Jaihind Comprehensive Educational Institute`s

JAIHIND COLLEGE OF ENGINEERING

Tal- Junnar, Dist- Pune, Pin Code -410511

DTE Code: EN6609, SPPU Code: CEGP015730

Data Science and Big Data Analytics (310251)

LABORATORY MANUAL

DEPARMTENT OF COMPUTER ENGINEERING

Institute Vision

- To enrich the role of nation building by imparting the qualitative technical education.

Institute Mission

- Impart technical knowledge through prescribed curriculum of university.
- Inculcate ethical and moral values in students for environmental and sustainable development.
- Equip the aspirants through co-curricular and extra- curricular activities to excel in career.

Department Vision

- To transmit the advance knowledge in computer engineering to sustain skills for the emerging trends in society and industry

Department Mission

- To deliver quality technical education to students through creative methods and skilled faculty.
- To create computer engineering experts who serves as a resource at global level.
- To promote the research vein among the students of computer engineering.

Program Educational Objectives (PEOs)

- **PEO1**- Graduates will apply their knowledge and skills to succeed in their careers and/or obtain an advanced degree.
- **PEO2**- Graduates will communicate effectively, demonstrate leadership, and work collaboratively in diverse teams/organizations
- **PEO3**- graduates will demonstrate commitment towards sustainable development for the betterment of society

Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool Usage: Create, Select and apply appropriate techniques, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

- **PSO1-** Graduates design, develop, test and maintain the software system that fulfills the needs of the industry and society.
- **PSO2-** Graduates maintain computer engineering related software and hardware systems.
- **PSO3-** Graduates apply the concept of networking data storage and development methodologies of software systems.

Objectives of DS&BDA Lab

To understand principles of Data Science for the analysis of real time problems

To develop in depth understanding and implementation of the key technologies in Data Science and Big Data Analytics

To analyze and demonstrate knowledge of statistical data analysis techniques for decision- making

To gain practical, hands-on experience with statistics programming languages and

Course Outcomes (COs)

On completion of the course, learner will be able to

CO1: Apply principles of Data Science for the analysis of real time problems

CO2: Implement data representation using statistical methods

CO3: Implement and evaluate data analytics algorithms

CO4: Perform text preprocessing

CO5: Implement data visualization techniques

CO6: Use cutting edge tools and technologies to analyze Big Data

JCEI's JAIHIND COLLEGE ENGINEERING, KURAN

"PRACTICAL LAB MANUAL"

UNDER THE SUBJECT

DATA SCIENCE AND BIG DATA ANALYTICS

(310256)

TE- COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

SEMESTER-VI

Name of Faculty:

TEACHING SCHEME	EXAMINATION SCHEME
Practical: 4Hrs/Week	Practical Assessment: 25 Marks
Lectures: 4Hrs/Week	Term Work: 50 Marks

INDEX

Group A: Data Science (All Compulsory)

Sr. No.	Date	TITLE	Marks	Sign.
1		**Data Wrangling, I** Perform the following operations using Python on any open source dataset (e.g., data.csv) 1. Import all the required Python Libraries. 2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site). 3. Load the Dataset into pandas data frame. 4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame. 5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions. 6. Turn categorical variables into quantitative variables in Python. In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.		
2		**Data Wrangling II** Create an "Academic performance" dataset of students and perform the following operations using Python. 1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them. 2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them. 3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the		

		distribution into a normal distribution. Reason and document your approach properly.		
3		**Descriptive Statistics - Measures of Central Tendency and variability** Perform the following operations on any open source dataset (e.g., data.csv) 1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable. 2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris-versicolor' of iris.csv dataset. Provide the codes with outputs and explain everything that you do in this step.		
4		**Data Analytics I** Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.		
5		**Data Analytics II** 1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.		
6		**Data Analytics III** 1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.		
7		**Text Analytics** 1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.		

		2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.		
8		**Data Visualization I** 1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data. 2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.		
9		**Data Visualization II** 1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age') 2. Write observations on the inference from the above statistics.		
10		**Data Visualization III** Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as: 1. List down the features and their types (e.g., numeric, nominal) available in the dataset. 2. Create a histogram for each feature in the dataset to illustrate the feature distributions. 3. Create a box plot for each feature in the dataset. 4. Compare distributions and identify outliers.		

INDEX

Group B: Big Data Analytics (Any 3)

Sr. No.	Date	TITLE	Sign	Remark
11		Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up.		
12		Design a distributed application using Map-Reduce which processes a log file of a system.		
13		Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.		
14		Write a simple program in SCALA using Apache Spark framework.		

INDEX

Group C: Mini Project (Any 2)

Sr. No.	Date	TITLE	Sign	Remark
1		Write a case study on Global Innovation Network and Analysis (GINA). Components of analytic plan are 1. Discovery business problem framed, 2. Data, 3. Model planning analytic technique and 4. Results and Key findings.		
2		Use the following dataset and classify tweets into positive and negative tweets. https://www.kaggle.com/ruchi798/data-science-Tweets		
3		Develop a movie recommendation model using the scikit-learn library in python. Refer dataset https://github.com/rashida048/Some-NLP-Projects/blob/master/movie_dataset.csv		
4		Use the following covid_vaccine_statewise.csv dataset and perform following analytics on the given dataset https://www.kaggle.com/sudalairajkumar/covid19-in-india?select=covid_vaccine_statewise.csv a. Describe the dataset b. Number of persons state wise vaccinated for first dose in India c. Number of persons state wise vaccinated for second dose in India d. Number of Males vaccinated d. Number of females vaccinated		
5		Write a case study to process data driven for Digital Marketing **OR** Health care systems with Hadoop Ecosystem components as shown. (Mandatory) ● HDFS: Hadoop Distributed File System ● YARN: Yet Another Resource Negotiator ● MapReduce: Programming based Data Processing ● Spark: In-Memory data processing ● PIG, HIVE: Query based processing of data services ● Hbase: NoSQL Database (Provides real-time reads and writes) ● Mahout, Spark MLLib: (Provides analytical tools) Machine Learning algorithm libraries ● Solar, Lucene: Searching and Indexing		

Practical No.1	Group A
Title	**Data Wrangling, I** Perform the following operations using Python on any open source dataset (e.g., data.csv) 1. Import all the required Python Libraries. 2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site). 3. Load the Dataset into pandas data frame. 4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame. 5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions. 6. Turn categorical variables into quantitative variables in Python. In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 1

Title: Data Wrangling, I

- **Problem Statement:**

Perform the following operations using Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.

2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a clear

Description of the data and its source (i.e., URL of the web site).

3. Load the Dataset into pandas data frame.

4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.

6. Turn categorical variables into quantitative variables in Python.

In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

- **Prerequisites:**

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

- **Objectives:**

To learn concepts of Data Wrangling, import /read/scrap meaningful insights from dataset.

- **Theory:**

Data Wrangling:
Data wrangling is the process of cleaning, structuring and enriching raw data into a desired format for better decision making in less time.

Importance Of Data Wrangling
Data Wrangling is a very important step. The below example will explain its importance as :

Books selling Website want to show top-selling books of different domains, according to user preference. For example, a new user search for motivational books, then they want to show those motivational books which sell the most or having a high rating, etc.

But on their website, there are plenty of raw data from different users. Here the concept of Data Munging or Data Wrangling is used. As we know Data is not Wrangled by System. This process is done by Data Scientists. So, the data Scientist will wrangle data in such a way that they will sort that motivational books that are sold more or have high ratings or user buy this book with these package of Books, etc. On the basis of that, the new user will make choice. This will explain the importance of Data wrangling.

Data Wrangling in Python Data Wrangling is a crucial topic for Data Science and Data Analysis. Pandas Framework of Python is used for Data Wrangling. Pandas is an open-source library specifically developed for Data Analysis and Data Science. The process like data sorting or filtration, Data grouping, etc.
Data wrangling in python deals with the below functionalities:

1. **Data exploration:** In this process, the data is studied, analyzed and understood by visualizing representations of data.
2. **Dealing with missing values:** Most of the datasets having a vast amount of data contain missing values of *NaN, they are needed to be taken* care of by replacing them with mean, mode, the most frequent value of the column or simply by dropping the row having a *NaN* value.
3. **Reshaping data:** In this process, data is manipulated according to the requirements, where new data can be added or pre-existing data can be modified.
4. **Filtering data:** Some times datasets are comprised of unwanted rows or columns which are required to be removed or filtered
5. **Other:** After dealing with the raw dataset with the above functionalities we get an efficient dataset as per our requirements and then it can be used for a required purpose like data analyzing, machine learning, data visualization, model training etc.
   Below is an example which implements the above functionalities on a raw dataset:

- **Data exploration**, here we assign the data, and then we visualize the data in a tabular format.

Python3

```
# Import pandas package
```

```
import pandas as pd



# Assign data

data = {'Name': ['Jai', 'Princi', 'Gaurav',

                 'Anuj', 'Ravi', 'Natasha', 'Riya'],

        'Age': [17, 17, 18, 17, 18, 17, 17],

        'Gender': ['M', 'F', 'M', 'M', 'M', 'F', 'F'],

        'Marks': [90, 76, 'NaN', 74, 65, 'NaN', 71]}



# Convert into DataFrame

df = pd.DataFrame(data)



# Display data

df
```

Output:

	Name	Age	Gender	Marks
0	Jai	17	M	90
1	Princi	17	F	76
2	Gaurav	18	M	NaN
3	Anuj	17	M	74
4	Ravi	18	M	65
5	Natasha	17	F	NaN
6	Riya	17	F	71

- **Dealing with missing values**, as we can see from the previous output, there are *NaN* values present in the *MARKS* column which are going to be taken care of by replacing them with the column mean.

Python3

```
# Compute average

c = avg = 0

for ele in df['Marks']:

    if str(ele).isnumeric():

        c += 1

        avg += ele

avg /= c



# Replace missing values

df = df.replace(to_replace="NaN",

                value=avg)



# Display data

df
```

Output:

	Name	Age	Gender	Marks
0	Jai	17	M	90.0
1	Princi	17	F	76.0
2	Gaurav	18	M	75.2
3	Anuj	17	M	74.0
4	Ravi	18	M	65.0
5	Natasha	17	F	75.2
6	Riya	17	F	71.0

- **Reshaping data**, in the *GENDER* column, we can reshape the data by categorizing them into different numbers.

Python3

```
# Categorize gender

df['Gender'] = df['Gender'].map({'M': 0,

                                 'F': 1, }).astype(float)

 # Display data

df
```

Output:

	Name	Age	Gender	Marks
0	Jai	17	0.0	90.0
1	Princi	17	1.0	76.0
2	Gaurav	18	0.0	75.2
3	Anuj	17	0.0	74.0
4	Ravi	18	0.0	65.0
5	Natasha	17	1.0	75.2
6	Riya	17	1.0	71.0

- **Filtering data**, suppose there is a requirement for the details regarding name, gender, marks of the top-scoring students. Here we need to remove some unwanted data.

Python3

```
# Filter top scoring students

df = df[df['Marks'] >=  75]

# Remove age row

df = df.drop(['Age'], axis=1)

 # Display data

df
```

Output:

	Name	Gender	Marks
0	Jai	0.0	90.0
1	Princi	1.0	76.0
2	Gaurav	0.0	75.2
5	Natasha	1.0	75.2

Hence, we have finally obtained an efficient dataset which can be further used for various purposes.

Now that we know the basics of data wrangling. Below we will discuss various operations using which we can perform data wrangling:

Wrangling Data Using Merge Operation:
Merge operation is used to merge raw data and into the desired format.

Syntax:
pd.merge(data_frame1,data_frame2, on="field ")

Here the field is the name of the column which is similar on both data-frame.

For example: Suppose that a Teacher has two types of Data, first type of Data consist of Details of Students and Second type of Data Consist of Pending Fees Status which is taken from Account Office. So The Teacher will use merge operation here in order to merge the data and provide it meaning. So that teacher will analyze it easily and it also reduces time and effort of Teacher from Manual Merging.

FIRST TYPE OF DATA:

Python3

```
# import module

import pandas as pd

 # creating DataFrame for Student Details

details = pd.DataFrame({

   'ID': [101, 102, 103, 104, 105, 106,

         107, 108, 109, 110],

   'NAME': ['Jagroop', 'Praveen', 'Harjot',

          'Pooja', 'Rahul', 'Nikita',

          'Saurabh', 'Ayush', 'Dolly', "Mohit"],

   'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE',

            'CSE', 'CSE', 'CSE', 'CSE', 'CSE']})



# printing details

print(details)
```

Output:

	ID	NAME	BRANCH
0	101	Jagroop	CSE
1	102	Praveen	CSE
2	103	Harjot	CSE
3	104	Pooja	CSE
4	105	Rahul	CSE
5	106	Nikita	CSE
6	107	Saurabh	CSE
7	108	Ayush	CSE
8	109	Dolly	CSE
9	110	Mohit	CSE

SECOND TYPE OF DATA

Python3

```
# Import module

import pandas as pd

# Creating Dataframe for Fees_Status

fees_status = pd.DataFrame(

    {'ID': [101, 102, 103, 104, 105,

            106, 107, 108, 109, 110],

     'PENDING': ['5000', '250', 'NIL',

                 '9000', '15000', 'NIL',

                 '4500', '1800', '250', 'NIL']})

# Printing fees_status

print(fees_status)
```

Output:

	ID	PENDING
0	101	5000
1	102	250
2	103	NIL
3	104	9000
4	105	15000
5	106	NIL
6	107	4500
7	108	1800
8	109	250
9	110	NIL

WRANGLING DATA USING MERGE OPERATION:

Python3

```
# Import module

import pandas as pd

 # Creating Dataframe

details = pd.DataFrame({

   'ID': [101, 102, 103, 104, 105,

         106, 107, 108, 109, 110],

   'NAME': ['Jagroop', 'Praveen', 'Harjot',

          'Pooja', 'Rahul', 'Nikita',

          'Saurabh', 'Ayush', 'Dolly', "Mohit"],

   'BRANCH': ['CSE', 'CSE', 'CSE', 'CSE', 'CSE',

           'CSE', 'CSE', 'CSE', 'CSE', 'CSE']})

 # Creating Dataframe

fees_status = pd.DataFrame(

   {'ID': [101, 102, 103, 104, 105,
```

```
        106, 107, 108, 109, 110],

    'PENDING': ['5000', '250', 'NIL',

            '9000', '15000', 'NIL',

            '4500', '1800', '250', 'NIL']})

# Merging Dataframe

print(pd.merge(details, fees_status, on='ID'))
```

Output:

	ID	NAME	BRANCH	PENDING
0	101	Jagroop	CSE	5000
1	102	Praveen	CSE	250
2	103	Harjot	CSE	NIL
3	104	Pooja	CSE	9000
4	105	Rahul	CSE	15000
5	106	Nikita	CSE	NIL
6	107	Saurabh	CSE	4500
7	108	Ayush	CSE	1800
8	109	Dolly	CSE	250
9	110	Mohit	CSE	NIL

Wrangling Data using Grouping Method

The grouping method in Data analysis is used to provide results in terms of various groups taken out from Large Data. This method of pandas is used to group the outset of data from the large data set.

Example: There is a Car Selling company and this company have different Brands of various Car Manufacturing Company like Maruti, Toyota, Mahindra, Ford, etc. and have data where different cars are sold in different years. So the Company wants to wrangle only that data where cars are sold during the year 2010. For this problem, we use another Wrangling technique that is *groupby()* method.

CARS SELLING DATA:

Python3

```
# Import module

import pandas as pd

# Creating Data

car_selling_data = {'Brand': ['Maruti', 'Maruti', 'Maruti',
```

```
                    'Maruti', 'Hyundai', 'Hyundai',

                    'Toyota', 'Mahindra', 'Mahindra',

                    'Ford', 'Toyota', 'Ford'],

                'Year':  [2010, 2011, 2009, 2013,

                    2010, 2011, 2011, 2010,

                    2013, 2010, 2010, 2011],

                'Sold': [6, 7, 9, 8, 3, 5,

                    2, 8, 7, 2, 4, 2]}

 # Creating Dataframe of car_selling_data

df = pd.DataFrame(car_selling_data)

 # printing Dataframe

print(df)
```

Output:

	Brand	Year	Sold
0	Maruti	2010	6
1	Maruti	2011	7
2	Maruti	2009	9
3	Maruti	2013	8
4	Hyundai	2010	3
5	Hyundai	2011	5
6	Toyota	2011	2
7	Mahindra	2010	8
8	Mahindra	2013	7
9	Ford	2010	2
10	Toyota	2010	4
11	Ford	2011	2

DATA OF THE YEAR 2010:

Python3

```
# Import module
```

```
import pandas as pd

# Creating Data

car_selling_data = {'Brand': ['Maruti', 'Maruti', 'Maruti',

                    'Maruti', 'Hyundai', 'Hyundai',

                    'Toyota', 'Mahindra', 'Mahindra',

                    'Ford', 'Toyota', 'Ford'],

                'Year':  [2010, 2011, 2009, 2013,

                    2010, 2011, 2011, 2010,

                    2013, 2010, 2010, 2011],

                'Sold': [6, 7, 9, 8, 3, 5,

                    2, 8, 7, 2, 4, 2]}

 # Creating Dataframe for Provided Data

df = pd.DataFrame(car_selling_data)

 # Group the data when year = 2010

grouped = df.groupby('Year')

print(grouped.get_group(2010))
```

Output:

	Brand	Year	Sold
0	Maruti	2010	6
4	Hyundai	2010	3
7	Mahindra	2010	8
9	Ford	2010	2
10	Toyota	2010	4

Wrangling data by removing Duplication:

Pandas *duplicates()* method helps us to remove duplicate values from Large Data. An important part of Data Wrangling is removing Duplicate values from the large data set.

Syntax:

DataFrame.duplicated(subset=None, keep='first')

Here subset is the column value where we want to remove Duplicate value.

In *keep*, we have 3 options :

- if *keep = 'first'* then the first value is marked as original rest all values if occur will be removed as it is considered as duplicate.
- if *keep= 'last'* then the last value is marked as original rest all above same values will be removed as it is considered as duplicate values.
- if *keep = 'false'* the all the values which occur more than once will be removed as all considered as a duplicate value.

For example, A University will organize the event. In order to participate Students have to fill their details in the online form so that they will contact them. It may be possible that a student will fill the form multiple time. It may cause difficulty for the event organizer if a single student will fill multiple entries. The Data that the organizers will get can be Easily Wrangles by removing duplicate values.

DETAILS STUDENTS DATA WHO WANT TO PARTICIPATE IN THE EVENT:

Python3

```
# Import module

import pandas as pd

# Initializing Data

student_data = {'Name': ['Amit', 'Praveen', 'Jagroop',

                         'Rahul', 'Vishal', 'Suraj',

                         'Rishab', 'Satyapal', 'Amit',

                         'Rahul', 'Praveen', 'Amit'],

                'Roll_no': [23, 54, 29, 36, 59, 38,

                            12, 45, 34, 36, 54, 23],

                'Email': ['xxxx@gmail.com', 'xxxxxx@gmail.com',

                          'xxxxxx@gmail.com', 'xx@gmail.com',

                          'xxxx@gmail.com', 'xxxxx@gmail.com',

                          'xxxxx@gmail.com', 'xxxxx@gmail.com',

                          'xxxxx@gmail.com', 'xxxxxx@gmail.com',
```

```
                'xxxxxxxxxx@gmail.com', 'xxxxxxxxxx@gmail.com']}

 # Creating Dataframe of Data

df = pd.DataFrame(student_data)

 # Printing Dataframe

print(df)
```

Output:

	Name	Roll_no	Email
0	Amit	23	xxxx@gmail.com
1	Praveen	54	xxxxxx@gmail.com
2	Jagroop	29	xxxxxx@gmail.com
3	Rahul	36	xx@gmail.com
4	Vishal	59	xxxx@gmail.com
5	Suraj	38	xxxxx@gmail.com
6	Rishab	12	xxxxx@gmail.com
7	Satyapal	45	xxxxx@gmail.com
8	Amit	34	xxxxx@gmail.com
9	Rahul	36	xxxxxx@gmail.com
10	Praveen	54	xxxxxxxxxx@gmail.com
11	Amit	23	xxxxxxxxxx@gmail.com

DATA WRANGLED BY REMOVING DUPLICATE ENTRIES:

Python3

```
# import module

import pandas as pd

# initializing Data

student_data = {'Name': ['Amit', 'Praveen', 'Jagroop',

                'Rahul', 'Vishal', 'Suraj',

                'Rishab', 'Satyapal', 'Amit',

                'Rahul', 'Praveen', 'Amit'],

           'Roll_no': [23, 54, 29, 36, 59, 38,
```

```
                    12, 45, 34, 36, 54, 23],

            'Email': ['xxxx@gmail.com', 'xxxxxx@gmail.com',

                    'xxxxxx@gmail.com', 'xx@gmail.com',

                    'xxxx@gmail.com', 'xxxxx@gmail.com',

                    'xxxxx@gmail.com', 'xxxxx@gmail.com',

                    'xxxxx@gmail.com', 'xxxxxx@gmail.com',

                    'xxxxxxxxxx@gmail.com', 'xxxxxxxxxx@gmail.com']}

 # creating dataframe

df = pd.DataFrame(student_data)

 # Here df.duplicated() list duplicate  Entries in ROllno.

# So that ~(NOT) is placed in order to get non duplicate values.

non_duplicate = df[~df.duplicated('Roll_no')]

 # printing non-duplicate values

print(non_duplicate)
```

Output:

Practical No.2	Group A
Title	**Data Wrangling II** Create an "Academic performance" dataset of students and perform the following operations using Python. 1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them. 2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them. 3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution. Reason and document your approach properly.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 2

Title: Data Wrangling, II

- ## Problem Statement:

Data Wrangling II

Create an "Academic performance" dataset of students and perform the following operations using Python.

- 1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.
- 2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.
- 3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to decrease the skewness and convert the distribution into a normal distribution. Reason and document your approach properly.

- ## Prerequisites:

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software**

Requirements: Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware**

Requirements: Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:**

64 Bit Windows/Linux/Mac OS

Programming Language: Python 3.8, PIP

Datasets: Any Open Source Dataset/CSV Files

Contents for Theory:

1. **Creation of Dataset using Microsoft Excel.**
2. **Identification and Handling of Null Values**
3. **Identification and Handling of Outliers**
4. **Data Transformation for the purpose of :**
    a. **To change the scale for better understanding**
    b. **To decrease the skewness and convert distribution into normal distribution**

Theory:

1. **Creation of Dataset using Microsoft Excel.**

   The dataset is created in "CSV" format.

   - The name of dataset is **StudentsPerformance**
   - **The features of the dataset are:** Math_Score, Reading_Score, Writing_Score, Placement_Score, Club_Join_Date .
   - **Number of Instance**s: 30
   - **The response variable is**: Placement_Offer_Count .
   - **Range of Values:**
     Math_Score [-80], Reading_Score[75-,95], ,Writing_Score [,80], Placement_Score[75-100], Club_Join_Date [2018-2021].
   - **The response variable is** the number of placement offers facilitated to particular students, which is largely depend on Placement_Score

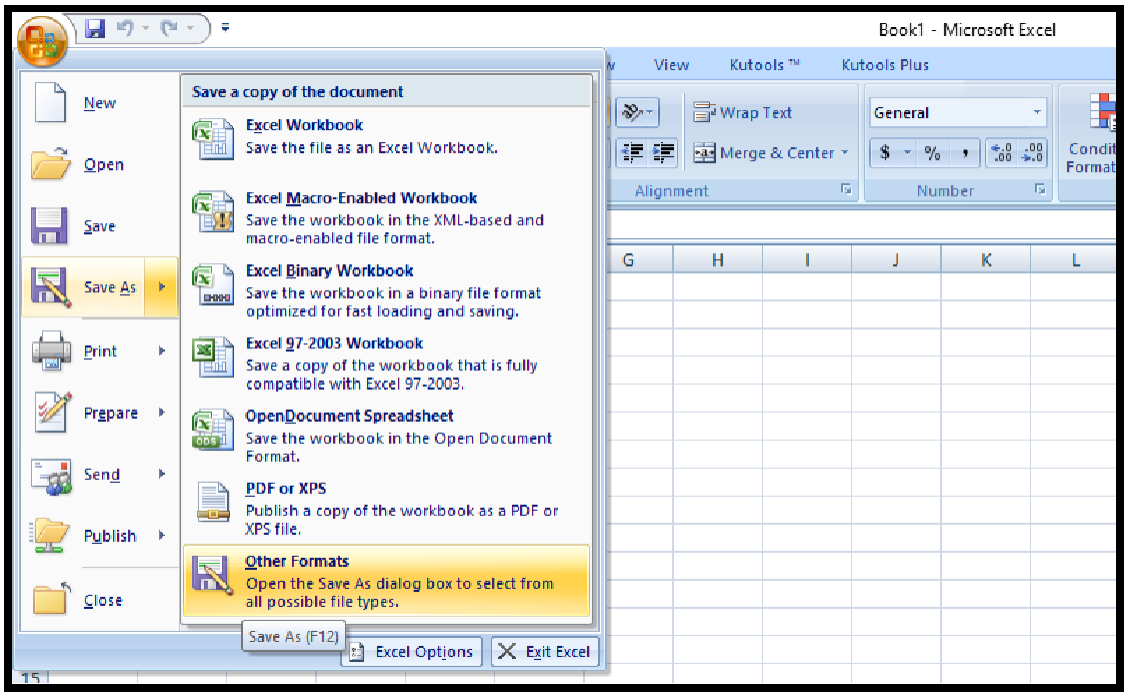
   To fill the values in the dataset the **RANDBETWEEN** is used. Returns a random integer number between the numbers you specify

   Syntax : RANDBETWEEN(bottom, top) Bottom The smallest integer and
   Top The largest integer RANDBETWEEN will return.

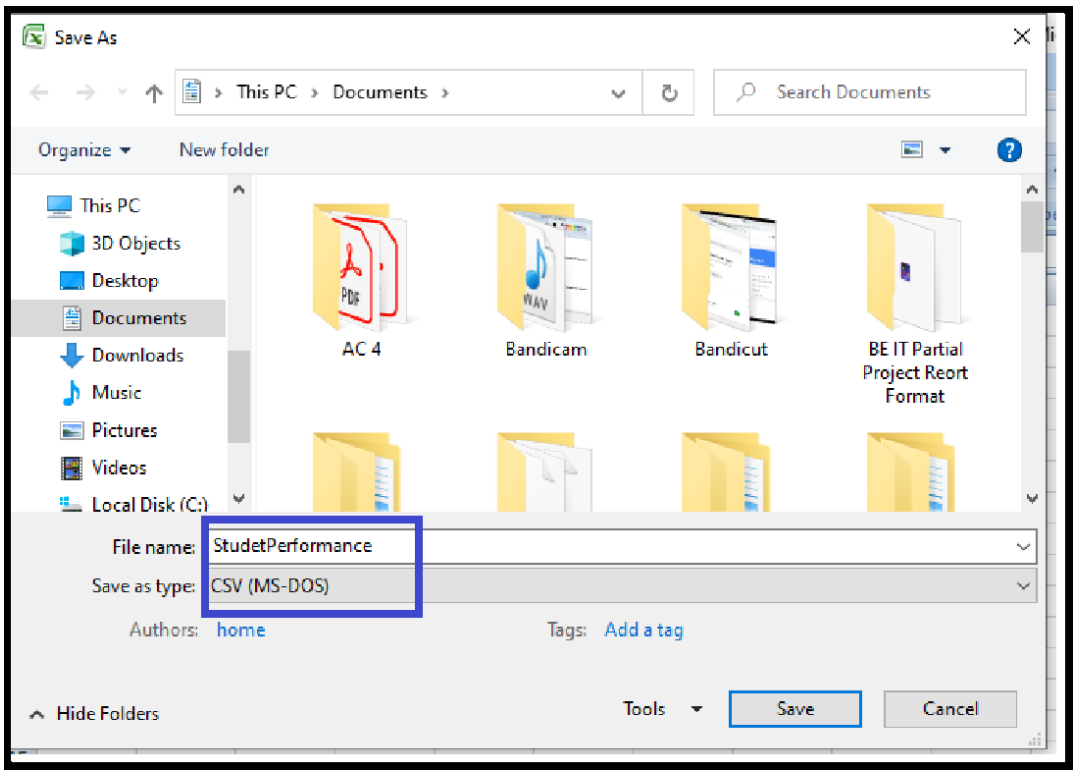
For better understanding and visualization, 20% impurities are added into each variable to the dataset.

The step to create the dataset are as follows:

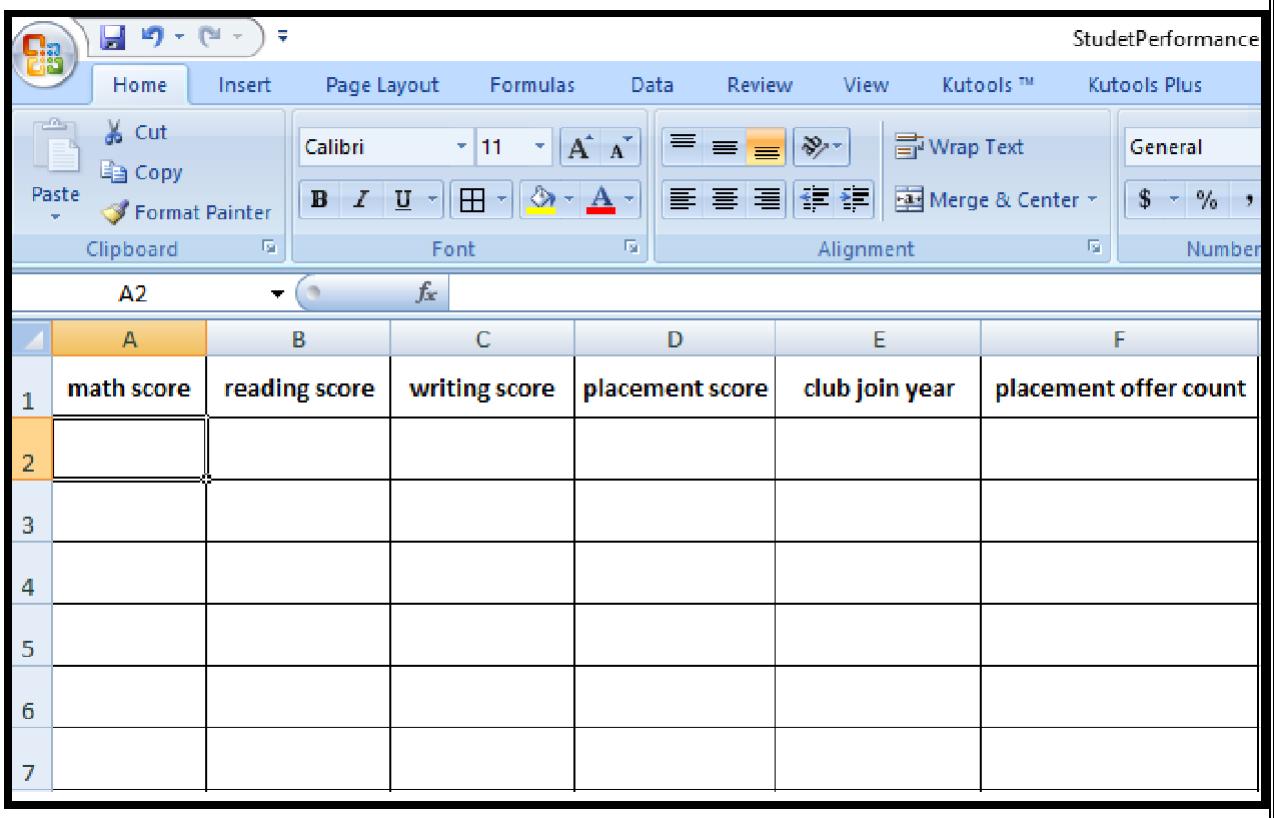
Step 1: Open Microsoft Excel and click on Save As. Select Other .Formats



Step 2: Enter the name of the dataset and Save the dataset astye CSV(MS-DOS).

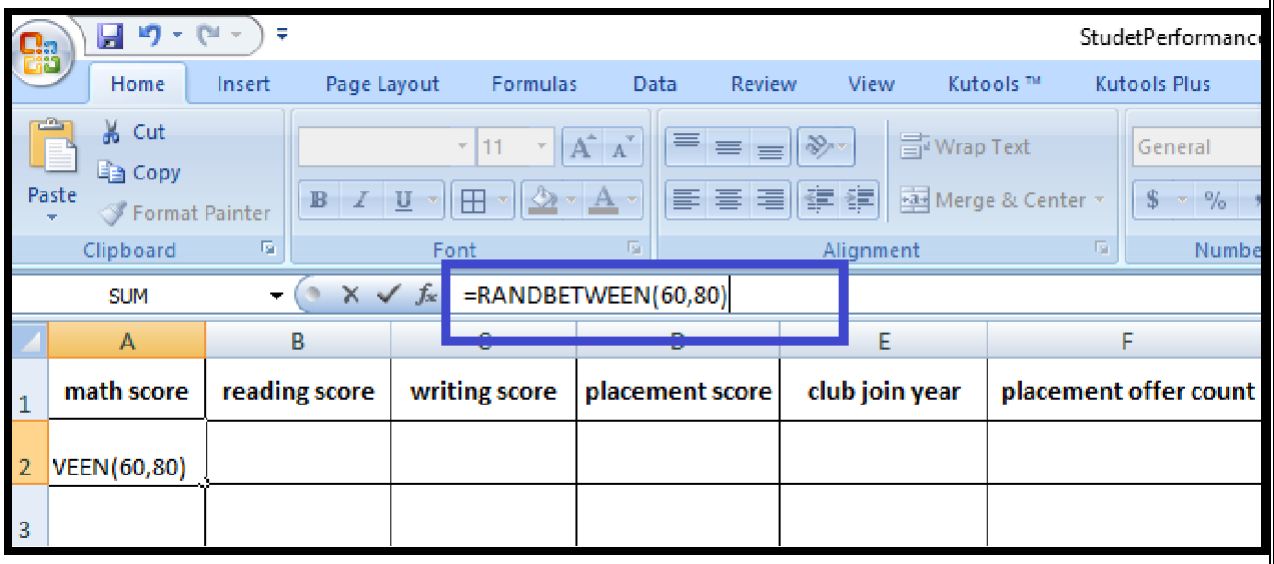


Step 3: Enter the name of features as column header.



