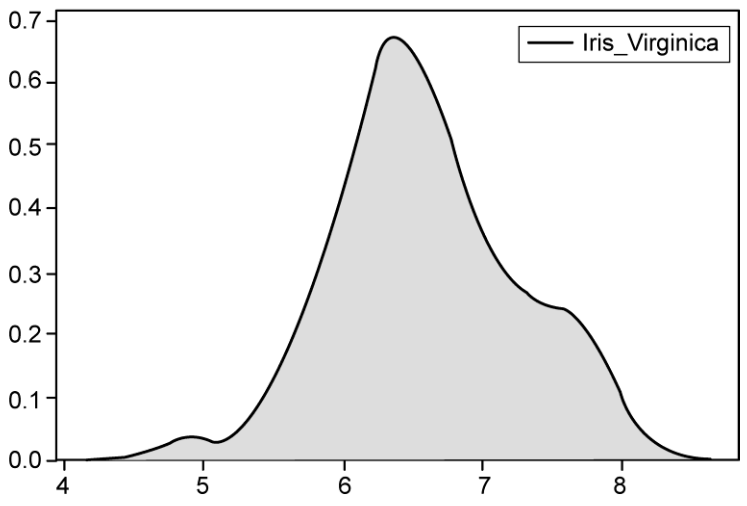


Fig. 2.29.

Bivariate and Univariate data using seaborn and pandas:

Bivariate data: This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship between the two variables.

Univariate data: This type of data consists of only one variable. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.

Let's see an example of Bivariate data :

Example 1: Using the box plot.

```
# import module
import seaborn as sns
import pandas
 # read csv and plotting
data = pandas.read_csv( "nba.csv" )
sns.boxplot( data['Age'], data['Height'])
```

Output:

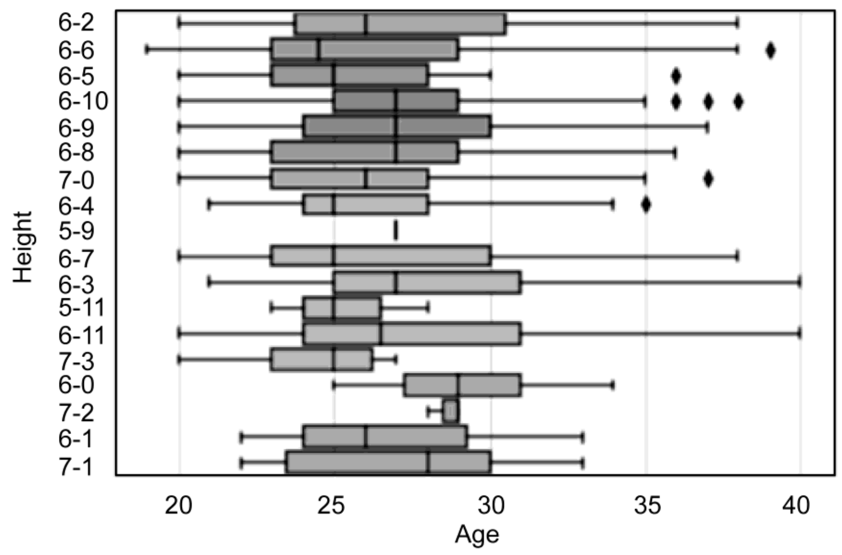


Fig. 2.30.

Let's see an example of univariate data distribution:

Example: Using the dist plot

```
# import module
import seaborn as sns
import pandas
 # read top 5 column
data = pandas.read_csv("nba.csv").head()
 sns.distplot( data['Age'])
```

Output:

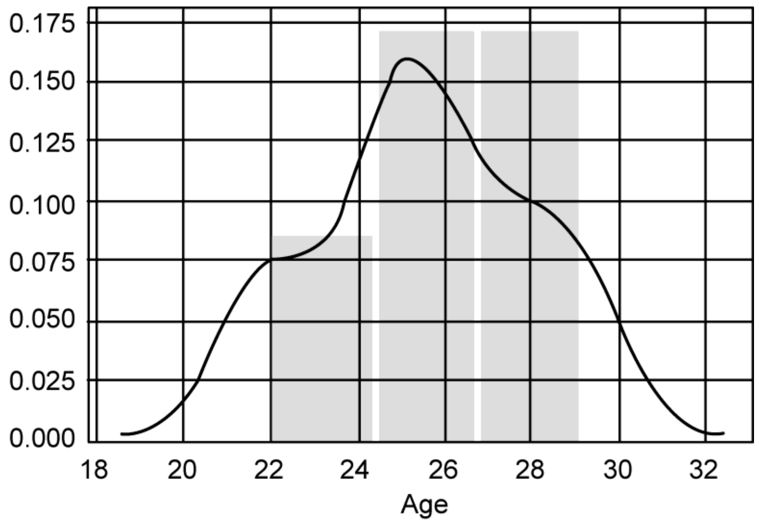


Fig. 2.31.

TWO MARKS QUESTION AND ANSWERS (PART-A)

1. ***What is a Matplotlib?***

   Matplotlib is a low level graph plotting library in python that serves as a visualization utility.Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility. Matplotlib is a Python library that helps to plot graphs. It is used in data visualization and graphical plotting.

2. ***What is a simple line plot?***

   The simplest of all plots is the visualization of a single function y=f(x)y=f(x). Here we will take a first look at creating a simple plot of this type. we'll start by plotting and importing the packages we will use:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

3. *List down the steps involved in line plot*

- ❖ Adjusting the Plot: Line Colors and Styles
- ❖ Adjusting the Plot: Axes Limits
- ❖ Labeling Plots

4. *What is a simple scatter plot?*

Another commonly used plot type is the simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

5. *How do you visualize error?*

For any scientific measurement, accurate accounting for errors is nearly as important, if not more important, than accurate reporting of the number itself. In visualization of data and results, showing these errors effectively can make a plot convey much more complete information

6. *What is a contour plot?*

Sometimes it is useful to display three-dimensional data in two dimensions using contours or color-coded regions. There are three Matplotlib functions that can be helpful for this task: plt.contour for contour plots, plt.contourf for filled contour plots, and plt.imshow for showing images.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np
```

7. *How will you create a histogram in matplotlib?*

To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

8. ***What is a Subplot?***

There are many cases where you will want to generate a plot that contains several smaller plots within it. That is exactly what a subplot is! A common version of the subplot is the 4x4 subplot. Subplots can be very complicated to create when done properly.

9. ***How To Create Subplots in Python Using Matplotlib?***

We can create subplots in Python using matplotlib with the subplot method, which takes three arguments:

nrows: The number of rows of subplots in the plot grid.

ncols: The number of columns of subplots in the plot grid.

index: The plot that you have currently selected.

10. ***How do you annotate text and graph?***

Annotate using text

Matplotlib offers the ability to place text within a chart. The only condition is it requires the positioning co-ordinate of the x and y-axis to place the text.

Annotate using graph: plt.annotate()

To input text using matplotlib's "plt.annotate()" we need to declare two things, which is the "xy" coordinates which tells matplotlib where we want to input our text and the "s" attribute.

There is also an added attribute aswell, this is called the "arrowprops" attribute, which basically allows us to input an arrow pointing towards a specific point in our graph.

11. ***Brief three-dimensional Plotting in Python using Matplotlib***

Matplotlib was introduced keeping in mind, only two-dimensional plotting. But at the time when the release of 1.0 occurred, the 3d utilities were developed upon the 2d and thus, we have 3d implementation of data available today! The

3d plots are enabled by importing the mplot3d toolkit. In this article, we will deal with the 3d plots using matplotlib.

Example:

```
 import numpy as np
import matplotlib.pyplot as plt
 fig = plt.figure()
ax = plt.axes(projection ='3d')
```

12. ***What is Geographic data with basemap***

One common type of visualization in data science is that of geographic data. Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits which lives under the mpl_toolkits namespace. Admittedly, Basemap feels a bit clunky to use, and often even simple visualizations take much longer to render than you might hope. More modern solutions such as leaflet or the Google Maps API may be a better choice for more intensive map visualizations. Still, Basemap is a useful tool for Python users to have in their virtual toolbelts.

13. ***What is Visualization with seaborn***

Data Visualization is the presentation of data in pictorial format. It is extremely important for Data Analysis, primarily because of the fantastic ecosystem of data-centric Python packages. And it helps to understand the data, however, complex it is, the significance of data by summarizing and presenting a huge amount of data in a simple and easy-to-understand format and helps communicate information clearly and effectively

14. ***Define a KDE plot?***

KDE Plot described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization.

Syntax: seaborn.kdeplot(x=None, *, y=None, vertical=False, palette=None, **kwargs)

15. *Differentiate Bivariate and Univariate data using seaborn and pandas:*

Bivariate data: This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship between the two variables.

Univariate data: This type of data consists of only one variable. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.

16. *What is a Box plot?*

A box plot (or box-and-whisker plot) s is the visual representation of the depicting groups of numerical data through their quartiles against continuous/categorical data.

A box plot consists of 5 things.

Minimum

First Quartile or 25%

Median (Second Quartile) or 50%

Third Quartile or 75%

Maximum

17. *What are the different map projections available?*

Cylindrical projections

Pseudo-cylindrical projections

Perspective projections

Conic projections

18. *Define Customization*

Customization challenges the boundaries of business analytics by constantly reinventing the delicate configurations between a user's creativity and fundamental design principles. For that reason, it aims to showcase better pictorial representations with more flexible and interactive use of the underlying data.

19. ***Distinguish plot vs scatter***

As datasets get larger than a few thousand points, plt.plot can be noticeably more efficient than plt.scatter. The reason is that plt.scatter has the capability to render a different size and/or color for each point, so the renderer must do the extra work of constructing each point individually. In plt.plot, on the other hand, the points are always essentially clones of each other, so the work of determining the appearance of the points is done only once for the entire set of data. For large datasets, the difference between these two can lead to vastly different performance, and for this reason, plt.plot should be preferred over plt.scatter for large datasets.

20. ***How do you Install Matplotlib?***

Matplotlib can be installed using pip. The following command is run in the command prompt to install Matplotlib.

```
pip install matplotlib
```

This command will start downloading and installing packages related to the matplotlib library.

PART- B & C

1. Explain the simple line plot using matplotlib
2. Elaborate the simple scatter plot in matplotlib
3. Detail the process of visualizing errors in matplotlib
4. What is a sub-plot. Explain using mat plot lib
5. Explain geographic data three dimensional plotting with base map
6. Elucidate visualization with sea-born

****

UNIT III

UNIVARIATE ANALYSIS

SYLLABUS

Introduction to Single variable: Distributions and Variables - Numerical Summaries of Level and Spread - Scaling and Standardizing – Inequality - Smoothing Time Series.

- ❖ Introduction to Single Variable
- ❖ Distributions and Variables
- ❖ Numerical Summaries of Level and Spread
- ❖ Scaling and Standardizing
- ❖ Inequality
- ❖ Smoothing Time Series

UNIT III
UNIVARIATE ANALYSIS

3.1. INTRODUCTION TO SINGLE VARIABLE

3.1.1. DISTRIBUTIONS AND VARIABLES

How many households have no access to a car? What is a typical household income in Britain? Which country in Europe has the longest working hours? To answer these kinds of questions we need to collect information from a large number of people, and we need to ensure that the people questioned are broadly representative of the population we are interested in.

Conducting large-scale surveys is a time-consuming and costly business. However, increasingly information or data from survey research in the social sciences are available free of charge to researchers and students. The development of the worldwide web and the ubiquity and power of computers makes accessing these types of data quick and easy.

The aim is to explore data. We can use the 'Statistical Package for the Social Sciences' (SPSS) package to start analysing data and answering the questions posed above.

Preliminaries

Two organizing concepts have become the basis of the language of data analysis: cases and variables. The cases are the basic units of analysis, the things about which information is collected. The word variable expresses the fact that this feature varies across different cases.

We will look at some useful techniques for displaying information about the values of single variables, and will also introduce the differences between interval level and ordinal level variables.

Variables on Household Survey

It is a multipurpose survey carried out by the social survey division of the Office for National Statistics (ONS). The main aim of the survey is to collect data on a range of core topics, covering household, family and individual information. Government departments and other organizations use this information for planning, policy and monitoring purposes, and to present a picture of households, family and people in Great Britain.

Person-id	Age	Sex	Units of alcohol per week	Drinking Classification	NC-SEC5
1	27	1	24	4	5
2	27	2	8	3	1
3	27	2	27	3	4
4	6	1	.	.	–6
5	5	1	.	.	–6
6	77	1	8	2	1
7	65	2	14	3	2
8	51	2	3	2	2
9	33	1	9	3	5
10	25	1	9	2	5
11	49	1	352	5	3
12	16	1	2	1	97
13	66	1	0	1	4
14	65	2	0	1	5
15	47	2	0	–9	2
16	42	1	6	1	2
17	15	1	.	.	–6
18	13	2	.	.	–6
19	47	2	0	1	1
20	44	1	5	1	1

Fig. 3.1. Specimen data from the 2005GHS (Individual file)

Column 5 contains a variable that indicates individuals classification of themselves in terms of the amount of alcohol they usually drink. It has five ranked categories:

1. hardly drink at all
2. drink a little
3. drink a moderate amount
4. drink quite a lot
5. drink heavily

Column 5 indicates the social class of individual based on the occupation.

1. Managerial and professional occupations
2. Intermediate occupations
3. Small employers and own account workers
4. Lower supervisory and technical occupations
5. Semi-routine occupations

Bar Charts and Pie Charts

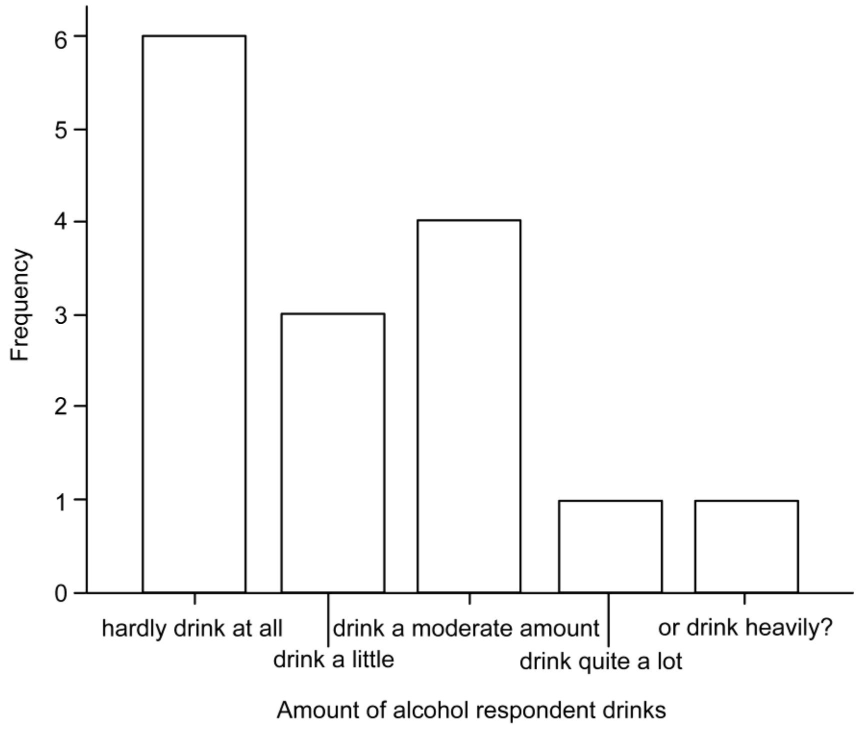


Fig. 3.2. Amount of alcohol respondent drinks

One simple device is the bar chart, a visual display in which bars are drawn to represent each category of a variable such that the length of the bar is proportional to the number of cases in the category.

A pie chart can also be used to display the same information. It is largely a matter of taste whether data from a categorical variable are displayed in a bar chart or a pie chart. In general, pie charts are to be preferred when there are only a few categories and when the sizes of the categories are very different.

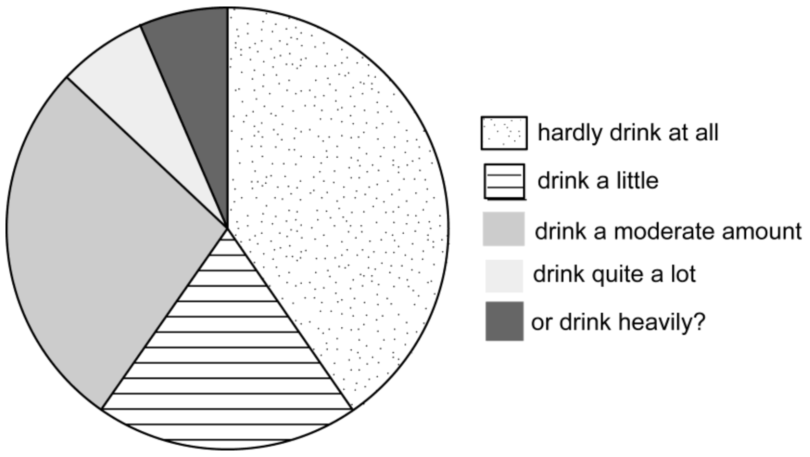


Fig. 3.3.

Bar charts and pie charts can be an effective medium of communication if they are well drawn.

Histograms

Charts that are somewhat similar to bar charts can be used to display interval level variables grouped into categories and these are called histograms. They are constructed in exactly the same way as bar charts except that the ordering of the categories is fixed, and care has to be taken to show exactly how the data were grouped.

3.2. NUMERICAL SUMMARIES OF LEVEL AND SPREAD

Let focus on the topic of working hours to demonstrate how simple descriptive statistics can be used to provide numerical summaries of level and spread. The chapter will begin by examining data on working hours in Britain taken from the

General Household Survey discussed in the previous chapter. These data are used to illustrate measures of level such as the mean and the median and measures of spread or variability such as the standard deviation and the midspread.

Working hours of couples in Britain

The histograms of the working hours distributions of men and women in the 2005 General Household Survey are shown in figures 3.1 and 3.2. We can compare these two distributions in terms of the four features introduced in the previous chapter, namely level, spread, shape and outliers. We can then see that:

- ❖ The male batch is at a higher level than the female batch
- ❖ The two distributions are somewhat similarly spread out
- ❖ The female batch is bimodal suggesting there are two rather different underlying populations
- ❖ The male batch is uni-modal

Summaries of level

The level expresses where on the scale of numbers found in the dataset the distribution is concentrated

Residuals

Another way of expressing this is to say that the residual is the observed data value minus the predicted value and in this case 45 – 40 = 5. Any data value such as a measurement of hours worked or income earned can be thought of as being composed of two components: a fitted part and a residual part. This can be expressed as an equation:

$$\text{Data} = \text{Fit} + \text{Residual}$$

The median

The value of the case at the middle of an ordered distribution would seem to have an intuitive claim to typicality. Finding such a number is easy when there are very few cases. In the example of hours worked by a small random sample of 15 men (figure 3.4), the value of 48 hours per week fits the bill. There are six men who work fewer hours and seven men who work more hours while two men work exactly 48 hours per week. Similarly, in the female data, the value of the middle case is 3 7 hours. The data value that meets this criterion is called the median: the value of the case that has equal numbers of data points above and below it. The median hours worked by women in this very small sample is 11 hours less than the median for men. This numeric summary of the level of the data therefore confirms our first impressions from simply looking at the histograms in figures 3.1 and 3.2 that women generally work shorter hours than men.0020

Men's working hours (ranked)
30
37
39
40
45
47
48
Median value 48
50
54
55
55
67
70
80

Fig. 3.4. Men's working hours ranked to show the median

The arithmetic mean

Another commonly used measure of the centre of a distribution is the arithmetic mean. Indeed, it is so commonly used that it has even become known as the average. It is conventionally written as A (pronounced 'A bar'). To calculate it, first all of the values are summed, and then the total is divided by the number of data points. In more mathematical terms:

$$\frac{1}{N} \sum_{i=1}^{N} \gamma_i$$

We have come across N before. The symbol Y is conventionally used to refer to an actual variable. The subscript *i* is an index to tell us which case is being referred to. So, in this case, Y; refers to all the values of the hours variable. The Greek letter 2, pronounced 'sigma', is the mathematician's way of saying 'the sum of'.

Summaries of Spread

The second feature of a distribution visible in a histogram is the degree of variation or spread in the variable.

Once again, there are many candidates we could think of to summarize the spread. One might be the distance between the two extreme values (the range). Or we might work out what was the most likely difference between any two cases drawn at random from the dataset.

The midspread

The range of the middle 5 0 per cent of the distribution is a commonly used measure of spread because it concentrates on the middle cases. It is quite stable from sample to sample. The points which divide the distribution into quarters are called the quartiles (or sometimes 'hinges' or 'fourths'). The lower quartile is usually denoted QL and the upper quartile Q0. (The middle quartile is of course the median.) The distance between QL and Q0 is called the midspread (sometimes the 'interquartile range'), or the dQ for short.

Men's working hours (ranked)	
30	
37	
39	
40	
	$Q_L = 42.5$
45	
47	
48	
48	
50	
54	
55	
	$Q_U = 55$
55	
67	
70	
80	

Fig. 3.5. Men's working hours ranked and showing the upper and lower quartiles

There is a measure of spread which can be calculated from these squared distances from the mean. The standard deviation essentially calculates a typical value of these distances from the mean. It is conventionally denoted s, and defined as:

$$s = \sqrt{\left[\frac{\Sigma (Y_i - \bar{Y})^2}{(N-1)}\right]}$$

The deviations from the mean are squared, summed and divided by the sample size and then the square root is taken to return to the original units. The order in which the calculations are performed is very important. As always, calculations within brackets are performed first, then multiplication and division, then addition (including summation) and subtraction. Without the square root, the measure is called the variance, s2• The layout for a worksheet to calculate the standard deviation of the hours worked by this small sample of men is shown in figure 3.6.

Y	**Y – Y**	**(Y – Y)^2**
54	3	9
30	–21	441
47	–4	16
39	–12	144
50	–1	1
48	–3	9
45	–6	36
40	–11	121
37	–14	196
48	–3	9
67	16	256
55	4	16
55	4	16
80	29	841
70	19	361
Sum = 765		Sum of squared residuals = 2472

Fig. 3.6. Worksheet for standard deviation of men's weekly working hours

$$s = \sqrt{\left[\frac{\Sigma\,(Y_i - \bar{Y})^2}{(N-1)}\right]} = \sqrt{\frac{2472}{14}} = 13.29$$

Interpreting Locational Summaries

In the examples discussed above the locational statistics for only a very small subsample of data of 15 cases from the GHS 2005 have been calculated by hand. It is useful to experiment with calculating locational statistics in this way in order to reach a better understanding of the meaning of these summary statistics. However, with larger batches of data the median, quartiles (and deciles) can be calculated very easily using a package such as Excel or SPSS.

Total Work Hours (Men)

N	Valid	6392
	Missing	8188
Median		39. 000
Minimum		.00
Maximum		97.00
Percentiles	25	37.0000
	50	39.0000
	75	42.8750

Total Work Hours (Women)

N	Valid	6127
	Missing	9362
Median		35.0000
Minimum		.00
Maximum		97.00
Percentiles	25	20.0000
	50	35.0000
	75	37.5000

We can see that on average men tend to work more hours per week than women (39.2 hours vs 29.6 hours) and also the higher standard deviation for women, 12.3 vs 11.6 for men indicates that there is more variation among women in terms of the hours they usually work per week. It should also be noted that the figures for the means and standard deviations are pasted directly from the SPSS output. We can see that in each case the number of decimal places provided is four for the mean and five for the standard deviation.

Total Work Hours (Men)

N	Valid	6392
	Missing	8188
Mean		39.2268
Std. Deviation		11.64234

Total Work Hours (Women)

N	Valid	6127
	Missing	9362
Mean		29.5977
Std. Deviation		12.31122

3.3. SCALING AND STANDARDIZING

Data are produced not given

The word 'data' must be treated with caution. Literally translated, it means 'things that are given'.

There are often problems with using official statistics, especially those which are the by-products of some administrative process like, for example, reporting deaths to the Registrar-General or police forces recording reported crimes. Data analysts have to learn to be critical of the measures available to them, but in a constructive manner. As well as asking 'Are there any errors in this measure?' we also have to ask 'Is there anything better available?' and, if not, 'How can I improve what I've got?'

Improvements can often be made to the material at hand without resorting to the expense of collecting new data.

We must feel entirely free to rework the numbers in a variety of ways to achieve the following goals:

- ❖ to make them more amenable to analysis
- ❖ to promote comparability
- ❖ to focus attention on differences.

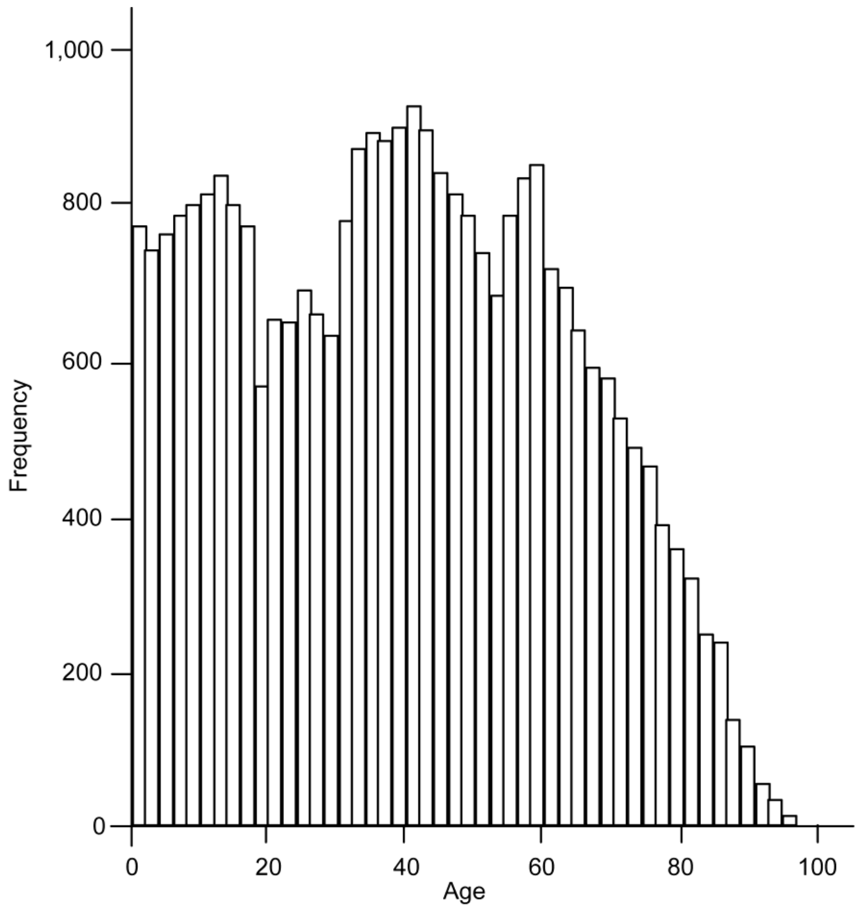


Fig. 3.7. Histogram

Consider various manipulations that can be applied to the data to achieve the above goals:

(*i*) Adding or subtracting a constant

One way of focusing attention on a particular feature of a dataset is to add or subtract a constant from every data value.

For example, in a set of data on weekly family incomes, it would be possible to subtract the median from each of the data values, thus drawing attention to which families had incomes below or above a hypothetical typical family.

The change made to the data by adding or subtracting a constant is fairly trivial. Only the level is affected; spread, shape and outliers remain unaltered. The reason for doing it is usually to force the eye to make a division above and below a particular point. A negative sign would be attached to all those incomes which were below the median in the example above. However, we sometimes add or subtract a constant to bring the data within a particular range.

(*ii*) Mu1ultiplying or dividing by a constant

Instead of adding a constant, we could change each data point by multiplying or dividing it by a constant.

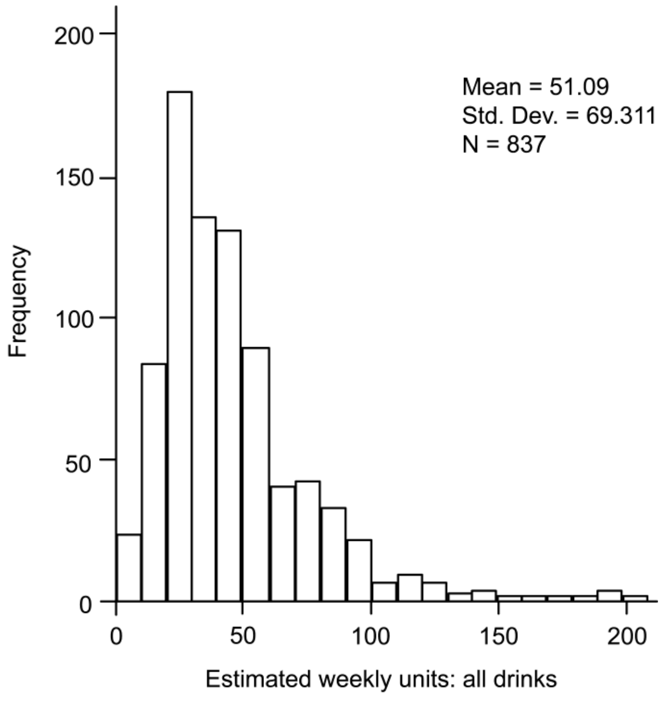


Fig. 3.8.(a) Histogram of weekly alcohol consumption of men who describe themselves as 'drinking quite a lot' or 'heavy drinkers'

A common example of this is the re-expression of one currency in terms of another. For example, in order to convert pounds to US dollars, the pounds are multiplied by the current exchange rate. Multiplying or dividing each of the values has a more powerful effect than adding or subtracting. The result of multiplying or dividing by a constant is to scale the entire variable by a factor, evenly stretching or shrinking the axis like a piece of elastic. To illustrate this, let us see what happens if data from the General Household Survey on the weekly alcohol consumption of men who classify themselves as moderate or heavy drinkers are divided by seven to give the average daily alcohol consumption.

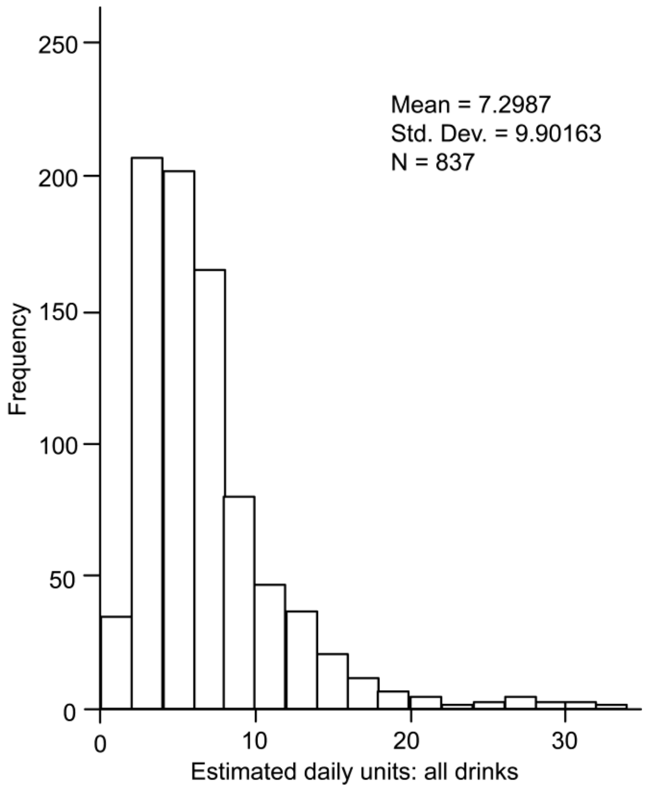


Fig. 3.8. (b) Histogram of daily alcohol consumption of men who describe themselves as 'drinking quite a lot' or 'heavy drinkers'

The overall shape of the distributions in figures 3.8 (*a*) and 3.8 (*b*) are the same. The data points are all in the same order, and the relative distances between them have not been altered apart from the effects of rounding. The whole distribution has simply been scaled by a constant factor.

In SPSS it is very straightforward to multiply or divide a set of data by a constant value. For example, using syntax, the command to create the variable drday 'Average daily alcohol consumption' from the variable drating 'Average weekly alcohol consumption' is as follows:

```
COMPUTE DRDAY — DRATING/7.
```

Alternatively, to create a new variable 'NEWVAR' by multiplying an existing variable 'OLDVAR' by seven the syntax would be:

```
COMPUTE NEWVAR = OLDVAR*7.
```

The 'Compute' command can also be used to add or subtract a constant,

for example:

```
COMPUTE NEWVAR = OLDVAR + 100.
COMPUTE NEWVAR = OLDVAR – 60.
```

The value of multiplying or dividing by a constant is often to promote comparability between datasets where the absolute scale values are different. For example, one way to compare the cost of a loaf of bread in Britain and the United States is to express the British price in dollars. Percentages are the result of dividing frequencies by one particular constant - the total number of cases.

(*iii*) Standardized Variables

In sections 3.2 and 3.3, we saw that subtracting a constant from every data value altered the level of the distribution and dividing by a constant scaled the values by a factor. In this section we will look at how these two ideas may be combined to produce a very powerful tool which can render any variable into a form where it can be compared with any other. The result is called a standardized variable.

To standardize a variable, a typical value is first subtracted from each data point, and then each point is divided by a measure of spread. It is not crucial which numerical summaries of level and spread are picked. The mean and standard deviation could be used, or the median and midspread:

$$\frac{Y_i - \bar{Y}}{s} \quad \text{or} \quad \frac{Y_i - M(Y)}{dQ}$$

A variable which has been standardized in this way is forced to have a mean or median of 0 and a standard deviation or midspread of 1.

Two different uses of variable standardization are found in social science literature. The first is in building causal models, where it is convenient to be able to compare the effect that two different variables have on a third on the same scale.

The second use which is more immediately intelligible: standardized variables are useful in the process of building complex measures based on more than one indicator. In order to illustrate this, we will use some data drawn from the National Child Development Study (NCDS). This is a longitudinal survey of all children born in a single week of 1958.

There is a great deal of information about children's education in this survey. Information was sought from the children's schools about their performance at state examinations, but the researchers also decided to administer their own tests of attainment.

Rather than attempt to assess knowledge and abilities across the whole range of school subjects, the researchers narrowed their concern down to verbal and mathematical abilities. Each child was given a reading comprehension test which was constructed by the National Foundation for Educational Research for use in the study, and a test of mathematics devised at the University of Manchester.

The two tests were administered at the child's school and had very different methods of scoring. As a result they differed in both level and spread. As can be seen from the descriptive statistics in figure 3.4, the sixteen-year-olds in the National Child Development Study apparently found the mathematics test rather more difficult than the reading comprehension test. The reading comprehension was scored out of a total of 35 and sixteen-year- olds gained a mean score of 25.37, whereas the mathematics test was scored out of a possible maximum of 31, but the 16-year-olds only gained a mean score of 12.75.

Descriptive Statistics

	N	**Minimum**	**Maximum**	**Mean**	**Std.Deviation**
Age 16 Test 1–reading Comprehension	11920	0	35	25.37	7.024
Age 16 Test 2-Mathematics Comprehension	11920	0	31	12.75	6.997
Valid N (listwise)	11920				

Fig. 3.9. Descriptive statistics for reading comprehension and mathematics test scores from NCDS age 16

The first two columns of figure 3.11 show the scores obtained on the reading and mathematics test by fifteen respondents in this study. There is nothing inherently interesting or intelligible about the raw numbers. The first score of 31 for the reading test can only be assessed in comparison with what other children obtained. Both tests can be thought of as indicators of the child's general attainment at school. It might be useful to try to turn them into a single measure of that construct.

1 Raw reading score	**2** Raw maths score	**3** Standardized reading score	**4** Standardized maths score	**5** Composite score of attainment
31	17	0.8	0.61	1.41
33	20	1.09	1.04	2.12
31	21	0.8	1.18	1.98
30	14	0.66	0.18	0.84
28	14	0.37	0.18	0.55
31	11	0.8	–0.25	0.55
29	8	0.52	–0.68	–0.16
28	17	0.37	0.61	0.98
23	8	–0.34	–0.68	–1.02
25	13	–0.05	0.04	–0.02
19	8	–0.91	–0.68	–1.59
32	25	0.94	1.75	2.69
31	22	0.80	1.32	2.12
29	8	0.52	–0.68	–0.16
30	17	0.66	0.61	1.27

Fig. 3.10. Scores of reading and mathematics tests at age 16

In order to create such a summary measure of attainment at age 16, we want to add the two scores together. But this cannot be done as they stand, because as we saw before, the scales of measurement of these two tests are different. If this is not immediately obvious try the following thought experiment. A 16-year-old who is average at reading but terrible at mathematics will perhaps score 25.4 (i.e. the mean score) on the reading comprehension test and 0 on the mathematics test. If these were summed the total is 25.4. However, a 16-year-old who is average at mathematics but can't read is likely to score 12.7 (i.e. the mean score) on the maths score and 0 on the reading comprehension. If these are summed the total would only be 12.7. If the two tests can be forced to take the same scale, then they can be summed.

This is achieved by standardizing each score. One common way of standardizing is to first subtract the mean from each data value, and then divide the result by the standard deviation. This process is summarized by the following formula, where the original variable 'Y' becomes the standardized variable 'Z'

$$Z = (Y_i - \hat{Y}) / \text{St.Dev.}$$

For example, the first value of 31 in the reading test becomes:

$$(31 - 25.37) / 7 \text{ or } 0.8$$

The same individual's mathematics score becomes (17 – 12.75) / 7, or 0.61. This first respondent is therefore above average in both reading and maths. To summarize, we can add these two together and arrive at a score of 1.41 for attainment in general.

Similar calculations for the whole batch are shown in columns 3 and 4 of figure 3.11. We can see that the sixth person in this extract of data is above average in reading but slightly below average (by a quarter of a standard deviation) in mathematics. It should also be noted that any individual scoring close to the mean for both their reading comprehension and their mathematics test will have a total score close to zero. For example, the tenth case in figure 3.11 has a total score of – 0.02.

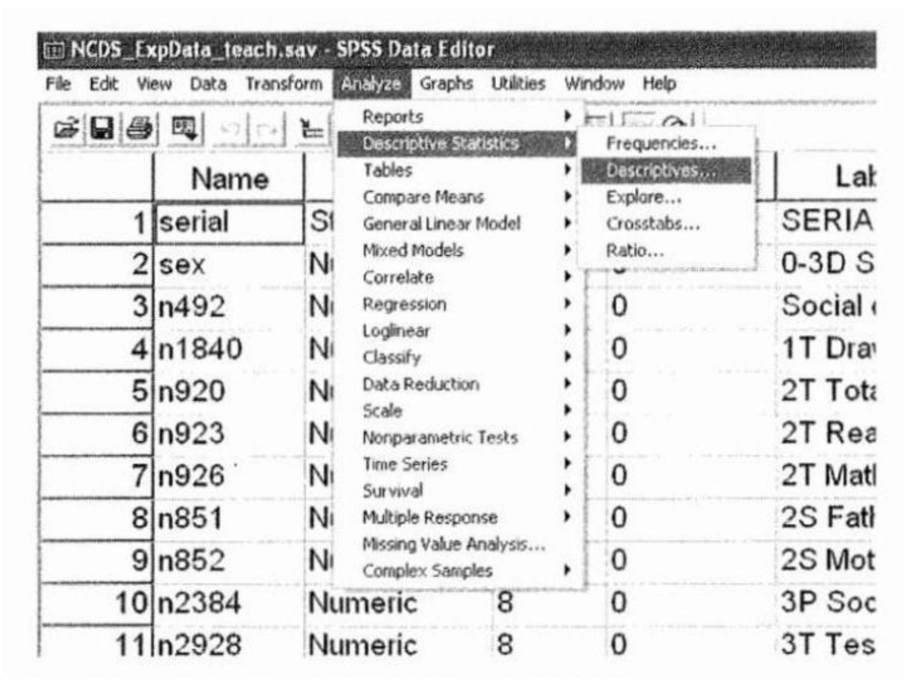
The final column of figure 3.11 now gives a set of summary scores of school attainment, created by standardizing two component scores and summing them, so attainment in reading and maths have effectively been given equal weight.

It is very straightforward to create standardized variables using SPSS. by using the Descriptives command, the SPSS package will automatically save a standardized version of any variable.

First select the menus.

Analyze > Descriptive Statistics > Descriptives

The next stage is to select the variables that you wish to standardize, in this case N2928 and N2930, and check the box next to 'Save standardized values as variables.' The SPSS package will then automatically save new standardized variables with the suffix Z. In this example, two new variables ZN2928 and ZN2930 are created.



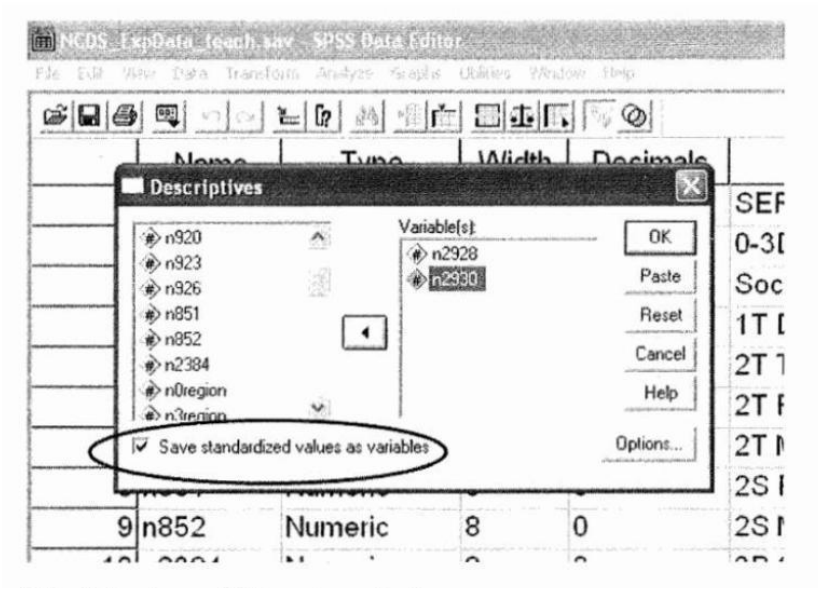
	Name				Lak
1	serial	St			SERIA
2	sex	Ni			0-3D S
3	n492	Ni		0	Social
4	n1840	Ni		0	1T Dra
5	n920	Ni		0	2T Tot
6	n923	Ni		0	2T Rea
7	n926	Ni		0	2T Matl
8	n851	Ni		0	2S Fatl
9	n852	Ni		0	2S Mot
10	n2384	Numeric	8	0	3P Soc
11	n2928	Numeric	8	0	3T Tes

Fig. 3.11. Creating standardized variables using SPSS

The syntax to achieve this is as follows:

```
DESCRIPTIVES
VARIABLES  = n2928 n2930 /
SAVE /STATISTICS = MEAN
STDDEV MIN MAX.
```

Standardizing the variables was a necessary, but not a sufficient condition for creating a simple summary score. It is also important to have confidence that the components are both valid indicators of the underlying construct of interest.



Variable(s): n2928, n2930

✓ Save standardized values as variables

Fig. 3.12. Selecting variables to standardize

(*iv*) The Gaussian distribution

We are now ready to turn to the third feature of distributions, their shape. With level and spread taken care of, the shape of the distribution refers to everything that is left. In order to summarize the shape of a distribution, it would need to be simple enough to be able to specify how it should be drawn in a very few statements. For example, if the distribution were completely flat (a uniform distribution), this would be possible. We would only need to specify the value of the extremes an the number of cases for it to be reproduced accurately, and it would be possible to say exactly what proportion of the cases fell above and below a certain level.

However, many distributions do have a characteristic shape — a lump in the middle and tails straggling out at both ends. How convenient it would be if there was an easy way to define a more complex shape like this and to know what proportion of the distribution would lie above and below different levels.

One such shape, investigated in the early nineteenth century by the German mathematician and astronomer, Gauss, and therefore referred to as the Gaussian distribution, is commonly used. It is possible to define a symmetrical, bell-shaped curve which looks like those in figure 3.14, and which contains fixed proportions of the distribution at different distances from the centre. The two curves in figure 3.14 look different — (*a*) has a smaller spread than (*b*) — but in fact they only differ by a scaling factor.

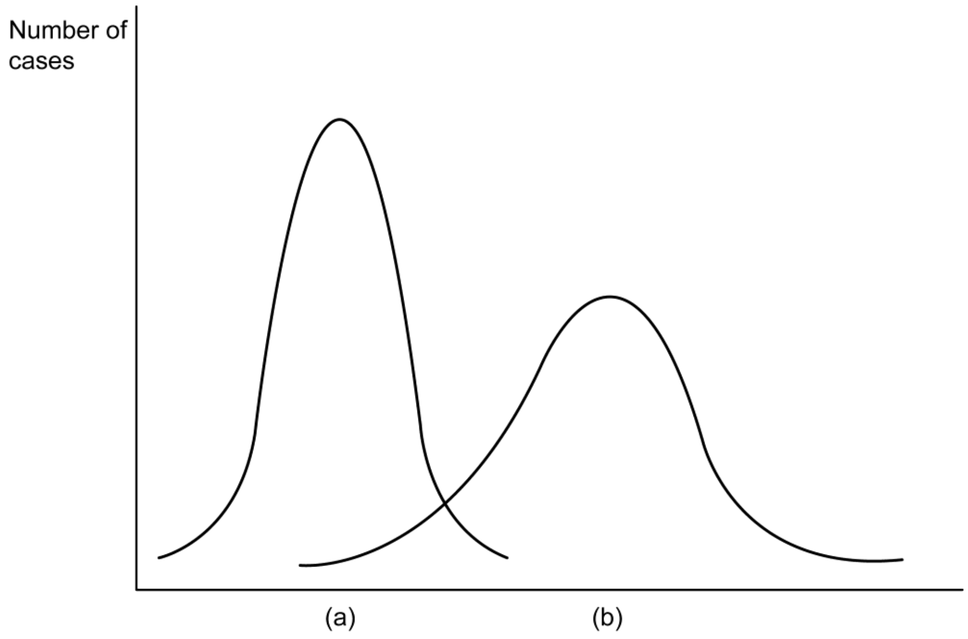


Fig. 3.13. The Gaussian distribution

Any Gaussian distribution has a very useful property: it can be defined uniquely by its mean and standard deviation. Given these two pieces of information, the exact shape of the curve can be reconstructed, and the proportion of the area under the curve falling between various points can be calculated.

This bell-shaped curve is often called ‘the normal distribution’. Its discovery was associated with the observation of errors of measurement. If sufficient repeated measurements were made of the same object, it was discovered that most of them centred around one value (assumed to be the true measurement), quite a few were fairly near the centre, and measurements fairly wide of the mark were unusual but did occur. The distribution of these errors of measurement often approximated to the bell-shape in figure 3.14.

(*v*)) Standardizing with respect to an appropriate base

In the scaling and standardizing techniques considered up to now, the same numerical adjustment has been made to each of the values in a batch of data. Sometimes, however, it can be useful to make the same conceptual adjustment to each data value, which may involve a different number in each case.

A batch of numbers may be reworked in several different ways in order to reveal different aspects of the story they contain. A dataset which can be viewed from several angles is shown in figure 3.15: the value of the lower quartile, the median and the upper quartile of male and female earnings in the period between 1990 and 2000. The data are drawn from the New Earnings Survey that collects information about earnings in a fixed period each year from the employers of a large sample of employees.

Year	Male Earnings			Female Earnings		
	QL	M	Qu	QL	M	Qu
1990	193.4	258.2	347.5	136.2	177.5	244.7
1991	206.9	277.5	376.5	150.6	195.7	271.6
1992	219.3	295.9	401.9	161.4	211.3	295.9
1993	226.0	304.6	417.3	168.2	221.6	309.1
1994	231.1	312.8	427.3	174.6	229.4	320.1
1995	237.1	323.2	442.7	179.5	237.2	332.5
1996	245.2	334.9	460.7	186.8	248.1	347.3
1997	256.4	349.7	480.0	196.1	260.5	364.7
1998	265.3	362.8	499.0	203.6	270.0	379.1
1999	274.5	374.3	517.3	213.3	284.0	398.2
2000	284.7	389.7	537.7	223.6	296.7	417.6

Fig. 3.14. Male and female earnings 1990-2000 gross earnings in pounds per week for full-time workers on adult rates whose pay was not affected by absence

As the figures stand, the most dominant feature of the dataset is a rather uninteresting one: the change in the value of the pound. While the median and mid-spreads of the money incomes each year have increased substantially in this period, real incomes and differentials almost certainly have not. How could we present the data in order to focus on the trend in real income differentials over time?

One approach would be to treat the distribution of incomes for each sex in each year as a separate distribution, and express each of the quartiles relative to the median. The result of doing this is given in figure 3.16.

Year	**Male Earnings**			**Female Earnings**		
	QL	**M**	**Qu**	**QL**	**M**	**Qu**
1990	75	100	135	77	100	138
1991	75	100	136	77	100	139
1992	74	100	136	76	100	140
1993	74	100	137	76	100	139
1994	74	100	137	76	100	140
1995	73	100	137	76	100	140
1996	73	100	138	75	100	140
1997	73	100	137	75	100	140
1998	73	100	138	75	100	140
1999	73	100	138	75	100	140
2000	73	100	138	75	100	141

Fig. 3.15. Male and female earnings relative to medians for each sex

3.4. INEQUALITY

Prosperity and Inequality :

There are a number of reasons why we might want to reduce inequality in society. For example, as Layard (2005) argues, if we accept that extra income has a bigger impact on increasing the happiness of the poor than the rich, this means that if some money is transferred from the rich to the poor this will increase the happiness of the poor more than it diminishes the happiness of the rich. This in turn suggests that the overall happiness rating of a country will go up if income is distributed more equally. Of course, as Layard acknowledges, the problem with this argument is that it only works if it is possible to reduce inequality without raising taxes to such an extent that there is no longer an incentive for individuals to strive to make money so that the total income is reduced as a result of policies aimed at redistribution. It is clearly important to understand the principal ways of measuring inequality if we are to monitor the consequences of changing levels of inequality in society. This chapter will focus on how we can measure inequality in such a way as to make it possible to compare levels of inequality in different societies and to look at changes in levels of inequality over time.

Income and Wealth :

Considered at the most abstract level, income and wealth are two different ways of looking at the same thing. Both concepts try to capture ways in which members of society have different access to the goods and services that are valued in that society. Wealth is measured simply in pounds, and is a snapshot of the stock of such valued goods that any person owns, regardless of whether this is growing or declining. Income is measured in pounds per given period, and gives a moving picture, telling us about the flow of revenue over time.

For the sake of simplicity, we restrict our focus to the distribution of income. We will look in detail at the problems of measuring income and then consider some of the distinctive techniques for describing and summarizing inequality that have evolved in the literature on economic inequality.

There are four major methodological problems encountered when studying the distribution of income:

1. How should income be defined?
2. What should be the unit of measurement?
3. What should be the time period considered?
4. What sources of data are available?

Definition of Income

To say that income is a flow of revenue is fine in theory, but we have to choose between two approaches to making this operational. One is to follow accounting and tax practices, and make a clear distinction between income and additions to wealth. With this approach, capital gains in a given period, even though they might be used in the same way as income, would be excluded from the definition. This is the approach of the Inland Revenue, which has separate taxes for income and capital gains. In this context a capital gain is defined as the profit obtained by selling an asset that has increased in value since it was obtained. However, interestingly, in most cases this definition (for the purposes of taxation) does not include any profit made when you sell your main home.

The second approach is to treat income as the value of goods and services consumed in a given period plus net changes in personal wealth during that period. This approach involves constantly monitoring the value of assets even when they do not come to the market. That is a very hard task.
So, although the second approach is theoretically superior, it is not very practical and the first is usually adopted.

The definition of income usually only includes money spent on goods and services that are consumed privately. But many things of great value to different people are organized at a collective level: health services, education, libraries, parks, museums, even nuclear warheads.

The benefits which accrue from these are not spread evenly across all members of society. If education were not provided free, only families with children would need to use their money income to buy schooling.

Sources of income are often grouped into three types:

- earned income, from either employment or self-employment;
- unearned income which increases from ownership of investments, property, rent and so on;
- transfer income, that is benefits and pensions transferred on the basis of entitlement, not on the basis of work or ownership, mainly by the government but occasionally by individuals .

Lower boundary of group (£ per week gross income)	**2003/4**
2nd decile	£ 124
3rd decile	£ 193
4th decile	£ 263
5th decile	£ 351
6th decile	£ 445
7th decile	£ 558
8th decile	£ 673
9th decile	£ 828
10th decile	£ 1092

Fig. 3.16. Lower boundaries of each gross income decile group

Measuring inequality: quantiles and quantileshares :

Figure 3.17 illustrates one method for summarizing data on the income received by households. It displays the gross income of different deciles of the distribution (gross income is defined as income from employment, self-employment, investments, pensions, etc. plus any cash benefits or tax credits). For example, figure 3.17 shows that in 2003/4 the poorest ten per cent of households had a gross income of less than 124 pounds per week, while the richest ten per cent of households had a gross income of over 1,092 pounds per week. The median gross income is 445 pounds per week.

An alternative technique for examining the distribution of incomes is to adopt the quantile shares approach. This is illustrated in figure 3.18, which is a modified version of a table produced as part of the annual report from the Office for National Statistics ’The effects of taxes and benefits on household:1 income’. The income of all units falling in a particular quantile group — for example, all those with income above the top decile, is summed and expressed as a proportion of the total income received by everyone.

	Percentage shares of equivalized income for ALL households			
	Original income	**Gross income**	**Disposable Income**	**Post-tax income**
Quintile group				
Bottom	3	7	8	7
2nd	7	11	12	12
3rd	15	16	17	16
4th	24	22	22	22
Top	51	44	42	44
All households	100	100	100	100
Decile group				
Bottom	1	3	3	2
Top	33	29	27	29

Fig. 3.17. Percentage shares of household income, 2003-4

Cumulative income shares and Lorenz curves :

Neither quantiles nor quantile shares lend themselves to an appealing way of presenting the distribution of income in a graphical form. This is usually achieved by making use of cumulative distributions. The income distribution is displayed by plotting cumulative income shares against the cumulative percentage of the population.

The cumulative distribution is obtained by counting in from one end only. Income distributions are traditionally cumulated from the lowest to the highest incomes. To see how this is done, consider the worksheet in figure 3.19. The bottom 5 percent receive 0.47 percent of the total original income, and the next 5 percent receive 0.51 percent. In summing these, we can say that the bottom 10 per cent receive 0.98 per cent of the total original income. We work our way up through the incomes in this fashion. It can be noted that the first two columns of this table are simply a more detailed version of the data presented in figure 3.18. For example, from figure 3.18 we can see that the top quintile group receives 51 per cent of original income; this figure is also obtained if you sum the first three numbers in the first column of figure 3.19.

The cumulative percentage of the population is then plotted against the cumulative share of total income. The resulting graphical display is known as a Lorenz curve. It was first introduced in 1905 and has been repeatedly used for visual communication of income and wealth inequality. The Lorenz curve for pre-tax income in 2003/4 in the UK is shown in figure 3.20.

	Percentage of total income received by the quantile		**Cumulative share of total income**	
Cumulative share of population	**Original income**	**Post-tax income**	**Original income**	**Post-tax income**
100	21.6	18.9	100	100
95	11.8	9.8	78.4	81.1
90	17.6	15	66.6	71.3
80	13.5	12.1	49	56.3
70	10.5	10	35.5	44.2
60	8.5	8.6	25	34.2
50	6.3	7.4	16.5	25.6
40	4.6	6.3	10.2	18.2
30	2.9	5.3	5.6	11.9
20	1.72	4.3	2.7	6.6
10	0.51	1.76	0.98	2.3
5	0.47	0.54	0.47	0.54

Fig. 3.18. Cumulative income shares : 2003-4

Lorenz curves have visual appeal because they portray how near total equality or total inequality a particular distribution falls. If everyone in society had the same income, then the share received by each decile group, for example, would be 10 per cent, and the Lorenz curve would be completely straight, described by the diagonal line.

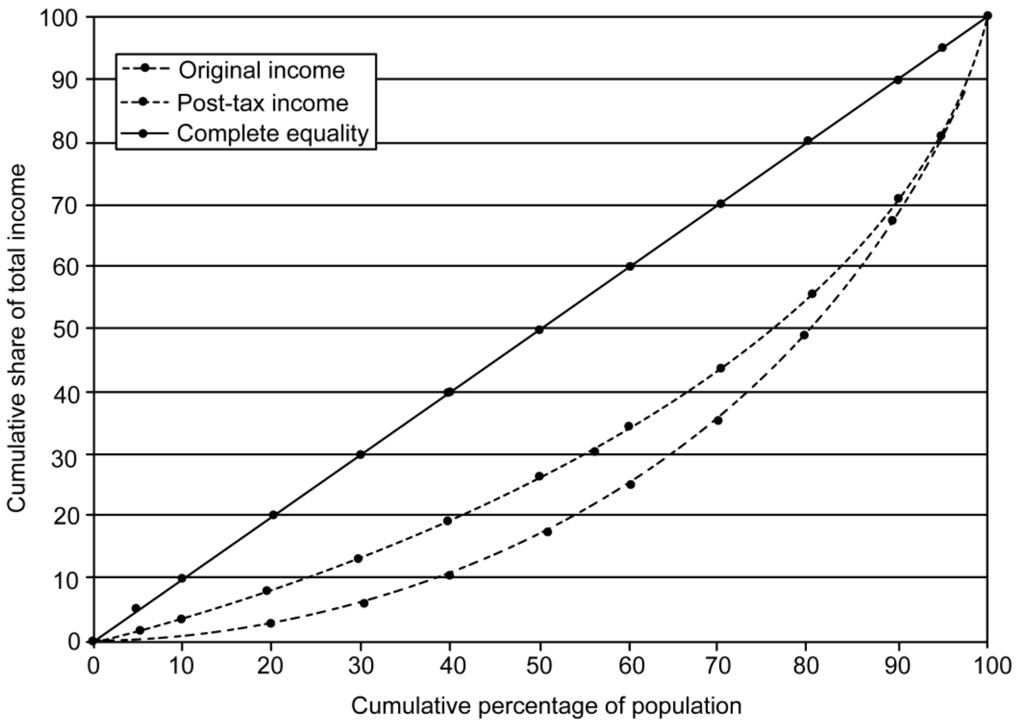


Fig. 3.19. Lorenz curves of income: 2003-4

Desirable properties in a summary measure of inequality

Scale independence

However, it is important that the measure be sensitive to the level of the distribution. Imagine a hypothetical society containing three individuals who earned 5,000, 10,000 and 15,000 pounds respectively. If they all had an increase in their incomes of l million pound, we would expect a measure of inequality to decline, since the differences between these individuals would have become trivial. The standard deviation and midspread would be unaffected. A popular approach is to log income data before calculating the numerical summaries of spread. If two distributions differ by a scaling factor, the logged distributions will differ only in level. However, if they differ by an arithmetic constant, they will have different spreads when logged. The existence of units with zero incomes leads to problems, since the log of zero cannot be defined mathematically. An easy technical solution to this problem is to add a very small number to each of the zeros. If a numerical summary of spread in a logged distribution met the other desirable features of a measure of inequality, we could stop here. Unfortunately, it does not.

The principle of transfers

It makes intuitive sense to require that a numerical summary of inequality should decline whenever money is given by a rich person to a poor person, regardless of how poor or how rich, and regardless of how much money is transferred (provided of course that the amount is not so big that the previously poor person becomes even richer than the previously rich person).One numerical summary — the income share of a selected quantile group — fails to meet this principle. By focusing on one part of the distribution only, perhaps the top 5 per cent, it would fail to record a change if a transfer occurred elsewhere in the distribution. Similar objections apply to another commonly used summary, the decile ratio, which simply expresses the ratio of the upper decile to the lower decile. Other inequality measures meet this principle, and so are to be preferred.

However, they unfortunately still fail to agree on an unambiguous ranking of different societies in terms of income inequality, because they are sensitive in different ways to transfers of varying amounts and at different points in the income scale. Cowell (1977) argues that the principle of transfers should be strengthened to specify that the measure of inequality should be sensitive only to the distance on the income scale over which the transfer is made, not to the amount transferred.

He also adds a third principle to the two considered here, that of decomposition: a decline in inequality in part of a distribution should lead to a decline in inequality overall. We shall return to these more stringent criteria below.

3.5. SMOOTHING

Time series such as that shown in the second column of figure 3.21 are displayed by plotting them against time, as shown in figure 3.22. When such trend lines are smoothed, the jagged edges are sawn off. A smoothed version of the total numbers of recorded crimes over the thirty years from the mid 1960s to the mid 1990s is displayed in figure 3.23.

Year	Total Recorded Crimes	Year	Total Recorded Crimes
1965	1,133,882	1980	2,688,235
1966	1,199,859	1981	2,963,764
1967	1,207.354	1982	3,262,422
1968	1,289,090	1983	3,247,030
1969	1,488,638	1984	3,499,107
1970	1,555,995	1985	3,611,883
1971	1,646,081	1986	3,847,410
1972	1,690,219	1987	3,892,201
1973	1,657,669	1988	3,715,767
1974	1,963,360	1989	3,870,748
1975	2,105,631	1990	4,543,611
1976	2,135,713	1991	5,276,173
1977	2,636,517	1992	5,591,717
1978	2,561,499	1993	5,526,255
1979	2,536,737	1994	5,252,980

Fig. 3.20. Total numbers of recorded crimes:1965-94

Most people, if asked to smooth the data by eye, would probably produce a curve similar to that in figure 3.23, which has been derived using a well-defined arithmetic procedure described later in the chapter. However, smoothing by an arithmetic procedure can sometimes reveal patterns not immediately obvious to the naked eye.

The aim of smoothing

Figure 3.22 was constructed by joining points together with straight lines. Only the points contain real information of course. The lines merely help the reader to see the points. The result has a somewhat jagged appearance. The sharp edges do not occur because very sudden changes really occur in numbers of recorded crimes.

They are an artefact of the method of constructing the plot, and it is justifiable to want to remove them. According to Tukey (1977, p. 205), the value of smoothing is 'the clearer view of the general, once it is unencumbered by detail'. The aim of smoothing is to remove any upward or downward movement in the series that is not part of a sustained trend.

Sharp variations in a time series can occur for many reasons. Part of the variation across time may be error. For example, it could be sampling error. The opinion-poll data used later in this chapter were collected in monthly sample surveys, each of which aimed to interview a cross-section of the general public, but each of which will have deviated from the parent population to some extent. Similarly, repeated measures may each contain a degree of measurement error. In such situations, smoothing aims to remove the error component and reveal the underlying true trend. But the variable of interest may of course genuinely swing around abruptly. For example, the monthly count of unemployed people rises very sharply when school-leavers come on to the register. In these cases, we may want to smooth to remove the effect of events which are unique, or which are simply not the main trend in which we are interested. It is good practice to plot the rough as well as the smooth values, to inspect exactly what has been discarded.

In engineering terms we want to recover the signal from a message by filtering out the noise. The process of smoothing time series also produces such a decomposition of the data. In other words, what we might understand in engineering as

Message = Signal +Noise

becomes

Data = Smooth+ Rough

This choice of words helps to emphasize that we impose no a priori structure on the form of the fit. The smoothing procedure may be determined in advance, but this is not the case for the shape and form of the final result: the data are allowed to speak for themselves. Put in another way, the same smoothing recipe applied to different time series will produce different resulting shapes for the smooth, which, as we will see in, is not the case when fitting straight lines.

As so often, this greater freedom brings with it increased responsibility. The choice of how much to smooth will depend on judgement and needs. If we smooth too much, the resulting rough will itself exhibit a trend. Of course, more work is required to obtain smoother results, and this is an important consideration when doing calculations by hand. The smoothing recipe described later in the chapter generally gives satisfactory results and involves only a limited amount of computational effort.

Most time series have a past, a present and a future. For example, the rising crime figures plotted in figure 3.22 and figure 3.23 are part of a story that begins well before the 1960s and continues to the present day. However, the goal of the smoothing recipes explained in this chapter is not the extrapolation of a given series into the future. The following section provides the next instalment in this story and discusses what happened after the very dramatic increases in total recorded crime in the early 1990s.

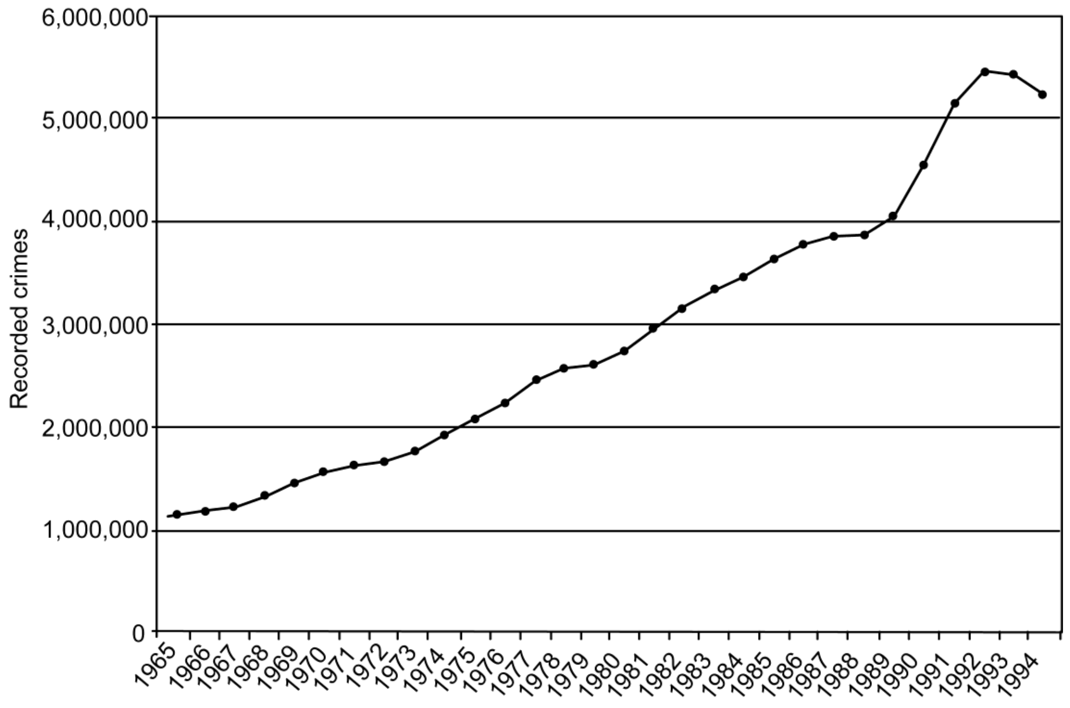


Fig. 3.21. Total number of recorded crimes: unsmoothed

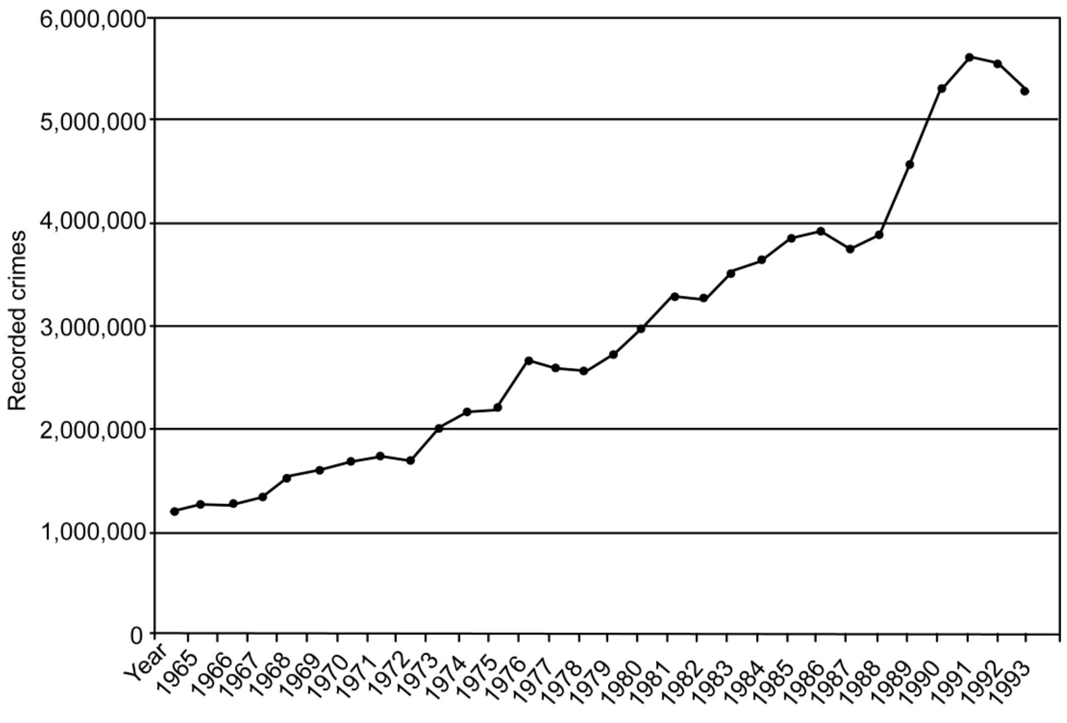


Fig. 3.22. Total recorded crimes 1965-94: smoothed

TWO MARKS QUESTION AND ANSWERS (PART- A)

1. ***What is a Univariate analysis***

   Among all the forms of analytical methods that data analysts practice, univariate analysis is considered one of the basic forms of analysis. It is typically the first step to understanding a dataset. The idea of univariate analysis is to first understand the variables individually. Then, you move into analyzing two or more variables simultaneously.

2. ***What is the basis for data analysis?***

   Two organizing concepts have become the basis of the language of data analysis: cases and variables. The cases are the basic units of analysis, the things about which information is collected. The word variable expresses the fact that this feature varies across different cases.

3. ***Distinguish Bar charts and pie charts***

One simple device is the bar chart, a visual display in which bars are drawn to represent each category of a variable such that the length of the bar is proportional to the number of cases in the category.

A pie chart can be used to display the above said information but in a different perspect as whether data from a categorical variable are displayed in a bar chart or a pie chart. In general, pie charts are to be preferred when there are only a few categories and when the sizes of the categories are very different.

4. ***Define level and spread in data exploration***

We will focus on the working hours to demonstrate how simple descriptive statistics can be used to provide numerical summaries of level and spread. We begin by examining data on working hours in Britain taken from the General Household Survey. These data are used to illustrate measures of level such as the mean and the median and measures of spread or variability such as the standard deviation and the midspread.

5. ***What is a mid-spread?***

The points which divide the distribution into quarters are called the quartiles (or sometimes 'hinges' or 'fourths'). The lower quartile is usually denoted QL and the upper quartile Q0. (The middle quartile is of course the median.) The distance between QL and Q0 is called the midspread (sometimes the 'interquartile range'), or the dQ for short.

6. ***Differentiate scaling and standardizing.***

Subtracting a constant from every data value altered the level of the distribution and dividing by a constant scaled the values by a factor. These two ideas may be combined to produce a very powerful tool which can render any variable into a form where it can be compared with any other. The result is called a standardized variable.

7. ***Define a Gaussian Distribution.***

Many distributions do have a characteristic shape a lump in the middle and tails straggling out at both ends. One such shape, investigated in the early nineteenth century by the German mathematician, Gauss, and therefore referred to as the Gaussian distribution, is commonly used. It is possible to define a symmetrical, bell-shaped curve which contains fixed proportions of the distribution at different distances from the centre.

8. ***Why there is a need to reduce in-equality?***

If we accept that extra income has a bigger impact on increasing the happiness of the poor than the rich, this means that if some money is transferred from the rich to the poor this will increase the happiness of the poor more than it diminishes the happiness of the rich. This in turn suggests that the overall happiness rating of a country will go up if income is distributed more equally.

9. ***What Is a Lorenz Curve?***

A Lorenz curve, developed by American economist Max Lorenz in 1905, is a graphical representation of income inequality or wealth inequality. The graph plots percentiles of the population on the horizontal axis according to income or wealth and plots cumulative income or wealth on the vertical axis.

10. ***Define smoothing time series***

Smoothing is usually done to help us better see patterns, trends for example, in time series. Generally smooth out the irregular roughness to see a clearer signal. For seasonal data, we might smooth out the seasonality so that we can identify the trend. Smoothing doesn't provide us with a model, but it can be a good first step in describing various components of the series. The term **filter** is sometimes used to describe a smoothing procedure. For instance, if the smoothed value for a particular time is calculated as a linear combination of observations for surrounding times, it might be said that we've applied a linear filter to the data

11. ***What is the Gini Coefficient?***

The Gini coefficient (Gini index or Gini ratio) is a statistical measure of economic inequality in a population. The coefficient measures the dispersion of income or distribution of wealth among the members of a population. What is the Gini Coefficient?

The Gini coefficient (Gini index or Gini ratio) is a statistical measure of economic inequality in a population. The coefficient measures the dispersion of income or distribution of wealth among the members of a population.

PART - B & C

1. Categorize variables and explain its distribution.
2. Define level and summarize in detail.
3. Summarize spread depicting an example.
4. Elaborate scaling and standardizing with an example.
5. Clarify in-equality with a proper example.
6. Explain the concept smoothing time series with an example.

****

UNIT IV
BIVARIATE ANALYSIS

SYLLABUS

Relationships between two Variables - Percentage Tables - Analyzing Contingency Tables - Handling Several Batches - Scatterplots and Resistant Lines – Transformations.

- ❖ Relationships between two Variables
- ❖ Percentage Tables
- ❖ Analyzing Contingency Tables
- ❖ Handling Several Batches
- ❖ Scatterplots and Resistant Lines
- ❖ Transformations

UNIT IV
BIVARIATE ANALYSIS

4.1. RELATIONSHIPS BETWEEN TWO VARIABLES

Relationships between two variables (bivariate relationships) are of interest because they can suggest hypotheses about the way in which the world works. In particular, they are interesting when one variable can be considered a cause and the other an effect. It is customary to call these variables by different names. We shall call the variable that is presumed to be the cause the explanatory variable (and denote it X) and the one that is presumed to be the effect the response variable (denoted Y); they are termed independent and dependent variables respectively.

Causal reasoning is often assisted by the construction of a schematic model of the hypothesized causes and effects: a causal path model. If we believe that the social class a child comes from is likely to have an effect on its school performance, we could model the relationship as in the sketch.

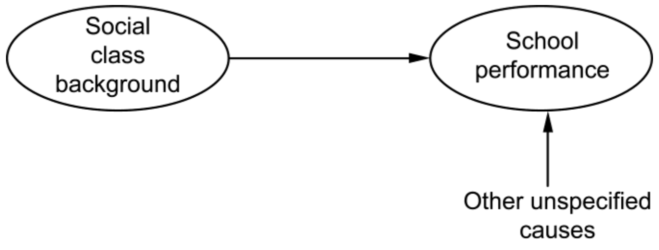


Fig. 4.1.

Such models are drawn up according to a set of conventions:

1. The variables are represented inside boxes or circles and labelled; in this example the variables are class background and performance at school.
2. Arrows run from the variables which we consider to be causes to those we consider to be effects; class background is assumed to have a causal effect on school performance.

3. Positive effects are drawn as unbroken lines and negative effects are drawn as dashed lines.
4. A number is placed on the arrow to denote how strong the effect of the explanatory variable is.
5. An extra arrow is included as an effect on the response variable, often unlabelled, to act as a reminder that not all the causes have been specified in the model.

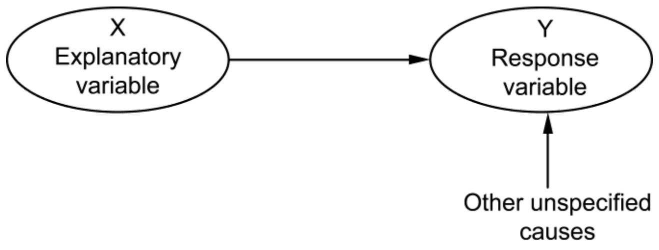


Fig. 4.2.

Proportions, Percentages and Probabilities

To express a variable in proportional terms, the number in each category is divided by the total number of cases N. Percentages are proportions multiplied by 100.

Figure 4.1 shows the proportions of young people aged 19 from different social class backgrounds, measured using family's socio-economic classification. The data were collected in 2005 and from the eleventh cohort of the Youth Cohort Study.

Parental Occupation (NS-SEC)	Number of Cases	Proportion	Percentage
Higher professional	1036	0.168	16.8
Lower professional	1708	0.276	27.6
Intermediate	1384	0.224	22.4
Lower supervisory	687	0.111	11.1
Routine	900	0.146	14.6
Other/unclassified	465	0.075	7.5
Total	6180	1.000	100.0

Fig. 4.3. Social class (NS-SEC) background of individuals ages 19 in 2005

Proportions and percentages are bounded numbers, in that they have a floor of zero, below which they cannot go, and a ceiling of 1.0 and 100 respectively.

Proportions can be used descriptively as in figure 4.1 to represent the relative size of different subgroups in a population. But they can also be thought of as probabilities. For example, we can say that the probability of an individual aged 19 in 2005 having a parent in a 'Higher professional' occupation is 0.168.

Contingency Tables

A contingency table does numerically what the three-dimensional bar chart does graphically. The Concise Oxford Dictionary defines contingent as 'true only under existing or specified conditions'. A contingency table shows the distribution of each variable conditional upon each category of the other. The categories of one of the variables form the rows, and the categories of the other variable form the columns. Each individual case is then tallied in the appropriate pigeonhole depending on its value on both variables. The pigeonholes are given the more scientific name cells, and the number of cases in each cell is called the cell frequency. Each row and column can have a total presented at the right-hand end and at the bottom respectively; these are called the marginals, and the univariate distributions can be obtained from the marginal distributions. Figure 4.4 shows a schematic contingency table with four rows and four columns.

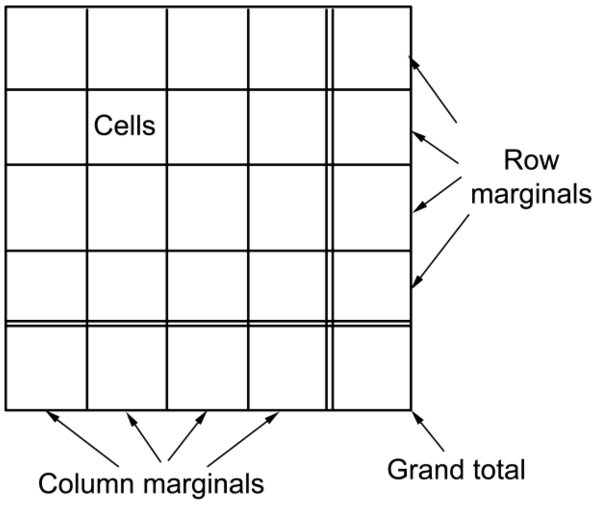


Fig. 4.4. Anatomy of a contingency table

Parental Occupation (NS-SEC)	Main activity at age 19							
	Full-time education	**Govt. supported training**	**Full-time job**	**Part-time Job**	**Out of work**	**Looking after home/ family**	**Other**	**Total**
Higher professional	663	41	249	41	21	0	21	1036
Lower professional	854	102	529	85	85	17	34	1706
Intermediate	498	97	554	69	83	42	28	1371
Lower supervisory	158	69	330	55	41	27	7	687
Routine	189	90	369	72	99	36	36	891
Other/unclassified	149	19	144	56	65	28	9	470
Total	2511	418	2175	378	394	150	135	6161

Fig. 4.5. Main activity by class background (frequencies)

The contingency table in figure 4.5 depicts the bivariate relationship between the two variables, but it is hard to grasp.

4.2. PERCENTAGE TABLES

The common way to make contingency tables readable is to cast them in percentage form. There are three different ways in which this can be done, as shown in the three panels of figure 4.6.

The first table, shown in panel (a) of figure 4.6, was constructed by dividing each cell frequency by the grand total. We now know that the 663 respondents with higher professional parents who were in full-time education at age 19 represented 10.8 per cent of the total population aged time education at age 19 represented 10.8 per cent of the total population aged 19 in 2005. But the table as a whole is scarcely more readable than the raw frequencies were, because there is nothing we can compare this 19 per cent with. For this reason, total percentage tables are not often constructed.