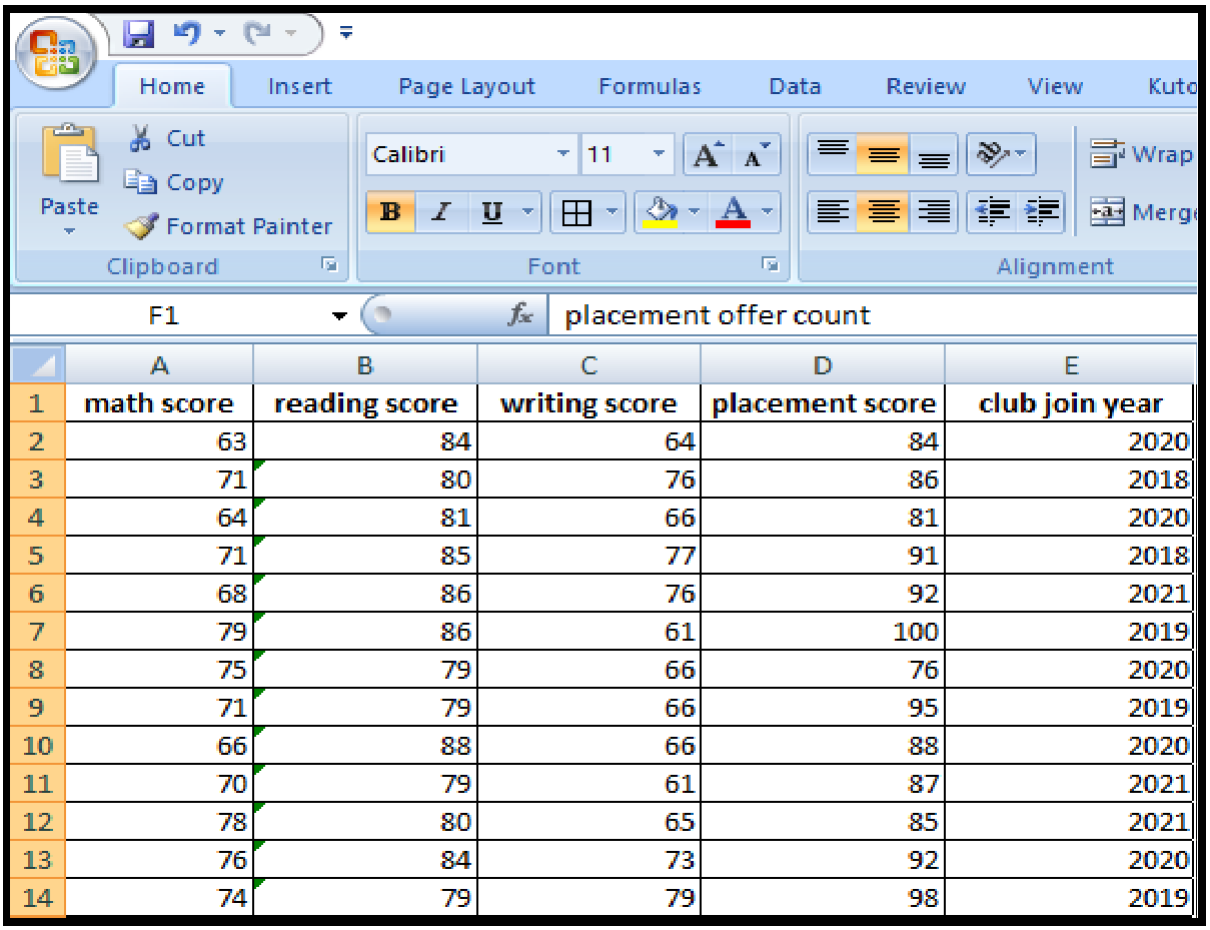
Step 3: Fill the dara by using **RANDOMBETWEEN** function. For every feature , fill the data by considering above spectified range.

one example is given:



Scroll down the cursor for 30 rows to create 30 instances.

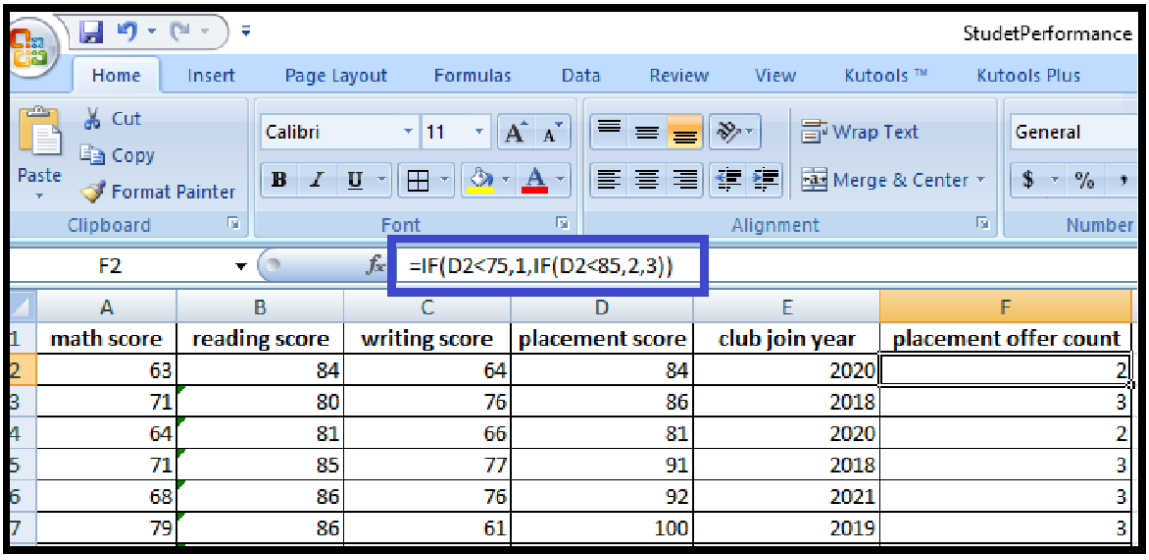
Repeat this for the features, Reading_Score, Writing_Score, Placement_Score, Club_Join_Date.



F1: placement offer count

	A	B	C	D	E
1	math score	reading score	writing score	placement score	club join year
2	63	84	64	84	2020
3	71	80	76	86	2018
4	64	81	66	81	2020
5	71	85	77	91	2018
6	68	86	76	92	2021
7	79	86	61	100	2019
8	75	79	66	76	2020
9	71	79	66	95	2019
10	66	88	66	88	2020
11	70	79	61	87	2021
12	78	80	65	85	2021
13	76	84	73	92	2020
14	74	79	79	98	2019

The placement count largely depends on the placement score. It is considered that if placement score <75, 1 offer is facilitated; for placement score >75 , 2 offer is facilitated and for else (>85) 3 offer is facilitated. Nested If formula is used for ease of data filling.



F2: =IF(D2<75,1,IF(D2<85,2,3))

	A	B	C	D	E	F
1	math score	reading score	writing score	placement score	club join year	placement offer count
2	63	84	64	84	2020	2
3	71	80	76	86	2018	3
4	64	81	66	81	2020	2
5	71	85	77	91	2018	3
6	68	86	76	92	2021	3
7	79	86	61	100	2019	3

Step 4: In 20% data, fill the impurities. The range of math score is [,80], updating a few instances values below or above 80. Repeat this for Writing_Score [,80], Placement_Score[75-100], Club_Join_Date [2018-2021].

	A	B	C	D	E
1	math score	reading score	writing score	placement score	club join year
2	68	94	64	90	2018
3	72	85	70	86	2018
4	94	90	64	91	2020

Step 5: To violate the ruleof response variable, update few valus . If placement scoreis greater then 85, facilated only 1 offer.

	A	B	C	D	E	F
1	math score	reading score	writing score	placement score	club join year	placement offer count
2	70	91	64	87	2019	3
3	77	75	67	81	2020	2
4	94	84	73	99	2019	3
5	78	84	77	96	2020	1

The dataset is created with the given description.

2. Identification and Handling of Null Values

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.

In Pandas missing data is represented by two value:

1. **None**: None is a Python singleton object that is often used for missing data in Python code.
2. **NaN** : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation.

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()
- dropna()
- fillna()
- replace()

1. Checking for missing values using isnull() and notnull()

- **Checking for missing values using isnull()**

In order to check null values in Pandas DataFrame, isnull() function is used. This function return dataframe of Boolean values which are True for NaN values.

Algorithm:

Step 1 : Import pandas and numpy in order to check missing values in Pandas DataFrame

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the data frame

```
df
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
1	female	69	90	88.0	NaN	2	na
2	female	90	95	93.0	74.0	2	Nashik
3	male	47	57	NaN	78.0	1	Na
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik
7	male	NaN	65	67.0	49.0	1	Pune
8	male	5	77	89.0	55.0	0	NaN

Step 4: Use isnull() function to check null values in the dataset.

```
df.isnull()
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	False	False	False	False	False	False	False
1	False	False	False	False	True	False	False
2	False	False	False	False	False	False	False
3	False	False	False	True	False	False	False
4	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False
7	False	True	False	False	False	False	False
8	False	False	False	False	False	False	True

Step 5: To create a series true for NaN values for specific columns. for example math score in dataset and display data with only math score as NaN

```
series = pd.isnull(df["math score"])
df[series]
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
7	male	NaN	65	67.0	49.0	1	Pune

- **Checking for missing values using notnull()**

In order to check null values in Pandas Dataframe, notnull() function is used. This function return dataframe of Boolean values which are False for NaN values.

Algorithm:

Step 1 : Import pandas and numpy in order to check missing values in Pandas DataFrame

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the data frame

```
df
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
1	female	69	90	88.0	NaN	2	na
2	female	90	95	93.0	74.0	2	Nashik
3	male	47	57	NaN	78.0	1	Na
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik
7	male	NaN	65	67.0	49.0	1	Pune
8	male	5	77	89.0	55.0	0	NaN

Step 4: Use notnull() function to check null values in the dataset.

```
df.notnull()
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	True	True	True	True	True	True	True
1	True	True	True	True	False	True	True
2	True	True	True	True	True	True	True
3	True	True	True	False	True	True	True
4	True	True	True	True	True	True	True
5	True	True	True	True	True	True	True
6	True	True	True	True	True	True	True
7	True	False	True	True	True	True	True
8	True	True	True	True	True	True	False

Step 5: To create a series true for NaN values for specific columns. for example math score in dataset and display data with only math score as NaN

```
series1 = pd.notnull(df["math score"])
df[series1]
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
1	female	69	90	88.0	NaN	2	na
2	female	90	95	93.0	74.0	2	Nashik
3	male	47	57	NaN	78.0	1	Na
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik
8	male	5	77	89.0	55.0	0	NaN

See that there are also categorical values in the dataset, for this, you need to use Label Encoding or One Hot Encoding.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])
newdf=df
df
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	0	72	72	74.0	78.0	1	Pune
1	0	69	90	88.0	NaN	2	na
2	0	90	95	93.0	74.0	2	Nashik
3	1	47	57	NaN	78.0	1	Na
4	1	na	78	75.0	81.0	3	Pune
5	0	71	Na	78.0	70.0	4	na
6	1	12	44	52.0	12.0	2	Nashik
7	1	NaN	65	67.0	49.0	1	Pune
8	1	5	77	89.0	55.0	0	NaN

2. **Filling missing values using dropna(), fillna(), replace()**

In order to fill null values in a datasets, fillna(), replace() functions are used. These functions replace NaN values with some value of their own. All these functions help in filling null values in datasets of a DataFrame.

- **For replacing null values with NaN**

```
missing_values = ["Na", "na"]
```

```
df = pd.read_csv("StudentsPerformanceTest1.csv", na_values = missing_values)
df
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72.0	72.0	74.0	78.0	1	Pune
1	female	69.0	90.0	88.0	NaN	2	NaN
2	female	90.0	95.0	93.0	74.0	2	Nashik
3	male	47.0	57.0	NaN	78.0	1	NaN
4	male	NaN	78.0	75.0	81.0	3	Pune
5	female	71.0	NaN	78.0	70.0	4	NaN
6	male	12.0	44.0	52.0	12.0	2	Nashik
7	male	NaN	65.0	67.0	49.0	1	Pune
8	male	5.0	77.0	89.0	55.0	0	NaN

- **Filling null values with a single value**

Step 1 : Import pandas and numpy in order to check missing values in Pandas DataFrame

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the data frame

```
df
```

Step 4: filling missing value using fillna()

```
ndf=df
ndf.fillna(0)
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
1	female	69	90	88.0	0.0	2	na
2	female	90	95	93.0	74.0	2	Nashik
3	male	47	57	0.0	78.0	1	Na
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik
7	male	0	65	67.0	49.0	1	Pune
8	male	5	77	89.0	55.0	0	0

Step 5: filling missing values using mean, median and standard deviation of that column.

```
data['math score'] = data['math score'].fillna(data['math score'].mean())

data["math score"] = data["math score"].fillna(data["math score"].median())

data['math score"] = data["math score"].fillna(data["math score"].std())
```

replacing missing values in forenoon column with minimum/maximum number of that column

```
data["math score"] = data["math score"].fillna(data["math score"].min())

data["math score"] = data["math score"].fillna(data["math score"].max())
```

- **Filling null values in dataset**
  To fill null values in dataset use inplace=true

```
m_v=df['math score'].mean()
df['math score'].fillna(value=m_v, inplace=True)
df
```

	gender	math score	reading score	writing score	Placement Score	placement offer count
0	female	72.000	72	74	78	1
1	female	69.000	90	88	70	2
2	female	90.000	95	93	74	2
3	male	47.000	57	44	78	1
4	male	11.000	78	75	81	3
5	female	71.000	83	78	70	4
6	male	12.000	44	52	12	2
7	male	47.125	65	67	49	1
8	male	5.000	77	89	55	0

- **Filling a null values using replace() method**

  Following line will replace Nan value in dataframe with value -99

```
ndf.replace(to_replace = np.nan, value = -99)
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
1	female	69	90	88.0	-99.0	2	na
2	female	90	95	93.0	74.0	2	Nashik
3	male	47	57	-99.0	78.0	1	Na
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik
7	male	-99	65	67.0	49.0	1	Pune
8	male	5	77	89.0	55.0	0	-99

- **Deleting null values using dropna() method**
  In order to drop null values from a dataframe, dropna() function is used. This function drops Rows/Columns of datasets with Null values in different ways.
  1. Dropping rows with at least 1 null value
  2. Dropping rows if all values in that row are missing
  3. Dropping columns with at least 1 null value.
  4. Dropping Rows with at least 1 null value in CSV file

Algorithm:

Step 1 : Import pandas and numpy in order to check missing values in Pandas DataFrame

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/StudentsPerformanceTest1.csv")
```

Step 3: Display the data frame

```
df
```

Step 4 : **To** drop rows with at least 1 null value

```
ndf.dropna()
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
2	female	90	95	93.0	74.0	2	Nashik
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik

Step 5: To Drop rows if all values in that row are missing

```
ndf.dropna(how = 'all')
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
1	female	69	90	88.0	NaN	2	na
2	female	90	95	93.0	74.0	2	Nashik
3	male	47	57	NaN	78.0	1	Na
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik
7	male	NaN	65	67.0	49.0	1	Pune
8	male	5	77	89.0	55.0	0	NaN

Step 6: To Drop columns with at least 1 null value.

```
ndf.dropna(axis = 1)
```

	gender	reading score	placement offer count
0	female	72	1
1	female	90	2
2	female	95	2
3	male	57	1
4	male	78	3
5	female	Na	4
6	male	44	2
7	male	65	1
8	male	77	0

Step 7 : To drop rows with at least 1 null value in CSV file.

making new data frame with dropped NA values

```
new_data = ndf.dropna(axis = 0, how ='any')
```

```
new_data
```

	gender	math score	reading score	writing score	Placement Score	placement offer count	Region
0	female	72	72	74.0	78.0	1	Pune
2	female	90	95	93.0	74.0	2	Nashik
4	male	na	78	75.0	81.0	3	Pune
5	female	71	Na	78.0	70.0	4	na
6	male	12	44	52.0	12.0	2	Nashik

3. Identification and Handling of Outliers

3.1 Identification of Outliers

One of the most important steps as part of data preprocessing is detecting and treating the outliers as they can negatively affect the statistical analysis and the training process of a machine learning algorithm resulting in lower accuracy.

1. What are Outliers?

We all have heard of the idiom ‘odd one out' which means something unusual in comparison to the others in a group.

Similarly, an Outlier is an observation in a given dataset that lies far from the rest of the observations. That means an outlier is vastly larger or smaller than the remaining values in the set.

2. Why do they occur?

An outlier may occur due to the variability in the data, or due to experimental error/human error.

They may indicate an experimental error or heavy skewness in the data(heavy-tailed distribution).

3. What do they affect?

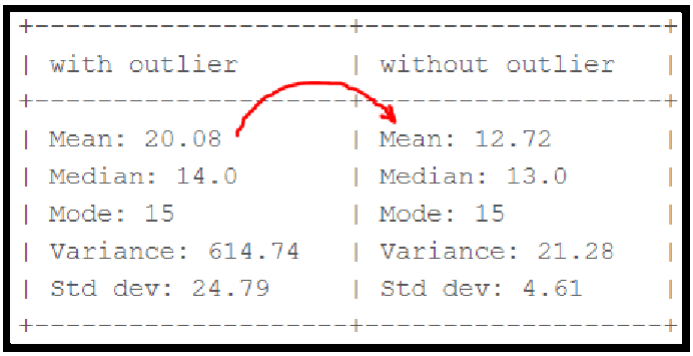
In statistics, we have three measures of central tendency namely Mean, Median, and Mode. They help us describe the data.

Mean is the accurate measure to describe the data when we do not have any outliers present. Median is used if there is an outlier in the dataset. Mode is used if there is an outlier AND about ½ or more of the data is the same.

‘Mean’ is the only measure of central tendency that is affected by the outliers which in turn impacts Standard deviation.

Example:

Consider a small dataset, sample= [15, 101, 18, 7, 13, 16, 11, 21, 5, 15, 10, 9]. By looking at it, one can quickly say '101' is an outlier that is much larger than the other values.



with outlier	without outlier
Mean: 20.08	Mean: 12.72
Median: 14.0	Median: 13.0
Mode: 15	Mode: 15
Variance: 614.74	Variance: 21.28
Std dev: 24.79	Std dev: 4.61

fig. Computation with and without outlier

From the above calculations, we can clearly say the Mean is more affected than the Median.

4. Detecting Outliers

If our dataset is small, we can detect the outlier by just looking at the dataset. But what if we have a huge dataset, how do we identify the outliers then? We need to use visualization and mathematical techniques.

Below are some of the techniques of detecting outliers

- Boxplots
- Scatterplots
- Z-score
- Inter Quantile Range(IQR)

4.1 Detecting outliers using Boxplot:

It captures the summary of the data effectively and efficiently with only a simple box and whiskers. Boxplot summarizes sample data using 25th, 50th, and 75th percentiles. One can just get insights(quartiles, median, and outliers) into the dataset by just looking at its boxplot.

Algorithm:

Step 1 : Import pandas and numpy libraries

```
import pandas as pd
```

```
import numpy as np
```

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/demo.csv")
```

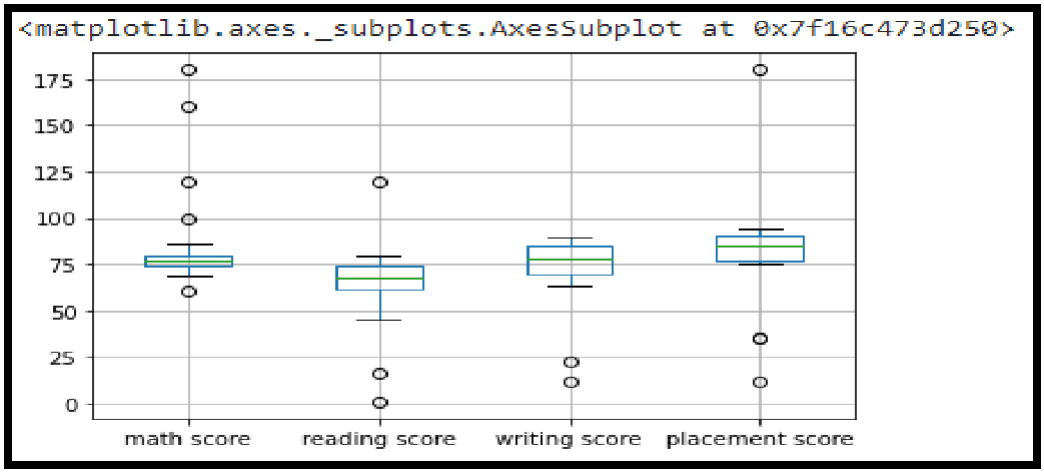
Step 3: Display the data frame

```
df
```

	math score	reading score	writing score	placement score	placement offer count
0	80	68	70	89	3
1	71	61	85	91	3
2	79	16	87	77	2
3	61	77	74	76	2
4	78	71	67	90	3
5	73	68	90	80	2
6	77	62	70	35	2
7	74	45	80	12	1
8	76	60	79	77	2
9	75	65	85	87	3
10	160	67	12	83	2
11	79	72	88	180	2
12	80	80	78	94	3

Step 4: **Select the columns for boxplot and draw the boxplot.**

```
col = ['math score', 'reading score' , 'writing
score','placement score']
df.boxplot(col)
```



Step 5: We can now print the outliers for each column with reference to the box plot.

```
print(np.where(df['math score']>90))
```

```
print(np.where(df['reading score']<25))
print(np.where(df['writing score']<30))
```

4.2 Detecting outliers using Scatterplot:

It is used when you have paired numerical data, or when your dependent variable has multiple values for each reading independent variable, or when trying to determine the relationship between the two variables. In the process of utilizing the scatter plot, one can also use it for outlier detection.

To plot the scatter plot one requires two variables that are somehow related to each other. So here Placement score and Placement count features are used.

Algorithm:

Step 1 : Import pandas , numpy and matplotlib libraries

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

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/demo.csv")
```

Step 3: Display the data frame

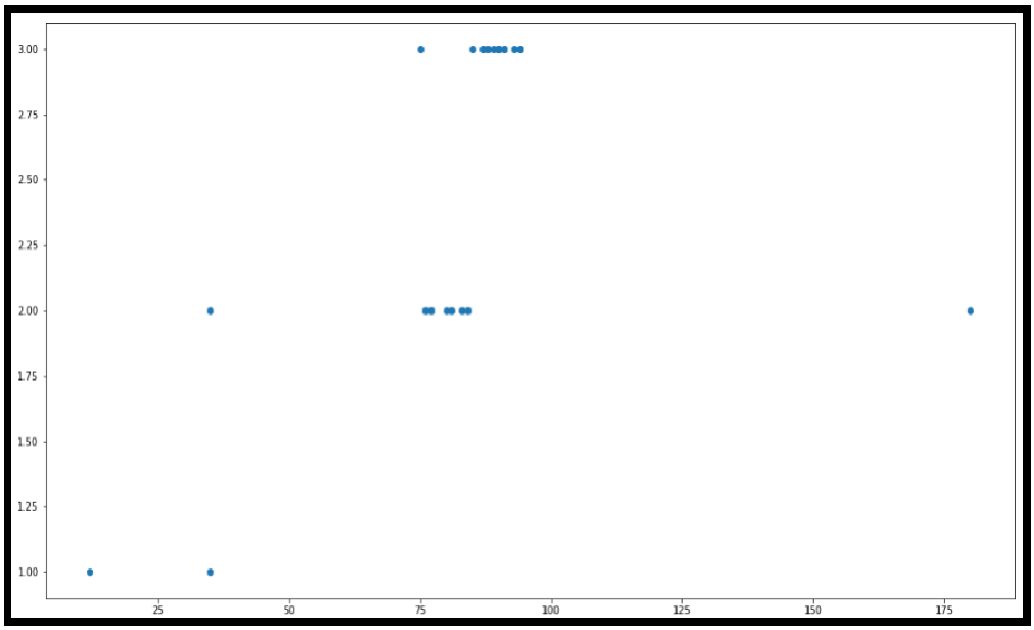
```
df
```

Step 4: Draw the scatter plot with placement score and placement offer count

```
fig, ax = plt.subplots(figsize = (18,10))
ax.scatter(df['placement score'], df['placement offer count'])
plt.show()
```

Labels to the axis can be assigned (Optional)

```
ax.set_xlabel('(Proportion non-retail business acres)/(town)')
ax.set_ylabel('(Full-value property-tax rate)/( $10,000)')
```



Step 5: We can now print the outliers with reference to scatter plot.

```
print(np.where((df['placement  score']<50)  &  (df['placement offer count']>1)))
print(np.where((df['placement  score']>85)  &  (df['placement offer count']<3)))
```

4.3 Detecting outliers using Z-Score:

Z-Score is also called a standard score. This value/score helps to understand how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outliers.

Zscore = (data_point -mean) / std. deviation

Algorithm:

Step 1 : Import numpy and stats from scipy libraries

```
import numpy as np

from scipy import stats
```

Step 2: Calculate Z-Score for mathscore column

```
z = np.abs(stats.zscore(df['math score']))
```

Step 3: Print Z-Score Value. It prints the z-score values of each data item of the column

```
print(z)
```

```
[0.17564553 0.5282877  0.21482799 0.92011234 0.25401045 0.44992277
 0.29319292 0.41074031 0.33237538 0.37155785 2.95895157 0.21482799
 0.17564553 0.25401045 0.37155785 0.25401045 0.05944926 0.17564553
 0.37155785 0.0972806  0.60665263 0.60800375 0.48910524 0.41074031
 0.37155785 3.74260085 0.48910524 0.5282877  1.39165302]
```

Step 4: Now to define an outlier threshold value is chosen.

```
threshold = 0.18
```

Step 5: Display the sample outliers

```
sample_outliers = np.where(z <threshold)
sample_outliers
```

```
(array([ 0, 12, 16, 17, 19]),)
```

4.4 Detecting outliers using Inter Quantile Range(IQR):

IQR (Inter Quartile Range) Inter Quartile Range approach to finding the outliers is the most commonly used and most trusted approach used in the research field.

IQR = Quartile3 – Quartile1

To define the outlier base value is defined above and below datasets normal range namely Upper and Lower bounds, define the upper and the lower bound (1.5*IQR value is considered) :

upper = Q3 +1.5*IQR

lower = Q1 – 1.5*IQR

In the above formula as according to statistics, the 0.5 scale-up of IQR (new_IQR = IQR + 0.5*IQR) is taken.

Algorithm:

Step 1 : Import numpy library

```
import numpy as np
```

Step 2: Sort Reading Score feature and store it into sorted_rscore.

```
sorted_rscore= sorted(df['reading score'])
```

Step 3: Print sorted_rscore

```
sorted_rscore
```

Step 4: Calculate and print Quartile 1 and Quartile 3

```
q1 = np.percentile(sorted_rscore, 25)
q3 = np.percentile(sorted_rscore, 75)
print(q1,q3)
```

```
62.0 74.0
```

Step 5: Calculate value of IQR (Inter Quartile Range)

```
IQR = q3-q1
```

Step 6: Calculate and print Upper and Lower Bound to define the outlier base value.

```
lwr_bound = q1-(1.5*IQR)
upr_bound = q3+(1.5*IQR)
print(lwr_bound, upr_bound)
```

```
44.0 92.0
```

Step 7: Print Outliers

```
r_outliers = []
for i in sorted_rscore:
  if (i<lwr_bound or i>upr_bound):
    r_outliers.append(i)
print(r_outliers)
```

```
[1, 16, 120]
```

3.2 Handling of Outliers:

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

Below are some of the methods of treating the outliers

- Trimming/removing the outlier
- Quantile based flooring and capping
- Mean/Median imputation

- **Trimming/removing the outlier:**

  In this technique, we remove the outliers from the dataset. Although it is not a good practice to follow.

```
new_df=df
for i in sample_outliers:
  new_df.drop(i,inplace=True)
new_df
```

	math score	reading score	writing score	placement score	placement offer count
1	71	61	85	91	3
2	79	16	87	77	2
3	61	77	74	76	2
4	78	71	67	90	3
5	73	68	90	80	2
6	77	62	70	35	2
7	74	45	80	12	1
8	76	60	79	77	2
9	75	65	85	87	3
10	160	67	12	83	2
11	79	72	88	180	2
13	78	69	71	90	3
14	75	1	71	81	2
15	78	62	79	93	3
18	75	62	86	87	3

Here Sample_outliers are `(array([ 0, 12, 16, 17 ]),)` So instances with index 0, 12 ,16 and 17 are deleted.

- **Quantile based flooring and capping:**

  In this technique, the outlier is capped at a certain value above the 90th percentile value or floored at a factor below the 10th percentile value

```
df=pd.read_csv("/demo.csv")
df_stud=df
ninetieth_percentile = np.percentile(df_stud['math score'], 90)
```

```
b = np.where(df_stud['math score']>ninetieth_percentile,
ninetieth_percentile, df_stud['math score'])
print("New array:",b)
```

```
New array: [ 80.  71.  79.  61.  78.  73.  77.  74.  76.  75. 104.  79.  80.  78.
  75.  78.  86.  80.  75.  82.  69. 100.  72.  74.  75. 104.  72.  71.
 104.]
```

```
df_stud.insert(1,"m score",b,True)
df_stud
```

	math score	m score	reading score	writing score	placement score	placement offer count
0	80	80.0	68	70	89	3
1	71	71.0	61	85	91	3
2	79	79.0	16	87	77	2
3	61	61.0	77	74	76	2
4	78	78.0	71	67	90	3
5	73	73.0	68	90	80	2
6	77	77.0	62	70	35	2
7	74	74.0	45	80	12	1

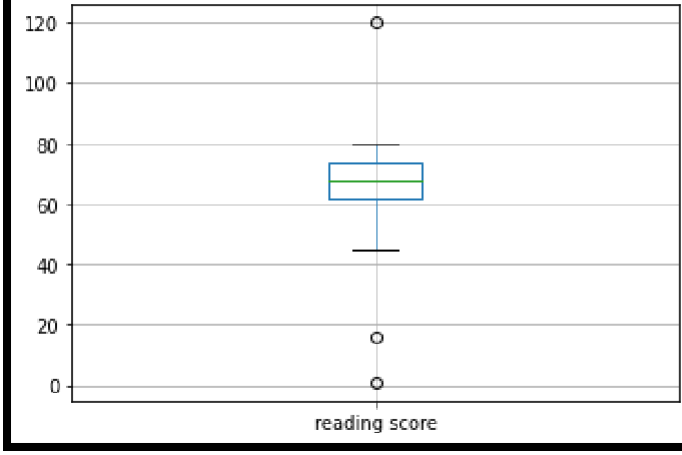
- **Mean/Median imputation:**

As the mean value is highly influenced by the outliers, it is advised to replace the outliers with the median value.

1. Plot the box plot for reading score

```
col = ['reading score']
df.boxplot(col)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f16affce150>
```



2. Outliers are seen in box plot.
3. Calculate the median of reading score by using sorted_rscore

```
median=np.median(sorted_rscore)
median
```

4. Replace the upper bound outliers using median value

```
refined_df=df
refined_df['reading score'] = np.where(refined_df['reading score'] >upr_bound, median,refined_df['reading score'])
```

5. Display redefined_df

	math score	m score	reading score	writing score	placement score	placement offer count
0	80	80.0	68.0	70	89	3
1	71	71.0	61.0	85	91	3
2	79	79.0	16.0	87	77	2
3	61	61.0	77.0	74	76	2
4	78	78.0	71.0	67	90	3
5	73	73.0	68.0	90	80	2
6	77	77.0	62.0	70	35	2
7	74	74.0	45.0	80	12	1
8	76	76.0	60.0	79	77	2
9	75	75.0	65.0	85	87	3
10	160	104.0	67.0	12	83	2

6. Replace the lower bound outliers using median value

```
refined_df['reading score'] = np.where(refined_df['reading score'] <lwr_bound, median,refined_df['reading score'])
```

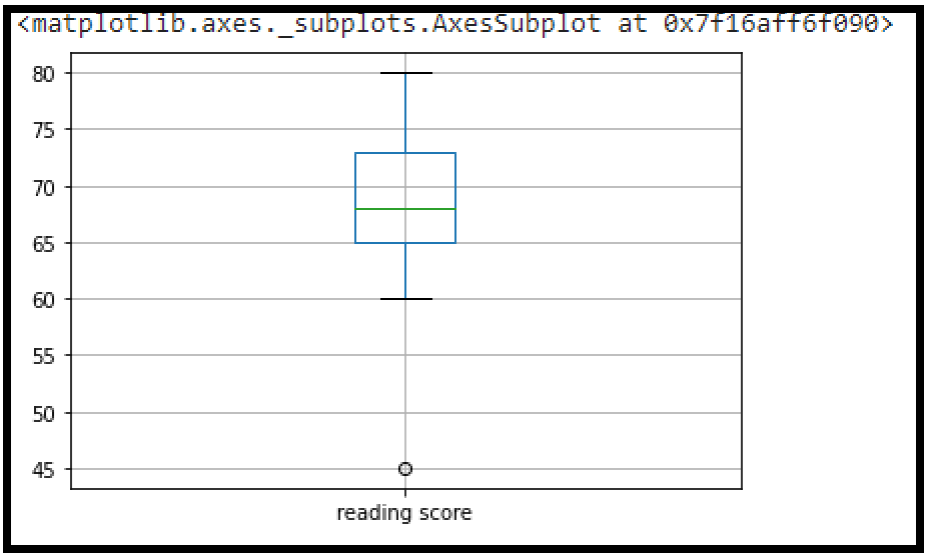
7. Display redefined_df

	math score	m score	reading score	writing score	placement score	placement offer count
0	80	80.0	68.0	70	89	3
1	71	71.0	61.0	85	91	3
2	79	79.0	68.0	87	77	2
3	61	61.0	77.0	74	76	2
4	78	78.0	71.0	67	90	3
5	73	73.0	68.0	90	80	2
6	77	77.0	62.0	70	35	2
7	74	74.0	45.0	80	12	1
8	76	76.0	60.0	79	77	2
9	75	75.0	65.0	85	87	3
10	160	104.0	67.0	12	83	2

8. Draw the box plot for redefined_df

```
col = ['reading score']
refined_df.boxplot(col)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f16aff6f090>
```



4. Data Transformation for the purpose of :

Data transformation is the process of converting raw data into a format or structure that would be more suitable for model building and also data discovery in general.The process of data transformation can also be referred to as extract/transform/load (ETL). The extraction phase involves identifying and pulling data from the various source systems that create data and then moving the data to a single repository. Next, the raw data is cleansed, if needed. It's then transformed into a target format that can be fed into operational systems or into a data warehouse, a date lake or another repository for use in business intelligence and analytics applications. The transformation The data are transformed in ways that are ideal for mining the data. The data transformation involves steps that are.