Panel (b) of figure 4.6 shows the percentage of young people within each category of social class background who are in each main activity grouping at age 19. The table was constructed by dividing each cell frequency by its appropriate row total. We can see that whereas nearly two-thirds of those with a parent in a higher professional occupation are still in full-time education at age 19, less than a quarter of those with parents in Lower supervisory or Routine occupations are still in full-time education by this age. Tables that are constructed by percentaging the rows are usually read down the columns (reading along the rows would probably only confirm two things we already know: the broad profile of the marginal distribution and the fact that the percentages sum to 100). This is sometimes called an 'outflow' table. The row percentages show the different outcomes for individuals with a particular social class background.

It is also possible to tell the story in a rather different way, and look at where people who ended up doing the same main activity at age 19 came from: the 'inflow table'. This is shown in panel (c) of figure 4.6.

(*a*) Total Percentages

Parental Occupation (NS-SEC)	**Main activity at age 19**							
	Full-time education	**Govt. supported training**	**Full-time job**	**Part-time Job**	**Out of work**	**Looking after home/family**	**Other**	**Total**
Higher professional	10.8	0.7	4.0	0.7	0.3	0.0	0.3	16.8
Lower professional	13.9	1.7	8.6	1.4	1.4	0.3	0.6	27.7
Intermediate	8.1	1.6	9.0	1.1	1.3	0.7	0.5	22.3
Lower supervisory	2.6	1.1	5.4	0.9	0.7	0.4	0.1	11.2
Routine	3.1	1.5	6.0	1.2	1.6	0.6	0.6	14.5
Other/unclassified	2.4	0.3	2.3	0.9	1.1	0.5	0.1	7.6
Total	40.8	6.8	35.3	6.1	6.4	2.4	2.2	100.0

(*b*)Row Percentages

Parental Occupation (NS-SEC)	Main activity at age 19							
	Full-time education	Govt. supported training	Full-time job	Part-time Job	Out of work	Looking after home/family	Other	Total
Higher professional	64	4	24	4	2	-	2	100.0
Lower professional	50	6	31	5	5	1	2	100.0
Intermediate	36	7	40	5	6	3	2	100.0
Lower supervisory	23	10	48	8	6	4	1	100.0
Routine	21	10	41	8	11	4	4	100.0
Other/unclassified	32	4	31	12	14	6	2	100.0

(*c*) Column Percentages

Parental Occupation (NS-SEC)	Main activity at age 19							
	Full-time education	Govt. supported training	Full-time job	Part-time Job	Out of work	Looking after home/family	Other	Total
Higher professional	26.4	9.8	11.4	10.8	5.3	0.0	15.6	16.8
Lower professional	34.0	24.4	24.3	22.5	21.6	11.3	25.2	27.7
Intermediate	19.8	23.2	25.5	18.3	21.1	28.0	20.7	22.3
Lower supervisory	6.3	16.5	15.2	14.6	10.4	18.0	5.2	11.2
Routine	7.5	21.5	17.0	19.0	25.1	24.0	26.7	14.5
Other/unclassified	5.9	4.5	6.6	14.8	16.5	18.7	6.7	7.6
Total	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0

Fig. 4.6. Main activity at age 19 by class background

Good Table Manners

A well-designed table is easy to read, but takes effort, time and perhaps many drafts to perfect. Clear display of data not only aids the final consumer of the research but also helps the data analyst. It pays to take care over the presentation of your own working and calculations, however preliminary. This can help reveal patterns in the data, and can save time at a later stage. Here are some guidelines on how to construct a lucid table of numerical data.

(*i*) Reproducibility versus Clarity

We are often trying to do two jobs at once when we present data: to tell a story while also allowing readers to check the conclusions by inspecting the data for themselves. These two jobs tend to work against one another, although the techniques of exploratory data analysis allow the researcher to pursue both at once to a much greater extent than more traditional techniques. For clarity we prefer visual displays, and we leave out extraneous detail to focus attention on the story line. To allow others to inspect and possibly reinterpret the results we want to leave as much of the original data as possible in numerical form. Think hard about which job any particular table is aiming to achieve. Dilemmas can often be solved by simplifying a table in the text and placing fuller details in an appendix, although in general it is desirable to place a table as near as possible to the text which discusses it. There are some elementary details which must always appear.

(*ii*) Labelling

The title of a table should be the first thing the reader looks at. A clear title should summarize the contents. It should be as short as possible, while at the same time making clear when the data were collected, the geographical unit covered, and the unit of analysis.

(*iii*) Sources

The reader needs to be told the source of the data. It is not good enough to say that it was from Social Trends. The volume and year, and either the table or page, and sometimes even the column in a complex table must be included. When the data are first collected from a published source, all these things should be recorded, or a return trip to the library will be needed.

(*iv*) Sample Data

If data are based on a sample drawn from a wider population, it always needs special referencing. The reader must be given enough information to assess the adequacy of the sample. The following details should be available somewhere: the method of sampling, the achieved sample size, the response rate or refusal rate, the geographical area which the sample overs and the frame from which it was drawn.

(*v*) Missing data

Providing details of the overall response rate in a survey does not usually tell the whole story about missing information. Many particular items in a survey attract refusals or responses that cannot be coded, and the extent of such item non response should be reported.

(*vi*) Definitions

There can be no hard and fast rule about how much definitional information to include in your tables. They could become unreadable if too much were included. If complex terms are explained elsewhere in the text, include a precise section or page reference.

(*vii*) Opinion Data

When presenting opinion data, always give the exact wording of the question put to respondents, including the response categories if these were read out. There can be big differences in replies to open questions such as: 'Who do you think is the most powerful person in Britain today?'

(*viii*) Ensuring frequencies can be reconstructed

It should always be possible to convert a percentage table back into the raw cell frequencies. To retain the clarity of a percentage table, present the minimum number of base Ns needed for the entire frequency table to be reconstructed.

(*ix*) Layout

The effective use of space and grid lines can make the difference between a table that is easy to read and one which is not. In general, white space is preferable, but grid lines can help indicate how far a heading or subheading extends in a complex table.

Tables of monthly data can be broken up by spaces between every December and January, for example. Labels must not be allowed to get in the way of the data. Set variable headings off from the table, and further set off the category headings.

Make a decision about which variable to put in the rows and which in the columns by combining the following considerations:

1. Closer figures are easier to compare
2. Comparisons are more easily made down a column
3. A variable with more than three categories is best put in the rows so that there is plenty of room for category labels.

4.3. ANALYSING CONTINGENCY TABLES

In the previous topic, we introduced percentage tables as a way of making contingency data more readable. The properties of percentages and proportions will be scrutinized more closely, and other ways of analysing contingency data considered in the quest for a summary measure of the effect of one variable upon another. First, however we must come back to the question of how to read a contingency table when one variable can be considered a likely cause of the other i.e. when one variable is interpreted as the explanatory variable and the other as the response or outcome variable.

(*i*) Which way should proportions run?

When we have a hypothesis about the possible causal relationship between variables, this can be conveyed by the choice of which proportions one uses in the analysis. Over the last two decades researchers have consistently found age to be associated with whether individuals feel safe walking alone after dark: older people, and particularly older women, are more likely to feel unsafe than younger individuals. In this example, the explanatory variable must be old age and the response or outcome variable is feeling unsafe, we would not suggest that feeling unsafe causes people to be old. This means that in a cross-tabulation of age by feeling unsafe, it is more natural to examine the proportion of each age group who feel unsafe, rather than the proportion of each category of a 'feeling safe walking alone after dark' variable who are old. This can be formalized into a rule when dealing with contingency data:

Construct the proportions so that they sum to one within the categories of the explanatory variable.

The rule is illustrated by the following diagram.

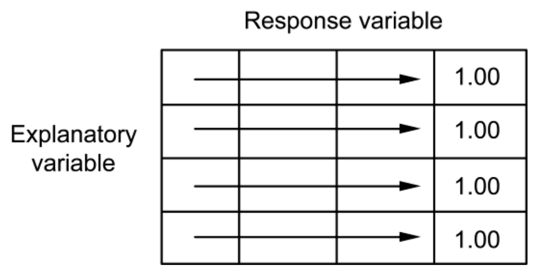


Fig. 4.7.

Note that it cannot be formulated as 'always calculate proportions along the rows'. This would only work if the explanatory variable was always put in the rows, and no such convention has been established.

(*ii*) The base for comparison

One category is picked to act as the base for comparison with all other categories. By making comparisons with this base, quantitative estimates of the likely causal effect of one variable on another can be made, and positive and negative relationships between nominal level variables can be distinguished.

Example:

Age group	**Very safe / fairly safe / a bit unsafe**		**Very unsafe**		**Total**	
	p	**N**	**p**	**N**	**p**	**N**
16-39	0.93	13,589	0.07	1083	1	14,672
40-59	0.93	13,861	0.07	1099	1	14,960
60+	0.84	12,722	0.16	2432	1	15,154
Total		40,172		4614		44,786

Fig. 4.8. How safe do you feel walking alone after dark? 2004-05

Which categories should be selected as bases for comparison among age groups feeling unsafe walking alone after dark? An important rule of thumb is to choose a category with a relatively large number of individuals within it. In this case, since the age-groups are all of similar size, any one of them could be used as the base category for the age-group variable.

If we select the youngest age group as the base and then pick feeling very unsafe as the base for comparison in the fear of walking alone after dark variable, we will almost certainly avoid too many negative relationships. In summary, each age group can be compared with those aged 16-39 in their feeling very unsafe when walking alone after dark.

In order to represent one three-category variable, like age group, in a causal path model, we have to present it as two dichotomous variables. Instead of coding the age of respondents as 1, 2 or 3 to denote 60 and over, 40-59, or 16-39, for example, the information is effectively presented as two dichotomous variables - whether someone is aged 60 and over or not, and aged 40-59 or not.

Someone who was in neither of these age groups would, by elimination, be in the youngest age group.

Age group as a three-category variable		**Age group as two dichotomies**	
		Aged 60+ or not	**Aged 40-59 or not**
60+	1	1	0
40-59	2	0	1
16-39	3	0	0

Choosing one category as a base effectively turns any polytomous variable into a series of dichotomous variables known as dummy variables. Figure 4.9 shows how the effect of a three - category explanatory variable on a dichotomous response variable can be portrayed in a causal path model. Age group is represented by two dummy variables. The effect of the first is denoted b1 and the effect of the second b2.

A line is drawn under which the base category of the explanatory variable is noted; the fact that some young people are afraid of walking alone after dark (path a) reminds us that there are some factors influencing feeling very unsafe that this particular model does not set out to explain.

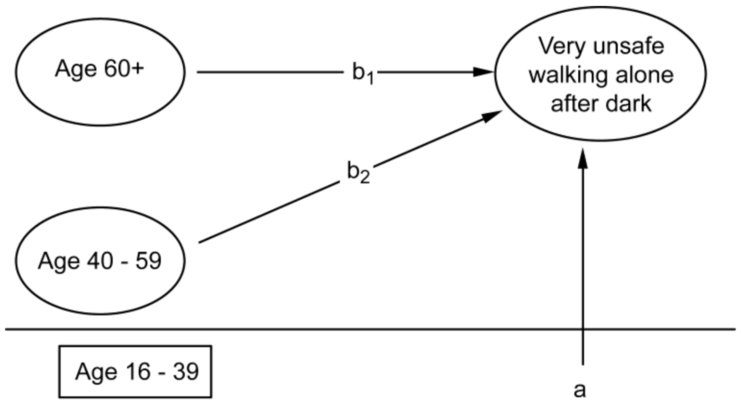


Fig. 4.9. Casual path model of age group and feeling unsafe walking alone after dark

(*iii*) Summarizing effects by Subtracting Proportions

In figure 4.9, the effect of being in the oldest age group on feeling unsafe when walking alone after dark is denoted b1 and the effect of being in the middle age group is denoted b2. How are these to be quantified? There is no answer to this question that commands universal acceptance. In this section we will consider d, the difference in proportions (Davis, 1976). This measure of effect has two virtues: it is simple and intuitively appealing.

The effect d is calculated by subtracting this proportion in the base category of the explanatory variable from this proportion in the non-base category of the explanatory variable.

In this particular example, path b represents the effect of being in the oldest age group as opposed to being in the youngest age group on the chances of feeling very unsafe walking alone after dark. It is found by subtracting the proportion of the youngest age group feeling very unsafe from the proportion of the oldest age group class giving the same response. If we look back at figure 7 .1, in this case, d = 0.16 - 0.07, or +0.09. The result is positive, as we expected: older people are more likely to be afraid of walking alone after dark than are the youngest age group.

lf we had selected different base categories, we could have ended up with negative values of d. For example, if we were trying to explain feeling safe when walking alone after dark, the d for the oldest age group would have been 0.84 - 0.93, or -0.09. The magnitude of effect would not have altered but the sign would have been reversed.

Path b2 represents the effect of being in the middle age group on feeling very unsafe walking alone after dark. We might expect this to be lower than the effect of being in the oldest age group. It is. In fact, d = 0.07 - 0.07, or O; the younger two age groups are extremely similar in their fear of walking alone after dark. While the paths b1 and b2 are the focus of our attention, it is also important to remember the other factors which lead to people being afraid to walk alone after dark: age group is not a complete determinant of who is fearful, since some in the youngest age group report feeling very unsafe about walking alone after dark. Path a reminds us of this.

The value of path a is given by the proportion of cases in the base category of the explanatory variable who fall in the non-base category of the response variable.

The quantified model is shown in figure 4.10. The model allows us to decompose the proportion of older people who are fearful of walking alone after dark (0.16) into a fitted component (0.07) and an effect (+0.09).

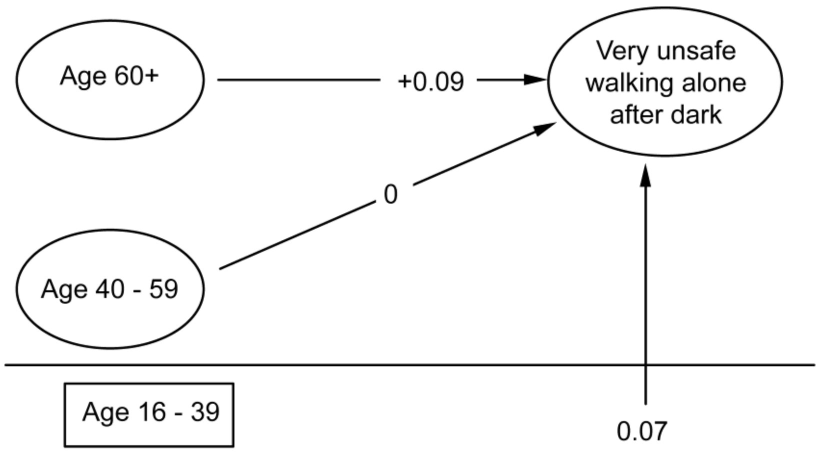


Fig. 4.10. Quantifying model in figure 4.5

A simple relationship between an explanatory variable X and a response variable Y as Y = a + bX. Lf the idea is familiar to you, you may like to note here that proportions can also be expressed in this way. The overall proportion Y who feel very unsafe when walking alone after dark is 4,614/44, 786, or 0.103 (figure 4.9).

Calculating the Chi-square Statistic

In order to understand how the chi-square statistic is calculated and how its value should be interpreted, it is helpful to refer to a concrete example, but one in which the numbers are very straightforward. Therefore let's start with an imaginary piece of research in which 100 men and 100 women are asked about their fear of walking alone after dark. Until we conduct the survey we have no information other than the number of men and women in our sample and therefore we have figure 4.11.

	Very safe / fairly safe / a bit unsafe		**Very unsafe**		**Total**	
	p	**N**	**p**	**N**	**p**	**N**
Male	?	?	?	?	1	100
Female	?	?	?	?	1	100
Total	?	?	?	?	1	200

Fig. 4.11. Feeling safe walking alone after dark by gender (hypothetical survey of 200 individuals)

Once we carry out the survey let us imagine that we find that in total 20 individuals i.e. 0.1 of the sample state that they feel very unsafe when walking alone after dark. We therefore now have some more information that we can add to our table and this is entered as the column marginals in figure 4.12 below.

	Very safe / fairly safe / a bit unsafe		**Very unsafe**		**Total**	
	p	**N**	**p**	**N**	**p**	**N**
Male	?	?	?	?	1	100
Female	?	?	?	?	1	100
Total	0.9	180	0.1	20		200

Fig. 4.12. Feeling safe walking alone after dark by gender (hypothetical survey of 200 individuals)

If, in the population as a whole, the proportion of men who feel very unsafe walking alone after dark is the same as the proportion of women who feel very unsafe walking alone after dark, we would expect this to be reflected in our sample survey. The expected proportions and frequencies would then be as shown in figure 4.13.

	Very safe / fairly safe / a bit unsafe		**Very unsafe**		**Total**	
	p	**N**	**p**	**N**	**p**	**N**
Male	0.9	90	0.1	10	1	100
Female	0.9	90	0.1	10	1	100
Total	0.9	180	0.1	20		200

Fig. 4.13. Feeling safe walking alone after dark by gender-expected values if men and women in the population are equally likely to feel unsafe (hypothetical survey of 200 individuals)

Once we have carried out our survey and cross-tabulated fear of walking alone after dark by gender we will have 'observed' values that we are able to put in our table as shown in figure 4.14.

	Very safe / fairly safe / a bit unsafe		**Very unsafe**		**Total**	
	p	**N**	**p**	**N**	**p**	**N**
Male	0.95	95	0.05	5	1	100
Female	0.85	85	0.15	15	1	100
Total	0.9	180	0.1	20		200

Fig. 4.14. Feeling safe walking alone after dark by gender – observed values following the survey (hypothetical survey of 200 individuals)

In order to be able to judge whether there is a relationship between gender and fear of walking alone after dark we need to compare the values we actually observed, following our survey, with the values that we would expect if there were no differences between men and women. The chi-square statistic provides a formalized way of making this comparison. The equation for chi-square is given below. In practical terms we need to find the difference between the observed and expected frequencies for each cell of the table. We then square this value before dividing it by the expected frequency for that cell. Finally we sum these values over all the cells of the table.

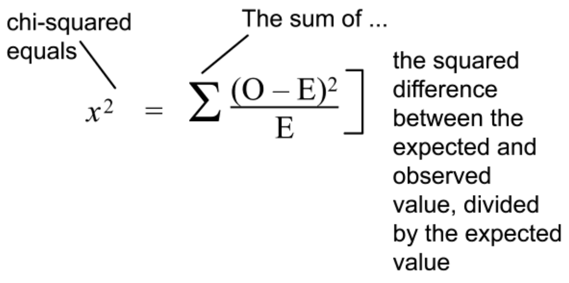

chi-squared equals
The sum of ...
$x^2 = \sum \left[\frac{(O - E)^2}{E} \right]$
the squared difference between the expected and observed value, divided by the expected value

Fig. 4.15.

For the previous example, the computational details are provided in figure 4.16. The total chi-square value is calculated as 5.56. Although this provides a measure of the difference between all the observed and expected values in the table.

Observed	Expected	O-E	$(O - E)^2$	$(O - E)^2 / E$
95	90	5	25	0.28
5	10	– 5	25	2.5
85	90	– 5	25	0.28
15	10	5	25	2.5
200	200			Total sum: 5.56

Fig. 4.16. Computation of chi-square from figure 4.13 and 4.14

Type I and Type 2 Errors

Clearly, using the method described above for deciding whether a result is statistically significant or not can never give us a definitive answer as to whether the relationship we observe in our sample reflects what we would observe if we could collect data on the population as a whole. However, the level of probability associated with a particular chi-square gives us a measure of how likely we are to be mistaken.

This probability is sometimes thought of as the likelihood that we will make what is called a 'Type 1' error.

In some surveys, particularly where the sample size is small, we may obtain what looks like an interesting difference between two groups, but find that the probability associated with the chi- square is above the conventional cut-off of 0.05. It is in this situation that we run the risk of making a 'Type 2' error.

Degrees of Freedom

A table with two rows and two columns is said have one degree of freedom because only one cell is known (e.g. once we know how many women are afraid to walk alone after dark) the values in the other cells can be calculated based on the row and column marginals. Similarly, a table with two columns and three rows is said to have two degrees of freedom. In formal terms the number of degrees of freedom for a table with r rows and c columns is given by the equation below:

$$\text{Degrees of freedom(Df)} = (r - 1) \times (c - 1)$$

4.4. HANDLING SEVERAL BATCHES

In this, a new graphical method the boxplot, will be presented which facilitates comparisons between distributions, and the idea of an unusual data value will be given more systematic treatment than previously.

Boxplots

Most people agree that it is important to display data well when communicating it to others. Pictures are better at conveying the story line than numbers. However, visual display also has a role that is less well appreciated in helping researchers themselves understand their data and in forcing them to notice features that they did not suspect. We have already looked at one pictorial representation of data, the histogram. Its advantage was that it preserved a great deal of the numerical information. For some purposes, however, it preserves too much.

The boxplot is a device for conveying the information in the five number summaries economically and effectively. The important aspects of the distribution are represented schematically as shown in figure 4.17.

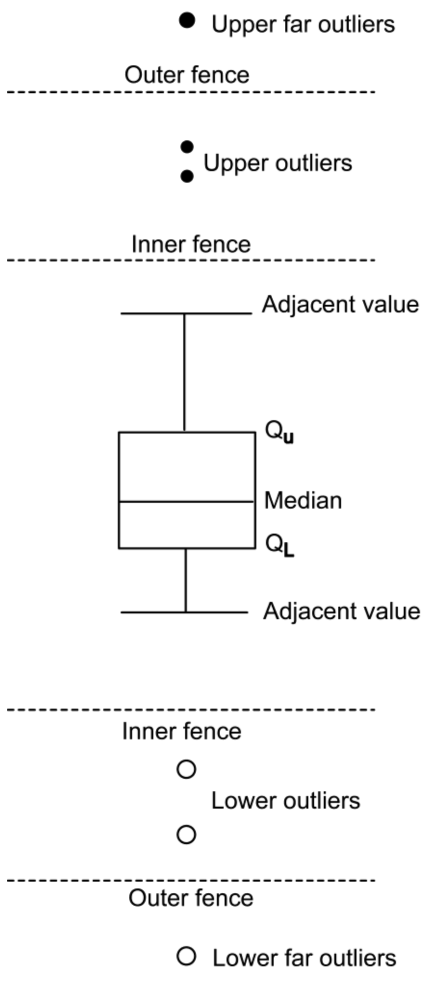


Fig. 4.17. Anatomy of a boxplot

The middle 50 per cent of the distribution is represented by a box. The median is shown as a line dividing that box. Whiskers are drawn connecting the box to the end of the main body of the data. They are not drawn right up to the inner fences because there may not be any data points that far out. They extend to the adjacent values, the data points which come nearest to the inner fence while still being inside or on them. The outliers are drawn in separately. They can be coded with symbols (such as those in figure 4.17) to denote whether they are ordinary or far outliers, and are often identified by name. Outliers are points that are unusually distant from the rest of the data. They are discussed in more detail in the next section. To identify the outliers in a particular dataset, a value 1.5 times the dQ, or a step, is calculated; as usual, fractions other than one-half are ignored. Then the points beyond which the outliers fall (the inner fences) and the points beyond which the far outliers fall (the outer fences) are identified; inner fences lie one step beyond the quartiles and outer fences lie two steps beyond the quartiles.

The boxplot of unemployment in the East Midlands is shown in figure 4.18. It contains the same data as figure 8.4:

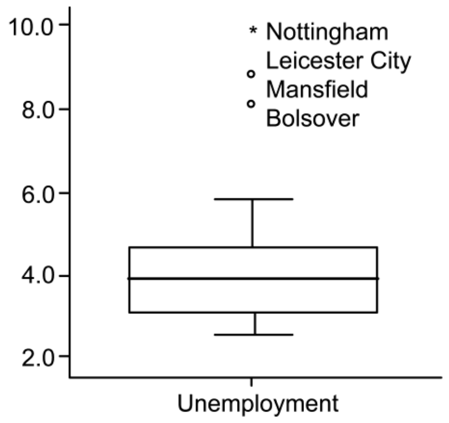


Fig. 4.18. Unemployment in the East Midlands in 2005 boxplot

Outliers

Some datasets contain points which are a lot higher or lower than the main body of the data. These are called outliers. They are always point that require the data analyst's special attention. They are important and arise for one of four reasons:

1. They may just result from a fluke of the particular sample that was drawn. The probability of this kind of fluke can be assessed by traditional statistical tests, if sensible assumptions can be made about the shape of the distribution.
2. They may arise through measurement or transcription errors, which can occur in official statistics as well as anywhere else. We always want to be alerted to such errors, so that they can be corrected, or so that the points can be omitted from the analysis.
3. They may occur because the whole distribution is strongly skewed. In this case they point to the need to transform the data. As we will see in, transformations such as logging or squaring the values may remove these outliers.

4. Most interesting of all, they may suggest that these particular data points do not really belong substantively to the same data batch.

Moving to the individual as a unit of analysis and using a statistical test

It is possible to use boxplots to carry out exploratory analysis of how unemployment rates vary between and within region.

Example: Comparing the mathematics scores of boys and girls

Let us now turn to a rather different topic, but use the same approach to examine whether there are any differences between the mathematics scores of boys and girls. Figure 4.19 displays two boxplots, one for girls' mathematics score and one for boys' mathematics score at age eleven.

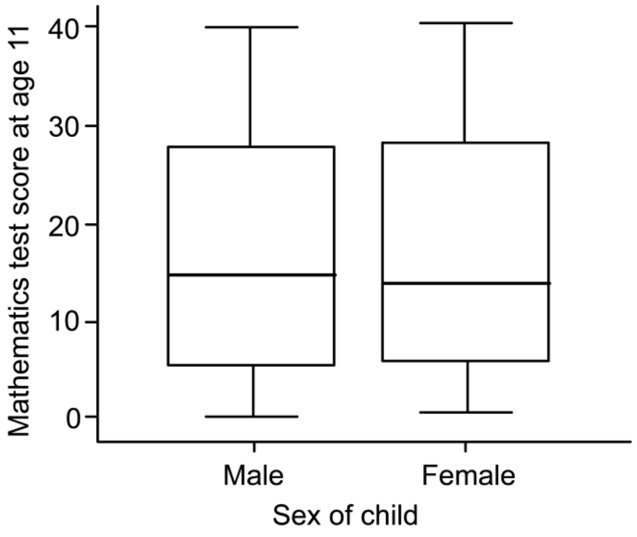


Fig. 4.19. Boxplots comparing girls and boys mathematics scores at age 11

In the example above, mathematics score is an interval level variable and we therefore need a different statistical test to check whether the results are significant. In this specific example we have two groups (boys and girls) defined by a dichotomous variable and we are comparing them on an interval level variable (mathematics score). In these circumstances the statistical test that we need to use is called the T-test.

The T-Test

T-Test provide a measure of the difference between the means of two groups.

T-test Formula

The formula for a two-sample *t*-test where the samples are independent as in the example of boys and girls mathematics test scores) is

$$t = \frac{\overline{X_1} - \overline{X_2}}{S_{X_1 X_2}\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

where X_1 and X_2 are the means of the two samples and $S_{X_1 X_2}$ is known as the pooled standard deviation and is calculated as follows:

$$S_{X_1 X_2} = \sqrt{\frac{(n_1 - 1)\, S_{X_1}^2 + (n_2 - 1)\, S_{X_2}^2}{n_1 + n_2 - 2}}$$

here S_{X_1} is the standard deviation of one sample and S_{X_2} is the standard deviation of the other sample. In these formulae n_1 is the sample size of the first sample and n_2 is the sample size of the second sample. In simple terms therefore the size of the *t*-statistic depends on the size of the difference between the two means adjusted for the amount of spread and the sample sizes of the two samples.

4.5. SCATTERPLOTS AND RESISTANT LINES

Scatterplots

To depict the information about the value of two interval level variables at once, each case is plotted on a graph known as a scatterplot, such as figure. Visual inspection of well-drawn scatterplots of paired data can be one of the most effective ways of spotting important features of a relationship.

A scatterplot has two axes – a vertical axis, conventionally labeled Y and a horizontal axis, labeled X. The variable that is thought of as a cause (the explanatory variable) is placed on the X-axis and the variable that is thought of as an effect (the response variable) is placed on the Y-axis. Each case is entered on the plot at the point representing its X and Y values.

Scatterplots depict bivariate relationships. To show a third variable would require a three-dimensional space, and to show four would be impossible.

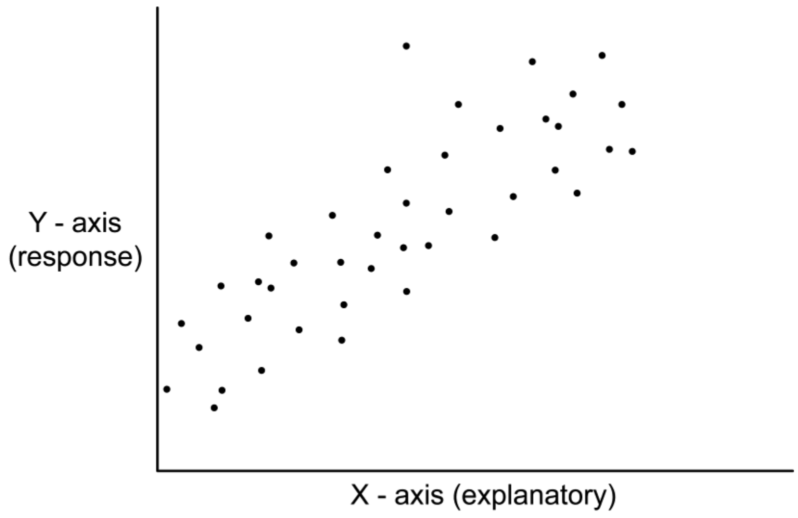


Fig. 4.20. A Scatterplot showing a moderately strong relationship

Scatterplots are inspected to see if there is any sort of pattern visible, to see if the value of Y could be predicted from the value of X, or if the relationship is patternless. If there does appear to be something interesting going on, there are several further useful questions that we can ask:

1. Is the relationship monotonic? In other words, does Y rise or fall consistently as X rises? The relationship in figure 9.1 is monotonic. A U-shaped relationship would not be.
2. Are the variables positively or negatively related? Do the points slope from bottom left to top right (positive) or from top left to bottom right (negative)?
3. Can the relationship be summarized as a straight line or will it need a curve?
4. How much effect does X have on Y? In other words, how much does Y increase (or decrease) for every unit increase of X?
5. How highly do the variables correlate? In other words, how tightly do the points cluster around a fitted line or curve?
6. Are there any gaps in the plot? Do we have examples smoothly ranged across the whole scale of X and Y, or are there gaps and discontinuities?

Caution may need to be exercised when one is making statements about the relationship in the gap.

7. Are there any obvious outliers? One of the major goals of plotting is to draw attention to any unusual data points.

Lone Parents

The data in figure 4.21 relate to the percentage of households that are headed by a lone parent and contain dependent children, and the percentage of households that have no car or van.

Government Office Region (2001)	% Lone parent households	% Households with no car or van
North East	7.35	35.94
North West	7.67	30.21
Yorkshire/Humber	6.58	30.31
East Midlands	6.08	24.25
West Midlands	6.73	26.77
Eastern	5.29	19.80
London	7.60	37.49
South East	5.22	19.43
South West	5.42	20.21
Wales	7.28	25.95

Fig. 4.21. Lone parent households and households with no car or van, % by region

Linear Relationships

$$Y = a + bX$$

always describe lines. In this equation, Y and X are the variables, and a and b are coefficients that quantify any particular line; figure shows this diagrammatically.

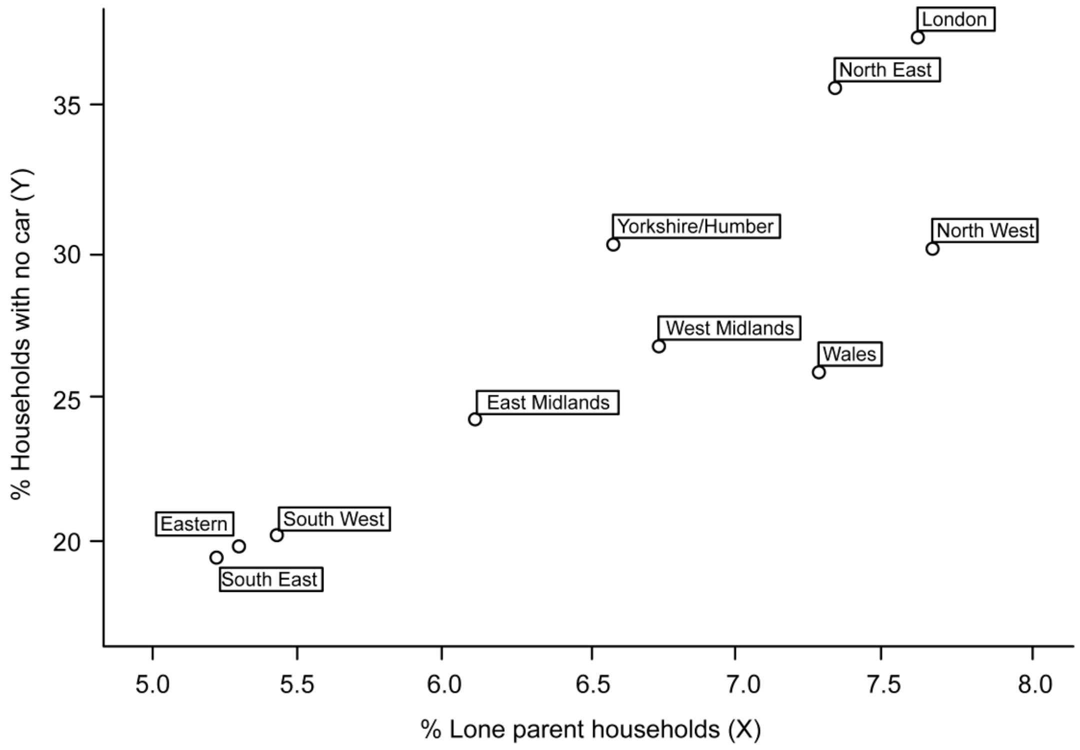


Fig. 4.22. Lone parent households with no car or van scatterplot

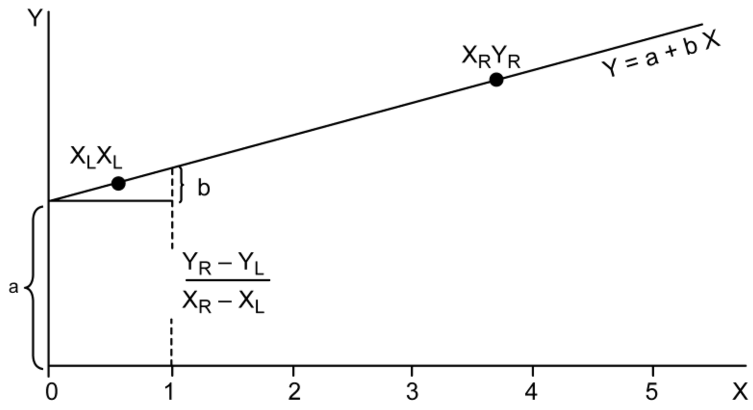


Fig. 4.23. Anatomy of a straight line

The degree of slope or gradient of the line is given by the coefficient b; the steeper the slope, the bigger the value of b. As we can see from figure 9.4, the coefficient b gives a measure of how much Y increases for a unit increase in the value of X. The slope is usually the item of scientific interest, showing how much change in Y is associated with a given change in X. The intercept a is the value of Y when X is zero, or where the line starts. Frequently, the intercept makes little substantive sense - for example, a mortality rate of zero is an impossibility. This value is also sometimes described as the constant.

The slope of a line can be derived from any two points on it. If we choose two points on the line, one on the left-hand side with a low X value (called XL, YL), and one on the right with a high X value (called XR, YR), then the slope is

$$\frac{Y_R - Y_L}{X_R - X_L}$$

If the line slopes from top left to bottom right, YR - YL will be negative and thus the slope will be negative.

4.6. TRANSFORMATIONS

We will consider a family of transformations of the scale of measurement which help make the variables easier to handle in data analysis.

Log Transformation

One method for transforming data or re-expressing the scale of measurement is to rake the logarithm of each data point. This keeps all the data points in the same order but stretches or shrinks the scale by varying amounts at different points.

	GNI per capita in 2000 ($US)	**Log GNI per capita**
Australia	20060	4.3
Benin	340	2.53
Burundi	120	2.08
China	930	2.97
Czech Republic	5690	3.76
Estonia	4070	3.61
Germany	25510	4.41
Haiti	490	2.69
Israel	17090	4.23
Korea, Rep.	9790	3.99

	GNI per capita in 2000 ($US)	Log GNI per capita
Lithuania	3180	3.5
Malta	9590	3.98
Mozambique	210	2.32
Nigeria	280	2.45
Philippines	1040	3.02
Sudan	310	2.49
United States	34400	4.54
Togo	270	2.43
Zimbabwe	460	2.66
Tanzania	260	2.41

Fig. 4.24. Logging the numbers in figure

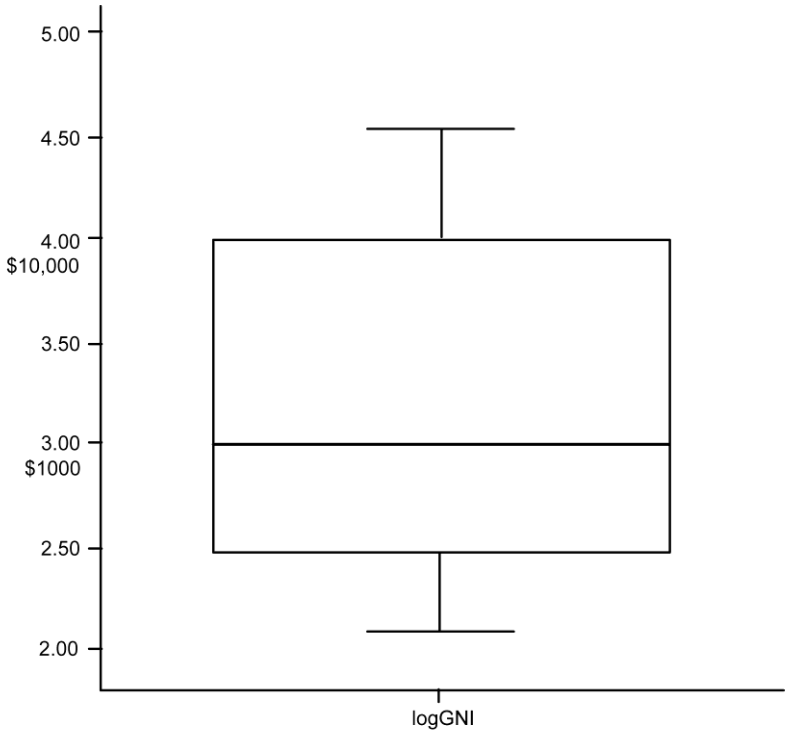


Fig. 4.25. Logging GNI per capita in 2000 in 20 selected countries

You will notice that all the GNI per capita figures between 100 and 1000 have been transformed, by taking logs, to lie between 2 and 3 (e.g. Benin with a GNI per capita of 340 has a log GNI per capita of 2.53). While all the data lying between 10,000 and 100,000 have been transformed to lie between 4 and 5 (e.g. Australia with a GNI per capita of 20,060 has a log GNI per capita of 4.3). The higher values have therefore been pulled down towards the centre of the batch, bringing the United States and Germany into the main body of the data, and the bottom of the scale has been stretched out correspondingly. The shape is now more symmetrical.

The Ladder of Powers

There are an infinite number of possible powers to which data can be raised. The commonest values are shown in figure 4.26, placed, as Tukey (1977) suggests, on a 'ladder' in terms of their effect on distributions. There are many other points besides the ones on this ladder of powers, both in between the values shown and above and beneath them, but we shall rarely have any need to go beyond those shown.

Power	Expression	Name
•		
•		
3	X^3	cube
2	X^2	square
1	X^1	raw data
0.5	$\sqrt{X}$	square root
0	X^0	raw data
–0.5	$1/\sqrt{X}$	reciprocal root
–1	$1/X$	reciprocal
–2	$1/X^2$	reciprocal square
•		
•		

Fig. 4.26. The ladder of powers

Going up the ladder of powers corrects downward straggle, whereas going down corrects upward straggle.

The goals of transformation

1. Data batches can be made more symmetrical.
2. The shape of data batches can be made more Gaussian.
3. Outliers that arise simply from the skewness of the distribution can be removed, and previously hidden outliers may be forced into view'.
4. Multiple batches can be made to have more similar spreads.
5. Linear, additive models may be fitted to the data.

Promoting Equality of Spread

It is important for the spread to be independent of level in data analysis, whether fitting lines, smoothing, or dealing with multiple boxplots. No simple statement can be made summarizing typical differences in GNI between the country groups in figure 10.3, for example, because they differ systematically in spread as well as in level.

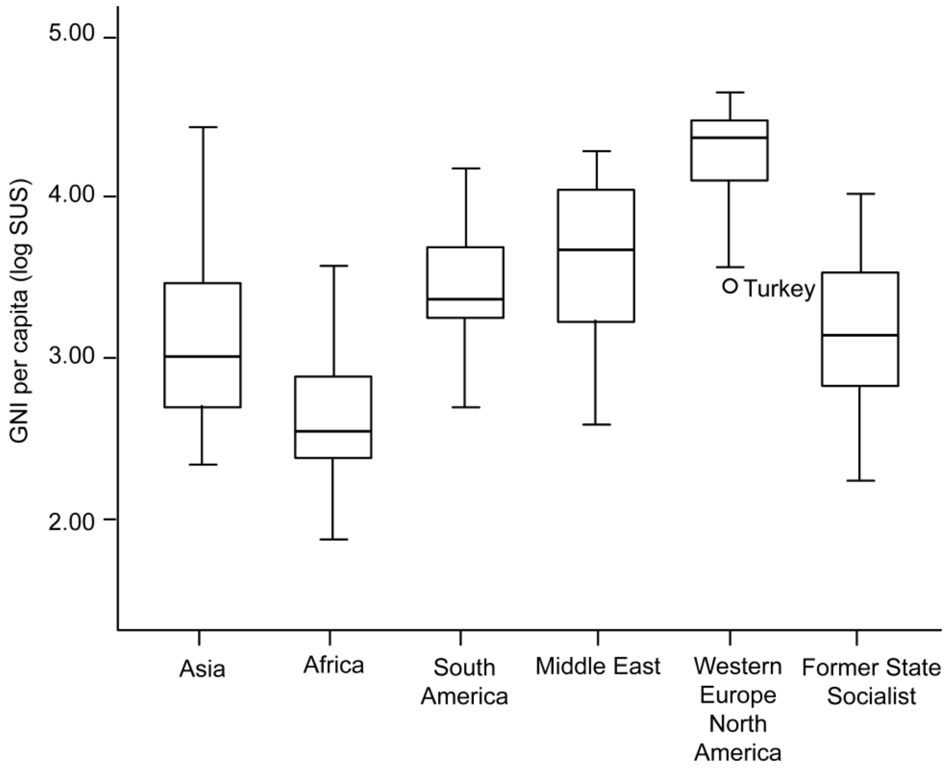


Fig. 4.27. Logged GNI per capita in 2000 by country group

Figure 10.9 shows the effect of taking logs on the distribution of GNI in the different country groups. Logging GNI per capita goes a long way towards holding the midspreads constant by making them similar in size. This means that statements can be made describing typical differences in wealth between the country groups without needing to mention the differences in spread in the same breath. But, by transforming, progress has also been made towards the first three goals: the batches are more symmetrical and bell-shaped, and some of the outliers in the original batch were not really unusual values, but merely a product of the upward straggle of the raw numbers.

TWO MARKS QUESTION AND ANSWERS (PART-A)

1. ***What is a bivariate analysis?***

   Bivariate analysis is one of the statistical analysis where two variables are observed. One variable here is dependent while the other is independent. These variables are usually denoted by X and Y. So, here we analyse the changes occured between the two variables and to what extent.

2. ***What is a causal path model?***

   Causal reasoning is often assisted by the construction of a schematic model of the hypothesized causes and effects: a causal path model. If we believe that the social class a child comes from is likely to have an effect on its school performance, we could model the relationship as in the sketch.

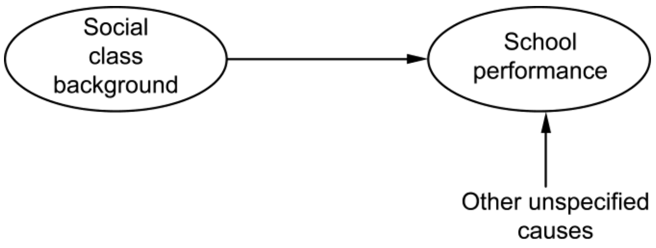


Fig. 4.28.

3. ***What is Proportions, percentages and probabilities.***

   To express a variable in proportional terms, the number in each category is divided by the total number of cases N. Percentages are proportions multiplied by

100. Proportions and percentages are bounded numbers, in that they have a floor of zero, below which they cannot go, and a ceiling of 1.0 and 100 respectively. Proportions can be used descriptively to represent the relative size of different subgroups in a population. But they can also be thought of as probabilities.

4. *Define a contingency table*

A contingency table does numerically what the three-dimensional bar chart does graphically. A contingency table shows the distribution of each variable conditional upon each category of the other. The categories of one of the variables form the rows, and the categories of the other variable form the columns. Each individual case is then tallied in the appropriate pigeonhole depending on its value on both variables. The pigeonholes are given the more scientific name cells, and the number of cases in each cell is called the cell frequency. Each row and column can have a total presented at the right-hand end and at the bottom respectively; these are called the marginals, and the univariate distributions can be obtained from the marginal distributions.

5. *What is a percentage table?*

The common way to make contingency tables readable is to cast them in percentage form. There are three different ways in which this can be done. The table was constructed by dividing each cell frequency by its appropriate row total. Tables that are constructed by percentaging the rows are usually read down the columns. This is sometimes called an 'outflow' table.

6. *What are the guidelines for a well designed table?*

(*i*) Reproducibility versus clarity

(*ii*) Labelling

(*iii*) Sources

(*iv*) Sample data

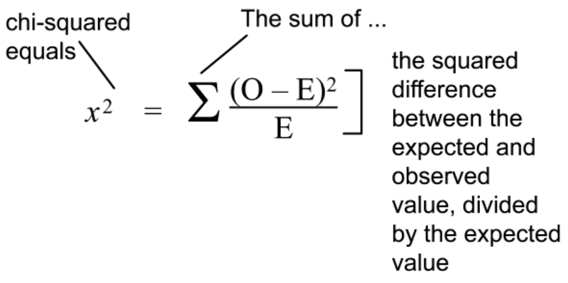
(*v*) Missing data

(*vi*) Opinion data

(*vii*) Layout

7. *What is a chi-square test?*

The chi-square statistic provides a formalized way of making this comparison. The equation for chi-square is given below. In practical terms we need to find the difference between the observed and expected frequencies for each cell of the table. We then square this value before dividing it by the expected frequency for that cell. Finally we sum these values over all the cells of the table.


chi-squared equals
The sum of ...
the squared difference between the expected and observed value, divided by the expected value

$$x^2 = \sum \left[\frac{(O - E)^2}{E} \right]$$

Fig. 4.29.

8. *Define degree of freedom*

A table with two rows and two columns is said have one degree of freedom because only one cell is known (e.g. once we know how many women are afraid to walk alone after dark) the values in the other cells can be calculated based on the row and column marginals. Similarly, a table with two columns and three rows is said to have two degrees of freedom. In formal terms the number of degrees of freedom for a table with r rows and c columns is given by the equation below:

9. *What is a Box plot?*

The method to summarize a set of data that is measured using an interval scale is called a box and whisker plot. These are maximum used for data analysis. We use these types of graphs or graphical representation to know:

- Distribution Shape
- Central Value of it
- Variability of it

10. *What is an outlier?*

In data analytics, outliers are values within a dataset that vary greatly from the others - they're either much larger, or significantly smaller. Outliers may indicate variabilities in a measurement, experimental errors, or a novelty. When going through the process of data analysis, outliers can cause anomalies in the results obtained. This means that they require some special attention and, in some cases, will need to be removed in order to analyze data effectively.

11. ***What is a T-test?***

T-test Formula

The formula for a two-sample t-test where the samples are independent as in the example of boys and girls mathematics test scores) is

$$t = \frac{\overline{X_1} - \overline{X_2}}{S_{X_1 X_2}\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

where X_1 and X_2 are the means of the two samples and $S_{X_1 X_2}$ is known as the pooled standard deviation and is calculated as follows:

$$S_{X_1 X_2} = \sqrt{\frac{(n_1 - 1)\, S^2_{X_1} + (n_2 - 1)\, S^2_{X_2}}{n_1 + n_2 - 2}}$$

here S_{X_1} is the standard deviation of one sample and S_{X_2} is the standard deviation of the other sample. In these formulae n_1 is the sample size of the first sample and n_2 is the sample size of the second sample. In simple terms therefore the size of the t-statistic depends on the size of the difference between the two means adjusted for the amount of spread and the sample sizes of the two samples.

12. ***What are scatter plots?***

Scatter plots are the graphs that present the relationship between two variables in a data-set. It represents data points on a two-dimensional plane or on a **Cartesian system**. The independent variable or attribute is plotted on the X-axis, while the dependent variable is plotted on the Y-axis. These plots are often called **scatter graphs** or **scatter diagrams**.

13. ***What is a resistant line?***

We explore paired data where you suspect a relationship between xx and yy. The focus here on how to fit a line to data in a “resistant” fashion, so the fit is relatively insensitive to extreme points. The first step to fitting a line,

- ❖ divides the data into three groups and then
- ❖ finds a summary point in each group

14. *What is meant by log transformation?*

One method for transforming data or re-expressing the scale of measurement is to rake the logarithm of each data point. This keeps all the data points in the same order but stretches or shrinks the scale by varying amounts at different points.

15. *What are the goals of transformation*

1. Data batches can be made more symmetrical.
2. The shape of data batches can be made more Gaussian.
3. Outliers that arise simply from the skewness of the distribution can be removed, and previously hidden outliers may be forced into view'.
4. Multiple batches can be made to have more similar spreads.
5. Linear, additive models may be fitted to the data.

PART- B & C

1. Explain the concept percentage table with a clear picture.
2. What are the process involved in analyzing contingency tables.
3. Explain scatter plots and resistant lines with a clear example.
4. Describe Box plots with an appropriate example
5. Explain the idea transformation in bi-variate analysis.

****

UNIT V

MULTIVARIATE AND TIME SERIES ANALYSIS

SYLLABUS

Introducing a Third Variable - Causal Explanations - Three-Variable Contingency Tables and Beyond - Longitudinal Data – Fundamentals of TSA – Characteristics of time series data – Data Cleaning – Time-based indexing – Visualizing – Grouping – Resampling.

- ❖ Introducing a Third Variable
- ❖ Causal Explanations
- ❖ Three-Variable Contingency Tables and Beyond
- ❖ Longitudinal Data
- ❖ Fundamentals of TSA
- ❖ Characteristics of time series data
- ❖ Data Cleaning
- ❖ Time-based Indexing
- ❖ Visualizing
- ❖ Grouping
- ❖ Resampling

UNIT V

MULTIVARIATE AND TIME SERIES ANALYSIS

5.1. INTRODUCING A THIRD VARIABLE

We consider ways of holding a third variable constant while assessing the relationship between two others.

5.1.1. CAUSAL EXPLANATIONS

We will now developed some experience of handling batches of data, summarizing features of their distributions, and investigating relationships between variables. We must now change gear somewhat and ask what it would take for such relationships to be treated as satisfactory explanations. Hume suggested that 'We may define a cause to be an object followed by another, and where all the objects, similar to the first, are followed by objects similar to the second. Or, in other words, where, if the first object had not been, the second never had existed'.

Direct and Indirect Effects

Causality should not necessarily be understood as a simple process in which one factor or variable has an impact on another. For example, it is likely in many cases that two or more factors will tend to work together to produce an effect. Moreover, the factors or variables contributing to the effect may themselves be causally related. For this reason, we have to keep a clear idea in our heads of the relationships between the variables in the whole causal process. In investigating the causes of absenteeism from work, for example, researchers have found different contributory factors. We will consider two possible causal factors: being female and being in a low status job. Let us construct a causal path diagram depicting one possible set of relationships between these variables.

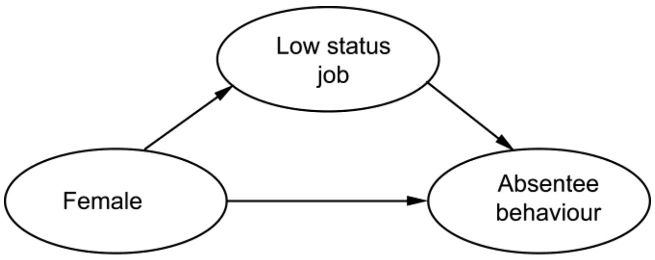


Fig. 5.1. Causes of absenteeism

The diagram in figure 11.1 represents a simple system of multiple causal paths. There is an arrow showing that those in low status jobs are more likely to go absent. Being female has a causal effect in two ways. There is an arrow straight to absentee behaviour; this says that women are more likely to be absent from work than men, regardless of the kind of job they are in. This is termed a direct effect of gender on absenteeism. There is also another way in which being female has an effect; women are more likely to be in the kind of low status, perhaps unpleasant, jobs where absenteeism is more likely, irrespective of gender. We can say that being female therefore also has an indirect effect on absenteeism, through the type of work performed. Without some empirical evidence we cannot be sure that this 'model' of the relationships between the variables is correct.

Controlling the world to learn about causes

It is one thing to declare confidently that causal chains exist in the world out there. However, it is quite another thing to find out what they are. Causal processes are not obvious. They hide in situations of complexity, in which effects may have been produced by several different causes acting together. When investigated, they will reluctantly shed one layer of explanation at a time, but only to reveal another deeper level of complexity beneath. For this reason, something that is accepted as a satisfactory causal explanation at one point in time can become problematic at another.

Researchers investigating the causes of psychological depression spent a long time carefully documenting how severe, traumatizing events that happen to people, such as bereavement or job loss, can induce it. Now that the causal effect of such life events has been established, the research effort is turning to ask how an event such as unemployment has its effect: is it through the loss of social esteem, through the decline of self evaluation and self-esteem, through lack of cash or through the sheer effect of inactivity?

Do opinion polls influence people?

Let us take an example to illustrate the different inferences which can be drawn from experiments and non-experiments.

Some people believe that hearing the results of opinion polls before an election always individuals towards the winning candidate. Imagine two ways in which empirical evidence could be collected for this proposition. An experiment could be conducted by taking a largish group of electors, splitting them into two at random, telling half that the polls indicated one candidate would win and telling the other half that they showed a rival would win. As long as there were a substantial number of people in each group, the groups would start the experiment having the same political preferences on average, since the groups were formed at random. If they differed substantially in their subsequent support for the candidates, then we could be almost certain that the phony poll information they were fed contributed to which candidate they supported.

Alternatively, the proposition could be researched in a non-experimental way. A survey could be conducted to discover what individuals believed recent opinion polls showed, and to find out which candidates the individuals themselves supported. The preferences of those who believed that one candidate was going to win would be compared with those who believed that the rival was going to win. The hypothesis would be that the former would be more sympathetic to the candidate than the latter.

If the second survey did reveal a strong relationship between individuals' perception of the state of public opinion and their own belief, should this be taken as evidence that opinion polls have a causal effect on people's voting decisions? Should policy-makers consider banning polls in pre-election periods as a result? Anyone who tried this line of argument would be taken to task by the pollsters, who have a commercial interest in resisting such reasoning. They would deny that the effect in any way proves that polls influence opinion; it could, for instance, be that supporters of a right-wing candidate are of a generally conservative predisposition, and purchase newspapers which only report polls sympathetic to their candidate.

In short, comparing individuals in a survey who thought that candidate A would win with those who believed that candidate B would win, would not be comparing two groups similar in all other possible respects, unlike the experiment discussed above. An experiment would have a better chance of persuading people that the publication of opinion polls affected individual views.

Assumptions required to infer causes

Imagine a common situation. A survey is conducted and an interesting statistical association between X and Y is discovered. There are two basic assumptions that have to be made if we wish to infer from this that X may cause Y. These involve the relationship between X and Y and other variables which might be operating. They are designed to ensure that when we compare groups which differ on X, we are comparing like with like. Before giving an exposition of these assumptions, we need a bit more terminology: other variables can be causally prior to both X and Y, intervene between X and Y, or ensue from X and Y, as shown in figure 5.2. These terms are only relative to the particular causal model in hand: in a different model we might want to explain what gave rise to the prior variable.

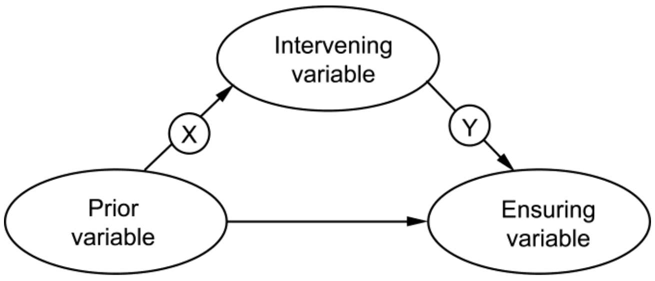


Fig. 5.2. Different casual relationships between variables

Let us discuss each of the two core assumptions in turn.

Assumption 1

X is casually prior to Y

There is nothing in the data to tell us whether X causes Y or Y causes X, so we have to make the most plausible assumption we can, based on our knowledge of the subject matter and our theoretical framework.

Assumption 2

Related prior variables have been controlled

All other variables which affect both X and Y must be held constant. In an experiment, we can be sure that there are no third variables which give rise to both X and Y because the only way in which the randomized control groups are allowed to vary is in terms of X. No such assumption can be made with non-experimental data.

Assumption 3

All variables intervening between X and Y have been controlled.

This assumption is not required before you can assume that there is a causal link between X and Y, but it is required if you aim to understand how X is causing Y.

Let us first consider a hypothetical example drawn from the earlier discussion of the causes of absenteeism. Suppose previous research had shown a positive bivariate relationship between low social status jobs and absenteeism. The question arises: is there something about such jobs that directly causes the people who do them to go off sick more than others? Before we can draw such a conclusion, two assumptions have to be made.

There are many possible outcomes once the relationship between all three variables is considered at once, four of which are shown in figure 5.3.

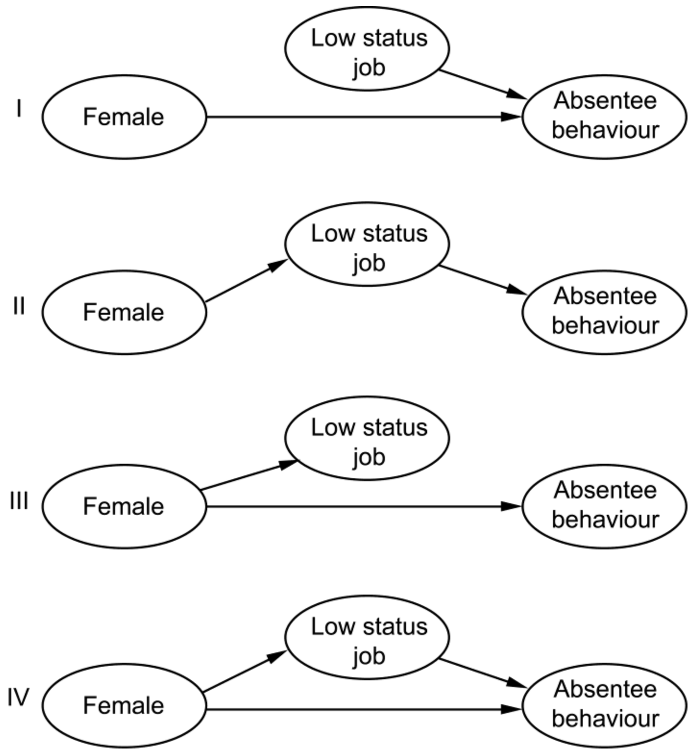


Fig. 5.3. The effect of job status on absenteeism: controlling a prior variable

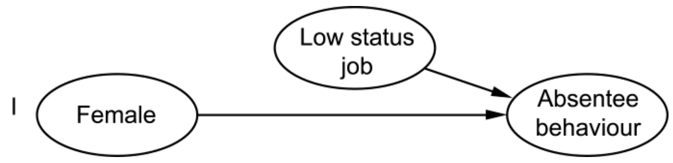


Fig. 5.4. Outcome I from figure 5.3

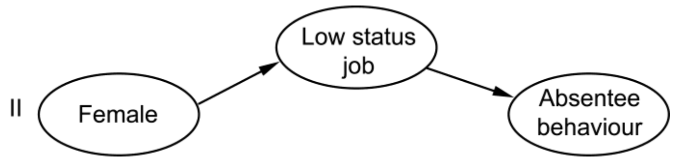


Fig. 5.5. Outcome II from figure 5.3

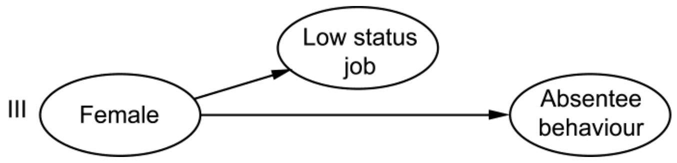


Fig. 5.6. Outcome III from figure 5.3

If the relationship between two variables entirely disappears when a causally prior variable is brought under control, we say that the original relationship was spurious. By this we do not mean that the bivariate effect did not really exist, but rather that any causal conclusions drawn from it would be incorrect. We can now introduce another meaning for that verb 'to explain': in this situation, many researchers say that the proportion of females in a job 'explains' the relationship between the status of the job and absenteeism, in the sense that it accounts for it entirely.

But what of the fourth situation which is actually the most likely outcome? It was the situation portrayed in figure 5.1.

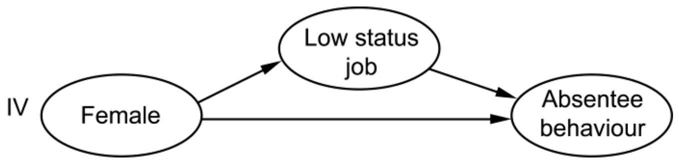


Fig. 5.7. Outcome IV from figure 5.3

Simpson's Paradox

In some cases the relationship between two variables is not simply reduced when a third, prior, variable is taken into account but indeed the direction of the relationship is completely reversed. This is often known as Simpson's paradox (named after Edward Simpson who wrote a paper describing the phenomenon that was published by the Royal Statistical Society in 1951). However, the insight that a third variable can be vitally important for understanding the relationship between two other variables is also credited to Karl Pearson in the late nineteenth century. Simpson's paradox can be succinctly summarized as follows: every statistical relationship between two variables may be reversed by including additional factors in the analysis.

Department	**Men**				**Women**			
	Accepted		**Rejected**		**Accepted**		**Rejected**	
	N	**%**	**N**	**%**	**N**	**%**	**N**	**%**
A	512	62%	311	38%	89	82%	19	18%
B	353	63%	207	37%	17	68%	8	32%
C	121	37%	204	63%	202	34%	391	66%
D	138	33%	279	67%	131	35%	244	65%
E	53	28%	138	72%	94	24%	299	76%
F	16	6%	256	94%	24	7%	317	93%
Total	3714	44%	4728	56%	1512	35%	2809	65%

Fig. 5.8. Success of application to graduate school by gender and department: an example of Simpson's paradox

5.2. THREE-VARIABLE CONTINGENCY TABLES AND BEYOND CAUSAL PATH MODELS FOR THREE VARIABLES

The set of paths of causal influence, both direct and indirect, that we want to begin to consider are represented in figure 12.5. In this causal model we are trying to explain social trust, the base is therefore the belief that 'You can't be too careful'. The base categories selected for the explanatory variables are having lower levels of qualifications and not being a member of a voluntary organization, to try and avoid negative paths. Each arrow linking two variables in a causal path diagram represents the direct effect of one variable upon the other, controlling all other relevant variables. The rule for identifying the relevant variables was given in chapter 11: when we are assessing the direct effect of one variable upon another, any third variable which is likely to be causally connected to both variables and prior to one of them should be controlled. Coefficient b in figure 12.5 shows the direct effect of being in a voluntary association on the belief that most people can be trusted. To find its value, we focus attention on the proportion who say that most people can be trusted, controlling for level of qualifications.

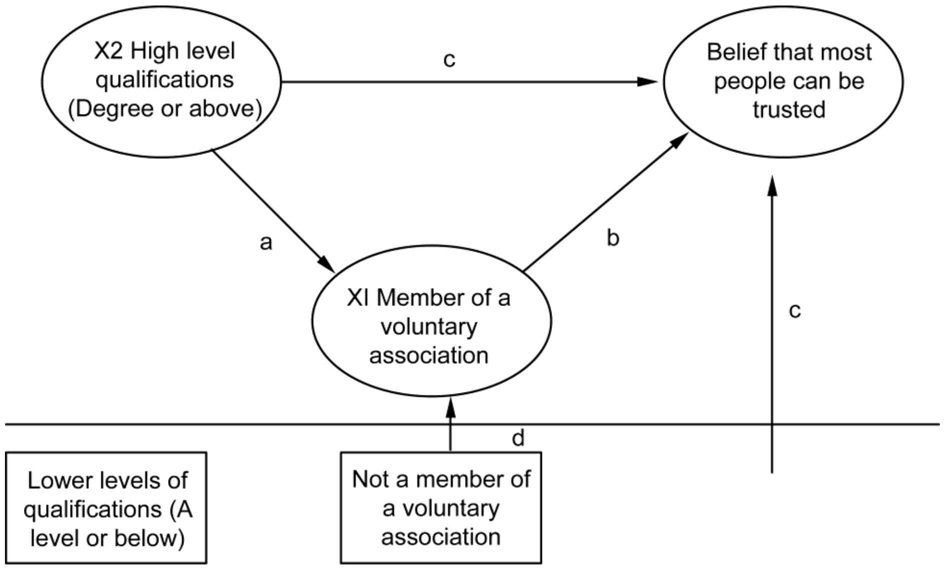

X2 High level qualifications (Degree or above)
c
Belief that most people can be trusted
a
b
XI Member of a voluntary association
c
d
Lower levels of qualifications (A level or below)
Not a member of a voluntary association

Fig. 5.9. Social trust by membership of voluntary association and level of qualifications: casual path diagram

More complex models: going beyond three variables

Clearly there are likely to be many other factors or 'variables' that will have an influence, both on volunteering behaviour and on social trust. For example, in the model discussed above we have not considered gender or age, and both of these may have an impact on all of the variables in our model.

As can be seen from the discussion above, it becomes quite complicated even to calculate the direct and indirect causal paths when we have a simple model with three variables. We therefore need to go beyond these paper and pencil techniques if we are going to build more complex models that aim to compare the impact of a number of different explanatory variables on an outcome variable such as social trust. The following section describes the conceptual foundations that underlie models to examine the factors influencing a simple dichotomous (two-category) variable.

Logistic Regression Models

Regression analysis is a method for predicting the values of a continuously distributed dependent variable from an independent, or explanatory, variable. The principles behind logistic regression are very similar and the approach to building models and interpreting the models is virtually identical. However, whereas regression (more properly termed Ordinary Least Squares regression, or OLS regression) is used when the dependent variable is continuous, a binary logistic regression model is used when the dependent variable can only take two values. In many examples this dependent variable indicates whether an event occurs or not and logistic regression is used to model the probability that the event occurs. In the example we have been discussing above, therefore, logistic regression would be used to model the probability that an individual believes that most people can be trusted. When we are just using a single explanatory variable, such as volunteering, the logistic regression can be written as

5.3. LONGITUDINAL DATA

It is important to distinguish longitudinal data from the time series data. Although time series data can provide us with a picture of aggregate change, it is only longitudinal data that can provide evidence of change at the level of the individual. Time series data could perhaps be understood as a series of snapshots of society, whereas longitudinal research entails following the same group of individuals over time and linking information about those individuals from one time point to another.

For example, in a study such as the British Household Panel Survey, individuals are interviewed each year about a range of topics including income, political preferences and voting. This makes it possible to link data about individuals over time and examine, for example, how an individual's income may rise (or fall) year on year and how their political preferences may change.

The first part provides a brief introduction to longitudinal research design and focuses on some of the issues in collecting longitudinal data and problems of attrition. The second part then provides a brief conceptual introduction to the analysis of longitudinal data.

Collecting longitudinal data

Prospective and retrospective research designs

Longitudinal data are frequently collected using a prospective longitudinal research design, i.e. the participants in a research study are contacted by researchers and asked to provide information about themselves and their circumstances on a number of different occasions. This is often referred to as a panel study. However, it is not necessary to use a longitudinal research design in order to collect longitudinal data and there is therefore a conceptual distinction between longitudinal data and longitudinal research. Indeed, the retrospective collection of longitudinal data is very common.

$$\text{Prob(event)} = \frac{1}{1 + e^{-\beta X}}$$

or more specifically $$\text{Prob (trust)} = \frac{1}{1 + e^{-\beta \text{ volunteering}}}$$

In particular, it has become an established method for obtaining basic information about the dates of key life events such as marriages, separations and divorces and the birth of any children (i.e. event history data). This is clearly an efficient way of collecting longitudinal data and obviates the need to re-contact the same group of individuals over a period of time.

A potential problem is that people may not remember the past accurately enough to provide good quality data. While some authors have argued that recall is not a major problem for collecting information about dates of significant life events, other research suggests that individuals may have difficulty remembering dates accurately, or may prefer not to remember unfavourable episodes or events in their lives. Large-scale quantitative surveys often combine a number of different data collection strategies so they do not always fit neatly into the classification of prospective or retrospective designs. In particular, longitudinal event history data are frequently collected retrospectively as part of an ongoing prospective longitudinal study.

5.4. FUNDAMENTALS OF TSA

Time series data is a collection of quantities that are assembled over even intervals in time and ordered chronologically. The time interval at which data is collected is generally referred to as the time series frequency.

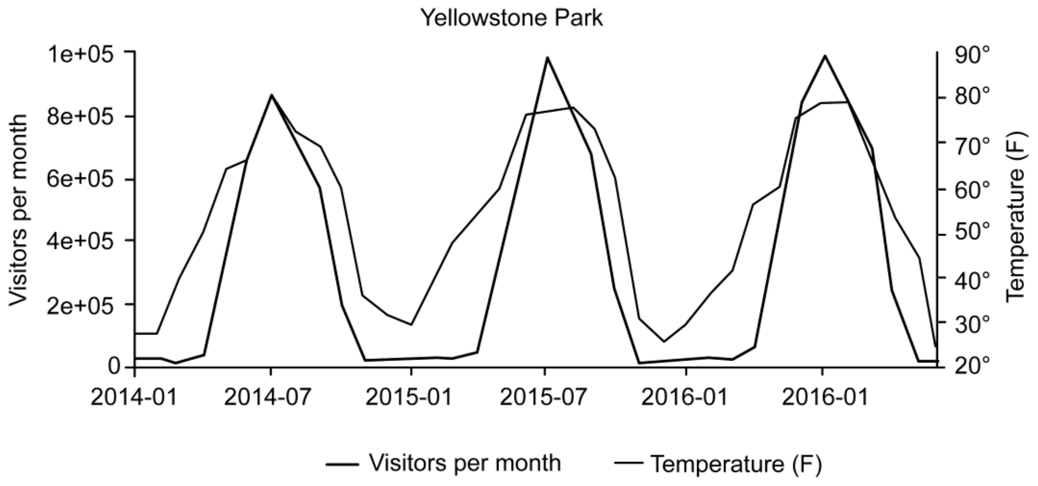


Fig. 5.10. Yellowstone Park

For example, the time series graph above plots the visitors per month to Yellowstone National Park with the average monthly temperatures. The data ranges from January 2014 to December 2016 and is collected at a monthly frequency.

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly. However, this type of analysis is not merely the act of collecting data over time.

5.5. CHARACTERISTICS OF TIME SERIES DATA

- ❖ Time-series data
- ❖ Structured data
- ❖ No updates on data
- ❖ Single data source
- ❖ The ratio of read/write is smaller

- The trend is more important
- Retention policy
- Aggregation over time or a set of devices
- Real-time computing or analysis is required
- Traffic is stable
- Special computing is needed
- Data volume is huge

5.6. DATA CLEANING

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset. But it is crucial to establish a template for your data cleaning process so you know you are doing it the right way every time.

How to clean data

While the techniques used for data cleaning may vary according to the types of data your company stores, you can follow these basic steps to map out a framework for your organization.

Step 1: Remove duplicate or irrelevant observations

Remove unwanted observations from your dataset, including duplicate observations or irrelevant observations. Duplicate observations will happen most often during data collection. When you combine data sets from multiple places, scrape data, or receive data from clients or multiple departments, there are opportunities to create duplicate data. De-duplication is one of the largest areas to be considered in this process. Irrelevant observations are when you notice observations that do not fit into the specific problem you are trying to analyze. For example, if you want to analyze data regarding millennial customers, but your dataset includes older generations, you might remove those irrelevant observations. This can make analysis more efficient and minimize distraction from your primary target—as well as creating a more manageable and more performant dataset.

Step 2: Fix structural errors

Structural errors are when you measure or transfer data and notice strange naming conventions, typos, or incorrect capitalization. These inconsistencies can cause mislabeled categories or classes. For example, you may find “N/A” and “Not Applicable” both appear, but they should be analyzed as the same category.

Step 3: Filter unwanted outliers

Often, there will be one-off observations where, at a glance, they do not appear to fit within the data you are analyzing. If you have a legitimate reason to remove an outlier, like improper data-entry, doing so will help the performance of the data you are working with. However, sometimes it is the appearance of an outlier that will prove a theory you are working on. Remember: just because an outlier exists, doesn’t mean it is incorrect. This step is needed to determine the validity of that number. If an outlier proves to be irrelevant for analysis or is a mistake, consider removing it.

Step 4: Handle missing data

You can’t ignore missing data because many algorithms will not accept missing values. There are a couple of ways to deal with missing data. Neither is optimal, but both can be considered.

1. As a first option, you can drop observations that have missing values, but doing this will drop or lose information, so be mindful of this before you remove it.
2. As a second option, you can input missing values based on other observations; again, there is an opportunity to lose integrity of the data because you may be operating from assumptions and not actual observations.
3. As a third option, you might alter the way the data is used to effectively navigate null values.

Step 5: Validate and QA

At the end of the data cleaning process, you should be able to answer these questions as a part of basic validation:

- ❖ Does the data make sense?
- ❖ Does the data follow the appropriate rules for its field?
- ❖ Does it prove or disprove your working theory, or bring any insight to light?
- ❖ Can you find trends in the data to help you form your next theory?
- ❖ If not, is that because of a data quality issue?

False conclusions because of incorrect or "dirty" data can inform poor business strategy and decision-making. False conclusions can lead to an embarrassing moment in a reporting meeting when you realize your data doesn't stand up to scrutiny. Before you get there, it is important to create a culture of quality data in your organization. To do this, you should document the tools you might use to create this culture and what data quality means to you.

Advantages and Benefits of Data Cleaning

Having clean data will ultimately increase overall productivity and allow for the highest quality information in your decision-making. Benefits include:

- ❖ Removal of errors when multiple sources of data are at play.
- ❖ Fewer errors make for happier clients and less-frustrated employees.
- ❖ Ability to map the different functions and what your data is intended to do.
- ❖ Monitoring errors and better reporting to see where errors are coming from, making it easier to fix incorrect or corrupt data for future applications.
- ❖ Using tools for data cleaning will make for more efficient business practices and quicker decision-making.

5.7. TIME-BASED INDEXING

A time series is a series of data points indexed in time order. If you index the dataset by date, you can easily carry out a time series analysis.

There are three index types of time series:

1. **DatetimeIndex** - The index type for timestamped data is DatetimeIndex.
2. **PeriodIndex** - The index type for period data, which shows the fixed interval date data, is PeriodIndex
3. **TimedeltaIndex** - The index type for the Timedelta data, which shows the time between two dates, is TimedeltaIndex

Creating a time series

To show how to create a time series, first, let me import pandas and numpy.

In [1]:
```
import pandas as pd ; import numpy as np.
```

Let's create a variable named date with the start date and end date.

In [2]:
```
date = pd.date_range(
      start =”2018”,end =”2019”, freq=”BM”)
```

Let's create a time series named ts using the date variable.

In [3]:
```
ts = pd.Series(
        np.random.randn(len(date)),index = date)
ts
```

Out [3]:
```
2018-01-31      – 1.977003
2018-02-28      – 0.339459
2018-03-30      – 0.587687
2018-04-30        1.141997
2018-05-31      – 0.125199
2018-06-29      – 1.090406
2018-07-31      – 0.435640
2018-08-31        0.181651
2018-09-28      – 2.518869
2018-10-31        1.428868
2018-11-30      – 0.357551
2018-12-31        0.612771
```