- **Smoothing:** It is a process that is used to remove noise from the dataset using some algorithms It allows for highlighting important features present in the dataset. It helps in predicting the patterns
- **Aggregation**: Data collection or aggregation is the method of storing and presenting data in a summary format. The data may be obtained from multiple data sources to integrate these data sources into a data analysis description. This is a crucial step since the accuracy of data analysis insights is highly dependent on the quantity and quality of the data used.
- **Generalization**:It converts low-level data attributes to high-level data attributes using concept hierarchy. For Example Age initially in Numerical form (22, 25) is converted into categorical value (young, old).
- **Normalization:** Data normalization involves converting all data variables into a given range. Some of the techniques that are used for accomplishing normalization are:
  - **Min–max normalization**: This transforms the original data linearly.
  - **Z-score normalization**: In z-score normalization (or zero-mean normalization) the values of an attribute (A), are normalized based on the mean of A and its standard deviation.
  - **Normalization by decimal scaling**: It normalizes the values of an attribute by changing the position of their decimal points
- **Attribute or feature construction**.
  - **New attributes constructed from the given ones**: Where new attributes are created & applied to assist the mining process from the given set of attributes. This simplifies the original data & makes the mining more efficient.

    In this assignment , The purpose of this transformation should be one of the following reasons:

a. To change the scale for better understanding (Attribute or feature construction)

Here the Club_Join_Date is transferred to Duration.

Algorithm:

Step 1 : Import pandas and numpy libraries

```
import pandas as pd
import numpy as np
```

Step 2: Load the dataset in dataframe object df

```
df=pd.read_csv("/content/demo.csv")
```

Step 3: Display the data frame

```
df
```

```
df
```

	math score	reading score	writing score	placement score	placement offer count	club join year
0	80	68	70	89	3	2019
1	71	61	85	91	3	2019
2	79	16	87	77	2	2018
3	61	77	74	76	2	2020
4	78	71	67	90	3	2019

Step 3: Change the scale of Joining year to duration.

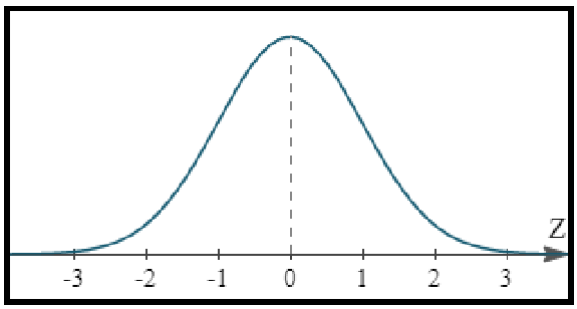
```
[11] df
```

	math score	reading score	writing score	placement score	placement offer count	club join year	Duration
0	80	68	70	89	3	2019	3
1	71	61	85	91	3	2019	3
2	79	16	87	77	2	2018	4
3	61	77	74	76	2	2020	2
4	78	71	67	90	3	2019	3

b. To decrease the skewness and convert distribution into normal distribution (Normalization by decimal scaling)

Data Skewness: It is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution.

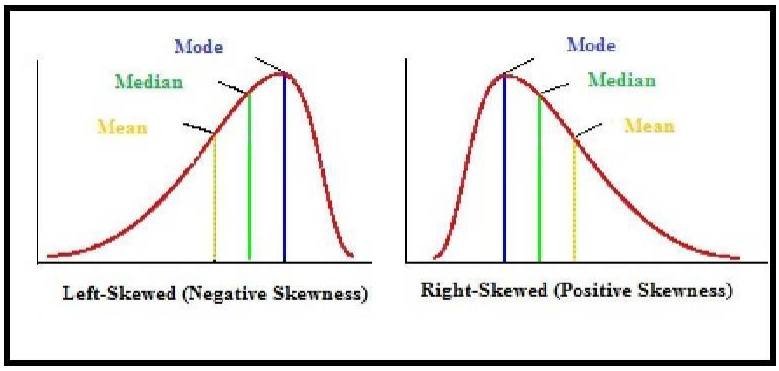
Normal Distribution: In a normal distribution, the graph appears as a classical, symmetrical "bell-shaped curve." The mean, or average, and the mode, or maximum point on the curve, are equal.



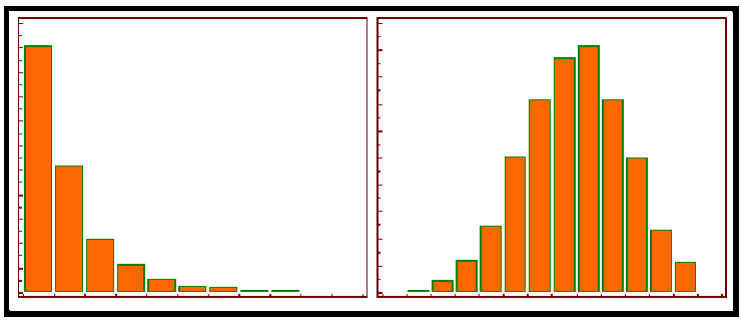
Positively Skewed Distribution

A positively skewed distribution means that the extreme data results are larger. This skews the data in that it brings the mean (average) up. The mean will be larger than the median in a Positively skewed distribution.

A negatively skewed distribution means the opposite: that the extreme data results are smaller. This means that the mean is brought down, and the median is larger than the mean in a negatively skewed distribution.



Reducing skewness A data transformation may be used to reduce skewness. A distribution that is symmetric or nearly so is often easier to handle and interpret than a skewed distribution. The logarithm, x to log base 10 of x, or x to log base e of x (ln x), or x to log base 2 of x, is a strong transformation with a major effect on distribution shape. It is commonly used for reducing right skewness and is often appropriate for measured variables. It can not be applied to zero or negative values.



Algorithm:

Step 1 : Detecting outliers using Z-Score for the Math_score variable and remove the outliers.

Step 2: Observe the histogram for math_score variable.

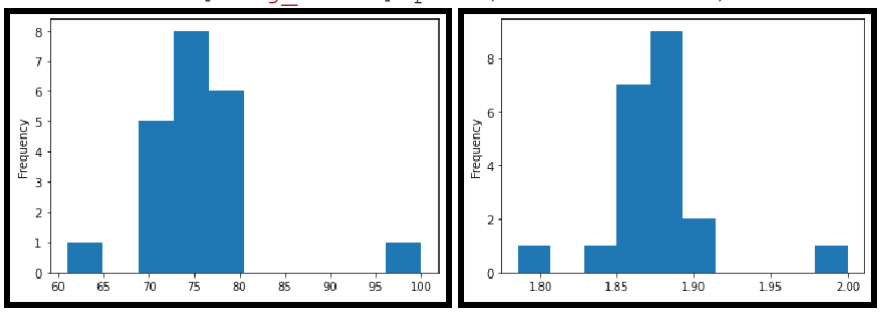
```
import matplotlib.pyplot as plt
new_df['math score'].plot(kind = 'hist')
```

Step 3: Convert the variables to logarithm at the scale 10.

```
df['log_math'] = np.log10(df['math score'])
```

Step 4: Observe the histogram for math_score variable.

```
df['log_math'].plot(kind = 'hist')
```



It is observed that skewness is reduced at some level.

Conclusion: In this way we have explored the functions of the python library for Data Identifying and handling the outliers. Data Transformations Techniques are explored with the purpose of creating the new variable and reducing the skewness from datasets.

Assignment Question:

1. **Explain the methods to detect the outlier.**
2. **Explain data transformation methods**
3. **Write the algorithm to display the statistics of Null values present in the dataset.**
4. **Write an algorithm to replace the outlier value with the mean of the variable.**

.

Practical No.3	Group A
Title	**Descriptive Statistics - Measures of Central Tendency and variability** Perform the following operations on any open source dataset (e.g., data.csv) 1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable. 2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris-versicolor' of iris.csv dataset. Provide the codes with outputs and explain everything that you do in this step.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 3

Title: Descriptive Statistics - Measures of Central Tendency and variability

- ## Problem Statement:

Descriptive Statistics - Measures of Central Tendency and variability

Perform the following operations on any open source dataset (e.g., data.csv)

1. Provide summary statistics (mean, median, minimum, maximum, standard deviation) for a dataset (age, income etc.) with numeric variables grouped by one of the qualitative (categorical) variable. For example, if your categorical variable is age groups and quantitative variable is income, then provide summary statistics of income grouped by the age groups. Create a list that contains a numeric value for each response to the categorical variable.

2. Write a Python program to display some basic statistical details like percentile, mean, standard deviation etc. of the species of 'Iris-setosa', 'Iris-versicolor' and 'Iris-versicolor' of iris.csv dataset.

Provide the codes with outputs and explain everything that you do in this step.

- ## Prerequisites:

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software**

Requirements: Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware**

Requirements: Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:**

64 Bit Windows/Linux/Mac OS

Programming Language: Python 3.8, PIP

Datasets: Any Open Source Dataset/CSV Files

Introduction:

Descriptive Statistics is the building block of data science. Advanced analytics is often incomplete without analyzing descriptive statistics of the key metrics. In simple terms, descriptive statistics can be defined as the measures that summarize a given data, and these measures can be broken down further into the measures of central tendency and the measures of dispersion. Measures of central tendency include mean, median, and the mode, while the measures of variability include standard deviation, variance, and the interquartile range. In this guide, you will learn how to compute these measures of descriptive statistics and use them to interpret the data.

We will cover the topics given below:

1. Mean
2. Median
3. Mode
4. Standard Deviation
5. Variance
6. Interquartile Range
7. Skewness

We will begin by loading the dataset to be used in this guide.

Data:

In this guide, we will be using fictitious data of loan applicants containing 0 observations and 10 variables, as described below:

1. Marital_status: Whether the applicant is married ("Yes") or not ("No").
2. Dependents: Number of dependents of the applicant.
3. Is_graduate: Whether the applicant is a graduate ("Yes") or not ("No").
4. Income: Annual Income of the applicant (in USD).
5. Loan_amount: Loan amount (in USD) for which the application was submitted.
6. Term_months: Tenure of the loan (in months).
7. Credit_score: Whether the applicant's credit score was good ("Satisfactory") or not ("Not_satisfactory").
8. Age: The applicant’s age in years.
9. Sex: Whether the applicant is female (F) or male (M).
10. approval_status: Whether the loan application was approved ("Yes") or not ("No").

Let's start by loading the required libraries and the data.

```
1import pandas as pd
2import numpy as np
3import statistics as st
4
5# Load the data
6df = pd.read_csv("data_desc.csv")
7print(df.shape)
```

```
8print(df.info())
```

python
Output:

```
1 (  0, 10)
2
3 <class 'pandas.core.frame.DataFrame'>
4 RangeIndex:   0 entries, 0 to 599
5 Data columns (total 10 columns):
6 Marital_status       0 non-null object
7 Dependents           0 non-null int64
8 Is_graduate          0 non-null object
9 Income               0 non-null int64
10 Loan_amount          0 non-null int64
11 Term_months          0 non-null int64
12 Credit_score         0 non-null object
13 approval_status      0 non-null object
14 Age                  0 non-null int64
15 Sex                  0 non-null object
16 dtypes: int64(5), object(5)
17 memory usage: 47.0+ KB
18 None
```

Five of the variables are categorical (labelled as 'object') while the remaining five are numerical (labelled as 'int').

Measures of Central Tendency:

Measures of central tendency describe the center of the data, and are often represented by the mean, the median, and the mode.

Mean: Mean represents the arithmetic average of the data. The line of code below prints the mean of the numerical variables in the data. From the output, we can infer that the average age of the applicant is 49 years, the average annual income is USD 705,541, and the average tenure of loans is 183 months. The command df.mean(axis = 0) will also give the same output.

```
1df.mean()
```

python
Output:

```
1 Dependents          0.748333
```

```
2 Income         705541.333333
3 Loan_amount    323793.666667
4 Term_months       183.350000
5 Age                49.450000
6 dtype: float64
```

It is also possible to calculate the mean of a particular variable in a data, as shown below, where we calculate the mean of the variables 'Age' and 'Income'.

```
1print(df.loc[:,'Age'].mean())
2print(df.loc[:,'Income'].mean())
```

python
Output:

```
1 49.45
2 705541.33
```

In the previous sections, we computed the column-wise mean. It is also possible to calculate the mean of the rows by specifying the (axis = 1) argument. The code below calculates the mean of the first five rows.

```
1df.mean(axis = 1)[0:5]
```

python
Output:

```
1 0     70096.0
2 1    161274.0
3 2    125113.4
4 3    119853.8
5 4    120653.8
6 dtype: float64
```

Median

In simple terms, median represents the 50th percentile, or the middle value of the data, that separates the distribution into two halves. The line of code below prints the median of the numerical variables in the data. The command df.median(axis = 0) will also give the same output.

```
1df.median()
```

python
Output:

```
1 Dependents          0.0
2 Income         508350.0
3 Loan_amount     7  00.0
```

```
4 Term_months      192.0
5 Age              51.0
6 dtype: float64
```

From the output, we can infer that the median age of the applicants is 51 years, the median annual income is USD 508,350, and the median tenure of loans is 192 months. There is a difference between the mean and the median values of these variables, which is because of the distribution of the data. We will learn more about this in the subsequent sections.
It is also possible to calculate the median of a particular variable in a data, as shown in the *first two lines of code* below. We can also calculate the median of the rows by specifying the (axis = 1) argument. The *third line* below calculates the median of the first five rows.

```
1#to calculate a median of a particular column
2print(df.loc[:,'Age'].median())
3print(df.loc[:,'Income'].median())
4
5df.median(axis = 1)[0:5]
```

python
Output:

```
1 51.0
2 508350.0
3
4 0    102.0
5 1    192.0
6 2    192.0
7 3    192.0
8 4    192.0
9 dtype: float64
```

Mode: Mode represents the most frequent value of a variable in the data. This is the only central tendency measure that can be used with categorical variables, unlike the mean and the median which can be used only with quantitative data.

The line of code below prints the mode of all the variables in the data. The .mode() function returns the most common value or most repeated value of a variable. The command df.mode(axis = 0) will also give the same output.

```
1df.mode()
```

python
Output:

```
1|        | Marital_status    | Dependents      | Is_graduate      | Income | Loan_amount    | Term_month
2|        |                   |                 |                  |        |                |
```

```
3| 0        | Yes                | 0        | Yes                | 333300 | 70000                | 192.0
         | Satisfactory       | Yes                | 55       | M      |
```

The interpretation of the mode is simple. The output above shows that most of the applicants are married, as depicted by the 'Marital_status' value of "Yes". Similar interpretation could be done for the other categorical variables like 'Sex' and 'Credit-Score'. For numerical variables, the mode value represents the value that occurs most frequently. For example, the mode value of 55 for the variable 'Age' means that the highest number (or frequency) of applicants are 55 years old.

Measures of Dispersion:

In the previous sections, we have discussed the various measures of central tendency. However, as we have seen in the data, the values of these measures differ for many variables. This is because of the extent to which a distribution is stretched or squeezed. In statistics, this is measured by dispersion which is also referred to as variability, scatter, or spread. The most popular measures of dispersion are standard deviation, variance, and the interquartile range.

Standard Deviation:

Standard deviation is a measure that is used to quantify the amount of variation of a set of data values from its mean. A low standard deviation for a variable indicates that the data points tend to be close to its mean, and vice versa. The line of code below prints the standard deviation of all the numerical variables in the data.

```
1df.std()
```

python Output:

```
1 Dependents         1.026362
2 Income        711421.814154
3 Loan_amount   724293.480782
4 Term_months       31.933949
5 Age               14.728511
6 dtype: float64
```

While interpreting standard deviation values, it is important to understand them in conjunction with the mean. For example, in the above output, the standard deviation of the variable 'Income' is much higher than that of the variable 'Dependents'. However, the unit of these two variables is different and, therefore, comparing the dispersion of these two variables on the basis of standard deviation alone will be incorrect. This needs to be kept in mind.

It is also possible to calculate the standard deviation of a particular variable, as shown inthe *first two lines of code* below. The *third line* calculates the standard deviation for the first five rows.

```
1print(df.loc[:,'Age'].std())
2print(df.loc[:,'Income'].std())
```

```
3
4#calculate the standard deviation of the first five rows
5df.std(axis = 1)[0:5]
```

python
Output:

```
1 14.728511412020659
2 711421.814154101
3
4 0    133651.842584
5 1    3056  .733951
6 2    244137.726597
7 3    233466.2050
8 4    202769.786470
9 dtype: float64
```

Variance

Variance is another measure of dispersion. It is the square of the standard deviation and the covariance of the random variable with itself. The line of code below prints the variance of all the numerical variables in the dataset. The interpretation of the variance is similar to that of the standard deviation.

```
1df.var()
```

python
Output:

```
1 Dependents     1.053420e+00
2 Income         5.061210e+11
3 Loan_amount    5.24  10e+11
4 Term_months    1.019777e+03
5 Age            2.169290e+02
6 dtype: float64
```

Interquartile Range (IQR)

The Interquartile Range (IQR) is a measure of statistical dispersion, and is calculated as the difference between the upper quartile (75th percentile) and the lower quartile (25th percentile). The IQR is also a very important measure for identifying outliers and could be visualized using a boxplot.

IQR can be calculated using the iqr() function. The *first line of code* below imports the 'iqr' function from the scipy.stats module, while the *second line* prints the IQR for the variable 'Age'.

```
1from scipy.stats import iqr
2iqr(df['Age'])
```

python
Output:

```
1 25.0
```

Skewness

Another useful statistic is skewness, which is the measure of the symmetry, or lack of it, for a real-valued random variable about its mean. The skewness value can be positive, negative, or undefined. In a perfectly symmetrical distribution, the mean, the median, and the mode will all have the same value. However, the variables in our data are not symmetrical, resulting in different values of the central tendency.
We can calculate the skewness of the numerical variables using the skew() function, as shown below.

```
1print(df.skew())
```

python
Output:

```
1 Dependents     1.169632
2 Income         5.344587
3 Loan_amount    5.006374
4 Term_months  -2.471879
5 Age          -0.055537
6 dtype: float64
```

The skewness values can be interpreted in the following manner:

- Highly skewed distribution: If the skewness value is less than −1 or greater than +1.
- Moderately skewed distribution: If the skewness value is between −1 and −½ or between +½ and +1.
- Approximately symmetric distribution: If the skewness value is between −½ and +½.

Putting Everything Together

We have learned the measures of central tendency and dispersion, in the previous sections. It is important to analyse these individually, however, because there are certain useful functions in python that can be called upon to find these values. One such important function is the .describe() function that prints the summary statistic of the numerical variables. The line of code below performs this operation on the data.

```
1df.describe()
```

python

Output:

```
1|       | Dependents      | Income          | Loan_amount     | Term_months     | Age
         |
2|       |       -         |                 |                 |                 |       -
---      |
3| count |   0.000000      | 6.000000e+02    | 6.000000e+02    |   0.000000      |
  0.000000       |
4| mean  | 0.748333        | 7.055413e+05    | 3.237937e+05    | 183.350000      |
49.450000        |
5| std   | 1.026362        | 7.114218e+05    | 7.242935e+05    | 31.933949       |
14.728511        |
6| min   | 0.000000        | 3.000000e+04    | 1.090000e+04    | 18.000000       |
22.000000        |
7| 25%   | 0.000000        | 3.849750e+05    | 6.100000e+04    | 192.000000      |
36.000000        |
8| 50%   | 0.000000        | 5.083500e+05    | 7.  0000e+04    | 192.000000      |
51.000000        |
9| 75%   | 1.000000        | 7.661000e+05    | 1.302500e+05    | 192.000000      |
61.000000        |
10| max          | 6.000000        | 8.444900e+06    | 7.780000e+06    | 252.000000      | 76.000000
```

The above output prints the important summary statistics of all the numerical variables like the mean, median (50%), minimum, and maximum values, along with the standard deviation. We can also calculate the IQR using the 25th and 75th percentile values.
However, the 'describe()' function only prints the statistics for the quantitative or numerical variable. In order to print the similar statistics for all the variables, an additional argument, include='all', needs to be added, as shown in the line of code below.

```
1df.describe(include='all')
```

python
Output:

```
1|       | Marital_status  | Dependents      | Is_graduate     | Income          |
Loan_amount     | Term_months     | Credit_score    | approval_status | Age             | Sex     |
2|       |                 |       -         |                 |                 |       -
-       |                 |                 |                 |       -         |   -     |
3|  ount | 0               |  0.000000       |  0              | 6.000000e+02    |
6.000000e+02    |  0.000000       |  0              |  0              |  0.000000       |  0      |
4| unique          | 2               | NaN             | 2               | NaN             | NaN
         | NaN             | 2               | 2               | NaN             | 2       |
5| top     | Yes             | NaN             | Yes             | NaN             | NaN
         | NaN             | Satisfactory    | Yes             | NaN             | M       |
6| freq    | 391             | NaN             | 470             | NaN             | NaN
         | NaN             | 472             | 410             | NaN             | 489     |
```

7 mean	NaN	0.748333	NaN	7.055413e+05	3.237937e+
8 std	NaN	1.026362	NaN	7.114218e+05	
7.242935e+05	31.933949	NaN	NaN	14.728511	NaN
9 min	NaN	0.000000	NaN	3.000000e+04	
1.090000e+04	18.000000	NaN	NaN	22.000000	NaN
10 25%	NaN	0.000000	NaN	3.849750e+05	6.100000e+
11 50%	NaN	0.000000	NaN	5.083500e+05	7. 0000e+
12 75%	NaN	1.000000	NaN	7.661000e+05	1.302500e+
13 max	NaN	6.000000	NaN	8.444900e+06	7.780000e+

Now we have the summary statistics for all the variables. For qualitative variables, we will not have the statistics such as the mean or the median, but we will have statistics like the frequency and the unique label.

Conclusion:

In this guide, you have learned about the fundamentals of the most widely used descriptive statistics and their calculations with Python. We covered the following topics in this guide:

1. Mean
2. Median
3. Mode
4. Standard Deviation
5. Variance
6. Interquartile Range
7. Skewness

It is important to understand the usage of these statistics and which one to use, depending on the problem statement and the data. To learn more about data preparation and building machine learning models using Python's 'scikit-learn' library, please refer to the following guides:

1. Scikit Machine Learning
2. Linear, Lasso, and Ridge Regression with scikit-learn
3. Non-Linear Regression Trees with scikit-learn
4. Machine Learning with Neural Networks Using scikit-learn
5. Validating Machine Learning Models with scikit-learn
6. Ensemble Modeling with scikit-learn
7. Preparing Data for Modeling with scikit-learn

Practical No.4	Group A
Title	**Data Analytics I** Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 4

Title: Data Analytics, I

- **Problem Statement:**

- Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (https://www.kaggle.com/c/boston-housing). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset.
- The objective is to predict the value of prices of the house using the given features.

- **Prerequisites:**

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Theory: Boston Housing with Linear Regression

With this data our objective is create a model using linear regression to predict the houses price

The data contains the following columns:

- 'crim': per capita crime rate by town.
- 'zn': proportion of residential land zoned for lots over 25,000 sq.ft.
- 'indus': proportion of non-retail business acres per town.
- 'chas':Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
- 'nox': nitrogen oxides concentration (parts per 10 million).
- 'rm': average number of rooms per dwelling.
- 'age': proportion of owner-occupied units built prior to 1940.
- 'dis': weighted mean of distances to five Boston employment centres.
- 'rad': index of accessibility to radial highways.
- 'tax': full-value property-tax rate per $10,000.
- 'ptratio': pupil-teacher ratio by town
- 'black': 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
- 'lstat': lower status of the population (percent).
- 'medv': median value of owner-occupied homes in $$1000s

Ps: this is my first analysis, i'm learning how to interpret the plots.

Lets Start

First we need to prepare our enviroment importing some librarys

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
# Importing DataSet and take a look at Data
BostonTrain = pd.read_csv("../input/boston_train.csv")
```

Here we can look at the BostonTrain data

In [3]:
```
BostonTrain.head()
```

Out[3]:

In [4]:

```
BostonTrain.info()
BostonTrain.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 333 entries, 0 to 332
Data columns (total 15 columns):
ID         333 non-null int64
crim       333 non-null float64
zn         333 non-null float64
indus      333 non-null float64
chas       333 non-null int64
nox        333 non-null float64
rm         333 non-null float64
age        333 non-null float64
dis        333 non-null float64
rad        333 non-null int64
tax        333 non-null int64
ptratio    333 non-null float64
black      333 non-null float64
lstat      333 non-null float64
medv       333 non-null float64
dtypes: float64(11), int64(4)
memory usage: 39.1 KB
```

Out[4]:

Now, or goal is think about the columns, and discovery which columns is relevant to build our model, because if we consider to put columns with not relevant with our objective "medv" the model may be not efficient

In [5]:

```
#ID columns does not relevant for our analysis.
BostonTrain.drop('ID', axis = 1, inplace=True)
```

In [6]:

```
BostonTrain.plot.scatter('rm', 'medv')
```

Out[6]:
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe883a8080>

In this plot its clearly to see a linear pattern. Wheter more average number of rooms per dwelling, more expensive the median value is.

Now lets take a loot how the all variables relate to each other.

In [7]:
```
plt.subplots(figsize=(12,8))
sns.heatmap(BostonTrain.corr(), cmap = 'RdGy')
```

Out[7]:
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe883530b8>

At this heatmap plot, we can do our analysis better than the pairplot.

Lets focus ate the last line, where y = medv:

When shades of Red/Orange: the more red the color is on X axis, smaller the medv. Negative correlation
When light colors: those variables at axis x and y, they dont have any relation. Zero correlation
When shades of Gray/Black : the more black the color is on X axis, more higher the value med is. Positive correlation

Lets plot the paiplot, for all different correlations

Negative Correlation.

When x is high y is low and vice versa.

To the right less negative correlation.

In [8]:
```
sns.pairplot(BostonTrain, vars = ['lstat', 'ptratio', 'indus', 'tax', 'crim', 'nox', 'rad', 'age', 'medv'])
```

Out[8]:
<seaborn.axisgrid.PairGrid at 0x7fbe88285c50>

Zero Correlation. When x and y are completely independent

Positive Correlation. When x and y go together

to the right more independent.

In [9]:

```
sns.pairplot(BostonTrain, vars = ['rm', 'zn', 'black', 'dis', 'chas','medv'])
```

Out[9]:
<seaborn.axisgrid.PairGrid at 0x7fbdf20f9d30>

Trainning Linear Regression Model

Define X and Y

X: Varibles named as predictors, independent variables, features.
Y: Variable named as response or dependent variable

In [10]:
```
X = BostonTrain[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'black', 'lstat']]
y = BostonTrain['medv']
```

Import sklearn librarys:
train_test_split, to split our data in two DF, one for build a model and other to validate.
LinearRegression, to apply the linear regression.

In [11]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

In [13]:
```
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Out[13]:
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [14]:
```
predictions = lm.predict(X_test)
```

In [15]:
```
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[15]:
Text(0,0.5,'Predicted Y')

In [16]:
```
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
```

```
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

MAE: 3.53544941908
MSE: 20.8892997114
RMSE: 4.57048134351

Considering the RMSE: we can conclude that this model average error is RMSE at medv, which means RMSE *1000 in money

In [17]:

```
sns.distplot((y_test-predictions),bins=50);
```

As more normal distribution, better it is.

In [18]:

```
coefficients = pd.DataFrame(lm.coef_,X.columns)
coefficients.columns = ['coefficients']
coefficients
```

Out[18]:

	Coefficients
crim	-0.116916
zn	0.017422
indus	-0.001589
chas	3.267698
nox	-17.405512
rm	3.242758
age	0.006570

	coefficients
dis	-1.414341
rad	0.404683
Tax	-0.013598
Ptratio	-0.724007
Black	0.007861
Lstat	-0.711690

How to interpret those coefficients: they are in function of Medv, so for one unit that nox increase, the house value decrease 'nox'*1000 (Negative correlation) money unit, for one unit that rm increase, the house value increase 'rm'*1000 (Positive correlation) money unit.

*1000 because the medv is in 1000 and this apply to the other variables/coefficients.

ML | Boston Housing Kaggle Challenge with Linear Regression

Boston Housing Data: This dataset was taken from the StatLib library and is maintained by Carnegie Mellon University. This dataset concerns the housing prices in the housing city of Boston. The dataset provided has 506 instances with 13 features. The Description of the dataset is taken from

```
1. CRIM      per capita crime rate by town
2. ZN        proportion of residential land zoned for lots over
             25,000 sq.ft.
3. INDUS     proportion of non-retail business acres per town
4. CHAS      Charles River dummy variable (= 1 if tract bounds
             river; 0 otherwise)
5. NOX       nitric oxides concentration (parts per 10 million)
6. RM        average number of rooms per dwelling
7. AGE       proportion of owner-occupied units built prior to 1940
8. DIS       weighted distances to five Boston employment centres
9. RAD       index of accessibility to radial highways
10. TAX      full-value property-tax rate per $10,000
11. PTRATIO  pupil-teacher ratio by town
12. B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks
             by town
13. LSTAT    % lower status of the population
14. MEDV     Median value of owner-occupied homes in $1000's
```

Let’s make the Linear Regression Model, predicting housing prices
Inputting Libraries and dataset.

- Python3

```
# Importing Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot  as plt



# Importing Data

from sklearn.datasets import load_boston

boston = load_boston()
```

The shape of input Boston data and getting feature_names

- Python3

```
boston.data.shape
```

```
(506, 13)
```

- Python3

```
boston.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],
      dtype='<U7')
```

Converting data from nd-array to data frame and adding feature names to the data

- Python3

```
data = pd.DataFrame(boston.data)

data.columns = boston.feature_names



data.head(10)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311

Adding ‘Price’ column to the dataset

- Python3

```
# Adding 'Price' (target) column to the data

boston.target.shape
```

```
(506,)
```

- Python3

```
data['Price'] = boston.target

data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222

Description of Boston dataset

- Python3

```
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.2
std	8.596783	23.322453	6.860353	0.253994	0.115878	0.7
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.5
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.8
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.2
75%	3.647423	12.500000	18.100000	0.000000	0.624000	6.6
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.7

Info of Boston Dataset

- Python3

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

Getting input and output data and further splitting data to training and testing dataset.

- Python3

```
# Input Data

x = boston.data
```

```
# Output Data

y = boston.target




# splitting data to training and testing dataset.



#from sklearn.cross_validation import train_test_split

#the submodule cross_validation is renamed and reprecated to
model_selection

from sklearn.model_selection  import train_test_split



xtrain, xtest, ytrain, ytest = train_test_split(x,  y,  test_size =0.2,

                                                    random_state = 0)



print("xtrain shape : ", xtrain.shape)

print("xtest shape : ", xtest.shape)

print("ytrain shape : ", ytrain.shape)

print("ytest shape  : ", ytest.shape)
```

```
xtrain shape :  (404, 13)
xtest shape  :  (102, 13)
ytrain shape :  (404,)
ytest shape  :  (102,)
```

Applying Linear Regression Model to the dataset and predicting the prices.

- Python3

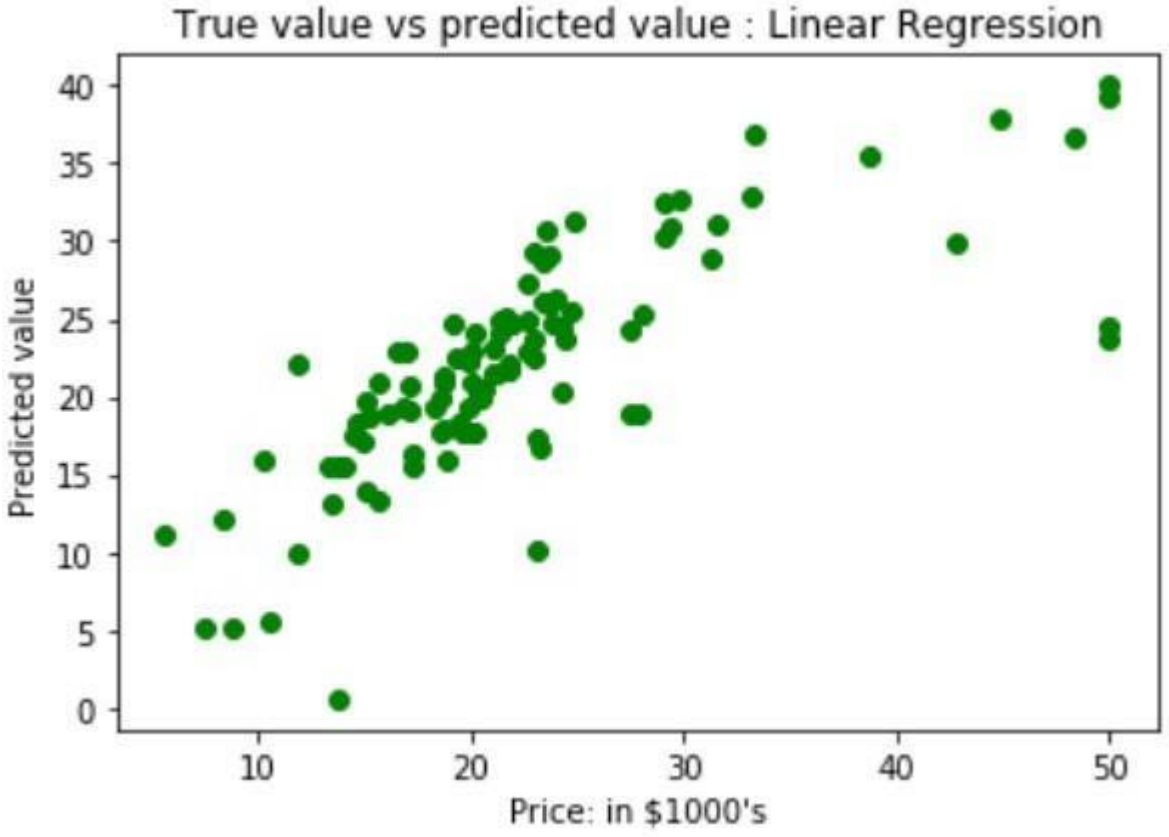
```
# Fitting Multi Linear regression model to training model

from sklearn.linear_model  import  LinearRegression

regressor = LinearRegression()

regressor.fit(xtrain, ytrain)



# predicting the test set results

y_pred  = regressor.predict(xtest)
```

Plotting Scatter graph to show the prediction results – ‘ytrue’ value vs ‘y_pred’ value

- Python3

```
# Plotting Scatter graph to show the prediction

# results - 'ytrue' value vs 'y_pred' value
```

```
plt.scatter(ytest,  y_pred, c  = 'green')

plt.xlabel("Price: in $1000's")

plt.ylabel("Predicted value")

plt.title("True value vs predicted value : Linear Regression")

plt.show()
```



Results of Linear Regression i.e. Mean Squared Error.

Python3

```
# Results of Linear Regression.

from sklearn.metrics import mean_squared_error

mse = mean_squared_error(ytest, y_pred)

print("Mean Square Error : ", mse)
```

```
Mean Square Error :  33.4507089677
```

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta\left(x^i\right) - y_i \right]^2$$

As per the result, our model is only 66.55% accurate. So, the prepared model is not very good for predicting housing prices. One can improve the prediction results using many other possible machine learning algorithms and techniques.

Practical No.5	Group A
Title	**Data Analytics II** 1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 5

Title: Data Analytics II

- **Problem Statement:**

Data Analytics II

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

- **Prerequisites:**

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Theory: Logistic Regression: Social Network Ads

This project will be a walkthrough of a simple Logistic Regression model in an attempt to strategize a basic ad-targeting campaign for a social media network/website. One of our sponsor's advertisements seems to be particularly successful among our older, wealthier users but seemingly less-so with our younger ones. We'd like to implement an appropriate model so that we know who our target audience is for this specific advertisement, thus maximizing our click-through rate. We'd like to show younger users this ad with a lower probability than we show it to our older/wealthier users, and use that time/space to expose the younger users to ads that they are more likely to be interested in.

Our dataset contains some information about all of our users in the social network, including their User ID, Gender, Age, and Estimated Salary. The last column of the dataset is a vector of booleans describing whether or not each individual ended up clicking on the advertisement (0 = False, 1 = True). Let's import the relevant libraries, the dataset, and establish which variables are either dependent or independent. We'll continually print out any changes that we've made to the data at the bottom of our code cells.

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

dataset = pd.read_csv('../input/Social_Network_Ads.csv')
dataset.head()
```

Sr. No.	User ID	Gender	Age	Estimated Salary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	4000	0
3	15 3246	Female	27	8000	0
4	15804002	Male	35	80000	0

If we wanted to determine the effect of more independent variables on the outcome (such as Gender), we would have to implement a Dimensionality Reduction aspect to the model because we can only describe so many dimensions visually. However, right now we are only worried about how the users' Age and Estimated Salary affect their decision to click or not click on the advertisement. To do this, we will extract the relevant vectors from our dataset: the independent variables (X) and the dependent variable (y). The following code segment

describes the selection of the entire third and fourth columns for X, as well as the entire fifth column for y. Again, we'll print out our data in order to help visualize the model.

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

print(X[:3, :])
print('-'*15)
print(y[:3])
```

```
[[   19 19000]
 [   35 20000]
 [ 26 43000]]
---------------
[0 0 0]
```

We now need to split our data into two sets: a training set for the machine to learn from, as well as a test set for the machine to execute on. This process is referred to as Cross Validation and we will be implementing SciKit Learn's appropriately named 'train_test_split' class to make it happen. Industry standard usually calls for a training set size of 70-80% so we'll split the two.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

print(X_train[:3])
print('-'*15)
print(y_train[:3])
print('-'*15)
print(X_test[:3])
print('-'*15)
print(y_test[:3])
```

```
[[   44  39000]
 [   32 120000]
 [   38  50000]]
---------------
[0 1 0]
---------------
[[   30 87000]
 [   38 50000]
 [ 35 75000]]
---------------
[0 0 0]
```

To get the most accurate results, a common tool within machine learning models is to apply Feature Scaling: "...a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step." -

Wikipedia https://en.wikipedia.org/wiki/Feature_scaling. SciKit Learn, again, has a helpful library called StandardScaler that will quickly preprocess the data for us in this manner.

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
/opt/conda/lib/python3.6/site-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [5]:

```
print(X_train[:3])
print('-'*15)
print(X_test[:3])
```

```
[[ 0.58164944 -0.88670699]
 [-0.  673761  1.46173768]
 [-0.01254409 -0.5677824 ]]
---------------
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085 0.1570462 ]]
```

Now we are ready to build our Logistic Regression Model. We create an object of the LogisticRegression() class and refer to it as our 'classifier' for obvious reasons. The random state variable simply allows us to all get the same outcome but can be changed to alter the results slightly. We then fit the classifier to the training set with the aptly named .fit() method so that it can understand the correlations between X and y. Lastly, we will test the classifier's predictive power on the test set. The Logistic Regression's .predict() method will give us a vector of predictions for our dataset, X_test.

In [6]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs' )
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print(X_test[:10])
```

```
print('-'*15)
print(y_pred[:10])
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.210  859 2.15757314] [-
 1.99318916 -0.04590581] [
 0.8787462  -0.77073441]]
```

```
---------------
[0 0 0 0 0 0 0 1 0 1]
```

We can see from the first ten values for y_pred that only the eighth and tenth individuals within the index are predicted to click on the advertisement. If you do some quick mental math, you'll also notice that these are the only two pairs with a positive summation. We can start to make some inferences about the relationship now. As the value of these independent variables increases (Age and Estimated Salary), the more likely it is that a given user will click on the ad. This also gives us a clue into how the model is appropriating a binary outcome (will click or will not click) for each individual based on the sum of these scaled values. We can think of 0 as the center of our normal distribution and consider any value falling below that threshold (any negative value) as being assigned a '0' boolean value because the probability that they click on the advertisement is below 50%. The machine is essentially rounding up (True) or down (False) based on if the individual's probability of clicking is closer to 0% or 100%.

We can also compare our prediction vector (y_pred) to the actual observable values in y_test. We'll print out the first 20 values adjacent to each other to get a sneak-peek of how close our classifier came to a 100% prediction rate. When you run the cell below, you'll notice that there is only one mismatch between the two, thus our model has a 19/20 or 95% prediction rate for the first 20 samples! This is a great start, but we'll have to look at the rest of the data, as well.

In [7]:

```
print(y_pred[:20])
print(y_test[:20])
```

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0]
```

Now that we've preprocessed the data, fit our classifier to the training set, and predicted the dependent values for our test set, we can use a Confusion Matrix to evaluate exactly how accurate our Logistic Regression model is. This function will compare the calculated results in our y_pred vector to the actual observed results in y_test to determine how similar they are. The more values that match, the higher the accuracy of the classifier.

Note: If you are unfamiliar with the structure, the top-left and bottom-right quadrants will tell us how many predictions were correct - while the other two values indicate how many were in correct. We can divide the summation of those first two numbers by the total number of samples to give us a percentage of accuracy.

In [8]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[65  3]
 [ 8 24]]
```

Conclusion: This Confusion Matrix tells us that there were 89 correct predictions and 11 incorrect ones, meaning the model overall accomplished an 89% accuracy rating. This is very good and there are many ways to improve the model by parameter tuning and sample size increasing, but those topics are outside the scope of this project. Our next step is to create visualizations to compare the training set and the test set. As we've stated throughout this discussion, seeing our data and being able to visualize our work in front of us is imperative to understanding each step of the model. Charts and graphs will also help us explain our findings in layman's terms so that others can comprehend the insights that we've derived and they can implement our findings into their business plans. The Matplotlib library provides some excellent tools to create visualizations so let's do that now. We'll start by plotting the training set results amidst our classifier. Most of the code in the next cell is relatively straightforward but feel free to visit https://matplotlib.org/ for more detail.

In [9]:

```
# Visualizing the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.6, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
```

```
plt.show()
```

This graph helps us see the clear correlations between the dependent and independent variables. It is obvious that, as Age and Estimated Salary increase, each individual has a higher likelihood of being green (they will click on the ad). Intuitively, this graph makes a lot of sense because we can quickly tell that about 80-90% of the observations have been correctly identified. There will almost always be some degree of error - or at least there should be, otherwise our model is probably guilty of overfitting. Now let's map the test set results to visualize where our Confusion Matrix came from.

In [10]:

```
# Visualizing the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.6, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

When we look at both models together, we can actually see that there is a shape to this data that's becoming increasingly apparent as the number of observations increases. Notice that that the positive (or green) data points seem to be almost wrapping around the most crowded area of red dots, inferring that we can probably improve our model and projections by implementing a non-linear model (one with a classifier that isn't restricted to being a straight line). The best X-intercept is probably closer to 1 than it is to 2 (as shown in this model), and the y-intercept is likely between 2 and 3. But while there is always room for improvement, we can be satisfied with this model as our final product. Our accuracy is high, but not so high that we need to be suspicious of any overfitting. We can safely say that an increase in both Age and Estimated Salary will lead to a higher probability of clicking the advertisement. As new users sign-up for the website, we can use this model to quickly determine whether or not to expose them to this particular ad or choose another that is more relevant to their profile.

Successfully ran in 11.1s

Output

Time	#	Log Message
2.8s	1	[NbConvertApp] Converting notebook __notebook_.ipynb to notebook
2.9s	2	[NbConvertApp] Executing notebook with kernel: python3
7.8s	3	[NbConvertApp] Writing 62764 bytes to __notebook__.ipynb
10.1s	4	[NbConvertApp] Converting notebook __notebook_.ipynb to html
10.5s	5	[NbConvertApp] Support files will be in__results__files/ [NbConvertApp] Making directory__re
10.5s	6	[NbConvertApp] Making directory__results__files [NbConvertApp] Writing 287968 bytes to __r

2.8s 1 [NbConvertApp] Converting notebook __notebook_.ipynb to notebook

2.9s 2 [NbConvertApp] Executing notebook with kernel: python3

7.8s 3 [NbConvertApp] Writing 62764 bytes to__notebook_.ipynb

10.1s 4 [NbConvertApp] Converting notebook __notebook_.ipynb to html

10.5s 5 [NbConvertApp] Support files will be in__results__files/

[NbConvertApp] Making directory __results__files

10.5s 6 [NbConvertApp] Making directory __results__files

[NbConvertApp] Writing 287968 bytes to __results_.html

Practical No.6	Group A
Title	**Data Analytics III** 1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset. 2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 6

Title: Data Analytics, III

- ## Problem Statement:

Data Analytics III

1. Implement Simple Naïve Bayes classification algorithm using Python/R on iris.csv dataset.

2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

- ## Prerequisites:

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Objective of the Assignment: Students should be able to data analysis using Naïve Bayes classification algorithm using Python for any open source dataset

---

Prerequisite:

1. Basic of Python Programming

2.Concept of Join and Marginal Probability.

Contents for Theory:

1. **Concepts used in Naïve Bayes classifier**
2. **Naive Bayes Example**
3. **Confusion Matrix Evaluation Metrics**

---

1. Concepts used in Naïve Bayes classifier

- Naïve Bayes Classifier can be used for Classification of categorical data.
    - Let there be a ‘j’ number of classes. C={1,2,….j}
    - Let, input observation is specified by ‘P’ features. Therefore input observation x is given , x = {F1,F2,…..Fp}
    - The Naïve Bayes classifier depends on Bayes' rule from probability theory.
- Prior probabilities: Probabilities which are calculated for some event based on no other information are called Prior probabilities.

  For example, P(A), P(B), P(C) are prior probabilities because while calculating P(A), occurrences of event B or C are not concerned i.e. no information about occurrence of any other event is used.

  Conditional Probabilities:

$$P\left(\frac{A}{B}\right) = \frac{P(A \cap B)}{P(B)} \quad if\ P(B) \neq 0 \qquad \ldots\ldots(1)$$

$$P\left(\frac{B}{A}\right) = \frac{P(B \cap A)}{P(A)} \qquad \ldots\ldots\ldots(2)$$

From equation (1) and (2) ,

$$P(A \cap B) = P\left(\frac{A}{B}\right).P(B) = P\left(\frac{B}{A}\right).P(A)$$

$$\therefore \quad P\left(\frac{A}{B}\right) = \frac{P\left(\frac{B}{A}\right).P(A)}{P(B)}$$

Is called the Bayes Rule.

2. Example of Naive Bayes

We have a dataset with some features Outlook, Temp, Humidity, and Windy, and the target here is to predict whether a person or team will play tennis or not.

Outlook	Temp	Humidity	Windy	Play
sunny	hot	high	FALSE	no
sunny	hot	high	TRUE	no
overcast	hot	high	FALSE	yes
rainy	mild	high	FALSE	yes
rainy	cool	normal	FALSE	yes
rainy	cool	normal	TRUE	no
overcast	cool	normal	TRUE	yes
sunny	mild	high	FALSE	no
sunny	cool	normal	FALSE	yes
rainy	mild	normal	FALSE	yes
sunny	mild	normal	TRUE	yes
overcast	mild	high	TRUE	yes
overcast	hot	normal	FALSE	yes
rainy	mild	high	TRUE	no

$$X = [\underbrace{Outlook}_{x_1}, \underbrace{Temp}_{x_2}, \underbrace{Humidity}_{x_3}, \underbrace{Windy}_{x_4}]$$

$$C_k = [\underbrace{Yes}_{C_1}, \underbrace{No}_{C_2}]$$

Conditional Probability

Here, we are predicting the probability of class1 and class2 based on the given condition. If I try to write the same formula in terms of classes and features, we will get the following equation

$$P(C_k \mid X) = \frac{P(X \mid C_k) * P(C_k)}{P(X)}$$

Now we have two classes and four features, so if we write this formula for class C1, it will be something like this.

$$P(C_1 \mid x_1 \cap x_2 \cap x_3 \cap x_4) = \frac{P(x_1 \cap x_2 \cap x_3 \cap x_4 \mid C_1) * P(C_1)}{P(x_1 \cap x_2 \cap x_3 \cap x_4)}$$

Here, we replaced Ck with C1 and X with the intersection of X1, X2, X3, X4. You might have a question, It's because we are taking the situation when all these features are present at the same time.

The Naive Bayes algorithm assumes that all the features are independent of each other or in other words all the features are unrelated. With that assumption, we can further simplify the above formula and write it in this form

$$P(C_1 \mid x_1 \cap x_2 \cap x_3 \cap x_4) = \frac{P(x_1 \mid C_1) * P(x_2 \mid C_1) * P(x_3 \mid C_1) * P(x_4 \mid C_1) * P(C_1)}{P(x_1) * P(x_2) * P(x_3) * P(x_4)}$$

This is the final equation of the Naive Bayes and we have to calculate the probability of both C1 and C2.For this particular example.

Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

$$P(Yes \mid X) = P(Rainy \mid Yes) \times P(Cool \mid Yes) \times P(High \mid Yes) \times P(True \mid Yes) \times P(Yes)$$

$$P(Yes \mid X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.00529 \rightarrow 0.2 = \frac{0.00529}{0.02057 + 0.00529}$$

$$P(No \mid X) = P(Rainy \mid No) \times P(Cool \mid No) \times P(High \mid No) \times P(True \mid No) \times P(No)$$

$$P(No \mid X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.02057 \rightarrow 0.8 = \frac{0.02057}{0.02057 + 0.00529}$$

P (N0 | Today) > P (Yes | Today) So, the prediction that golf would be played is 'No'.

Algorithm (Iris Dataset):

Step 1: Import libraries and create alias for Pandas, Numpy and Matplotlib

Step 2: Import the Iris dataset by calling URL.

Step 3: Initialize the data frame

Step 4: Perform Data Preprocessing

- `Convert Categorical to Numerical Values if applicable`
- `Check for Null Value`

- Divide the dataset into Independent(X) and Dependent(Y)variables.
- Split the dataset into training and testing datasets
- Scale the Features if necessary.

Step 5: Use Naive Bayes algorithm(Train the Machine) to Create Model

```
# import the class
from sklearn.naive_bayes import GaussianNB
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
```

Step 6: Predict the y_pred for all values of train_x and test_x

```
Y_pred = gaussian.predict(X_test)
```

Step 7:Evaluate the performance of Model for train_y and test_y

```
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
```

Step 8: Calculate the required evaluation parameters `from sklearn.metrics import precision_score,confusion_matrix,accuracy_score,recall_score`

```
cm = confusion_matrix(y_test, Y_pred)
```

Conclusion:

In this way we have done data analysis using Naive Bayes Algorithm for Iris dataset and evaluated the performance of the model.

Value Addition: Visualising Confusion Matrix using Heatmap

Assignment Question:

1) Consider the observation for the car theft scenario having 3 attributes colour, Type and origin.

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

Find the probability of car theft having scenarios Red SUV and Domestic.

2) Write python code for the preprocessing mentioned in step 4. and Explain every step in detail.

.

Practical No.7	Group A
Title	**Text Analytics** 1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization. 2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 7

Title: Text Analytics

- ## Problem Statement:

Text Analytics

1. Extract Sample document and apply following document preprocessing methods:

Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.

2. Create representation of document by calculating Term Frequency and Inverse Document Frequency. .

- ## Prerequisites:

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Contents for Theory:

1. **Basic concepts of Text Analytics**
2. **Text Analysis Operations using natural language toolkit**
3. **Text Analysis Model using TF-IDF.**
4. **Bag of Words (BoW)**

---

1. Basic concepts of Text Analytics

One of the most frequent types of day-to-day conversion is text communication. In our everyday routine, we chat, message, tweet, share status, email, create blogs, and offer opinions and criticism. All of these actions lead to a substantial amount of unstructured text being produced. It is critical to examine huge amounts of data in this sector of the online world and social media to determine people's opinions.

Text mining is also referred to as text analytics. Text mining is a process of exploring sizable textual data and finding patterns. Text Mining processes the text itself, while NLP processes with the underlying metadata. Finding frequency counts of words, length of the sentence, presence/absence of specific words is known as text mining. Natural language processing is one of the components of text mining. NLP helps identify sentiment, finding entities in the sentence, and category of blog/article. Text mining is preprocessed data for text analytics. In Text Analytics, statistical and machine learning algorithms are used to classify information.

2. Text Analysis Operations using natural language toolkit

NLTK(natural language toolkit) is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning and many more.
Analysing movie reviews is one of the classic examples to demonstrate a simple NLP Bag-of-words model, on movie reviews.

2.1. Tokenization:

Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or sentences is called Tokenization. Token is a single entity that is the building blocks for a sentence or paragraph.

- Sentence tokenization : split a paragraph into **list of sentences** using

  sent_tokenize() method

- Word tokenization : split a sentence into **list of words** using **word_tokenize()** method

2.2. Stop words removal

Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc. In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words.

2.3. Stemming and Lemmatization

Stemming is a normalization technique where lists of tokenized words are converted into shortened root words to remove redundancy. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form.

A computer program that stems word may be called a stemmer.

E.g.

A stemmer reduces the words like fishing, fished, and fisher to the stem fish.

The stem need not be a word, for example the Porter algorithm reduces, argue, argued, argues, arguing, and argus to the stem argu .

Lemmatization in NLTK is the algorithmic process of finding the lemma of a word depending on its meaning and context. Lemmatization usually refers to the morphological analysis of words, which aims to remove inflectional endings. It helps in returning the base or dictionary form of a word known as the lemma.

Eg. Lemma for studies is study

Lemmatization Vs Stemming

Stemming algorithm works by cutting the suffix from the word. In a broader sense cuts either the beginning or end of the word.

On the contrary, Lemmatization is a more powerful operation, and it takes into consideration morphological analysis of the words. It returns the lemma which is the base form of all its inflectional forms. In-depth linguistic knowledge is

required to create dictionaries and look for the proper form of the word. Stemming is a general operation while lemmatization is an intelligent operation where the proper form will be looked in the dictionary. Hence, lemmatization helps in forming better machine learning features.

2.4. POS Tagging

POS (Parts of Speech) tell us about grammatical information of words of the sentence by assigning specific token (Determiner, noun, adjective , adverb , verb,Personal Pronoun etc.) as tag (DT,NN ,JJ,RB,VB,PRP etc) to each words.

Word can have more than one POS depending upon the context where it is used. We can use POS tags as statistical NLP tasks. It distinguishes a sense of word which is very helpful in text realization and infer semantic information from text for sentiment analysis.

3. Text Analysis Model using TF-IDF.

Term frequency–inverse document frequency(TFIDF) , is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

- **Term Frequency (TF)**

  It is a measure of the frequency of a word (w) in a document (d). TF is defined as the ratio of a word's occurrence in a document to the total number of words in a document. The denominator term in the formula is to normalize since all the corpus documents are of different lengths.

$$TF(w,d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

Example:

Documents	Text	Total number of words in a document
A	Jupiter is the largest planet	5
B	Mars is the fourth planet from the sun	8

The initial step is to make a vocabulary of unique words and calculate TF for each document. TF will be more for words that frequently appear in a document and less for rare words in a document.

- **Inverse Document Frequency (IDF)**

  It is the measure of the importance of a word. Term frequency (TF) does not consider the importance of words. Some words such as' of', 'and', etc. can be most frequently present but are of little significance. IDF provides weightage to each word based on its frequency in the corpus D.

$$IDF(w, D) = \ln\left(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w}\right)$$

In our example, since we have two documents in the corpus, N=2.

Words	TF (for A)	TF (for B)	IDF
Jupiter	1/5	0	ln(2/1) = 0.69
Is	1/5	1/8	ln(2/2) = 0
The	1/5	2/8	ln(2/2) = 0
largest	1/5	0	ln(2/1) = 0.69
Planet	1/5	1/8	ln(2/2) = 0
Mars	0	1/8	ln(2/1) = 0.69
Fourth	0	1/8	ln(2/1) = 0.69
From	0	1/8	ln(2/1) = 0.69
Sun	0	1/8	ln(2/1) = 0.69

- **Term Frequency — Inverse Document Frequency (TFIDF)**

  It is the product of TF and IDF.

  TFIDF gives more weightage to the word that is rare in the corpus (all the documents).

  TFIDF provides more importance to the word that is more frequent in the document.

$$TFIDF\ (w, d, D) = TF(w, d) * IDF(w, D)$$

Words	TF (for A)	TF (for B)	IDF	TFIDF (A)	TFIDF (B)
Jupiter	1/5	0	ln(2/1) = 0.69	0.138	0
Is	1/5	1/8	ln(2/2) = 0	0	0
The	1/5	2/8	ln(2/2) = 0	0	0
largest	1/5	0	ln(2/1) = 0.69	0.138	0
Planet	1/5	1/8	ln(2/2) = 0	0.138	0
Mars	0	1/8	ln(2/1) = 0.69	0	0.086
Fourth	0	1/8	ln(2/1) = 0.69	0	0.086
From	0	1/8	ln(2/1) = 0.69	0	0.086
Sun	0	1/8	ln(2/1) = 0.69	0	0.086

After applying TFIDF, text in A and B documents can be represented as a TFIDF vector of dimension equal to the vocabulary words. The value corresponding to each word represents the importance of that word in a particular document.

TFIDF is the product of TF with IDF. Since TF values lie between 0 and 1, not using ***ln*** can result in high IDF for some words, thereby dominating the TFIDF. We don't want that, and therefore, we use ***ln*** so that the IDF should not completely dominate the TFIDF.

- **Disadvantage of TFIDF**

It is unable to capture the semantics. For example, funny and humorous are synonyms, but TFIDF does not capture that. Moreover, TFIDF can be computationally expensive if the vocabulary is vast.

4. Bag of Words (BoW)

Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a

bucket. This approach is called a bag of words model or BoW for short. It's referred to as a "bag" of words because any information about the structure of the sentence is lost.

Algorithm for Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization:

Step 1: Download the required packages

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

Step 2: Initialize the text

```
text= "Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunkssuch as words or sentences is called Tokenization."
```

Step 3: Perform Tokenization

```
#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)

#Word  Tokenization
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

Step 4: Removing Punctuations and Stop Word

```
# print stop words of English
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)

text= "How to remove stop words with NLTK library in Python?"
text= re.sub('[^a-zA-Z]', ' ',text)
tokens = word_tokenize(text.lower())
filtered_text=[]
for w in tokens:
      if w not in stop_words:
            filtered_text.append(w)
print("Tokenized Sentence:",tokens)
print("Filterd  Sentence:",filtered_text)
```

Step 5 : Perform Stemming

```
from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
       rootWord=ps.stem(w)
 print(rootWord)
```

Step 6: Perform Lemmatization

```
 from nltk.stem import    WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
      print("Lemma       for       {}       is       {}".format(w,
      wordnet_lemmatizer.lemmatize(w)))
```

Step 7: Apply POS Tagging to text

```
import nltk
from nltk.tokenize import word_tokenize
data="The pink sweater fit her perfectly"
words=word_tokenize(data)
 for word in words:
     print(nltk.pos_tag([word]))
```

Algorithm for Create representation of document by calculating TFIDF

Step 1: Import the necessary libraries.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

Step 2: Initialize the Documents.

```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the  Sun'
```

Step 3: Create BagofWords (BoW) for Document A and B.

```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

Step 4: Create Collection of Unique words from Document A and B.

```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

Step 5: Create a dictionary of words and their occurrence for each document in the corpus

```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
      numOfWordsA[word] += 1
      numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
      numOfWordsB[word] += 1
```

Step 6: Compute the term frequency for each of our documents.

```
def computeTF(wordDict, bagOfWords):
tfDict = {}
bagOfWordsCount =  len(bagOfWords)
for word, count in  wordDict.items():
    tfDict[word] = count / float(bagOfWordsCount)
return tfDict
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB,  bagOfWordsB)
```

Step 7: Compute the term Inverse Document Frequency.

```
 def computeIDF(documents):
import math
N = len(documents)

idfDict = dict.fromkeys(documents[0].keys(), 0)
for document in documents:
    for word, val in document.items():
        if val > 0:
            idfDict[word] += 1

for word, val in idfDict.items():
    idfDict[word] = math.log(N / float(val))
return idfDict
idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

Step 8: Compute the term TF/IDF for all words.

```
def computeTFIDF(tfBagOfWords, idfs):
tfidf = {}
for word, val in tfBagOfWords.items():
    tfidf[word] = val * idfs[word]
return tfidf
```

```
tfidfA = computeTFIDF(tfA, idfs)
tfidfB =  computeTFIDF(tfB,  idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

Conclusion:

In this way we have done text data analysis using TF IDF algorithm

Assignment Question:

1) **Perform Stemming for** `text = "studies studying cries cry".` **Compare the results generated with Lemmatization. Comment on your answer howStemming and Lemmatization differ from each other.**

2) **Write Python code for removing stop words from the below documents, conver the documents into lowercase and calculate the TF, IDF and TFIDF score for each document.**

```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the  Sun'
```

.

Practical No.8	Group A
Title	**Data Visualization I** 1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data. 2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 8

Title: Data Visualization I

- **Problem Statement:**

Data Visualization I

1. Use the inbuilt dataset 'titanic'. The dataset contains 891 rows and contains information about the passengers who boarded the unfortunate Titanic ship. Use the Seaborn library to see if we can find any patterns in the data.

2. Write a code to check how the price of the ticket (column name: 'fare') for each passenger is distributed by plotting a histogram.

- **Prerequisites:**

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Introduction:

In this article we will look at Seaborn which is another extremely useful library for data visualization in Python. The Seaborn library is built on top of Matplotlib and offers many advanced data visualization capabilities.

Though, the Seaborn library can be used to draw a variety of charts such as matrix plots, grid plots, regression plots etc., in this article we will see how the Seaborn library can be used to draw distributional and categorial plots. In the second part of the series, we will see how to draw regression plots, matrix plots, and grid plots.

Downloading the Seaborn Library

The seaborn library can be downloaded in a couple of ways. If you are using pip installer for Python libraries, you can execute the following command to download the library:

```
pip install seaborn
```

Alternatively, if you are using the Anaconda distribution of Python, you can use execute the following command to download the seaborn library:

```
conda install seaborn
```

Theory:

The Dataset

The dataset that we are going to use to draw our plots will be the Titanic dataset, which is downloaded by default with the Seaborn library. Now we have to use the load_dataset function and pass it the name of the dataset.

Let's see what the Titanic dataset looks like. Execute the following script:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

dataset = sns.load_dataset( 'titanic' )

dataset.head()
```

The script above loads the Titanic dataset and displays the first five rows of the dataset using the head function. The output looks like this:

survived	pclass	sex	age	sibsp	parch	fare	embarked	class
0	3	male	22.0	1	0	7.2500	S	Third
1	1	female	38.0	1	0	71.2833	C	First
1	3	female	26.0	0	0	7.9250	S	Third
1	1	female	35.0	1	0	53.1000	S	First
0	3	male	35.0	0	0	8.0500	S	Third

The dataset contains 891 rows and 15 columns and contains information about the passengers who boarded the unfortunate Titanic ship. The original task is to predict whether or not the passenger survived depending upon different features such as their age, ticket, cabin they boarded, the class of the ticket, etc. We will use the Seaborn library to see if we can find any patterns in the data.

Distributional Plots:

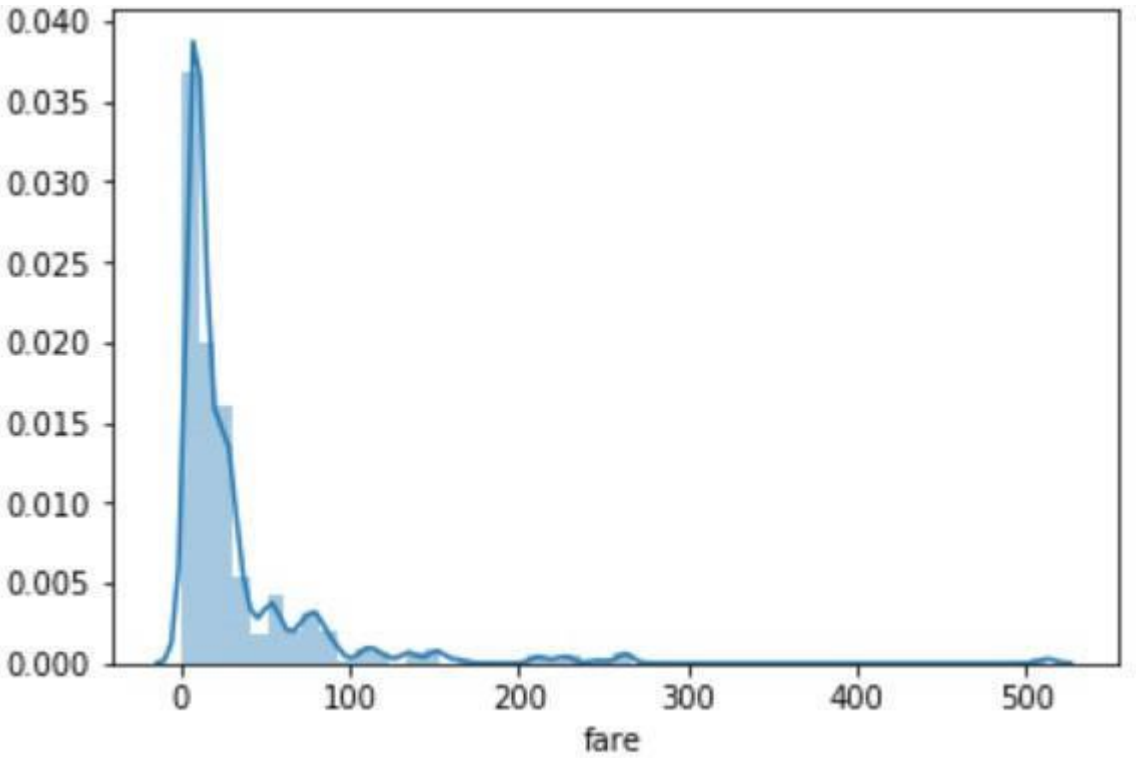
Distributional plots, as the name suggests are type of plots that show the statistical distribution of data. In this section we will see some of the most commonly used distribution plots in Seaborn.