```
Freq:  BM,  dtype:  float64
```

Let’s take a look at the index of this data.

```
In [4]:   1   ts.index
```

```
Out [4]:   DatetimeIndex([‘2018-01-31’, ‘2018-02-28’, ‘2018
           -03-30’, ‘2018-04-30’,
                          ‘2018-05-31’, ‘2018-06-29’, ‘2018’
           -07-31’, ‘2018-08-31’,
                          ‘2018-09-28’, ‘2018-10-31’, ‘2018
           -11-30’, ‘2018-12-31’],
                         dtype=’datetime64[ns]’, freq=’BM’)
```

As you can see, the index structure of the ts is DatetimeIndex. With these indexes, we can play like a ball. To slice indexes, let’s print the first 5 of the indexes.

```
In [5]:   1   ts [ :5] .index
```

```
Out [5]:   DatetimeIndex([‘2018-01-31’, ‘2018-02028’, 2018
           -03-30’, ‘2018-04-30’,
                          ‘2018-05-31’],
                        dtype=’datetime64[ns]’, freq=’BM’)
```

Reading a time series dataset

Now, let’s use a real-world dataset showing the stock market values of Facebook. Let me read this dataset with the read_csv method.

```
In [6]:   1   fb=pd.read_csv(“FB.csv”)
```

You can find this dataset here. Let’s see the first 5 rows of the dataset with the head method.

```
In [7]:   1   fb.head ( )
```

Out [7]:		**Date**	**Open**	**High**	**Low**	**Close**	
	0	2018-07-30	175.300003	175.300003	166.559998	171.059998	1
	1	2018-07-31	170.669998	174.240005	170.000000	172.580002	1
	2	2018-08-01	173.929993	175.080002	170.899994	171.649994	1
	3	2018-08-02	170.679993	176.789993	170.270004	176.369995	1
	4	2018-08-03	177.690002	178.850006	176.149994	177.779999	1

Converting dates into indexes

Let's take a look at the column types.

In [8]:
```
1   fb.dtypes
```

Out [8]:
```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

As you can see, the type of the date column is an object. Let's convert this date column to the DateTime type. To do this, I'm going to use the parse_dates parameter when reading the dataset.

In [9]:
```
fb=pd.read_csv(
      "FB.csv",parse_dates=["Date"])
```

Let's convert the date column into the index with the index_col parameter.

In [10]:
```
fb=pd.read_csv(
      "FB.csv",
      parse_dates=["Date"],
      index_col="Date")
```

Let's take a look at the indexes of the dataset.

In [11]:
```
fb.index
```

Out [11]:
```
DatetimeIndex(['2018-07-30', '2018-07-31', '2018
-08-01', '2018-08-02',
              '2018-08-03','2018-08-06', '2018
-08-07', '2018-08-08',
              '2018-08-09', '2018-08-10',
              …
              '2019-07-16', '2019-07-17', '2019
-07-18', '2019-07-19',
              '2019-07-22', '2019-07-23', '2019
- 07-24', '2019-07-25',
              '2019-07-26', '2019-07-29'],
              dtype='datetime64[ns]', name='Dat
e', 'length=251, freq=None)
```

As you can see, the index structure has changed and turned into a DatetimeIndex object. Let's see the first rows of the dataset.

In [12]:
```
fb.head ( )
```

Out [12]:

Date	Open	High	Low	Close	Adj
2018-07-30	175.300003	175.300003	166.559998	171.059998	171.0
2018-07-31	170.669998	174.240005	170.000000	172.580002	172.5
2018-08-01	173.929993	175.080002	170.899994	171.649994	171.6
2018-08-02	170.679993	176.789993	170.270004	176.369995	176.3
2018-08-03	177.690002	178.850006	176.149994	177.779999	177. 7

In [13]:

```
fb["2019-06"]
```

Out [13]:

Date	Open	High	Low	Close	Adj
2019-06-03	175.000000	175.050003	161.009995	164.149994	164.1
2019-06-04	163.710007	168.279999	160.839996	167.500000	167.5
2019-06-05	167.479996	168.720001	164.630005	168.169998	168.1
2019-06-06	168.300003	169.699997	167.229996	168.330002	168.3
2019-06-07	170.169998	173.869995	168.839996	173.350006	173.3
2019-06-10	174.750000	177.860001	173.800003	174.820007	174.8
2019-06-11	178.479996	179.979996	176.789993	178.100006	178.1

It is very useful to convert the dates into the DatetimeIndex structure. For example, you can easily select the values of the 6 months of 2019.