The Dist Plot:

The `distplot()` shows the histogram distribution of data for a single column. The column name is passed as a parameter to the `distplot()` function. Let's see how the price of the ticket for each passenger is distributed. Execute the following script:

```
sns.distplot(dataset[ 'fare' ])
```

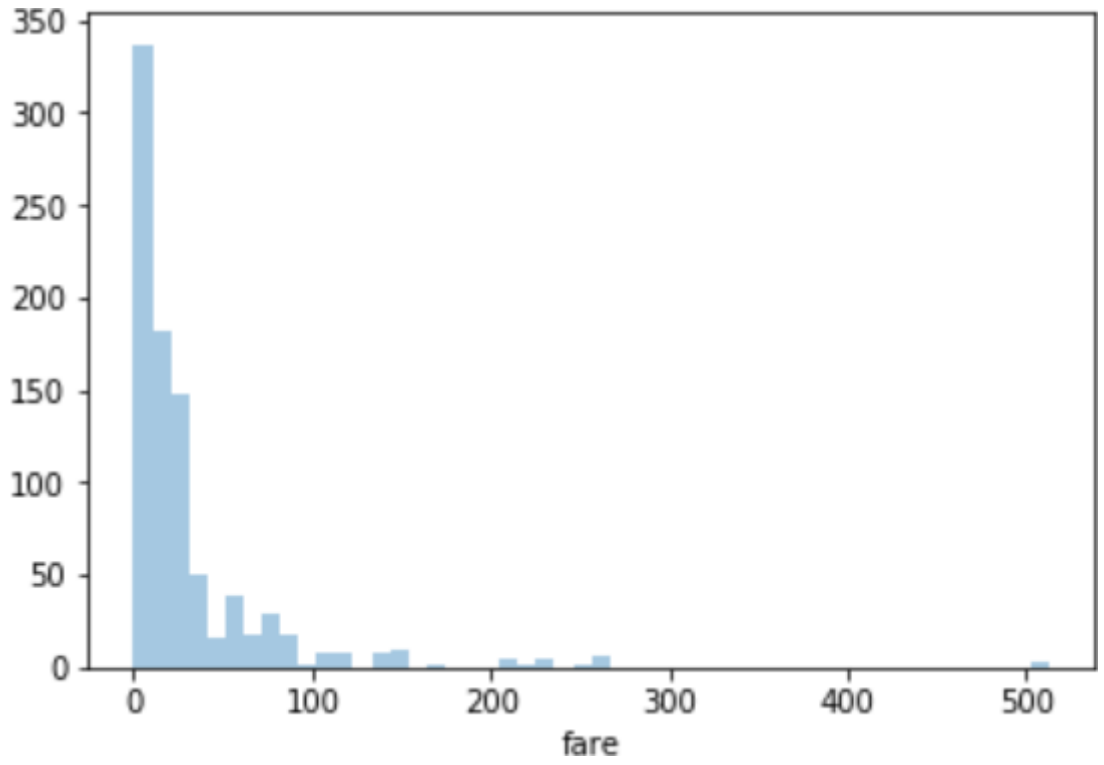
Output:



We can see that most of the tickets have been solved between 0-50 dollars. The line that we see represents the kernel density estimation. We can remove this line by passing `False` as the parameter for the `kde` attribute as shown below:

```
sns.distplot(dataset[ 'fare' ], kde= False )
```

Output:

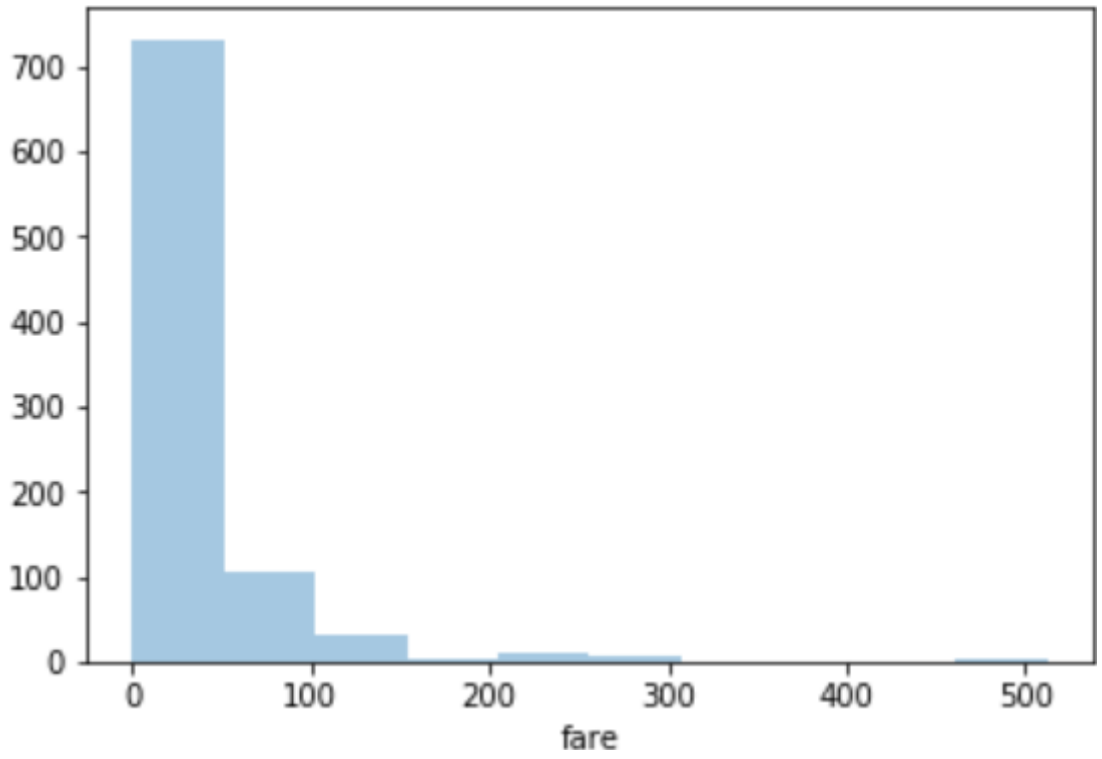


Now you can see there is no line for the kernel density estimation on the plot. We can also pass the value for the `bins` parameter in order to see more or less details in the graph. Take a look at the following script:

```
sns.distplot(dataset[ 'fare' ], kde= False , bins= 10 )
```

Here we set the number of bins to 10. In the output, you will see data distributed in 10 bins as shown below:

Output:



We can clearly see that for more than 700 passengers, the ticket price is between 0 and 50.

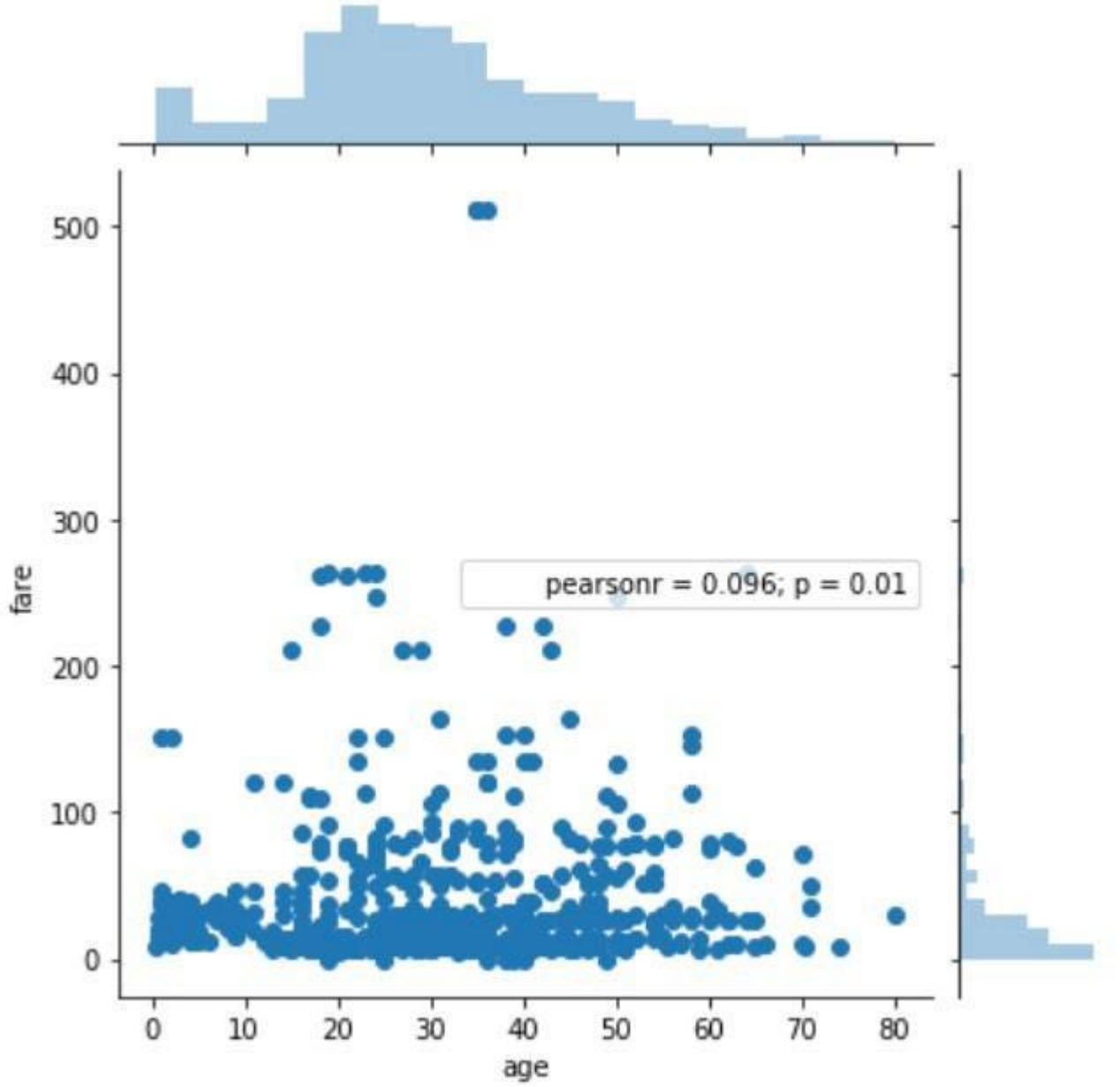
The Joint Plot

The `jointplot()` is used to display the mutual distribution of each column. You need to pass three parameters to `jointplot`. The first parameter is the column name for which you want to display the distribution of data on x-axis. The second parameter is the column name for which you want to display the distribution of data on y-axis. Finally, the third parameter is the name of the data frame.

Let's plot a joint plot of `age` and `fare` columns to see if we can find any relationship between the two.

```
sns.jointplot(x= 'age' , y= 'fare' , data=dataset)
```

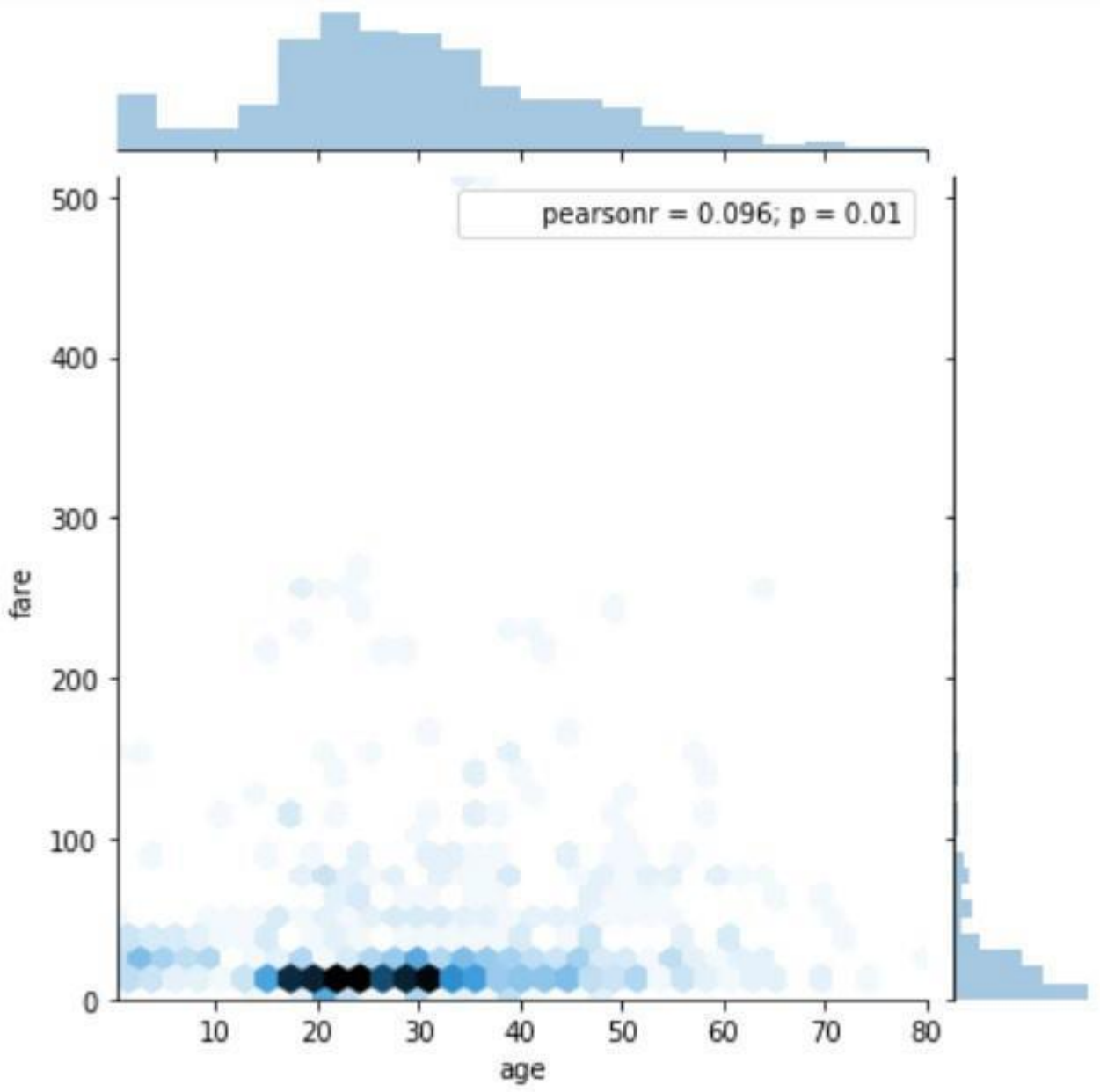
Output:



From the output, we can see that a joint plot has three parts. A distribution plot at the top for the column on the x-axis, a distribution plot on the right for the column on the y-axis and a scatter plot in between that shows the mutual distribution of data for both the columns. We can see that there is no correlation observed between prices and the fares. We can change the type of the joint plot by passing a value for the `kind` parameter. For instance, if instead of scatter plot, we want to display the distribution of data in the form of a hexagonal plot, we can pass the value `hex` for the `kind` parameter. Look at the following script:

```
sns.jointplot(x= 'age' , y= 'fare' , data=dataset, kind= 'hex' )
```

Output:

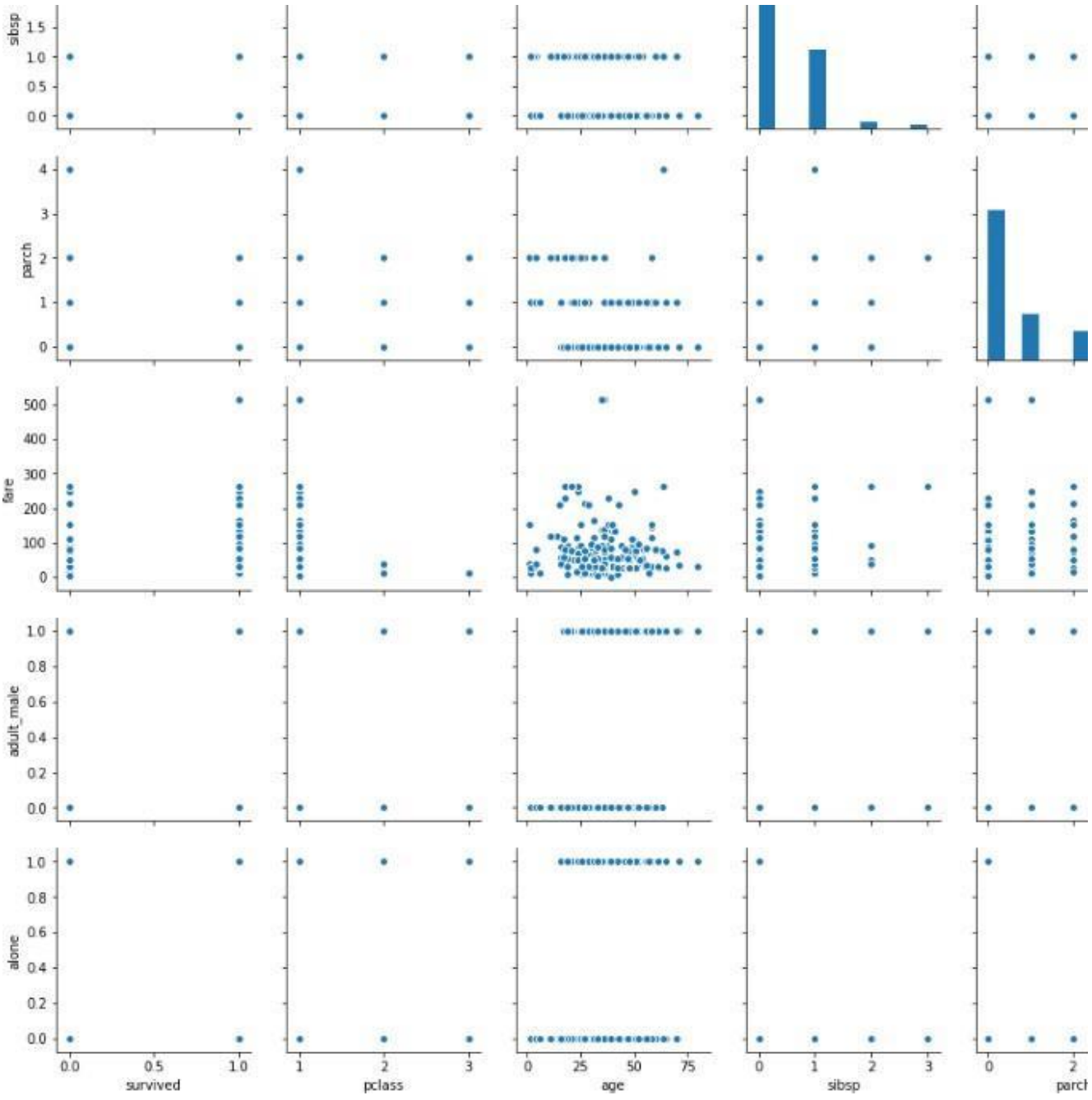


In the hexagonal plot, the hexagon with most number of points gets darker color. So if we look at the above plot, we can see that most of the passengers are between age 20 and 30 and most of them paid between 10-50 for the tickets.

The Pair Plot: The paitplot() is a type of distribution plot that basically plots a joint plot for all the possible combination of numeric and Boolean columns in the dataset. We need to pass the name of your dataset as the parameter to the pairplot() function as shown below:

```
sns.pairplot(dataset)
```

A snapshot of the portion of the output is shown below:

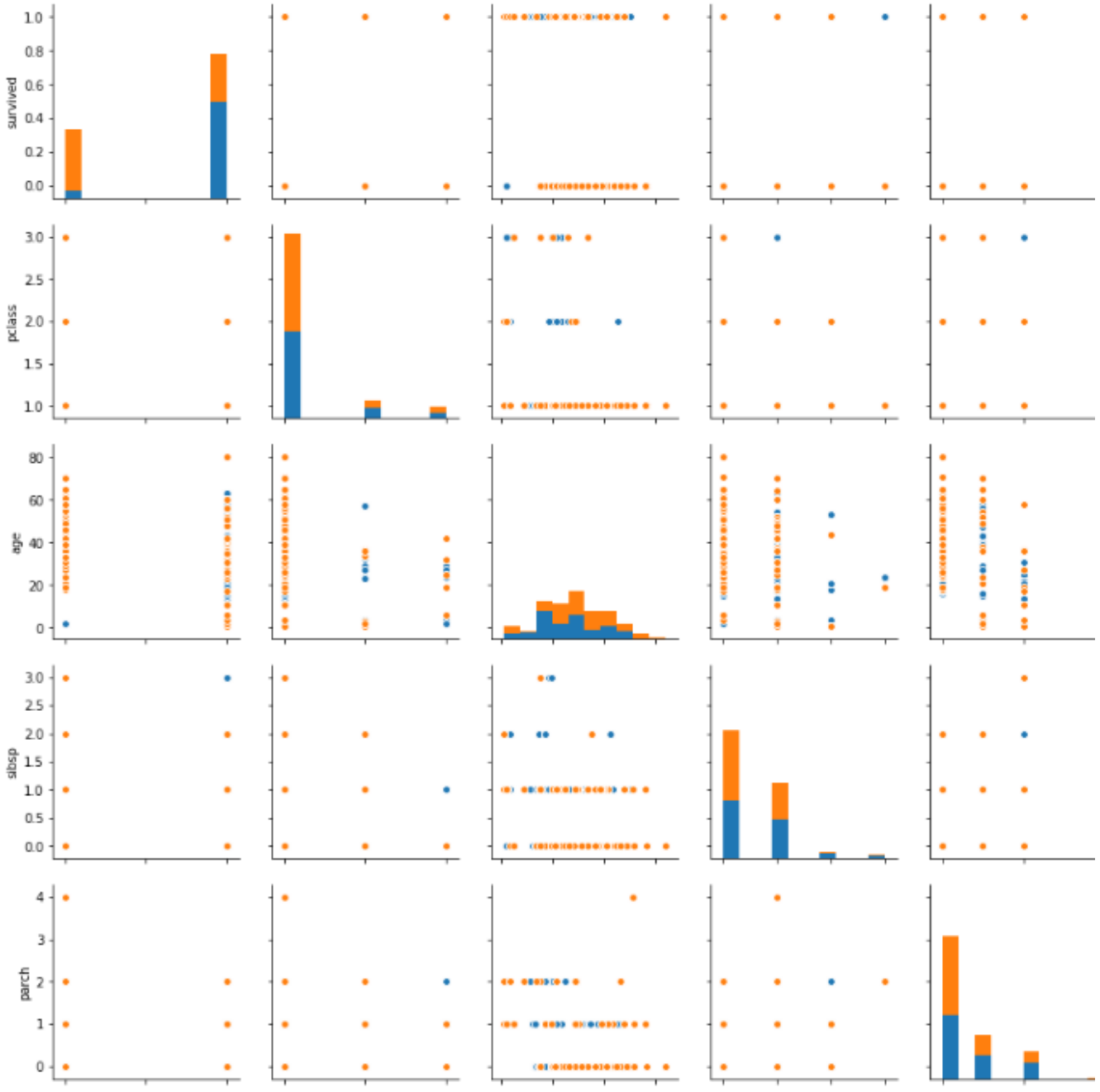


Note: Before executing the script above, remove all null values from the dataset using the following command: `dataset = dataset.dropna()` From the output of the pair plot you can see the joint plots for all the numeric and Boolean columns in the Titanic dataset.

To add information from the categorical column to the pair plot, you can pass the name of the categorical column to the hue parameter. For instance, if we want to plot the gender information on the pair plot, we can execute the following script:

```
sns.pairplot(dataset, hue='sex')
```

Output:

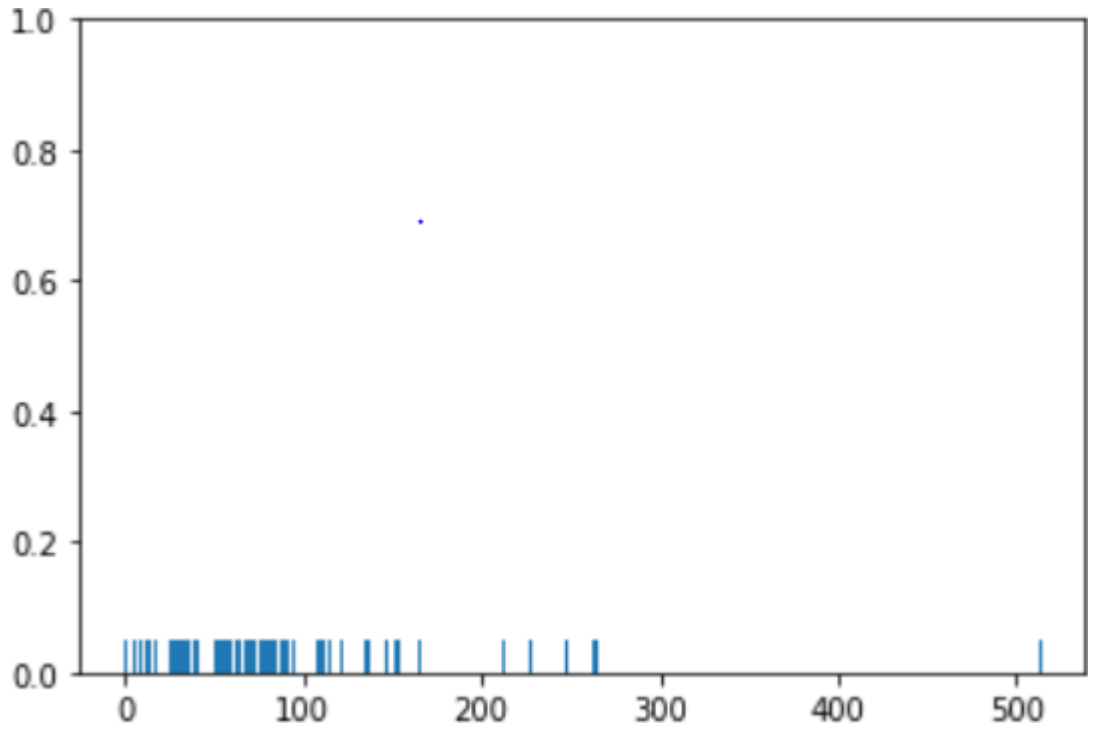


In the output, we can see the information about the males in orange and the information about the female in blue (as shown in the legend). From the joint plot on the top left, we can clearly see that among the surviving passengers, the majority were female.

The Rug Plot: The `rugplot()` is used to draw small bars along x-axis for each point in the dataset. To plot a rug plot, we need to pass the name of the column. Let's plot a rug plot for fare.

```
sns.rugplot(dataset[ 'fare' ])
```

Output:



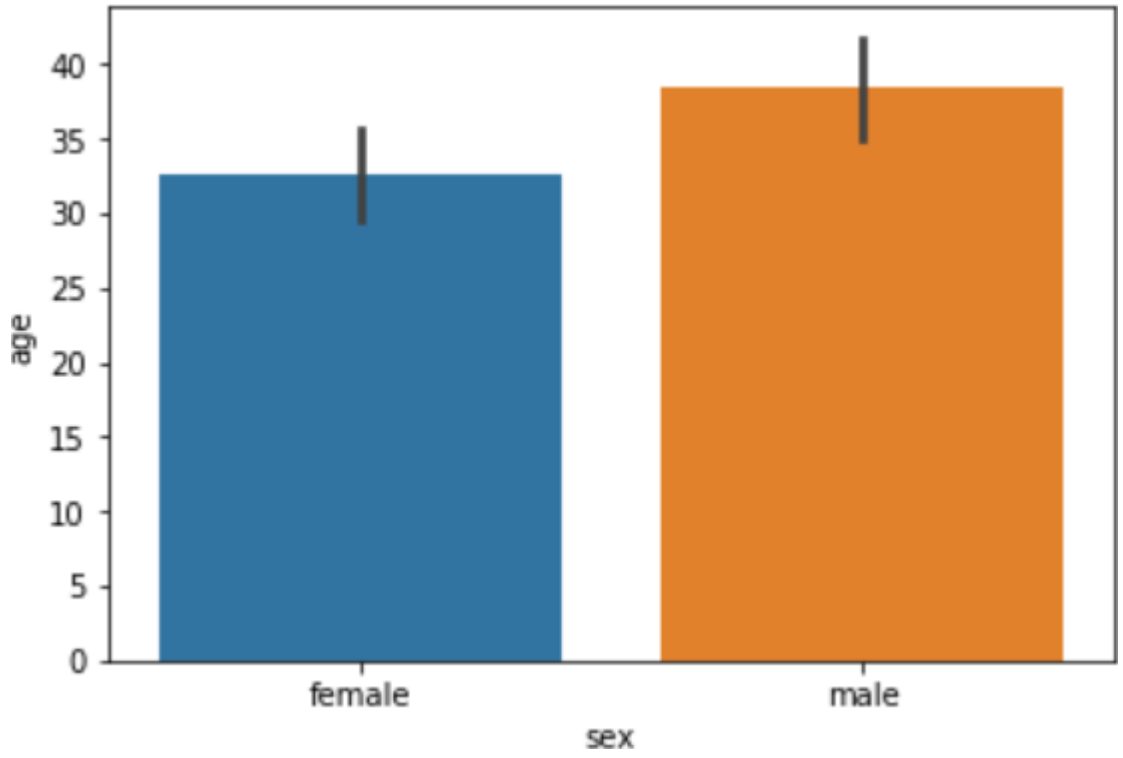
From the output, we can see that as was the case with the `distplot()`, most of the instances for the fares have values between 0 and 100. These are some of the most commonly used distribution plots offered by the Python's Seaborn Library. Let's see some of categorical plots in the Seaborn library.

Categorical Plots: Categorical plots, as the name suggests are normally used to plot categorical data. The categorical plots plot the values in the categorical column against another categorical column or a numeric column. Let's see some of the most commonly used categorical data.

The Bar Plot: The `barplot()` is used to display the mean value for each value in a categorical column, against a numeric column. The first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. For instance, if you want to know the mean value of the age of the male and female passengers, you can use the bar plot as follows.

```
sns.barplot(x= 'sex' , y= 'age' , data=dataset)
```

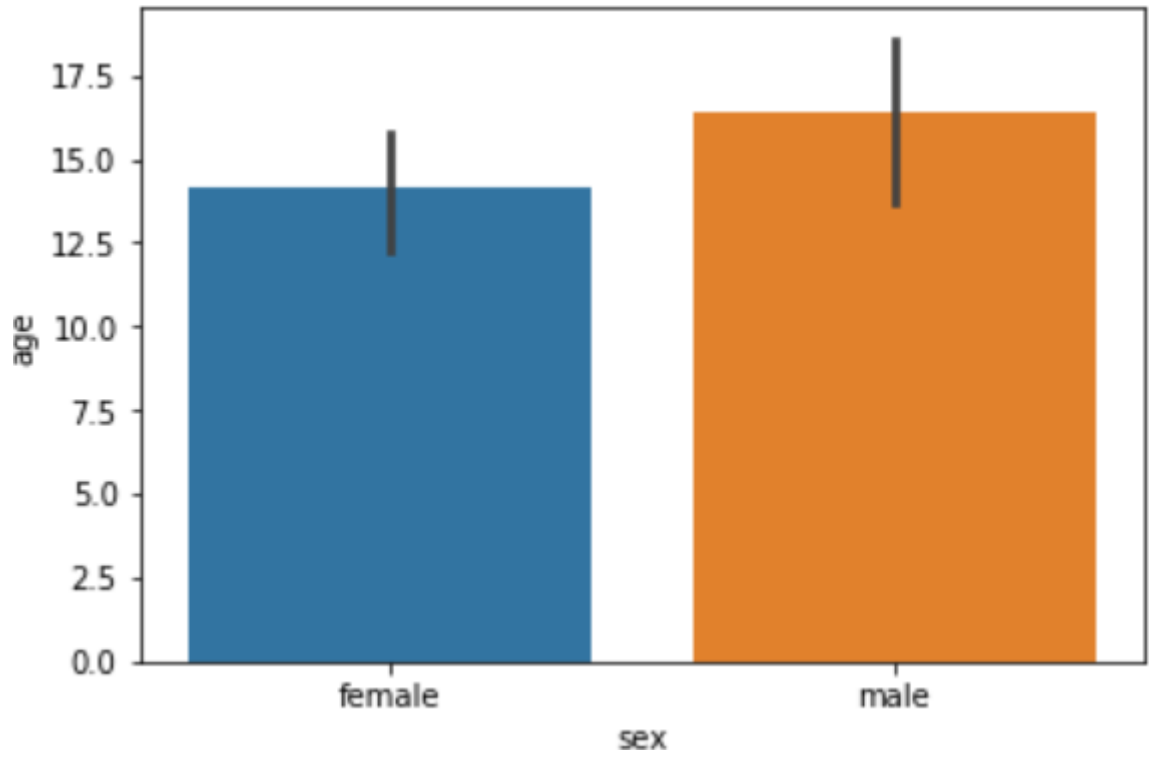
Output:



From the output, we can clearly see that the average age of male passengers is just less than 40 while the average age of female passengers is around 33. In addition to finding the average, the bar plot can also be used to calculate other aggregate values for each category. To do so, we need to pass the aggregate function to the `estimator`. For instance, we can calculate the standard deviation for the age of each gender as follows:

```
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

sns.barplot(x= 'sex' , y= 'age' , data=dataset, estimator=np.std)
```

In the above script, we use the `std` aggregate function from the `numpy` library to calculate the standard deviation for the ages of male and female passengers. The output looks like this:

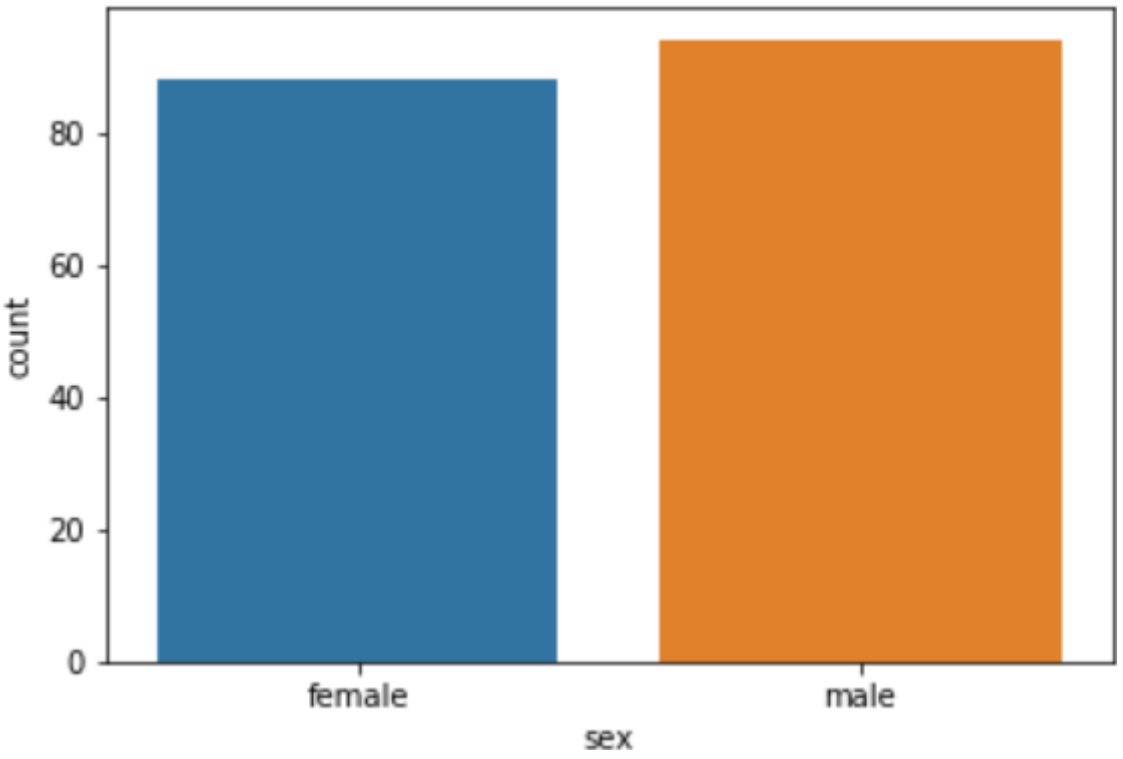


The Count Plot: The count plot is similar to the bar plot, however it displays the count of the categories in a specific column. For instance, if we want to count the number of males and women passenger we can do so using count plot as follows:

```
sns.countplot(x= 'sex' , data=dataset)
```

The output shows the count as follows:

Output:



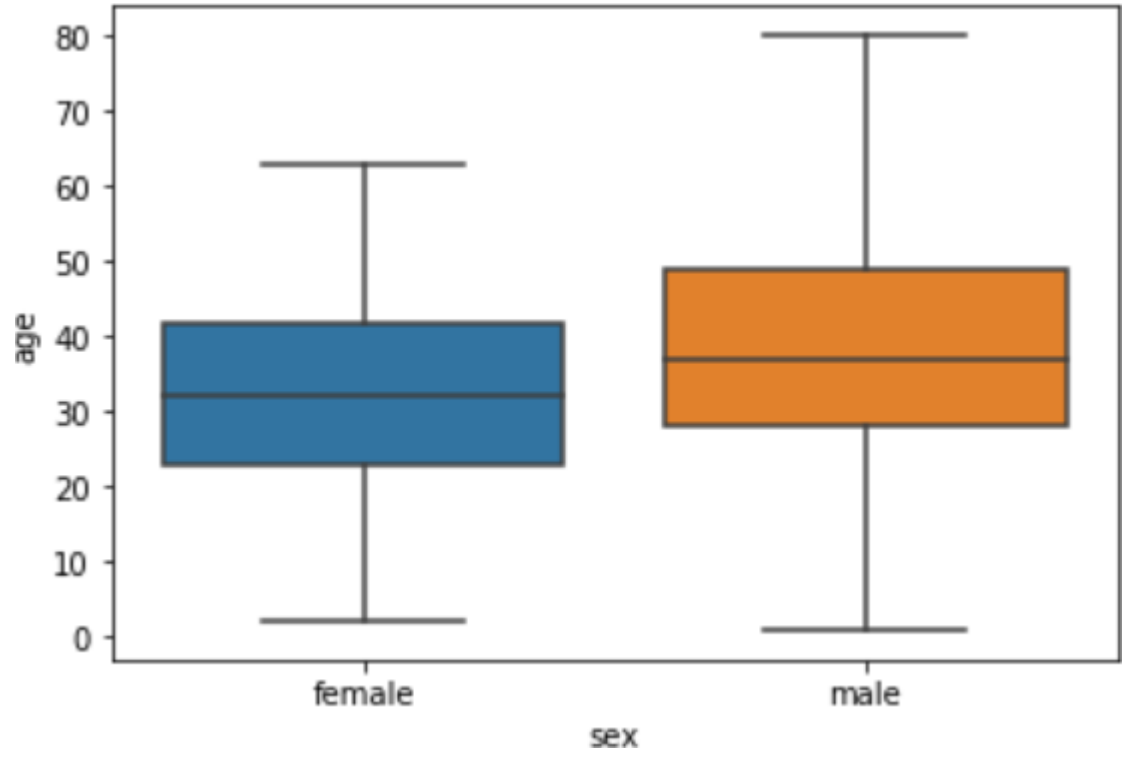
The Box Plot:

The box plot is used to display the distribution of the categorical data in the form of quartiles. The center of the box shows the median value. The value from the lower whisker to the bottom of the box shows the first quartile. From the bottom of the box to the middle of the box lies the second quartile. From the middle of the box to the top of the box lies the third quartile and finally from the top of the box to the top whisker lies the last quartile.

Now plot a box plot that displays the distribution for the age with respect to each gender. Here we need to pass the categorical column as the first parameter (which is sex in our case) and the numeric column (age in our case) as the second parameter. Finally, the dataset is passed as the third parameter, take a look at the following script:

```
sns.boxplot(x= 'sex' , y= 'age' , data=dataset)
```

Output:

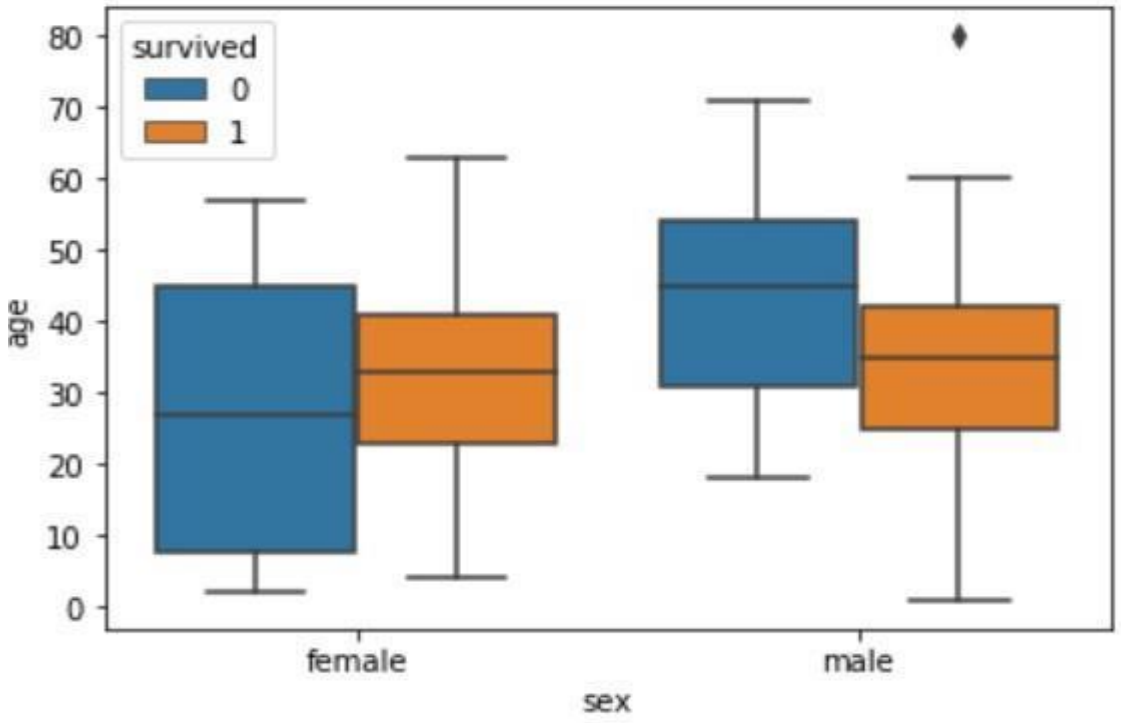


Now try to understand the box plot for female. The first quartile starts at around 5 and ends at 22 which means that 25% of the passengers are aged between 5 and 25. The second quartile starts at around 23 and ends at around 32 which means that 25% of the passengers are aged between 23 and 32. Similarly, the third quartile starts and ends between 34 and 42, hence 25% passengers are aged within this range and finally the fourth or last quartile starts at 43 and ends around 65. If there are any outliers or the passengers that do not belong to any of thequartiles, they are called outliers and are represented by dots on the box plot.

We can make our box plots more fancy by adding another layer of distribution. For instance, if we want to see the box plots of forage of passengers of both genders, along with the information about whether or not they survived, you can pass the `survived` as value to the `hue` parameter as shown below:

```
sns.boxplot(x= 'sex' , y= 'age' , data=dataset, hue= "survived" )
```

Output:



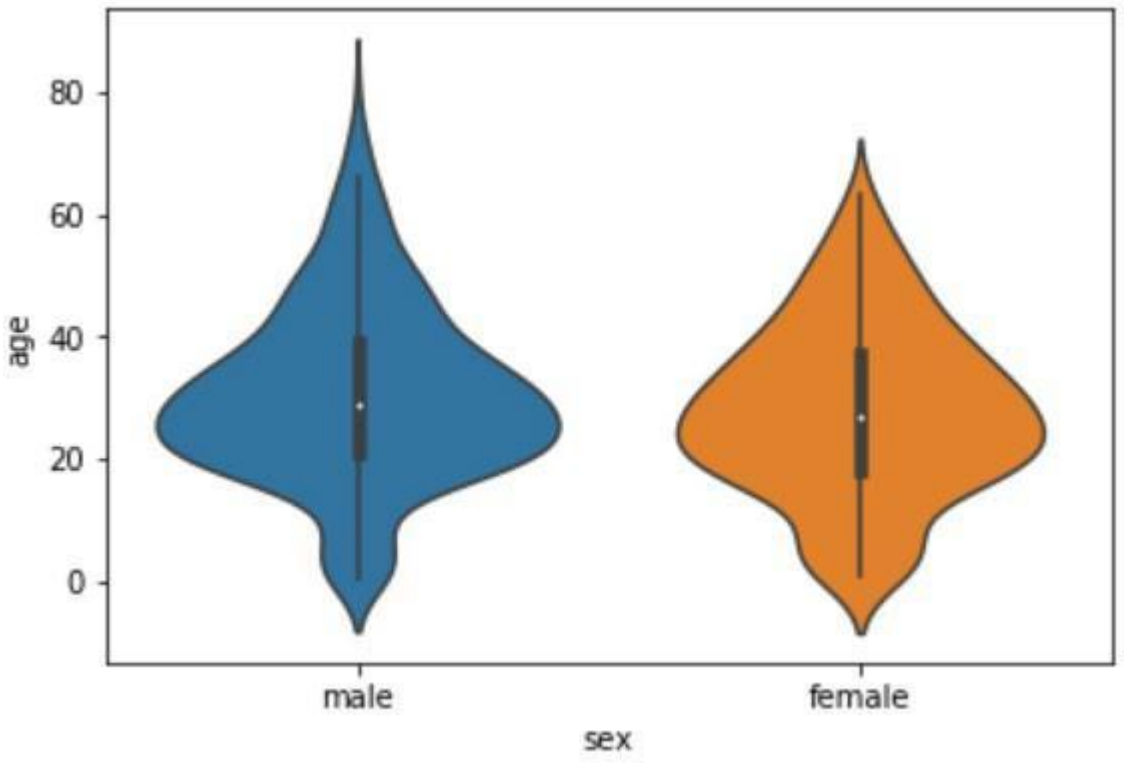
Now in addition to the information about the age of each gender, we can also see the distribution of the passengers who survived. For instance, we can see that among the male passengers, on average more younger people survived as compared to the older ones. Similarly, we can see that the variation among the age of female passengers who did not survive is much greater than the age of the surviving female passengers.

The Violin Plot: The violin plot is similar to the box plot, however, the violin plot allows us to display all the components that actually correspond to the data point.

The `violinplot()` function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column; the second parameter is the numeric column while the third parameter is the dataset. Now, plot a violin plot that displays the distribution for the age with respect to each gender.

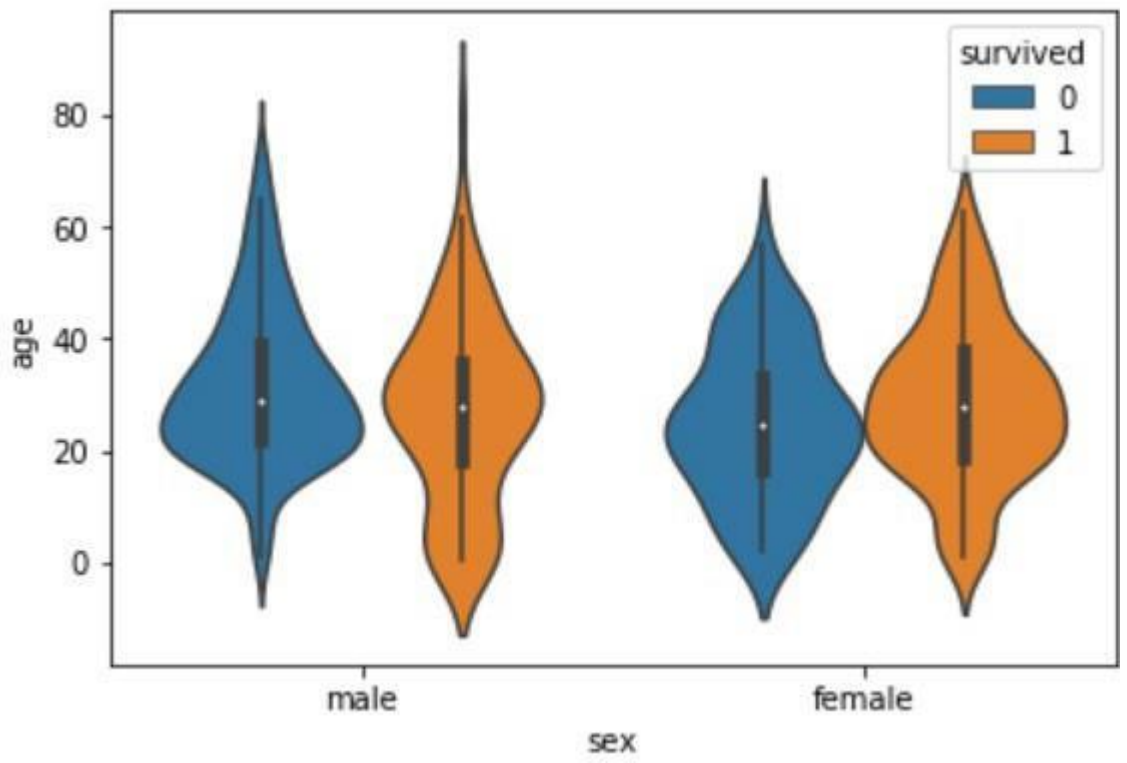
```
sns.violinplot(x= 'sex' , y= 'age' , data=dataset)
```

Output:



We can see from the figure above that violin plots provide much more information about the data as compared to the box plot. Instead of plotting the quartile, the violin plot allows us to see all the components that actually correspond to the data. The area where the violin plot is thicker has a higher number of instances for the age. For instance, from the violin plot for males, it is clearly evident that the number of passengers with age between 20 and 40 is higher than all the rest of the age brackets. Like box plots, we can also add another categorical variable to the violin plot using the `hue` parameter as shown below:

```
sns.violinplot(x= 'sex' , y= 'age' , data=dataset, hue= 'survived' )
```

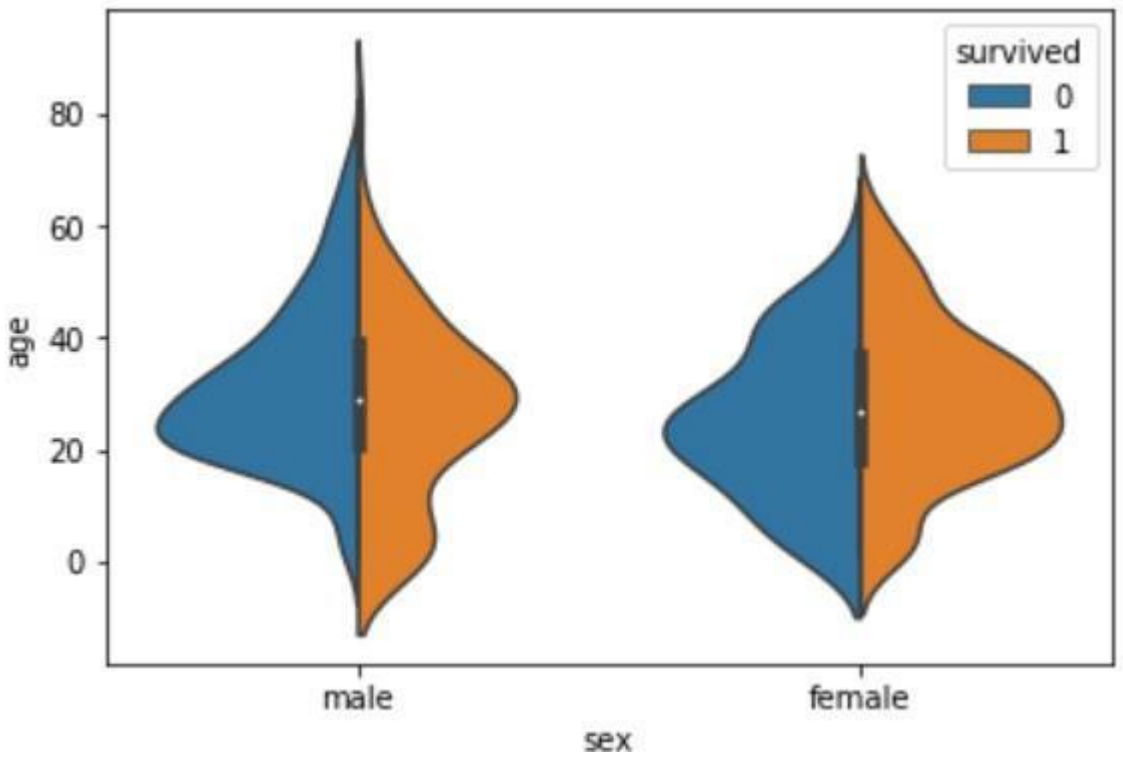


Now we can see a lot of information on the violin plot. For instance, if we look at the bottom of the violin plot for the males who survived (left-orange), you can see that it is thicker than the bottom of the violin plot for the males who didn't survive (left-blue). This means that the number of young male passengers who survived is greater than the number of young male passengers who did not survive. The violin plots convey a lot of information, however, on the downside, it takes a bit of time and effort to understand the violin plots.

Instead of plotting two different graphs for the passengers who survived and those who did not, you can have one violin plot divided into two halves, where one half represents survivingwhile the other half represents the non-surviving passengers. To do so, we need to pass `True` as value for the `split` parameter of the `violinplot()` function. Let's see how we can do this:

```
sns.violinplot(x= 'sex' , y= 'age' , data=dataset, hue= 'survived' , split= True )
```

The output looks like this:



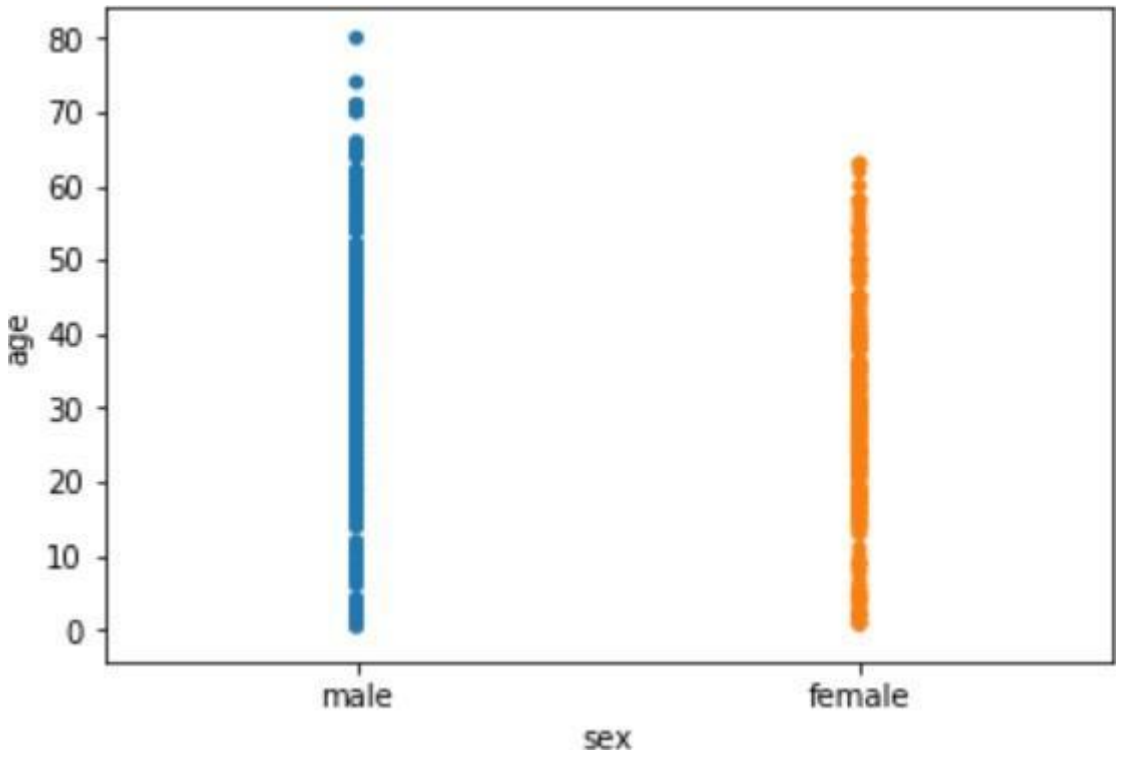
Now we can clearly see the comparison between the age of the passengers who survived and who did not for both males and females. Both violin and box plots can be extremely useful. However, as a rule of thumb if we are presenting our data to a non-technical audience, box plots should be preferred since they are easy to comprehend. On the other hand, if we are presenting our results to the research community it is more convenient to use violin plot to save space and to convey more information in less time.

The Strip Plot: The strip plot draws a scatter plot where one of the variables is categorical. We have seen scatter plots in the joint plot and the pair plot sections where we had two numeric variables. The strip plot is different in a way that one of the variables is categorical in this case, and for each category in the categorical variable, we will see scatter plot with respect to the numeric column.

The `stripplot()` function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. Look at the following script:

```
sns.stripplot(x= 'sex' , y= 'age' , data=dataset)
```

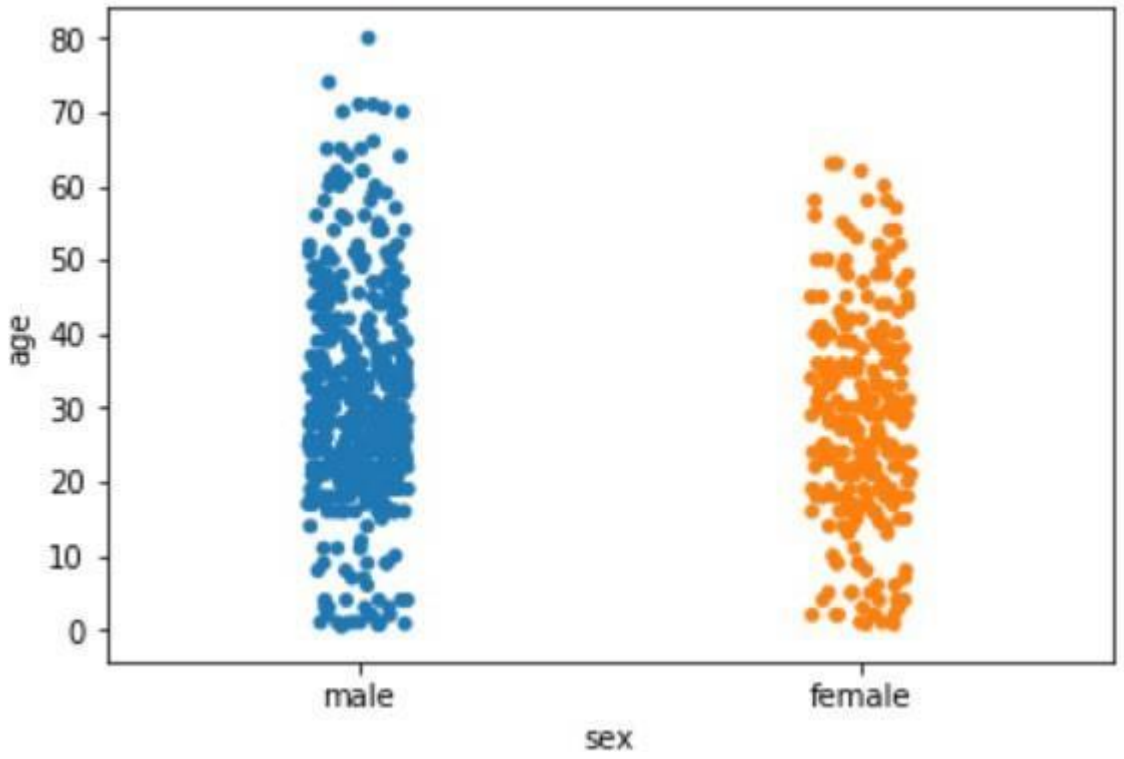
Output:



We can see the scattered plots of age for both males and females. The data points look like strips. It is difficult to comprehend the distribution of data in this form. To better comprehend the data, pass `True` for the `jitter` parameter which adds some random noise to the data. Look at the following script:

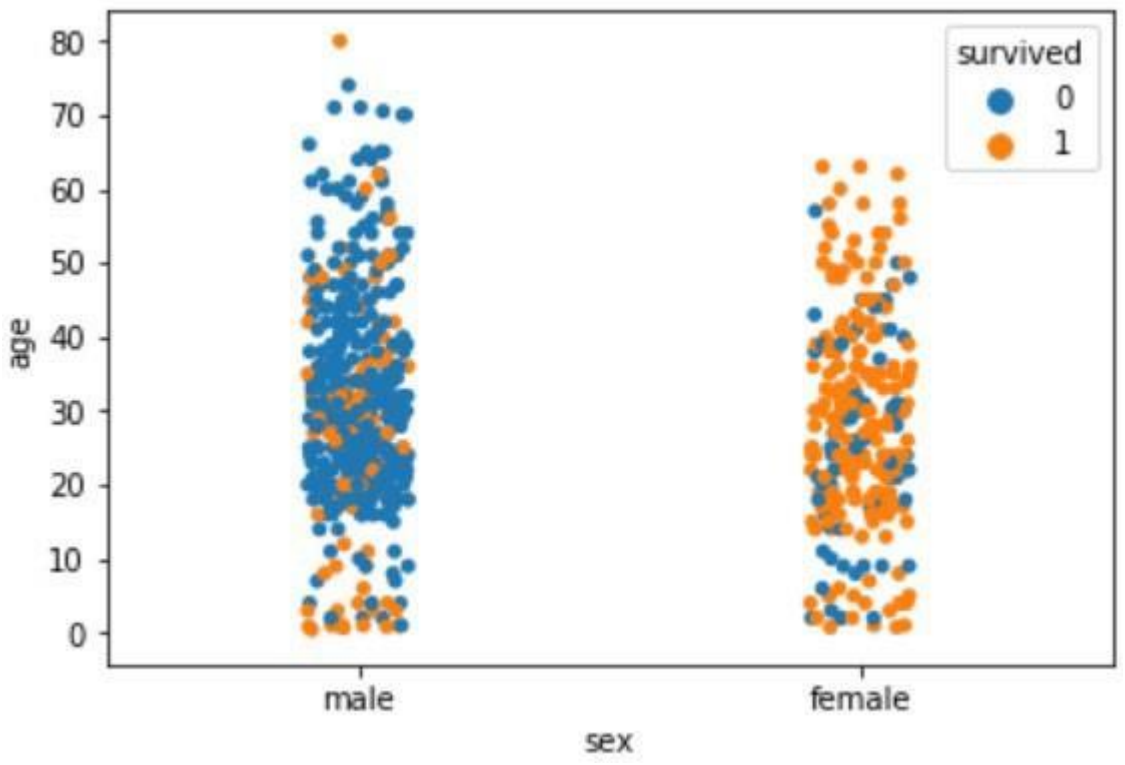
```
sns.stripplot(x= 'sex' , y= 'age' , data=dataset, jitter= True )
```

Output:



Now you have a better view for the distribution of age across the genders. Like violin and box plots, we can add an additional categorical column to strip plot using hue parameter as shown below:

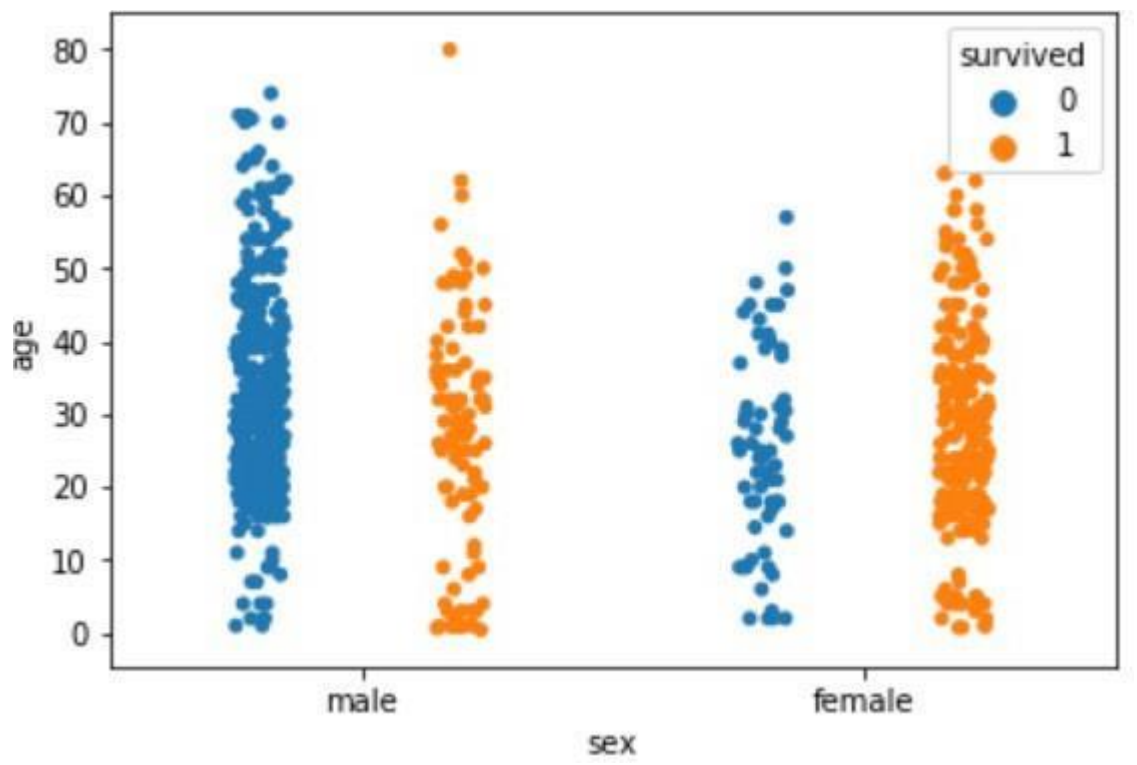
```
sns.stripplot(x= 'sex' , y= 'age' , data=dataset, jitter= True , hue= 'survived' )
```



Again we can see there are more points for the males who survived near the bottom of the plot compared to those who did not survive. Like violin plots, we can also split the strip plots. Execute the following script:

```
sns.stripplot(x= 'sex' , y= 'age' , data=dataset, jitter= True , hue= 'survived' , split= True )
```

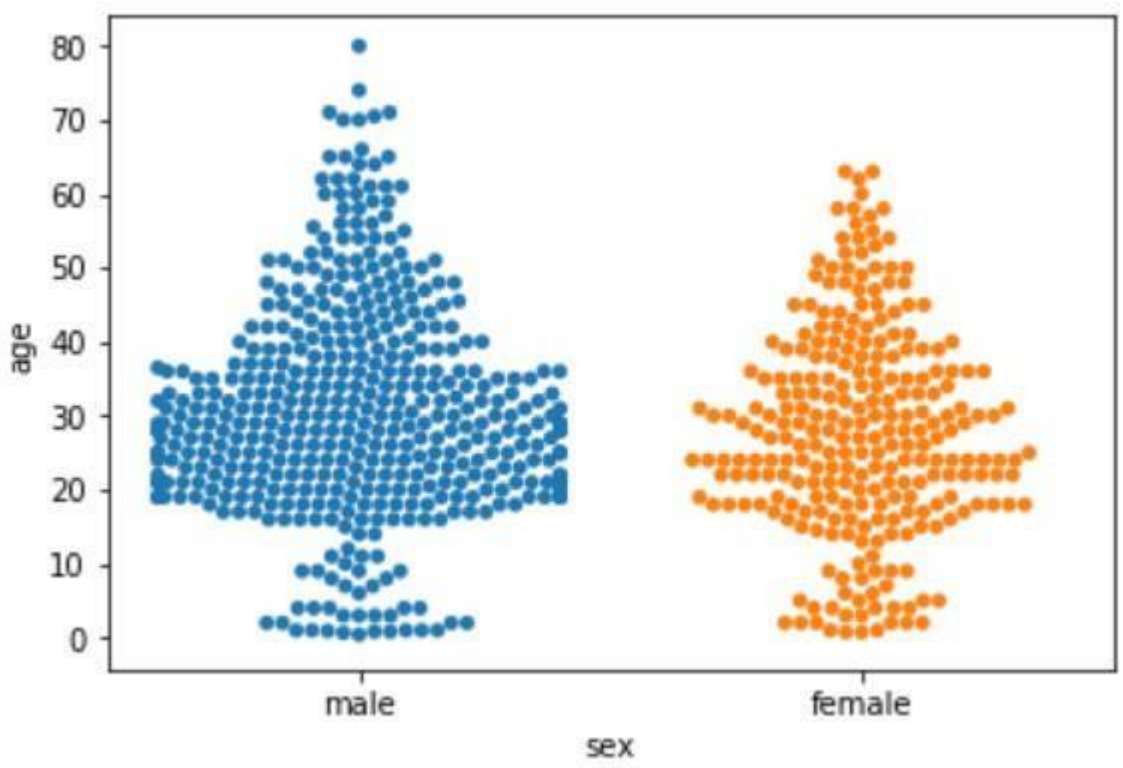
Output:



Now we can clearly see the difference in the distribution for the age of both male and female passengers who survived and those who did not survive.