Working with Indexes

To see the mean of the closing prices for a date, you first write the Close column and then use the mean function.

```
In  [14]:   fb[“2019-06”].Close.mean( )
```

```
Out[14]:    181.27450025000002
```

```
In  [15]:   fb[“2019-07-05” :”2019-07-10”]
```

Out [15]:	Date	Open	High	Low	Close	Adj
	2019-07-05	196.179993	197.070007	194.169998	196.399994	196.3
	2019-07-08	195.190002	196.679993	193.639999	195.759995	195.7
	2019-07-09	194.970001	199.460007	194.889999	199.210007	199.2
	2019-07-10	100.000000	202.960007	199669996	202.729996	202.7

Since dates are indexes, you can slice them as follows:

You can convert a date into the timestamp.

```
In  [16]:   t=pd.to_datetime(“7/22/2019”)
            t
```

```
Out[16]:  Timestamp(‘2019-07-22 00:00:00’)
```

You can compare this date with the dates in the dataset with the loc method.

```
In  [17]:   fb.loc]fb.index>=t,:]
```

Out [17]:	**Date**	**Open**	**High**	**Low**	**Close**	Adj
	2019-07-22	199.910004	202.570007	198.809996	202.320007	202.3
	2019-07-23	202.839996	204.240005	200.960007	202.360001	202.3
	2019-07-24	197.630005	204.809998	197.220004	204.660004	204.6
	2019-07-25	206.699997	208.660004	198.259996	200.710007	200.7
	2019-07-26	200.190002	202.880005	196.250000	199.750000	199.7
	2019-07-29	199.000000	199.590302	197.880005	198.059998	198.0

Dating a Dataset

To perform a time series analysis, you need to assign date values. To show this, I'm going to use a dataset without dates. Let's read this dataset.

In [18]:
```
fb1=pd.read_csv("FB-no-date.csv",sep=";")
```

Let's have a look at the first rows of the dataset.

In [19]:
```
fb1.head( )
```

Out [19]:		**Open**	**High**	**Low**	**Close**	**Adj Close**
	0	162600006	163130005	161690002	162279999	162279999
	1	163899994	167500000	163830002	167369995	167369995
	2	167369995	171880005	166550003	171259995	171259995
	3	172899994	173570007	171270004	172509995	172509995
	4	171500000	171740005	167610001	169130005	169130005

Notice that there is no date column in the dataset. Let's add a date column to this dataset. To do this, let me generate a date with the date_range function. I'm going to use the start, end, and freq parameters. Here, B represents business days.

In [20]:
```
dates=pd.date_range(start="03/01/2019",
                    end="03/29/2019",
                    freq="B")
dates
```

Out [20]:
```
DatetimeIndex(['2019-03-01', '2019-03-04', '2019
-03-05',   '2019-03-06',
               '2019-03-07','2019-03-08', '2019
-03-11',   '2019-03-12',
               '2019-03-13', '2019-03-14', '2019
-03-15',   '2019-03-18',
               '2019-03-19', '2019-03-20', '2019
-03-21',   '2019-03-22',
               '2019-03-25', '2019-03-26', '2019
-03-27',   '2019-03-28',
               '2019-03-29']
             dtype='datetime64[ns]', freq='B')
```

Now, let's assign this created date variable to the dataset as an index.

Let's see the first five rows of the dataset.

In [22]:
```
fb1.head( )
```

Out [22]:		**Open**	**High**	**Low**	**Close**	**Adj Close**
	2019-03-01	162600006	163130005	161690002	162279999	162279
	2019-03-04	163899994	167500000	163830002	167369995	167369
	2019-03-05	167369995	171880005	166550003	171259995	17259
	2019-03-06	172899994	173570007	171270004	172509995	172509
	2019-03-07	171500000	171740005	167610001	169130005	16913

As you can see, working days have been added to the dataset. Let’s look at the index of the dataset

Since the dataset is indexed with time, you can easily work with time series.

5.8. DATA VISUALIZATION

In [23]:
```
1    fb1.index
```

Out [23]:
```
DatetimeIndex([‘2019-03-01’, ‘2019-03-04’, ‘2019
-03-05’,   ‘2019-03-06’,
                  ‘2019-03-07’,‘2019-03-08’, ‘2019
-03-11’,   ‘2019-03-12’,
                  ‘2019-03-13’, ‘2019-03-14’, ‘2019
-03-15’,   ‘2019-03-18’,
                  ‘2019-03-19’, ‘2019-03-20’, ‘2019
-03-21’,   ‘2019-03-22’,
                  ‘2019-03-25’, ‘2019-03-26’, ‘2019
-03-27’,   ‘2019-03-28’,
                  ‘2019-03-29’]
                  dtype=’datetime64[ns]’, freq=’B’)
```

Let's draw a graph showing closing prices. First, I'm going to use the % matplotlib inline magic command to see the graph between lines.

```
In [24]:  1   %matplotlib inline
```

```
In [25]:  1   1 fb1.close.plot( )
```

Out[25]: <AxesSubplot:>

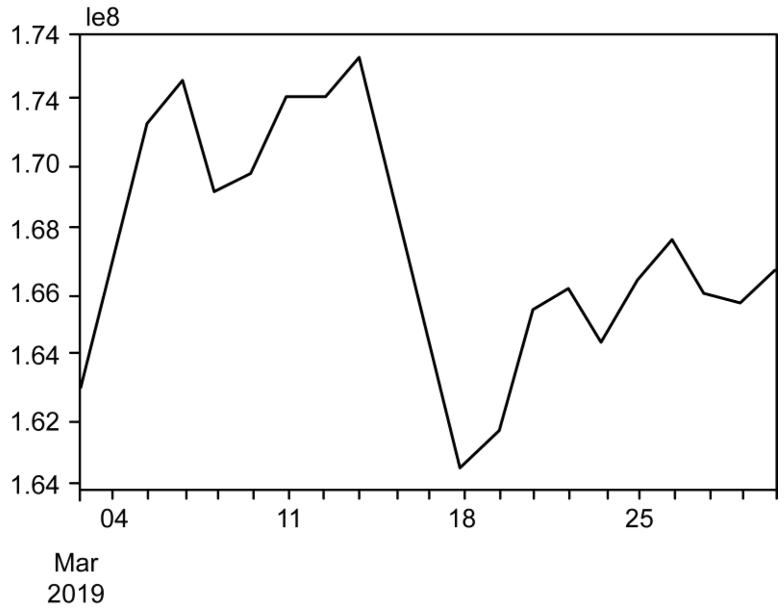


Fig. 5.11.

Let's draw a line plot.

How to use the asfreq method?

There are no stock values for holidays in our dataset since stock values are fixed during holidays. If you want to add these holidays to the dataset, you can use the as freq method. This method is optionally used to fill missing values.

```
In [26]:  1   fb1.asfreq("H",method="pad").head( )
```

Out [26]:		**Open**	**High**	**Low**	**Close**	**Adj Close**
	2019-03-01 00:00:00	162600006	163130005	161690002	162279999	1622
	2019-03-01 01:00:00	162600006	163130005	161690002	162279999	1622
	2019-03-01 02:00:00	162600006	163130005	161690002	162279999	1622
	2019-03-01 03:00:00	162600006	163130005	161690002	162279999	1622
	2019-03-01 04:00:00	162600006	163130005	161690002	162279999	1622

Let’s fill in the missing values weekly

In [27]:
```
fb1.asfreq(“W”,method=”pad”)
```

Out [27]:		**Open**	**High**	**Low**	**Close**	Adj Close
	2019-03-03	162600006	163130005	161690002	162279999	162279
	2019-03-10	166199997	169619995	165970001	169600006	169600
	2019-03-17	167160004	167580002	162509995	165979996	165979
	2019-03-24	165649994	167419998	164089996	164339996	164339

Creating time series the date range method

Let’s generate dates with the start, periods, and freq parameters.

In [29]:
```
z=pd.date_range(start=”3/1/2019”,
                periods=60 , freq=”B”)
z
```

Out [29]:
```
DatetimeIndex([‘2019-03-01’, ‘2019-03-04’, ‘2019
-03-05’,   ‘2019-03-06’,
               ‘2019-03-07’, ‘2019-03-08’, ‘2019
-03-11’,   ‘2019-03-12’,
               ‘2019-03-13’, ‘2019-03-14’, ‘2019
-03-15’,   ‘2019-03-18’,
               ‘2019-03-19’, ‘2019-03-20’, ‘2019
-03-21’,   ‘2019-03-22’,
               ‘2019-03-25’, ‘2019-03-26’, ‘2019
-03-27’,   ‘2019-03-28’,
               ‘2019-03-29’, ‘2019-04-01’, ‘2019
-04-02’,   ‘2019-04-03’,
               ‘2019-04-04’, ‘2019-04-05’, ‘2019
-04-08’,   ‘2019-04-09’,
               ‘2019-04-10’, ‘2019-04-11’, ‘2019
-04-12’,   ‘2019-04-15’,
```

You can set the frequency as hourly the H value.

In [30]:
```
z=pd.date_range
     (start=”3/1/2019”, periods=30 , freq=”H”)
z
```

Out [30]:
```
DatetimeIndex([‘2019-03-01   00:00:00’,    ‘2019-03-0
1  01:00:00’,
               ‘2019-03-01   02:00:00’,    ‘2019-03-0
1  03:00:00’,
               ‘2019-03-01   04:00:00’,    ‘2019-03-0
1  05:00:00’,
               ‘2019-03-01   06:00:00’,    ‘2019-03-0
1  07:00:00’,
               ‘2019-03-01   08:00:00’,    ‘2019-03-0
1  09:00:00’,
               ‘2019-03-01   10:00:00’,    ‘2019-03-0
1  11:00:00’,
               ‘2019-03-01   12:00:00’,    ‘2019-03-0
1  13:00:00’,
               ‘2019-03-01   14:00:00’,    ‘2019-03-0
1  15:00:00’,
               ‘2019-03-01   16:00:00’,    ‘2019-03-0
1  17:00:00’,
               ‘2019-03-01   18:00:00’,    ‘2019-03-0
```

Let’s create a time series with the z variable.

In [31]:
```
ts=pd.Series(
       np.random.randint(1,10,len(z)),index=z)
ts.head( )
```

Out [31]:
```
2019-03-01    00:00:00    7
2019-03-01    01:00:00    1
2019-03-01    02:00:00    6
2019-03-01    03:00:00    2
2019-03-01    04:00:00    7
Freq:  H,  dtype:  int32
```

5.9. GROUPING

Group Time Series (GTS) reports contain raw or aggregated data for a group of resources over a particular reporting period.

Raw data can be displayed for daily and weekly reporting periods only. Aggregated data can be displayed for any reporting period, but different reporting periods support different granularity values.

GTS reports support the following features:

- Near Real Time (NRT) data points. NRT data is raw data collected during the current hour that has not yet been written to the database.
- Access to all branches of a group hierarchy. Subelement groups are organized within a tree structure. When a GTS report is deployed against a particular group in a group tree, resources in that group and in groups at all levels of the tree below it are included in the aggregation. If a particular resource appears in multiple groups within the group tree, that resource is included in the aggregation only once.

Reports on GTS

- **Spatial aggregation** - Spatial aggregation is the aggregation of all data points for a group of resources over a specified period (the granularity). Data aggregations in Group Time Series reports are of the spatial aggregation type.
- **Sum of Average Reports** - A Sum of Average (**sumOfAvg**) report is an extension of the Group Time Series report. It calculates two data points for each granularity period.

5.10. RESAMPLING

While dealing with time-Series data analysis we need to combine data into certain intervals like with each day, a week, or a month.

We will solve these using only 2 Pandas APIs i.e. resample() and GroupBy().

The resample() function is used to resample time-series data. Convenience method for frequency conversion and resampling of time series. The object must have a DateTime-like index(DatetimeIndex, PeriodIndex, or TimedeltaIndex), or pass DateTime-like values to the on or level keyword.

If one wants to arrange the time series data in patterns like monthly, weekly, daily, etc., this function is very useful. This function is available in Pandas library.

Resampling

Resampling is for frequency conversion and resampling of time series. So, if one needs to change the data instead of daily to monthly or weekly etc. or vice versa. For this, we have resample option in pandas library[2]. In the resampling function, if we need to change the date to datetimeindex there is also an option of parameter "on" but the column must be datetime-like.

```
df.resample('w', on='LastUpdated').mean ( )
```

LastUpdated	Capacity	Occupancy
2016-10-09	1363.275862	546.699234
2016-10-16	1395.311828	612.520908
2016-10-23	1406.956522	597.105878
2016-10-30	1391.326531	628.676871
2016-11-06	1405.492228	600.865285
2016-11-10	1396.000000	609.621368
2016-11-20	1391.530612	627.790533
2016-11-27	1402.783505	678.044674
2016-12-04	1392.357143	713.621825
2016-12-11	1436.475410	712.380996
2016-12-18	1383.288645	696.309159
2016-12-25	1420.153846	844.256410

Below from resampling with option "D", the data got changed into daily data, i.e., all the dates will be taken into account. 375717 records downsampled to 77 records.

```
df3.resample("D").mean( ) # daily option
```

LastUpdated	Occupancy
2016-10-04	655.543651
2016-10-05	655.185185
2016-10-06	636.942130
2016-10-07	576.282407
2016-10-08	428.036232
-	-
2016-12-15	736.445110
2016-12-16	675.021073
2016-12-17	726.115385
2016-12-18	613.589583
2016-12-19	844.256410

77 rows × 1 columns

A resample option is used for two options, i.e., upsampling and downsampling.

Upsampling: In this, we resample to the shorter time frame, for example monthly data to weekly/biweekly/daily etc. Because of this, many bins are created with NaN values and to fill these there are different methods that can be used as pad method and bfill method. For example, changing weekly data to daily data and using bfill method following results are obtained, so bfill filling backward the new missing values in the resampled data:

```
dd.resample ('D').pad ( ) [:15]
```

LastUpdated	Capacity	Occupancy
2016-10-09	1363.275862	546.699234
2016-10-10	1363.275862	546.699234
2016-10-11	1363.275862	546.699234
2016-10-12	1363.275862	546.699234
2016-10-13	1363.275862	546.699234
2016-10-14	1363.275862	546.699234
2016-10-15	1363.275862	546.699234
2016-10-16	1395.311828	612.520908
2016-10-17	1395.311828	612.520908
2016-10-18	1395.311828	612.520908
2016-10-19	1395.311828	612.520908
2016-10-20	1395.311828	612.520908
2016-10-21	1395.311828	612.520908
2016-10-22	1395.311828	612.520908
2016-10-23	1406.956522	597.105878

```
dd.resample ('D').bfill ( ) [:15]
```

LastUpdated	Capacity	Occupancy
2016-10-09	1363.275862	546.699234
2016-10-10	1395.311828	612.520908
2016-10-11	1395.311828	612.520908
2016-10-12	1395.311828	612.520908
2016-10-13	1395.311828	612.520908
2016-10-14	1395.311828	612.520908
2016-10-15	1395.311828	612.520908

LastUpdated	Capacity	Occupancy
2016-10-16	1395.311828	612.520908
2016-10-17	1406.956522	597.105878
2016-10-18	1406.956522	597.105878
2016-10-19	1406.956522	597.105878
2016-10-20	1406.956522	597.105878
2016-10-21	1406.956522	597.105878
2016-10-22	1406.956522	597.105878
2016-10-23	1406.956522	597.105878

Other method is pad method, it forward fills the values as above right:

We can also use asfreq() or fillna() methods in upsamling.

Downsampling: In this we resample to the wider time frame, for example resample daily data to weekly/biweekly/monthly etc. For this we have options like sum(), mean(), max() etc. For example, daily data got resampled to month start data and mean function is used as below:

```
df3.resample("MS").mean( ) [ : ]
```

LastUpdated	Occupancy
2016-10-01	600.6633861
2016-11-01	637.142419
2016-12-01	714.497266

Graphical Representation of Resampling

After resampling data by four different rules, i.e., hourly, daily, weekly, and monthly, following graphs are obtained. We can clearly see the difference in shorter vs wider time frames. In the hourly plot, more noise is there and it is decreasing from daily to weekly to monthly. As per study objective, we can decide which option for rule would be best.

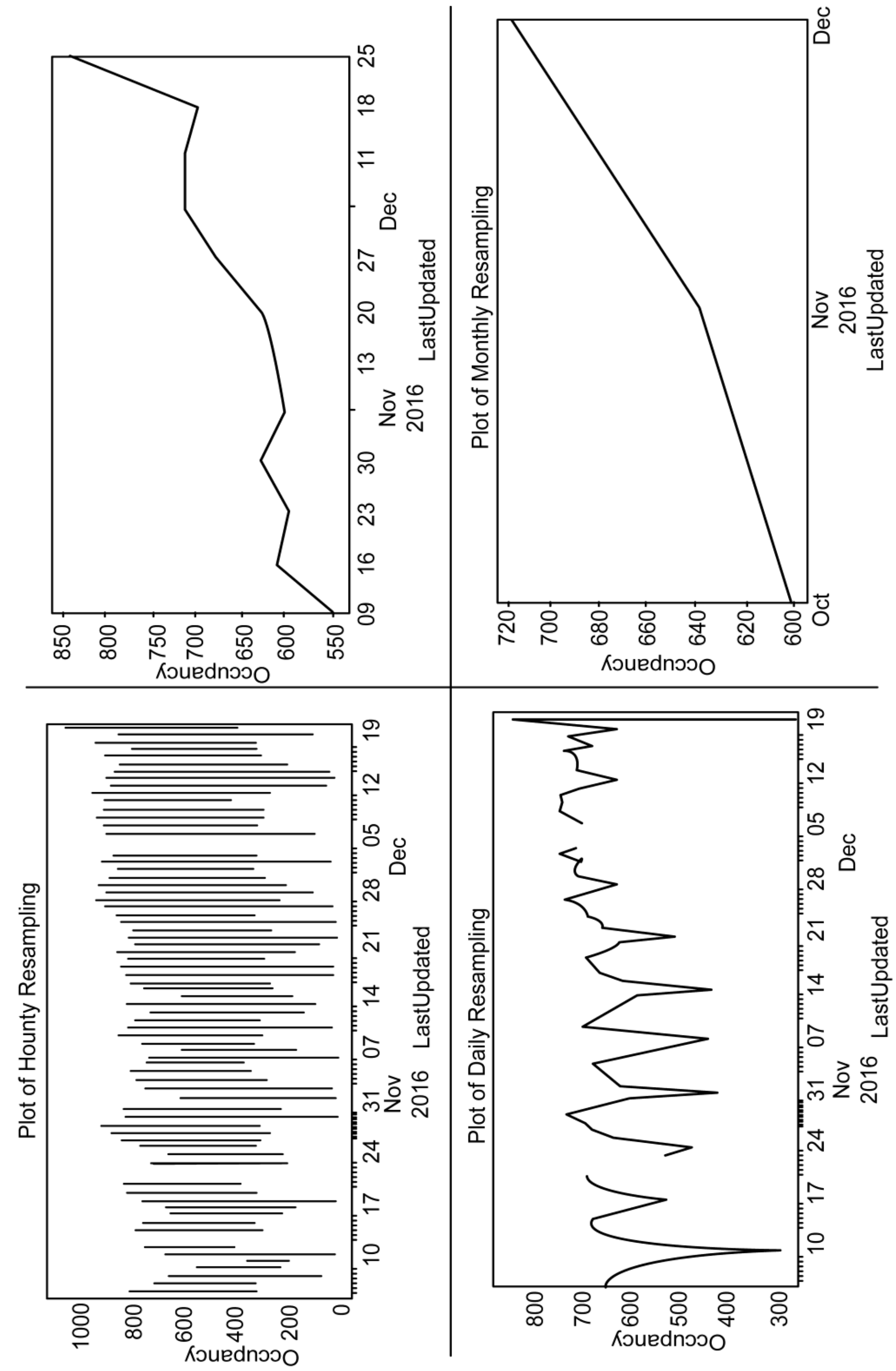

Plot of Hounty Resampling
Occupancy
LastUpdated
Nov
2016
Dec
Occupancy
LastUpdated
Nov
2016
Dec
Plot of Daily Resampling
Occupancy
LastUpdated
Nov
2016
Dec
Plot of Monthly Resampling
Occupancy
LastUpdated
Oct
Nov
2016
Dec

Fig. 5.12.

TWO MARKS QUESTION AND ANSWERS (PART-A)

1. ***Define a multivariate analysis.***

Multivariate data analysis is a type of statistical analysis that involves more than two dependent variables, resulting in a single outcome. Many problems in the world can be practical examples of multivariate equations as whatever happens in the world happens due to multiple reasons.

One such example of the real world is the weather. The weather at any particular place does not solely depend on the ongoing season, instead many other factors play their specific roles, like humidity, pollution, etc. Just like this, the variables in the analysis are prototypes of real-time situations, products, services, or decision-making involving more variables.

2. ***Differentiate longtidudinal data and time series data.***

It is important to distinguish longitudinal data from the time series data. Although time series data can provide us with a picture of aggregate change, it is only longitudinal data that can provide evidence of change at the level of the individual. Time series data could perhaps be understood as a series of snapshots of society, whereas longitudinal research entails following the same group of individuals over time and linking information about those individuals from one time point to another.

3. ***What are the fundamentals of TSA?***

Time series data is a collection of quantities that are assembled over even intervals in time and ordered chronologically. The time interval at which data is collected is generally referred to as the time series frequency.

4. ***Write in brief the characteristics of time series data.***

- Single data source
- The ratio of read/write is smaller
- The trend is more important
- Retention policy
- Aggregation over time or a set of devices
- Real-time computing or analysis is required
- Traffic is stable

- ❖ Special computing is needed
- ❖ Data volume is huge

5. ***List down the steps in data cleaning.***
    - ❖ Remove duplicate or irrelevant observations
    - ❖ Fix structural errors
    - ❖ Filter unwanted outliers
    - ❖ Handle missing data
    - ❖ Validate and QA

6. ***Write down the benefits of data cleaning.***
    - ❖ Removal of errors when multiple sources of data are at play.
    - ❖ Fewer errors make for happier clients and less-frustrated employees.
    - ❖ Ability to map the different functions and what your data is intended to do.
    - ❖ Monitoring errors and better reporting to see where errors are coming from, making it easier to fix incorrect or corrupt data for future applications.
    - ❖ Using tools for data cleaning will make for more efficient business practices and quicker decision-making.

7. ***State Time-based indexing.***

    A time series is a series of data points indexed in time order. If you index the dataset by date, you can easily carry out a time series analysis. There are three index types of time series:

    DatetimeIndex - The index type for timestamped data is DatetimeIndex.

    PeriodIndex - The index type for period data, which shows the fixed interval date data, is PeriodIndex

    TimedeltaIndex - The index type for the Timedelta data, which shows the time between two dates, is TimedeltaIndex

8. ***What do you mean by Grouping?***

    Group Time Series (GTS) reports contain raw or aggregated data for a group of resources over a particular reporting period. Raw data can be displayed for daily and weekly reporting periods only. Aggregated data can be displayed for any reporting period, but different reporting periods support different granularity values.

9. ***List the featutes of GTS.***

- ❖ Near Real Time (NRT) data points. NRT data is raw data collected during the current hour that has not yet been written to the database.
- ❖ Access to all branches of a group hierarchy. Subelement groups are organized within a tree structure. When a GTS report is deployed against a particular group in a group tree, resources in that group and in groups at all levels of the tree below it are included in the aggregation. If a particular resource appears in multiple groups within the group tree, that resource is included in the aggregation only once.

10. ***What is Resampling.***

While dealing with time-Series data analysis we need to combine data into certain intervals like with each day, a week, or a month. We will solve these using only 2 Pandas APIs i.e. resample() and GroupBy().

The resample() function is used to resample time-series data. Convenience method for frequency conversion and resampling of time series. The object must have a DateTime-like index(DatetimeIndex, PeriodIndex, or TimedeltaIndex), or pass DateTime-like values to the on or level keyword.

PART-B & C

1. Explain three variable contingency table with example.
2. Elaborate the term longitudinal data.
3. Define TSA, Explain broadly.
4. Explain the concept data cleaning and its steps involved.
5. Describe time based indexing.
6. Explain the methods of visualizing time series data.