The Swarm Plot: The swarm plot is a combination of the strip and the violin plots. In the swarm plots, the points are adjusted in such a way that they don't overlap. Let's plot a swarm plot for the distribution of age against gender. The `swarmplot()` function is used to plot the violin plot. Like the box plot, the first parameter is the categorical column, the second parameter is the numeric column while the third parameter is the dataset. Look at the following script:

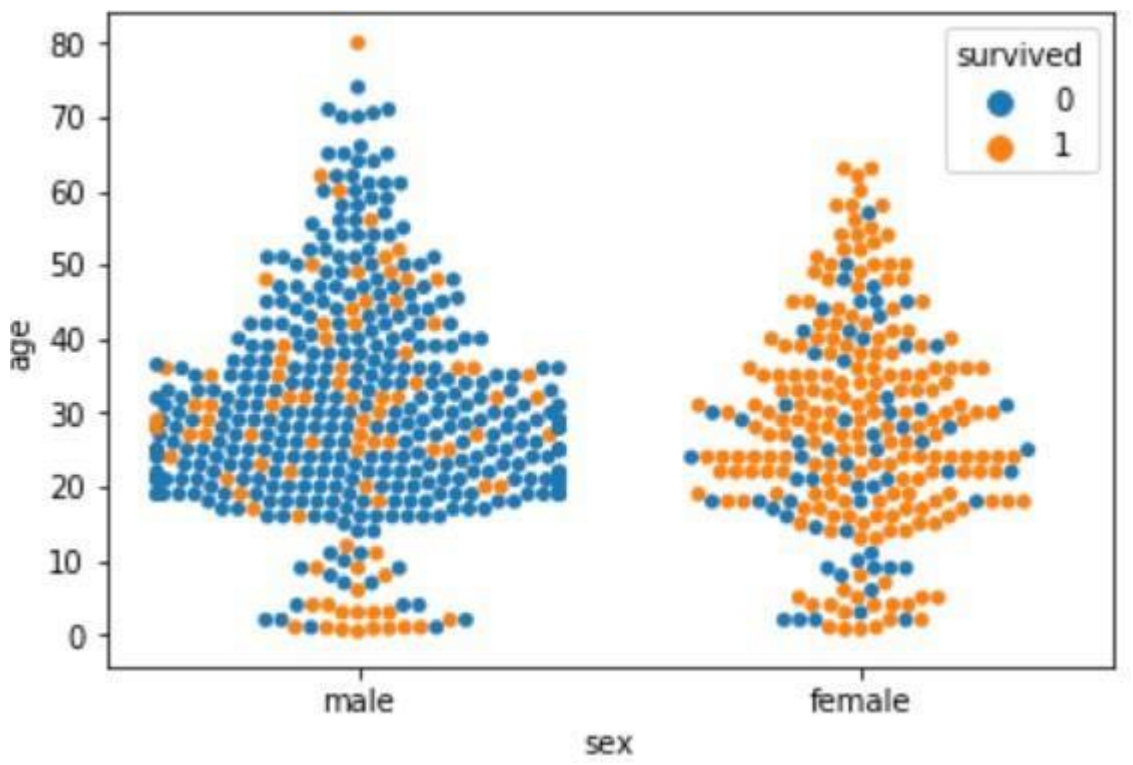
```
sns.swarmplot(x= 'sex' , y= 'age' , data=dataset)
```



We can clearly see that the above plot contains scattered data points like the strip plot and the data points are not overlapping. Rather they are arranged to give a view similar to that of a violin plot. Now add another categorical column to the swarm plot using the `hue` parameter.

```
sns.swarmplot(x= 'sex' , y= 'age' , data=dataset, hue= 'survived' )
```

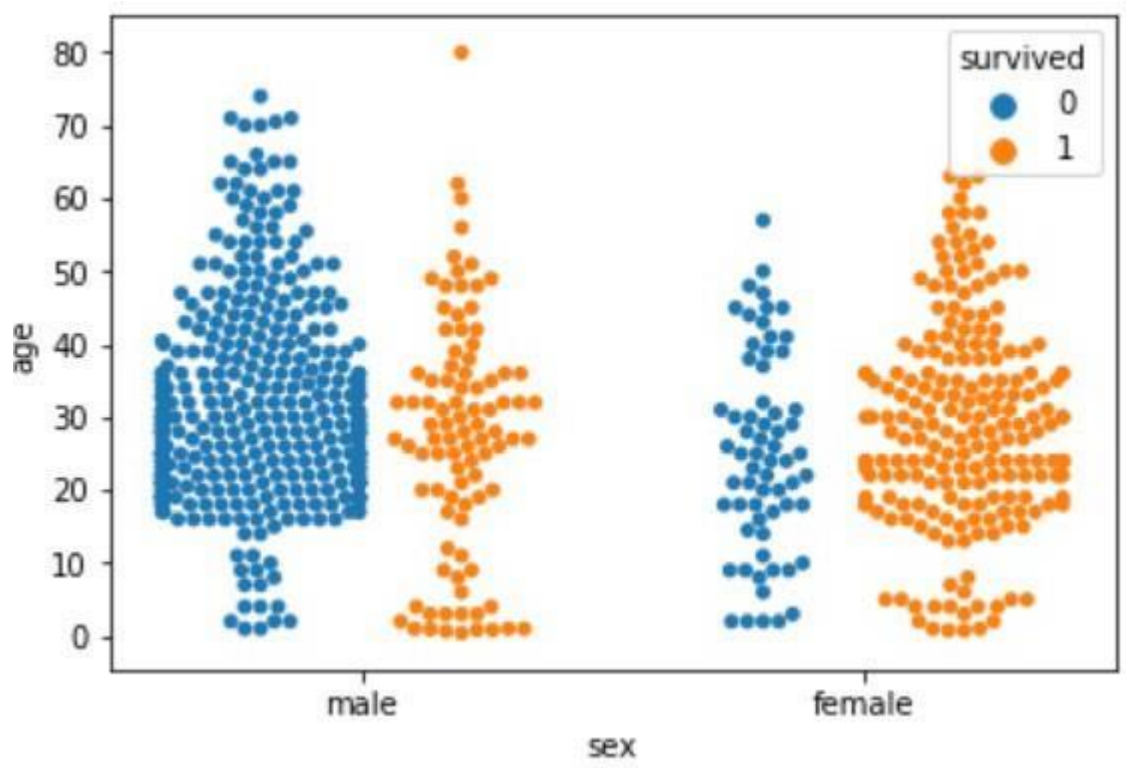
Output:



From the output, it is evident that the ratio of surviving males is less than the ratio of surviving females. Since for the male plot, there are more blue points and less orange points. On the other hand, for females, there are more orange points (surviving) than the blue points (not surviving). Another observation is that amongst males of age less than 10, more passengers survived as compared to those who didn't. We can also split swarm plots as we did in the case of strip and box plots. Execute the following script to do so:

```
sns.swarmplot(x= 'sex' , y= 'age' , data=dataset, hue= 'survived' , split= True )
```

Output:

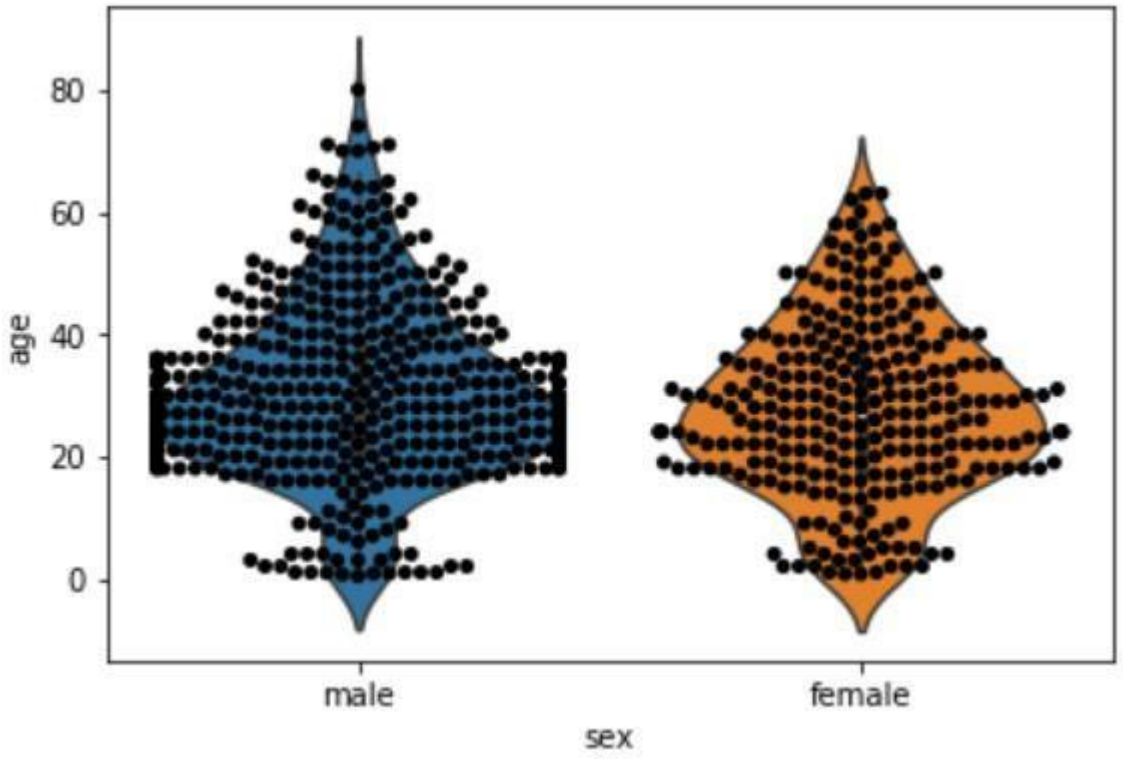


Now we can clearly see that more women survived, as compared to men.

Combining Swarm and Violin Plots: Swarm plots are not recommended if you have a huge dataset since they do not scale well because they have to plot each data point. If you really like swarm plots, a better way is to combine two plots. For instance, to combine a violin plot with swarm plot, you need to execute the following script:

```
sns.violinplot(x='sex', y='age', data=dataset)
sns.swarmplot(x='sex', y='age', data=dataset, color='black')
```

Output:



There are also a lot of other visualization libraries for Python that have features that go beyond what Seaborn can do.

Conclusion: Seaborn is an advanced data visualization library built on top of Matplotlib library. In this article, we looked at how we can draw distributional and categorical plots using Seaborn library. This is Part 1 of the series of article on Seaborn. In the second article of the series, we will see how we play around with grid functionalities in Seaborn and how we can draw Matrix and Regression plots in Seaborn.

Practical No.9	Group A
Title	**Data Visualization II** 1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names : 'sex' and 'age') 2. Write observations on the inference from the above statistics.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 9

Title: Data Visualization, II

- ## Problem Statement:

Data Visualization II

1. Use the inbuilt dataset 'titanic' as used in the above problem. Plot a box plot for distribution of age with respect to each gender along with the information about whether they survived or not. (Column names: 'sex' and 'age')

2. Write observations on the inference from the above statistics.

- ## Prerequisites:

Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Theory: Exploratory Data Analysis

There are various techniques to understand your data, And the basic need is you should have the knowledge of Numpy for mathematical operations and Pandas for data manipulation. We are using Titanic dataset. For demonstrating some of the techniques we will also use an inbuilt dataset of seaborn as tips data which explains the tips each waiter gets from different customers.

Import libraries and loading Data

```
import numpy as np
import pandas pd
import matplotlib.pyplot as plt
import seaborn as sns
from seaborn import load_dataset
#titanic dataset
data = pd.read_csv("titanic_train.csv")
#tips dataset
tips = load_dataset("tips")
```

Univariate Analysis

Univariate analysis is the simplest form of analysis where we explore a single variable. Univariate analysis is performed to describe the data in a better way. we perform Univariate analysis of Numerical and categorical variables differently because plotting uses different plots.

Categorical Data:

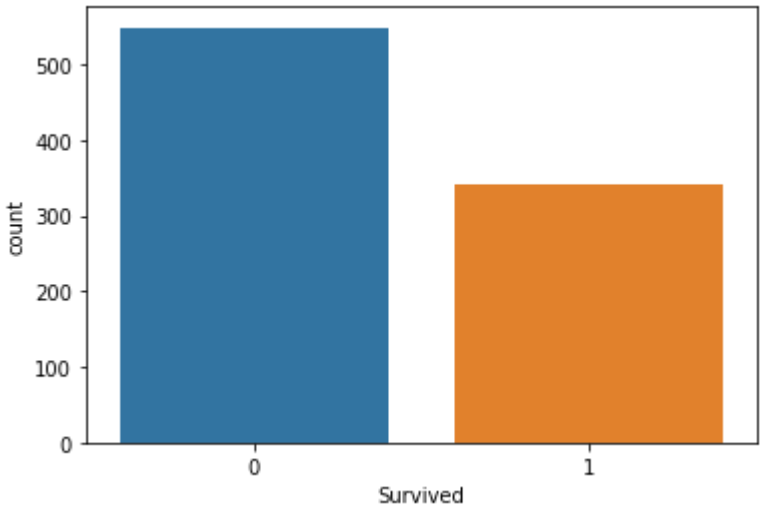
A variable that has text-based information is referred to as categorical variables. Now following are various plots which we can use for visualizing Categorical data.

1) CountPlot:

Countplot is basically a count of frequency plot in form of a bar graph. It plots the count of each category in a separate bar. When we use the pandas' value counts function on any

column. It is the same visual form of the value counts function. In our data-target variable is survived and it is categorical so plot a countplot of this.

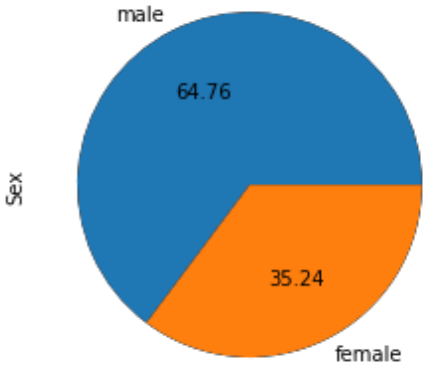
```
sns.countplot(data['Survived'])
plt.show()
```



2) Pie Chart:

The pie chart is also the same as the countplot, only gives us additional information about the percentage presence of each category in data means which category is getting how much weightage in data. Now we check about the Sex column, what is a percentage of Male and Female members traveling.

```
data['Sex'].value_counts().plot(kind="pie", autopct="%.2f")
plt.show()
```



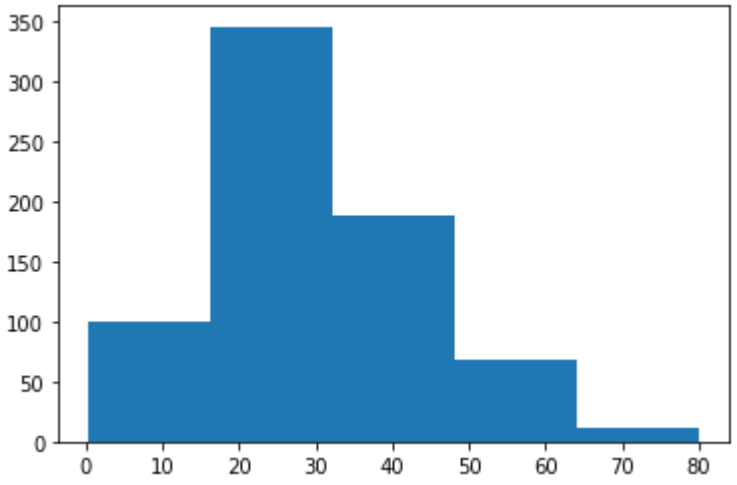
Numerical Data:

Analyzing Numerical data is important because understanding the distribution of variables helps to further process the data. Most of the time, we will find much inconsistency with numerical data so we have to explore numerical variables.

1) Histogram:

A histogram is a value distribution plot of numerical columns. It basically creates bins in various ranges in values and plots it where we can visualize how values are distributed. We can have a look where more values lie like in positive, negative, or at the center(mean). Let's have a look at the Age column.

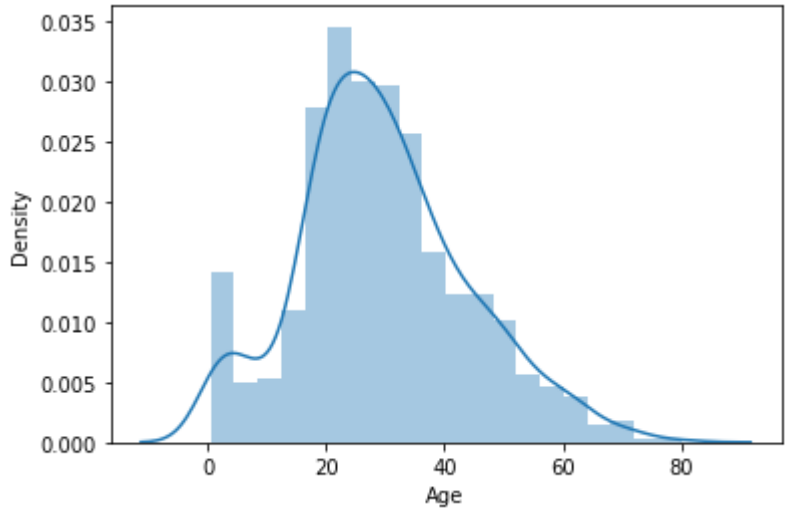
```
plt.hist(data['Age'], bins=5)
plt.show()
```



2) Distplot:

Distplot is also known as the second Histogram because it is a slight improvement version of the Histogram. Distplot gives us a KDE(Kernel Density Estimation) over histogram which explains PDF(Probability Density Function) which means what is the probability of each value occurring in this column.

```
sns.distplot(data['Age'])
plt.show()
```



3) Boxplot:

Boxplot is a very interesting plot that basically plots a 5 number summary. to get 5 number summary some terms we need to describe.

- Median – Middle value in series after sorting
- Percentile – Gives any number which is number of values present before this percentile like for example 50 under 25th percentile so it explains total of 50 values are there below 25th percentile
- Minimum and Maximum – These are not minimum and maximum values, rather they describe the lower and upper boundary of standard deviation which is calculated using Interquartile range(IQR).

```
IQR = Q3 - Q1
Lower_boundary = Q1 - 1.5 * IQR
Upper_bounday = Q3 +  1.5 * IQR
```

Here Q1 and Q3 is 1st quantile (25th percentile) and 3rd Quantile(75th percentile).

Bivariate/ Multivariate Analysis:

We have study about various plots to explore single categorical and numerical data. Bivariate Analysis is used when we have to explore the relationship between 2 different variables and we have to do this because, in the end, our main task is to explore the relationship between variables to build a powerful model. And when we analyze more than 2 variables together then it is known as Multivariate Analysis. we will work on different plots for Bivariate aswell on Multivariate Analysis.

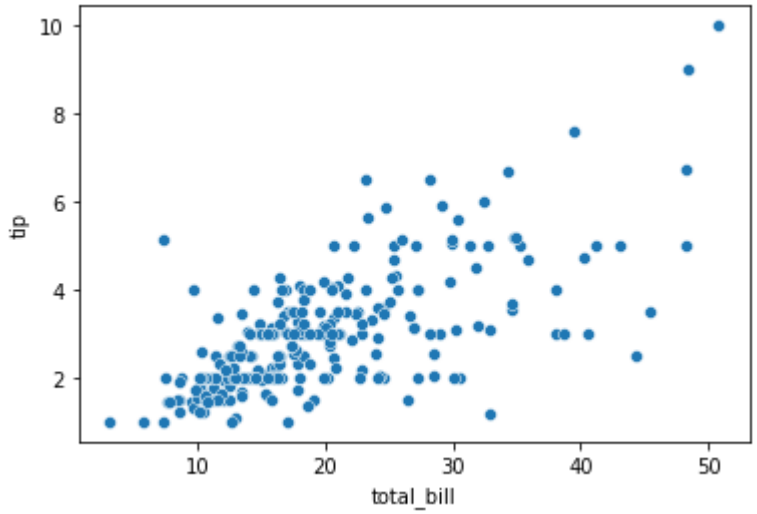
Explore the plots when both the variable is numerical.

1) Scatter Plot:

To plot the relationship between two numerical variables scatter plot is a simple plot to do.

Let us see the relationship between the total bill and tip provided using a scatter plot.

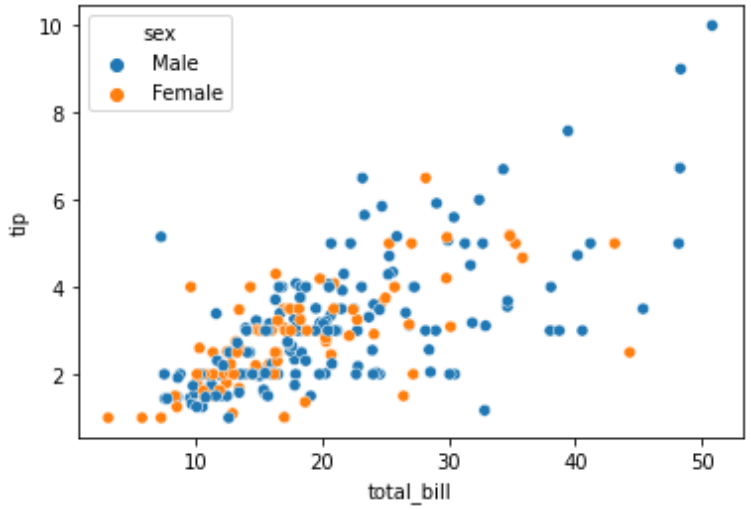
```
sns.scatterplot(tips["total_bill"], tips["tip"])
```



Multivariate analysis with scatter plot:

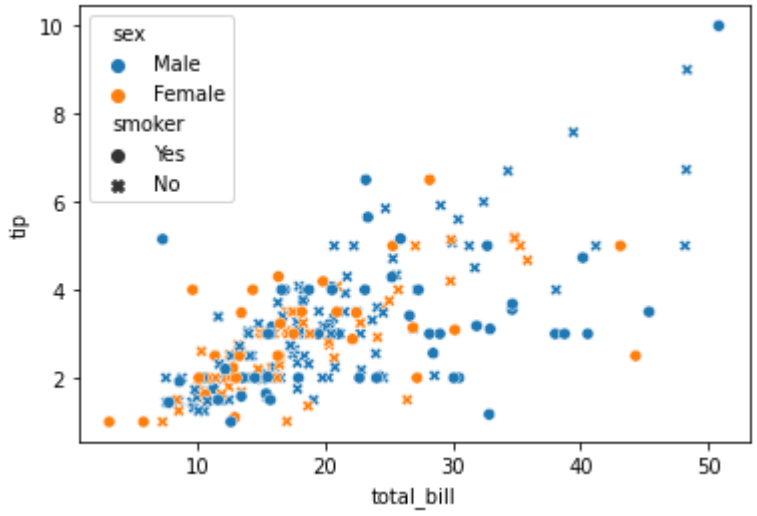
We can also plot 3 variable or 4 variable relationships with scatter plot. suppose we want to find the separate ratio of male and female with total bill and tip provided.

```
sns.scatterplot(tips["total_bill"], tips["tip"], hue=tips["sex"])
plt.show()
```



We can also see 4 variable multivariate analyses with scatter plots using style argument. Suppose along with gender we also want to know whether the customer was a smoker or not so we can do this.

```
sns.scatterplot(tips["total_bill"], tips["tip"], hue=tips["sex"],
style=tips['smoker'])
plt.show()
```



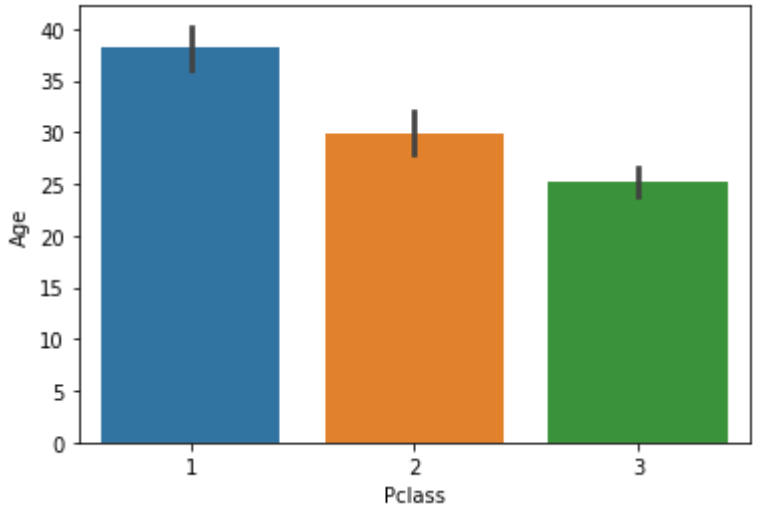
Numerical and Categorical:

If one variable is numerical and one is categorical then there are various plots that we can use for Bivariate and Multivariate analysis.

1) Bar Plot:

Bar plot is a simple plot which we can use to plot categorical variable on the x-axis and numerical variable on y-axis and explore the relationship between both variables. The blacktip on top of each bar shows the confidence Interval. let us explore P-Class with age.

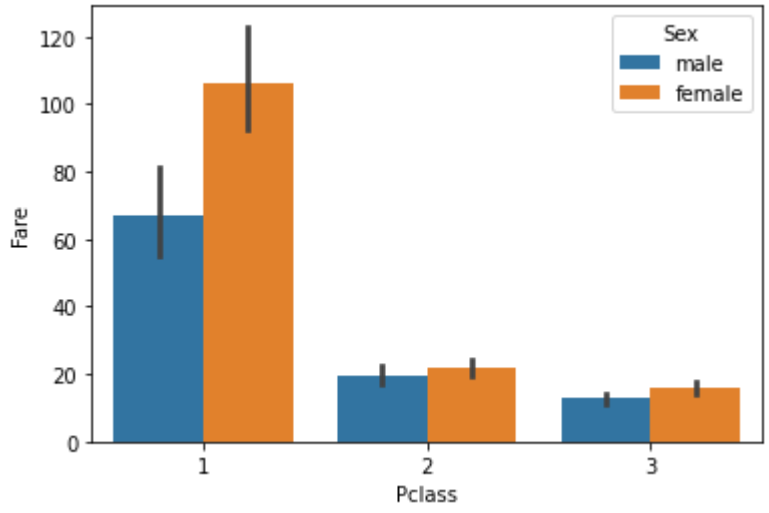
```
sns.barplot(data['Pclass'], data['Age'])
plt.show()
```



Multivariate analysis using Bar plot:

Hue's argument is very useful which helps to analyze more than 2 variables. Now along with the above relationship we want to see with gender.

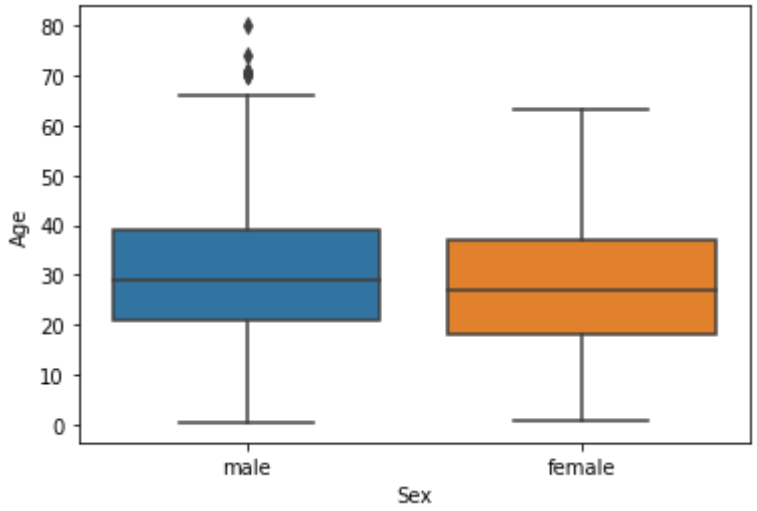
```
sns.barplot(data['Pclass'], data['Fare'], hue = data["Sex"])
plt.show()
```



2) Boxplot:

We have already study about boxplots in the Univariate analysis above. we can draw a separate boxplot for both the variable. let us explore gender with age using a boxplot.

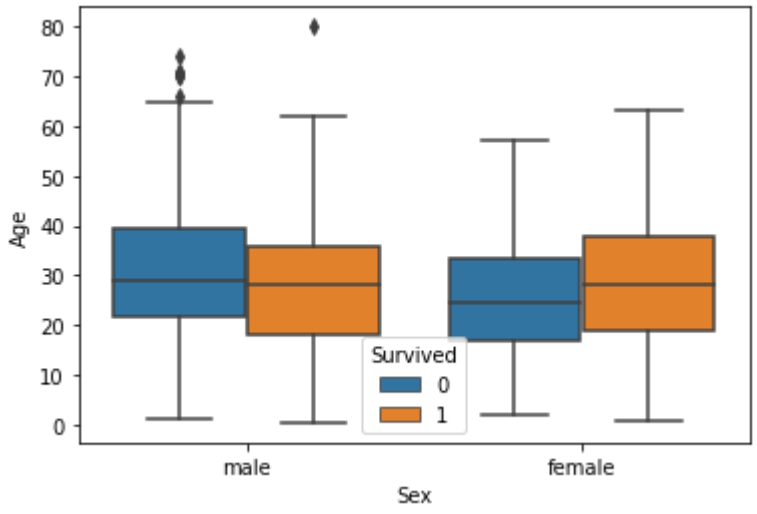
```
sns.boxplot(data['Sex'], data["Age"])
```



Multivariate analysis with boxplot:

Along with age and gender let's see who has survived and who has not.

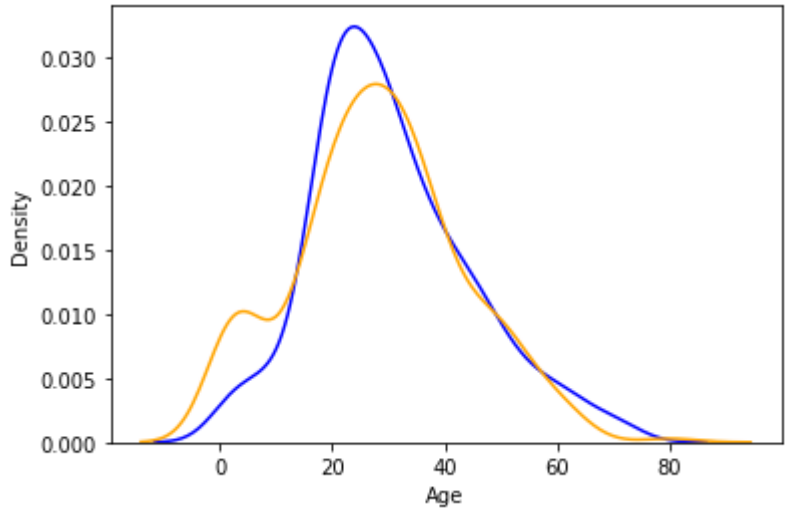
```
sns.boxplot(data['Sex'], data["Age"], data["Survived"])
plt.show()
```



3) Distplot:

Distplot explains the PDF function using kernel density estimation. Distplot does not have a hue parameter but we can create it. Suppose we want to see the probability of people with an age range that of survival probability and find out whose survival probability is high to the age range of death ratio.

```
sns.distplot(data[data['Survived'] == 0]['Age'], hist=False, color="blue")
sns.distplot(data[data['Survived'] == 1]['Age'], hist=False, color="orange")
plt.show()
```



In above graph, the blue one shows the probability of dying and the orange plot shows the survival probability. If we observe it we can see that children's survival probability is higher than death and which is the opposite in the case of aged peoples. This small analysis tells sometimes some big things about data and it helps while preparing data stories.