****

PRACTICAL EXERCISES

1. Install the data Analysis and Visualization tool: R/ Python /Tableau Public/ Power BI.

Program 1:

```
# importing the pands package
import pandas as pd
# creating rows
hafeez = ['Hafeez', 19]
aslan = ['Aslan', 21]
kareem = ['Kareem', 18]
# pass those Series to the DataFrame
# passing columns as well
data_frame = pd.DataFrame([hafeez, aslan, kareem], columns = ['Name', 'Age'])
# displaying the DataFrame
print(data_frame)
```

Output

If you run the above program, you will get the following results.

```
Name Age
0 Hafeez 19
1 Aslan 21
2 Kareem 18
```

Program 2:

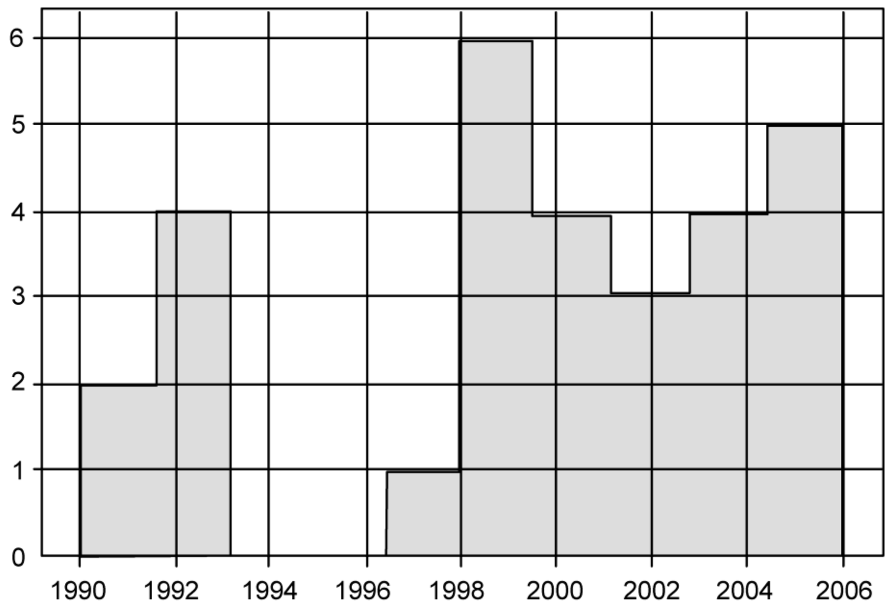
```
# importing the pyplot module to create graphs
import matplotlib.pyplot as plot
# importing the data using pd.read_csv() method
data = pd.read_csv('CountryData.IND.csv')
```

```
# creating a histogram of Time period
data['Time period'].hist(bins = 10)
```

Output

If you run the above program, you will get the following results.

```
<matplotlib.axes._subplots.AxesSubplot at 0x25e363ea8d0>
```



2. **Perform exploratory data analysis (EDA) on with datasets like email data set. Export all your emails as a dataset, import them inside a pandas data frame, visualize them and get different insights from the data.**

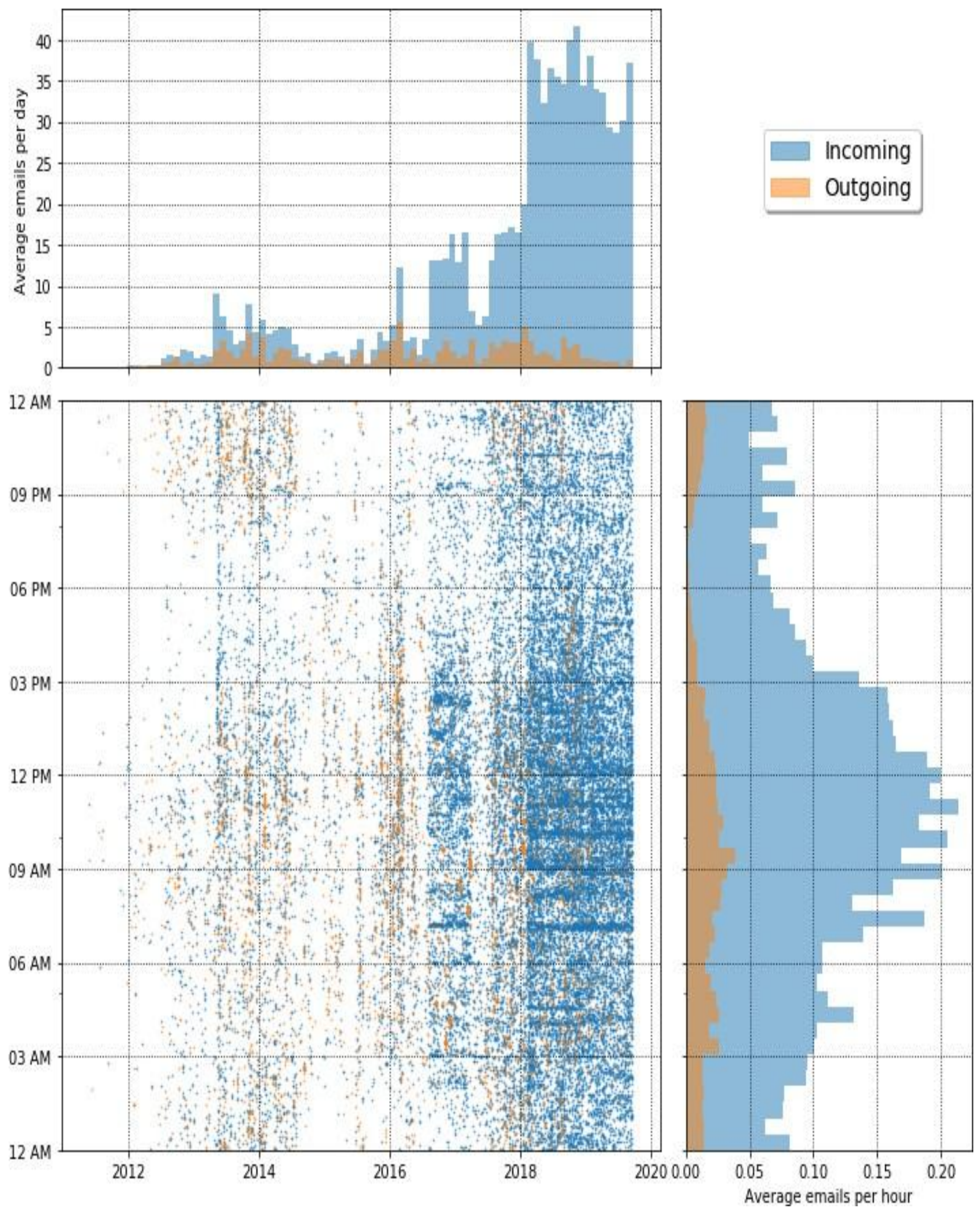
Create a CSV file with only the required attributes:

```
with open('mailbox.csv', 'w') as outputfile:
  writer =csv.writer(outputfile)
    writer.writerow(['subject','from','date','to','label','thread'])
    for message in mbox:
        writer.writerow([
            message['subject'],
            message['from'],
            message['date'],
            message['to'],
            message['X-Gmail-Labels'],
            message['X-GM-THRID']
```

The output of the preceding code is as follows:

```
subject object
from object date
object
to object label
object
thread float64
dtype: object
```

```
def plot_number_perdhour_per_year(df, ax, label=None, dt=1, smooth=False,
                    weight_fun=None, **plot_kwargs):

        tod = df[df['timeofday'].notna()]['timeofday'].values year =
        df[df['year'].notna()]['year'].values
        Ty = year.max() - year.min() T
        = tod.max() - tod.min() bins =
         int(T / dt)

        if weight_fun is None:
            weights = 1 / (np.ones_like(tod) * Ty * 365.25 / dt) else:
            weights = weight_fun(df) if
         smooth:
                hst, xedges = np.histogram(tod, bins=bins, weights=weights); x =
                np.delete(xedges, -1) + 0.5*(xedges[1] - xedges[0])
                hst = ndimage.gaussian_filter(hst, sigma=0.75) f =
                interp1d(x, hst, kind='cubic')
                x = np.linspace(x.min(), x.max(), 10000) hst =
                f(x)

                ax.plot(x, hst, label=label, **plot_kwargs) else:
                ax.hist(tod, bins=bins, weights=weights, label=label,
      **plot_kwargs);
         ax.grid(ls=':', color='k')
         orientation = plot_kwargs.get('orientation')
         if orientation is None or orientation == 'vertical':
             ax.set_xlim(0,                                          24)
             ax.xaxis.set_major_locator(MaxNLocator(8))
             ax.set_xticklabels([datetime.datetime.strptime(str(int(np.mod(ts,    24))),
"%H").strftime("%I %p")
                                          for ts in ax.get_xticks()]); elif
         orientation == 'horizontal':
                ax.set_ylim(0,                            24)
                ax.yaxis.set_major_locator(MaxNLocator(8))

                ax.set_yticklabels([datetime.datetime.strptime(str(int(np.mod(ts, 24))),
   "%H").strftime("%I %p")

                                          for ts in ax.get_yticks()]);
```



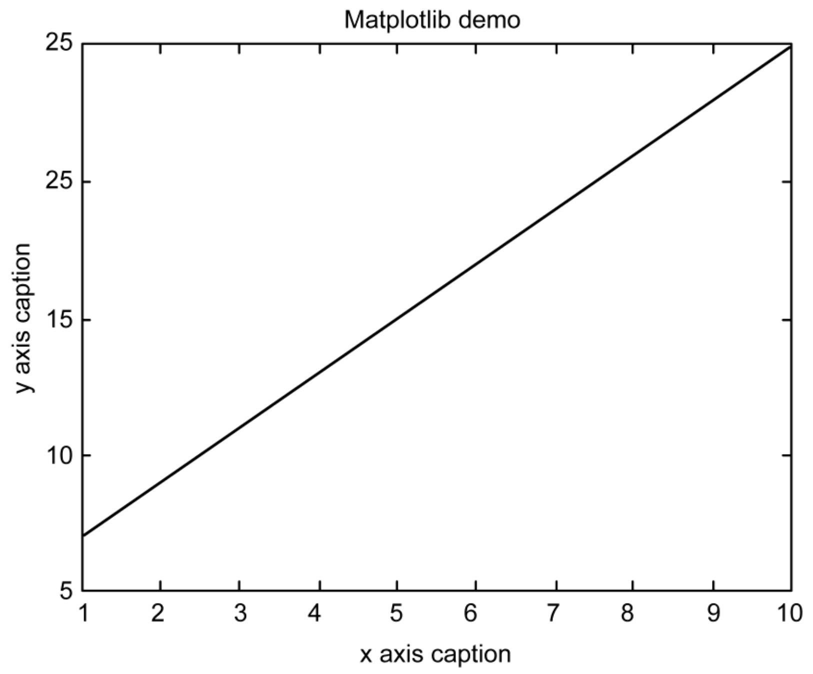
3. Working with Numpy arrays, Pandas data frames, Basic plots using Matplotlib.

Program 1:

```
import numpy as np
from matplotlib import pyplot as plt
```

```
x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y)
plt.show()
```

The above code should produce the following output −

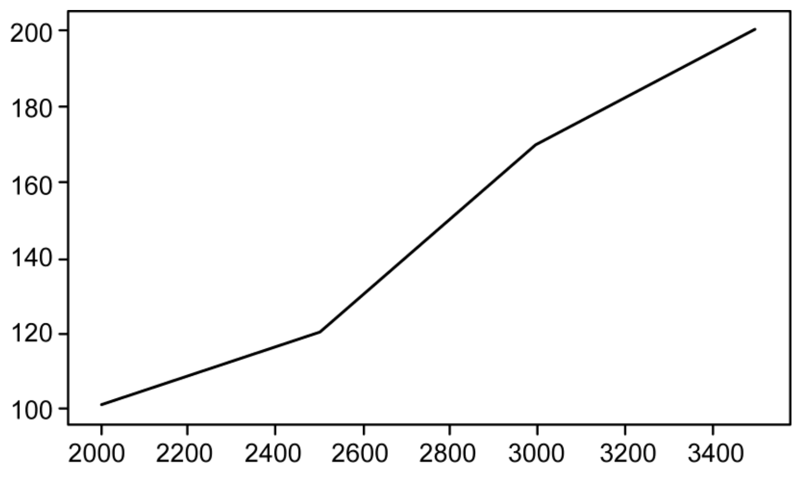


Program 2:

```
import pandas as pd
import matplotlib.pyplot as plt
# creating a DataFrame with 2 columns
dataFrame = pd.DataFrame(
  {
     "Car": ['BMW', 'Lexus', 'Audi', 'Mustang', 'Bentley', 'Jaguar'],
       "Reg_Price": [2000, 2500, 2800, 3000, 3200, 3500],
        "Units": [100, 120, 150, 170, 180, 200]
    }
)


# plot a line graph
plt.plot(dataFrame["Reg_Price"], dataFrame["Units"])
plt.show()
```

Output

This will produce the following output −



4. **Explore various variable and row filters in R for cleaning data. Apply various plot features in R on sample data sets and visualize.**

```
install.packages("data.table")                    # Install data.table package
library("data.table")                          # Load data.table
```

We also create some example data.

```
dt_all <- data.table(x = rep(month.name[1:3],    each = 3),
                y = rep(c(1, 2, 3),        times = 3),
                z = rep(c(TRUE, FALSE, TRUE), each = 3))  # Create data.table
head(dt_all)
```

Table 1

	x	*y*	*z*
1	January	1	TRUE
2	January	2	TRUE
3	January	3	TRUE
4	February	1	FALSE
5	February	2	FALSE
6	February	3	FALSE

Filter Rows by Column Values

In this example, I'll demonstrate how to select all those rows of the example data for which column *x* is equal to February. With the use of *%in%*, we can choose a set of values of *x*. In this example, the set only contains one value.

```
dt_all[x %in% month.name[c(2)], ]          # Rows where x is February
```

Table 2

	x	*y*	*z*
1	February	1	FALSE
2	February	2	FALSE
3	February	3	FALSE

Filter Rows by Column Values

In this example, I'll demonstrate how to select all those rows of the example data for which column *x* is equal to February. With the use of *%in%*, we can choose a set of values of *x*. In this example, the set only contains one value.

```
dt_all[x %in% month.name[c(2)], ]                        # Rows where x is February
```

Table 2

	x	*y*	*z*
1	February	1	FALSE
2	February	2	FALSE
3	February	3	FALSE

Filter Rows by Multiple Column Value

In the previous example, we addressed those rows of the example data for which one column was equal to some value. In this example, we condition on the values of multiple columns.

```
dt_all[x %in% month.name[c(2)] & y == 1, ]  # Rows, where x is February and y is 1
```

Table 3

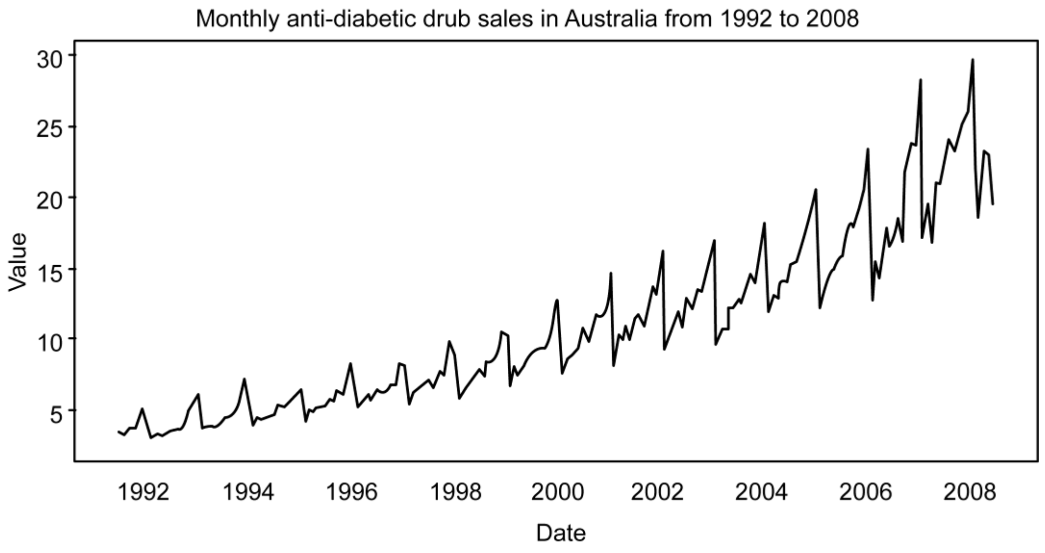
	x	*y*	*z*
1	February	1	FALSE

5. Perform Time Series Analysis and apply the various visualization techniques.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
plt.rcParams.update({'figure.figsize': (10, 7), 'figure.dpi': 120})
# Import as Dataframe
df=pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.csv',
parse_dates=['date'])
df.head()
```

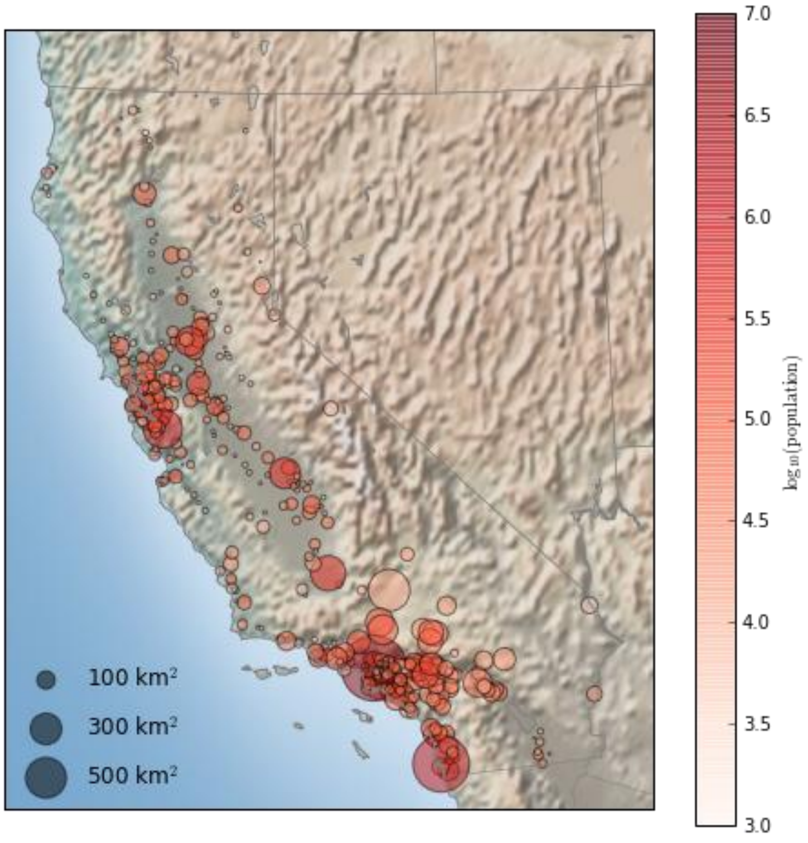
	Date	Value
0	1991-07-01	3.526591
1	1991-08-01	3.180891
2	1991-09-01	3.252221
3	1991-10-01	3.611003
4	1991-11-01	3.565869

```
# Time series data source: fpp pacakge in R.
import matplotlib.pyplot as plt
df=pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/a10.csv',
parse_dates=['date'], index_col='date')
# Draw Plot
def plot_df(df, x, y, title="", xlabel='Date', ylabel='Value', dpi=100):
    plt.figure(figsize=(16,5), dpi=dpi)
    plt.plot(x, y, color='tab:red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()
    plot_df(df, x=df.index, y=df.value, title='Monthly anti-diabetic drug sales in
Australia from 1992 to 2008.')
```



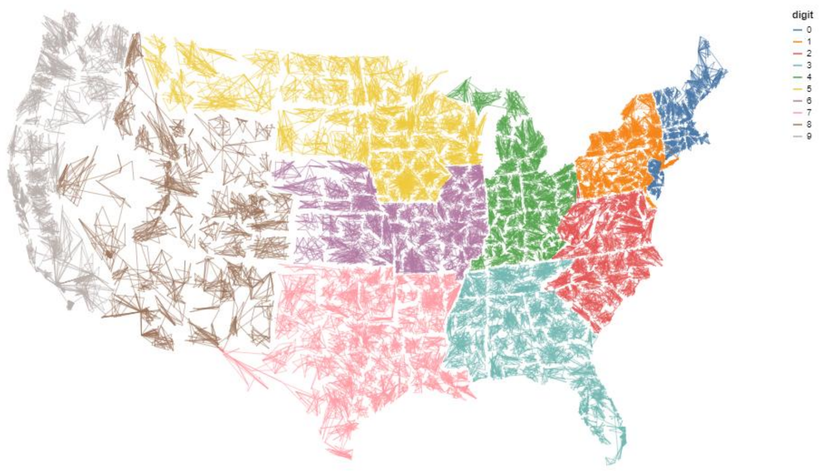
6. Perform Data Analysis and representation on a Map using various Map data sets with Mouse Rollover effect, user interaction, etc.. 22

```
# 1. Draw the map background
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution='h',
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
m.drawstates(color='gray')

# 2. scatter city data, with color reflecting population
# and size reflecting area
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap='Reds', alpha=0.5)

# 3. create colorbar and legend
plt.colorbar(label=r'$\log_{10}({\rm population})$')
plt.clim(3, 7)

# make legend with dummy points
for a in [100, 300, 500]:
    plt.scatter([], [], c='k', alpha=0.5, s=a,
                label=str(a) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc='lower left');
```

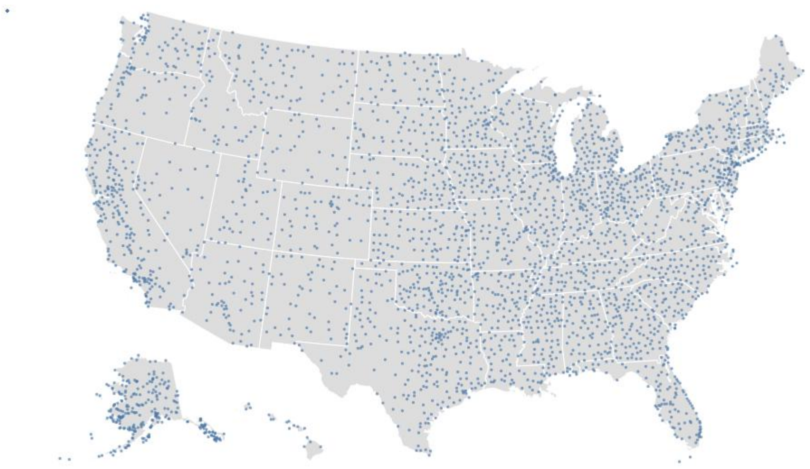


7. **Build cartographic visualization for multiple datasets involving various countries of the world;**

```
alt.Chart(zipcodes).transform_filter(
   '-150 < datum.longitude && 22 < datum.latitude && datum.latitude < 55'
).transform_calculate(
    digit='datum.zip_code[0]'
).mark_line(
    strokeWidth=0.5
).encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    color='digit:N',
    order='zip_code:O'
).project(
    type='albersUsa'
).properties(
    width=900,
    height=500
).configure_view(
    stroke=None
)
```



```
alt.layer(
    alt.Chart(alt.topo_feature(usa, 'states')).mark_geoshape(
        fill='#ddd', stroke='#fff', strokeWidth=1
    ),
    alt.Chart(airports).mark_circle(size=9).encode(
        latitude='latitude:Q',
        longitude='longitude:Q',
        tooltip='iata:N'
    )
).project(
    type='albersUsa'
).properties(
    width=900,
    height=500
).configure_view(
    stroke=None
)
```



8. Perform EDA on Wine Quality Data Set.

```
#importing libraries
import numpy as np
import pandas as pd
importmatplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
In [4]: | 1 | #features in data
        |   | df.columns
```

Out [4]:

```
Index([‘fixed acidity’, volatile acidity’, ‘citric acid’, ‘residual su
 gar’,
       ;chlorides’, ‘free sulfur dioxide’, total sulfur dioxide’, ‘den
 sity’,
       ‘pH’, ‘sulphates’, ‘alcohol’, ‘quality’],
                  dtype=’object’)
```

In [5]:

```
#few  datapoints

df.head( )
```

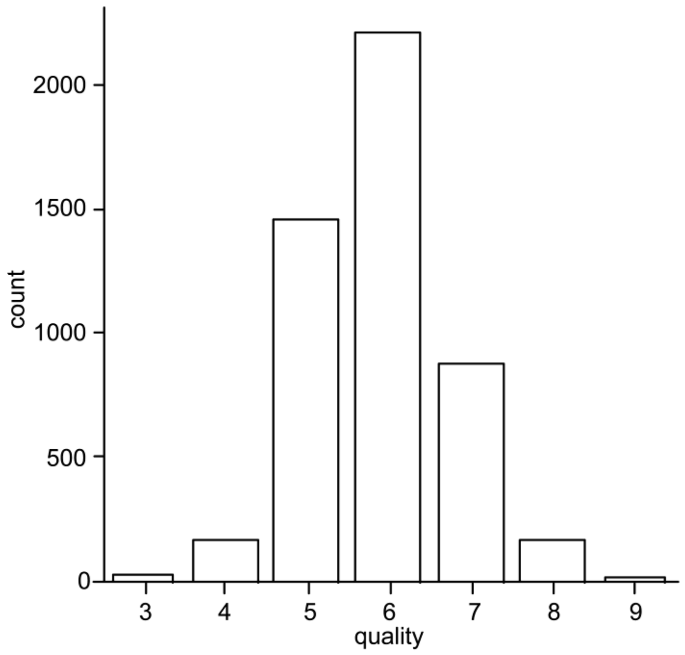
Out [5]:	Fixed acidity	Volatile acidity	citric acid	residual sugar	chlorides	free sulphur dioxide	total sulphur dioxide	density	pH	Sulphates	alcohol	Quality	
	0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
	1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
	2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
	3	7.2	0.23	0.32	8.5	0.068	47.0	186.0	0.9956	3.19	0.40	9.9	6
	4	7.2	0.23	0.32	8.5	0.068	47.0	186.0	0.9956	3.19	0.40	9.9	6

In [13]:

```
    sns.catplot(x=‘quality’,data=df,kind=‘count’)
```

Out [13]:

```
<seaborn.axisgrid.facegrid at0×22b7de0dba8  ??  >
```



9. Use a case study on a data set and apply the various EDA and visualization techniques and present an analysis report.

```
import datetime
import math
import pandas as pd
import random
import radar
from faker import Faker
fake = Faker()

def generateData(n):
    listdata = []
    start = datetime.datetime(2019, 8, 1)
    end = datetime.datetime(2019, 8, 30)
    delta = end - start
    for _ in range(n):
```

date = radar.random_datetime(start='2019-08-1', stop='2019-08-30').strftime("%Y-%m-%d")

price = round(random.uniform(900, 1000), 4)

Date	Price
2019-08-01	999.598900
2019-08-02	957.870150
2019-08-04	978.674200
2019-08-05	963.380375
2019-08-06	978.092900
2019-08-07	987.847700
2019-08-08	952.669900
2019-08-10	973.929400
2019-08-13	971.485600
2019-08-14	977.036200

listdata.append([date, price])

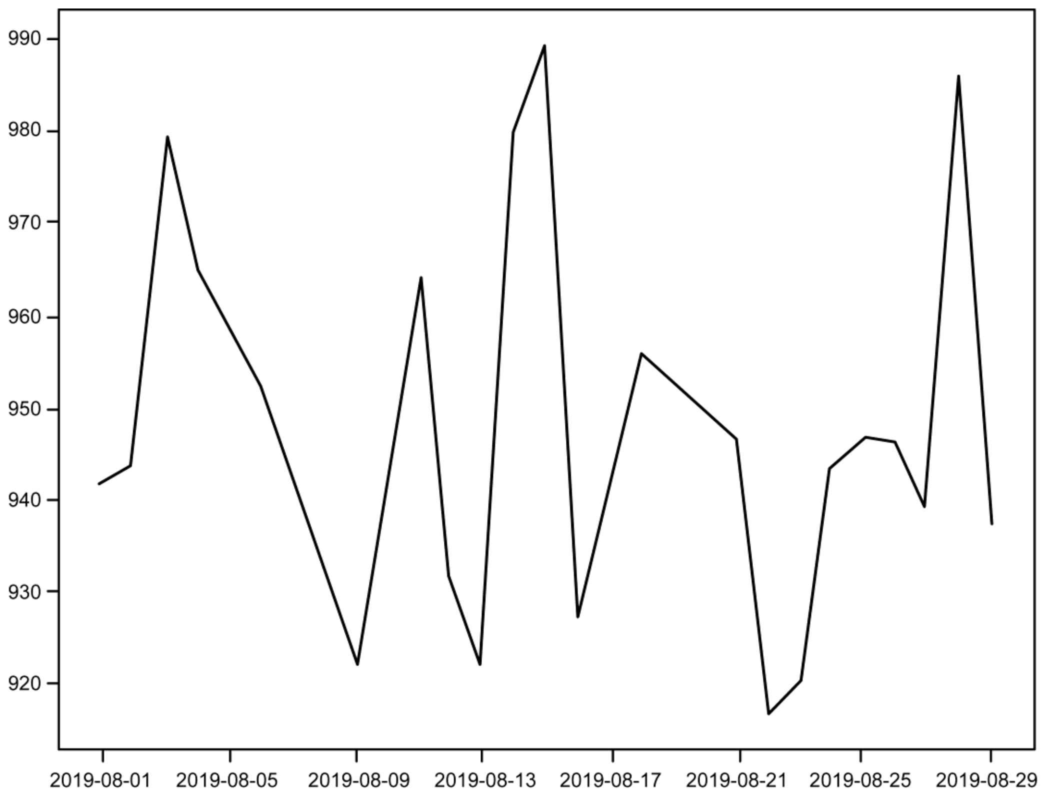
df = pd.DataFrame(listdata, columns = ['Date', 'Price']) df['Date']
= pd.to_datetime(df['Date'], format='%Y-%m-%d') df =
df.groupby(by='Date').mean()

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (14, 10)

plt.plot(df)

And the plotted graph looks something like this:



****