Categorical and Categorical:

Now, we will work on categorical and categorical columns.

1) Heatmap:

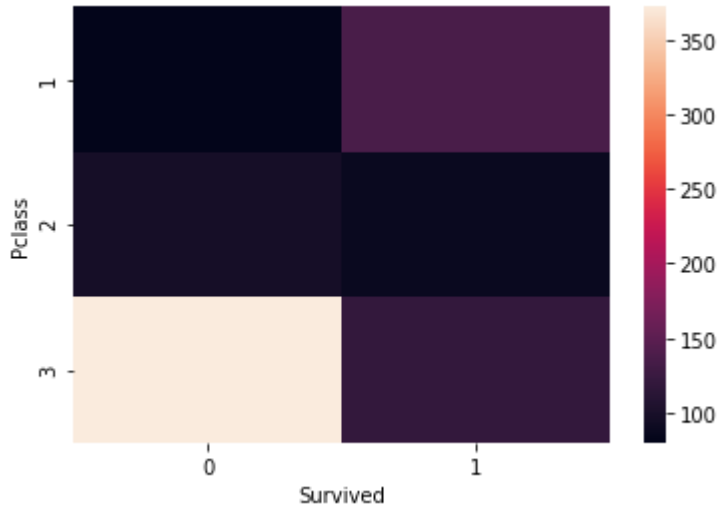
If you have ever used a crosstab function of pandas then Heatmap is a similar visual representation of that only. It basically shows that how much presence of one category concerning another category is present in the dataset. let me show first with crosstab and then with heatmap.

```
pd.crosstab(data['Pclass'], data['Survived'])
```

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

Now with heatmap, we have to find how many people survived and died.

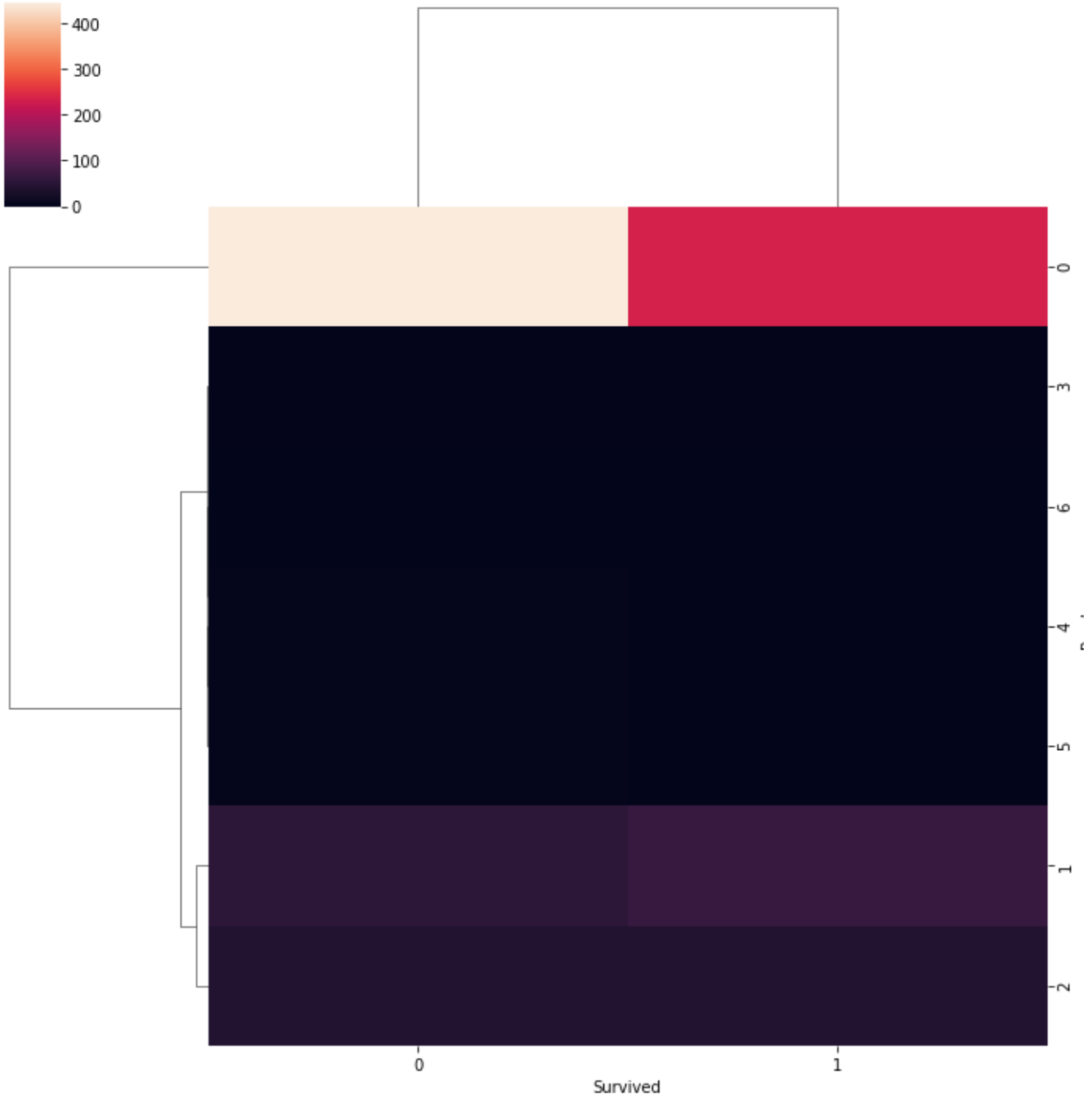
```
sns.heatmap(pd.crosstab(data['Pclass'], data['Survived']))
```



2) Cluster map:

We can also use a cluster map to understand the relationship between two categorical variables. A cluster map basically plots a dendrogram that shows the categories of similar behavior together.

```
sns.clustermap(pd.crosstab(data['Parch'], data['Survived']))
plt.show()
```


400
300
200
100
0
0
3
6
4
5
1
2
0
1
Survived

Practical No.10	Group A
Title	**Data Visualization III** Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as: 1. List down the features and their types (e.g., numeric, nominal) available in the dataset. 2. Create a histogram for each feature in the dataset to illustrate the feature distributions. 3. Create a box plot for each feature in the dataset. 4. Compare distributions and identify outliers.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 10

Title: Data Visualization III

- **Problem Statement:**

Data Visualization III

Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., https://archive.ics.uci.edu/ml/datasets/Iris). Scan the dataset and give the inference as:

1. List down the features and their types (e.g., numeric, nominal) available in the dataset.
2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
3. Create a box plot for each feature in the dataset.
4. Compare distributions and identify outliers.

- **Prerequisites:**

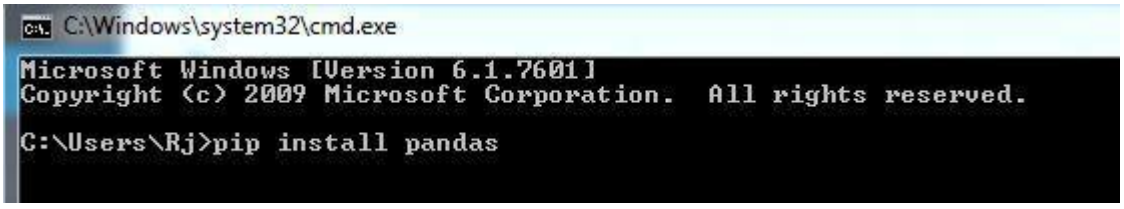
Subjects: Discrete Mathematics, Database Management Systems **Tools/Software Requirements:** Jupyter Notebook / Google Colab / Kaggle **Processor/Hardware Requirements:** Intel Core 2 Duo or above, 2 GB RAM or above **Operating System:** 64 Bit Windows/Linux/Mac OS
Programming Language: Python 3.8, PIP
Datasets: Any Open Source Dataset/CSV Files

Theory: Python – Basics of Pandas using Iris Dataset

Python language is one of the most trending programming languages as it is dynamic than others. Python is a simple high-level and an open-source language used for general-purpose programming. It has many open-source libraries and Pandas is one of them. Pandas is a powerful, fast, flexible open-source library used for data analysis and manipulations of data frames/datasets. Pandas can be used to read and write data in a dataset of different formats like CSV(comma separated values), txt, xls(Microsoft Excel) etc. Following are the various features of Pandas in Python and how to use it to practice.**Prerequisites:** Basic knowledge about coding in Python.

Installation:

So if you are new to practice Pandas, then firstly you should install Pandas on your system. Go to Command Prompt and run it as administrator. Make sure you are connected with an internet connection to download and install it on your system. Then type "**pip install pandas**", then press Enter key.



Download the Dataset **"Iris.csv"**.

Iris dataset is the Hello World for the Data Science, so if you have started your career in Data Science and Machine Learning you will be practicing basic ML algorithms on this famous dataset. Iris dataset contains five columns such as Petal Length, Petal Width, Sepal Length, Sepal Width and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flower sand recorded digitally.

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
1	5.1	3.5	1.4	0.2
2	4.9	3	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4
7	4.6	3.4	1.4	0.3
8	5	3.4	1.5	0.2
9	4.4	2.9	1.4	0.2
10	4.9	3.1	1.5	0.1
11	5.4	3.7	1.5	0.2
12	4.8	3.4	1.6	0.2
13	4.8	3	1.4	0.1
14	4.3	3	1.1	0.1
15	5.8	4	1.2	0.2
16	5.7	4.4	1.5	0.4
17	5.4	3.9	1.3	0.4
18	5.1	3.5	1.4	0.3
19	5.7	3.8	1.7	0.3

Getting Started with Pandas:
Code: Importing pandas to use in our code as pd.

- Python3

```
import pandas as pd
```

Code: Reading the dataset “Iris.csv”.

- Python3

```
data = pd.read_csv("your downloaded dataset location ")
```

```
import pandas as pd
data=pd.read_csv("C:\\Users\\Rj\\Desktop\\Iris.csv")
```

Code: Displaying up the top rows of the dataset with their columns
The function head() will display the top rows of the dataset, the default value of this function

is 5, that is it will show top 5 rows when no argument is given to it.

- Python3

```
data.head()
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Displaying the number of rows randomly.
In sample() function, it will also display the rows according to arguments given, but it will display the rows randomly.

- Python3

```
data.sample(10)
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
108	109	6.7	2.5	5.8	1.8	Iris-virginica
139	140	6.9	3.1	5.4	2.1	Iris-virginica
10	11	5.4	3.7	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
132	133	6.4	2.8	5.6	2.2	Iris-virginica
99	100	5.7	2.8	4.1	1.3	Iris-versicolor
140	141	6.7	3.1	5.6	2.4	Iris-virginica
1	2	4.9	3.0	1.4	0.2	Iris-setosa
107	108	7.3	2.9	6.3	1.8	Iris-virginica
42	43	4.4	3.2	1.3	0.2	Iris-setosa

Code: Displaying the number of columns and names of the columns.
The column() function prints all the columns of the dataset in a list form.

- Python3

```
data.columns
```

Output:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

Code: Displaying the shape of the dataset.
The shape of the dataset means to print the total number of rows or entries and the total number of columns or features of that particular dataset.

- Python3

```
#The first one is the number of rows and

# the other one is the number of columns.
```

data.shape

Output:

```
(150, 6)
```

Code: Display the whole dataset

- Python3

print(data)

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Code: Slicing the rows.
Slicing means if you want to print or work upon a particular group of lines that is from 10th row to 20th row.

- Python3

```
#data[start:end]

#start is inclusive whereas end is exclusive

print(data[10:21])

# it will print the rows from 10 to 20.



# you can also save it in a variable for further use in analysis

sliced_data=data[10:21]

print(sliced_data)
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa
15	16	5.7	4.4	1.5	0.4	Iris-setosa
16	17	5.4	3.9	1.3	0.4	Iris-setosa
17	18	5.1	3.5	1.4	0.3	Iris-setosa
18	19	5.7	3.8	1.7	0.3	Iris-setosa
19	20	5.1	3.8	1.5	0.3	Iris-setosa
20	21	5.4	3.4	1.7	0.2	Iris-setosa

Code: Displaying only specific columns.
In any dataset, it is sometimes needed to work upon only specific features or columns, so we can do this by the following code.

- Python3

```
#here in the case of Iris dataset
```

#we will save it in a another variable named "specific_data"

specific_data=data[["Id","Species"]]

#data[["column_name1","column_name2","column_name3"]]

#now we will print the first 10 columns of the specific_data dataframe.

print(specific_data.head(10))

Output:

	Id	Species
0	1	Iris-setosa
1	2	Iris-setosa
2	3	Iris-setosa
3	4	Iris-setosa
4	5	Iris-setosa
5	6	Iris-setosa
6	7	Iris-setosa
7	8	Iris-setosa
8	9	Iris-setosa
9	10	Iris-setosa

Filtering:Displaying the specific rows using “iloc” and “loc” functions.
The “loc” functions use the index name of the row to display the particular row of the dataset.
The “iloc” functions use the index integer of the row, which gives complete information about the row.
Code:

- Python3

#here we will use iloc

data.iloc[5]

#it will display records only with species "Iris-setosa".

data.loc[data["Species"] == "Iris-setosa"]

Output:

```
Id                          6
SepalLengthCm             5.4
SepalWidthCm              3.9
PetalLengthCm             1.7
PetalWidthCm              0.4
Species           Iris-setosa
Name: 5, dtype: object
```

iloc()[/caption]

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa

loc()

Code: Counting the number of counts of unique values using "value_counts()".
The value_counts() function, counts the number of times a particular instance or data has occurred.

- Python3

#In this dataset we will work on the Species column, it will count number of times a particular species has occurred.

data["Species"].value_counts()

#it will display in descending order.

Output:

```
Iris-versicolor    50
Iris-setosa        50
Iris-virginica     50
Name: Species, dtype: int64
```

Calculating sum, mean and mode of a particular column.
We can also calculate the sum, mean and mode of any integer columns as I have done in the following code.

- Python3

```
# data["column_name"].sum()



sum_data = data["SepalLengthCm"].sum()

mean_data = data["SepalLengthCm"].mean()

median_data = data["SepalLengthCm"].median()



print("Sum:",sum_data, "\nMean:", mean_data, "\nMedian:",median_data)
```

Output:

```
Sum: 876.5
Mean: 5.843333333333335
Median: 5.8
```

Code: Extracting minimum and maximum from a column.
Identifying minimum and maximum integer, from a particular column or row can also be done in a dataset.

- Python3

```
min_data=data["SepalLengthCm"].min()
```

```
max_data=data["SepalLengthCm"].max()



print("Minimum:",min_data, "\nMaximum:", max_data)
```

Output:

```
Minimum: 4.3
Maximum: 7.9
```

Code: Adding a column to the dataset.

If want to add a new column in our dataset, as we are doing any calculations or extracting some information from the dataset, and if you want to save it a new column. This can be done by the following code by taking a case where we have added all integer values of all columns.

- Python3

```
# For example, if we want to add a column let say "total_values",

# that means if you want to add all the integer value of that particular

# row and get total answer in the new column "total_values".

# first we will extract the columns which have integer values.

cols = data.columns



# it will print the list of column names.

print(cols)



# we will take that columns which have integer values.
```

```
cols = cols[1:5]



# we will save it in the new dataframe variable

data1 = data[cols]



# now adding new column "total_values" to dataframe data.

data["total_values"]=data1[cols].sum(axis=1)



# here axis=1 means you are working in rows,

# whereas axis=0 means you are working in columns.
```

Output:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species', 'total_values'],
      dtype='object')
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	total_valu
0	1	5.1	3.5	1.4	0.2	Iris-setosa	1
1	2	4.9	3.0	1.4	0.2	Iris-setosa	
2	3	4.7	3.2	1.3	0.2	Iris-setosa	
3	4	4.6	3.1	1.5	0.2	Iris-setosa	
4	5	5.0	3.6	1.4	0.2	Iris-setosa	1
...	...	...	...	...	...	...	
145	146	6.7	3.0	5.2	2.3	Iris-virginica	1
146	147	6.3	2.5	5.0	1.9	Iris-virginica	1
147	148	6.5	3.0	5.2	2.0	Iris-virginica	1
148	149	6.2	3.4	5.4	2.3	Iris-virginica	1
149	150	5.9	3.0	5.1	1.8	Iris-virginica	1

150 rows × 7 columns

Code: Renaming the columns.

Renaming our column names can also be possible in python pandas libraries. We have used the rename() function, where we have created a dictionary “newcols” to update our new column names. The following code illustrates that.

- Python3

```
newcols={ "Id":"id",

"SepalLengthCm":"sepallength"

"SepalWidthCm":"sepalwidth"}



data.rename(columns=newcols,inplace=True)
```

```
print(data.head())
```

Output:

	id	sepallength	sepalwidth	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Formatting and Styling:
Conditional formatting can be applied to your dataframe by using Dataframe.style function. Styling is used to visualize your data, and most convenient way of visualizing your dataset is in tabular form.
Here we will highlight the minimum and maximum from each row and columns.

- Python3

```
#this is an example of rendering a datagram,

which is not visualised by any styles.

data.style
```

Output:

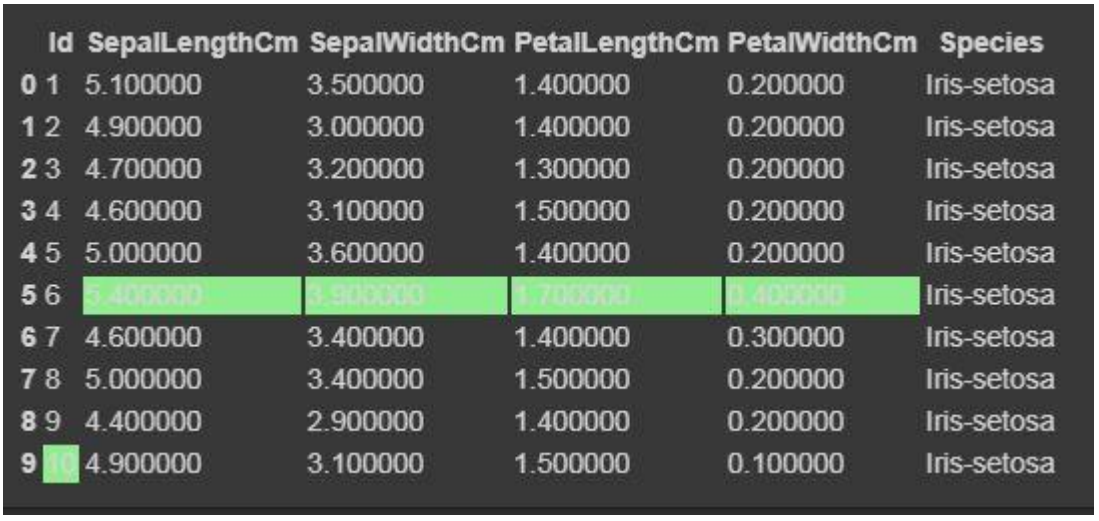
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa

Now we will highlight the maximum and minimum column-wise, row-wise, and the whole dataframe wise using Styler.apply function. The Styler.apply function passes each column or row of the dataframe depending upon the keyword argument axis. For column-wise use axis=0, row-wise use axis=1, and for the entire table at once use axis=None.

- Python3

```
# we will here print only the top 10 rows of the dataset,
# if you want to see the result of the whole dataset remove
#.head(10) from the below code

data.head(10).style.highlight_max(color='lightgreen', axis=0)

data.head(10).style.highlight_max(color='lightgreen', axis=1)

data.head(10).style.highlight_max(color='lightgreen', axis=None)
```

Output:



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa
5	6	5.400000	3.900000	1.700000	0.400000	Iris-setosa
6	7	4.600000	3.400000	1.400000	0.300000	Iris-setosa
7	8	5.000000	3.400000	1.500000	0.200000	Iris-setosa
8	9	4.400000	2.900000	1.400000	0.200000	Iris-setosa
9	10	4.900000	3.100000	1.500000	0.100000	Iris-setosa

for axis=0

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa
5	6	5.400000	3.900000	1.700000	0.400000	Iris-setosa
6	7	4.600000	3.400000	1.400000	0.300000	Iris-setosa
7	8	5.000000	3.400000	1.500000	0.200000	Iris-setosa
8	9	4.400000	2.900000	1.400000	0.200000	Iris-setosa
9	10	4.900000	3.100000	1.500000	0.100000	Iris-setosa

for axis=1

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa
5	6	5.400000	3.900000	1.700000	0.400000	Iris-setosa
6	7	4.600000	3.400000	1.400000	0.300000	Iris-setosa
7	8	5.000000	3.400000	1.500000	0.200000	Iris-setosa
8	9	4.400000	2.900000	1.400000	0.200000	Iris-setosa
9	10	4.900000	3.100000	1.500000	0.100000	Iris-setosa

for axis=None

Code: Cleaning and detecting missing values
In this dataset, we will now try to find the missing values i.e NaN, which can occur due to several reasons.

- Python3

```
data.isnull()

#if there is data is missing, it will display True else False.
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

isnull()

Code: Summarizing the missing values.
We will display how many missing values are present in each column.

- Python3

```
data.isnull.sum()
```

Output:

```
Id              0
SepalLengthCm   0
SepalWidthCm    0
PetalLengthCm   0
PetalWidthCm    0
Species         0
dtype: int64
```

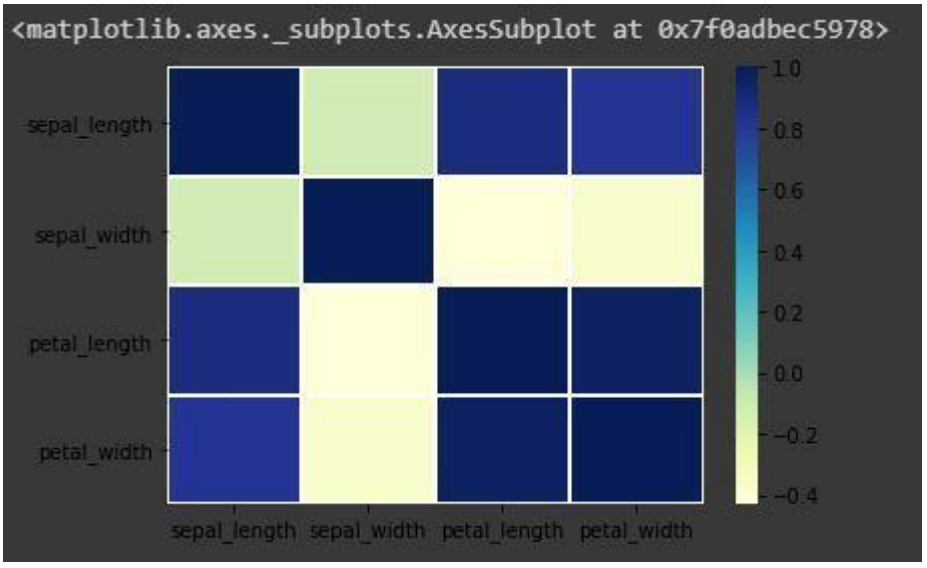
Heatmap: Importing seaborn
The heatmap is a data visualisation technique which is used to analyse the dataset as colors in two dimensions. Basically it shows correlation between all numerical variables in the dataset. Heatmap is an attribute of the Seaborn library.

Code:

- Python3

```
import seaborn as sns


iris = sns.load_dataset("iris")

sns.heatmap(iris.corr(),camp = "YlGnBu", linecolor = 'white', linewidths = 1)
```

Output:

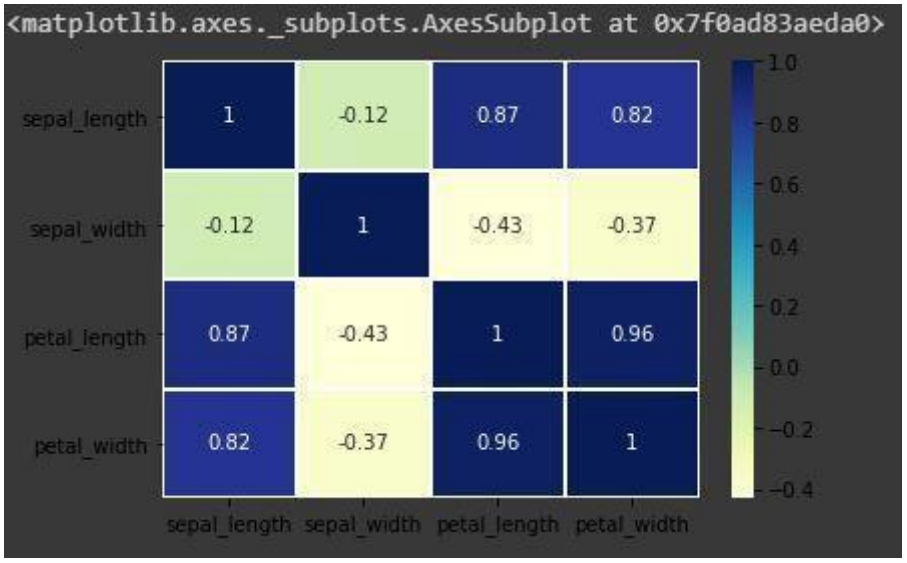


Code: Annotate each cell with the numeric value using integer formatting

- Python3

```
sns.heatmap(iris.corr(),camp = "YlGnBu", linecolor = 'white', linewidths = 1, annot = True )
```

Output:



heatmap with annot=True

Pandas Dataframe Correlation:
Pandas correlation is used to determine pairwise correlation of all the columns of the dataset. In dataframe.corr(), the missing values are excluded and non-numeric columns are also ignored.
Code:

- Python3

```
data.corr(method='pearson')
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

data.corr()

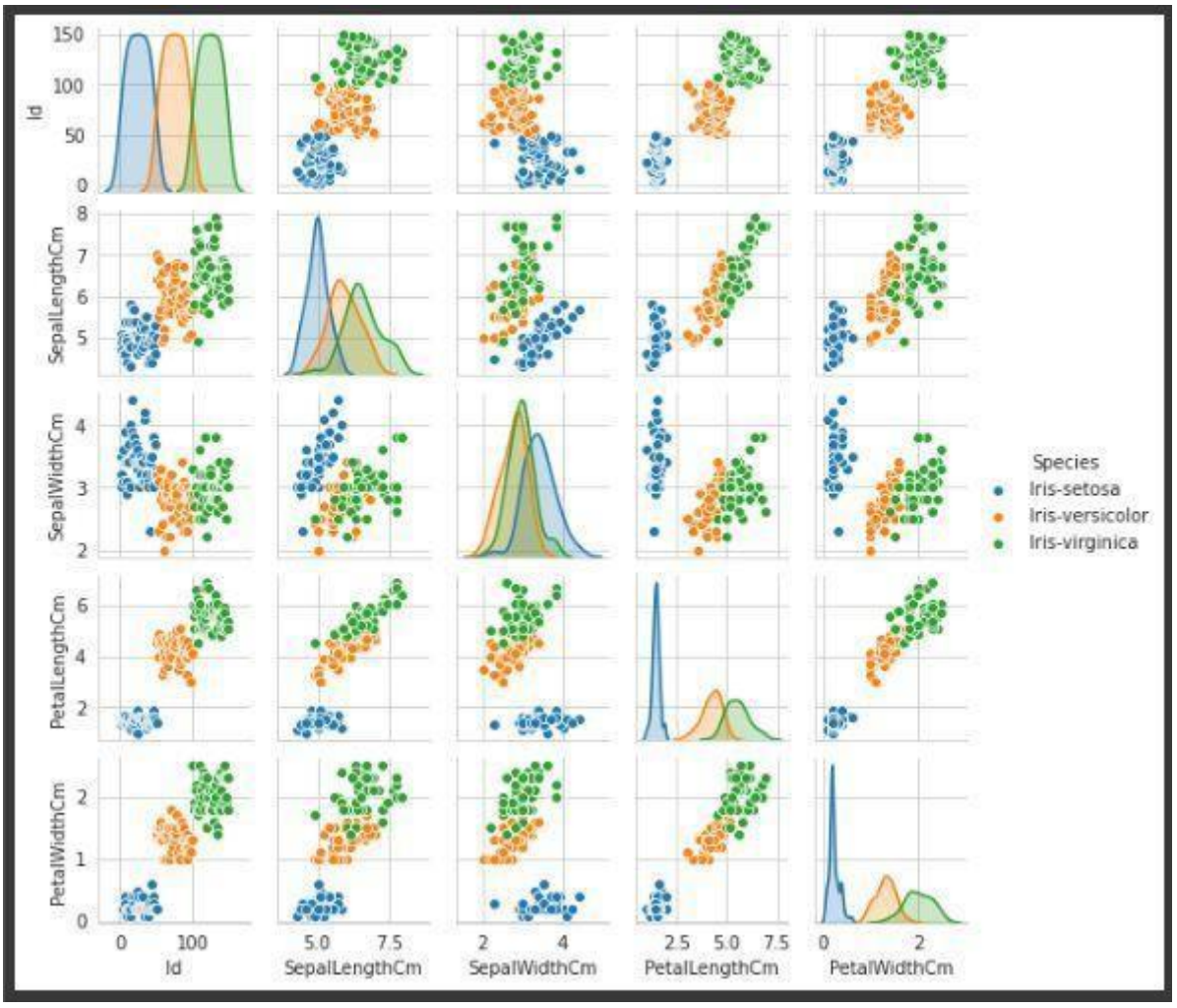
The output dataframe can be seen as for any cell, row variable correlation with the column variable is the value of the cell. The correlation of a variable with itself is 1. For that reason, all the diagonal values are 1.00.

Multivariate Analysis:
Pair plot is used to visualize the relationship between each type of column variable. It is implemented only by one line code, which is as follows :
Code:

- Python3

```
g = sns.pairplot(data,hue="Species")
```

Output:



Pairplot of variable "Species", to make it more understandable.

Practical No.11	Group B
Title	Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 11

Title: Data Wrangling, I

- **Problem Statement:**

  Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up.

- **Prerequisites:**

  Subjects: Discrete Mathematics, Database Management Systems
  Tools/Software Requirements: Apache Hadoop
  Processor/Hardware Requirements: Intel Core 2 Duo or above, 2 GB RAM or above
  Operating System: 64 Bit Windows/Linux/Mac OS
  Programming Language: JAVA

PART B

Title: Write a code in JAVA for a simple WordCount application that counts the number of occurrencesof each word in a given input set using the Hadoop MapReduce framework on local-standalone set-up

Pre-requisite

1. Java Installation – Java
(openjdk 11)java -version

2. Hadoop Installation – Hadoop 2 or higher.

Theory:

Hadoop MapReduce is a software framework for easily writing applications which process vast amountsof data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file- system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Word Count Example:

WordCount example reads text files and counts how often words occur. The input is text files and the output is text files, each line of which contains a word and the count of how often it occured, separatedby a tab.

Each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and each reducer sums the counts for each word and emits a single key/value with the word and sum.

As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amountof data sent across the network by combining each word into a single record

Steps to excute:

1. Create a text file in your local machine and write some text into it.
$ nano data.txt

2. In this example, we find out the frequency of each word exists in this text file.

3. Create a directory in HDFS, where to kept text file.
$ hdfs dfs -mkdir /test

4. Upload the data.txt file on HDFS in the specific directory.
$ hdfs dfs -put /home/codegyani/data.txt /test

5. Create the jar file of this program and name it countworddemo.jar.

6. Run the jar file
hadoop jar /home/codegyani/wordcountdemo.jar com.javatpoint.WC_Runner /test/data.txt /r_output

The output is stored in /r_output/part-00000

7. Now execute the command to see the
output.hdfs dfs -cat /r_output/part-00000

Program

File: WC_Mapper.java

```
package com.javatpoint;

import java.io.IOException;
import
java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,Int
Writable>{
    private final static IntWritable one = new IntWritable(1);private
    Text word = new Text();
    public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
            Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }

}
```

File: WC_Reducer.java

```
import
    java.io.IOException;
    import
    java.util.Iterator;
    import  org.apache.hadoop.io.IntWritable;
    import org.apache.hadoop.io.Text;
    import     org.apache.hadoop.mapred.MapReduceBase;
    import      org.apache.hadoop.mapred.OutputCollector;
    import org.apache.hadoop.mapred.Reducer;
    import  org.apache.hadoop.mapred.Reporter;

    public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritabl
e,Text,IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable>
output,
     Reporter reporter) throws IOException
    {int sum=0; while
    (values.hasNext()) {
    sum+=values.next().get(
    );
    }
    output.collect(key,new IntWritable(sum));
    }
    }
```

File: WC_Runner.java

```
 import
java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import    org.apache.hadoop.mapred.MapReduceBase;
import     org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritabl
e,Text,IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable>
output,
 Reporter reporter) throws IOException
{int sum=0; while
(values.hasNext()) {
sum+=values.next().get(
);
}
output.collect(key,new IntWritable(sum));
}
}
```

Out

put:

HD

FS

1

HADOOP 2

MapReduce 1

a 2

is 2

of 2

processing 1

storage 1

tool 1

unit 1

Practical No.12	Group B
Title	Design a distributed application using Map-Reduce which processes a log file of a system.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 12

Title: Map-Reduce

- **Problem Statement:**
  Design a distributed application using Map-Reduce which processes a log file of a system.

- **Prerequisites:**

  Subjects: Discrete Mathematics, Database Management Systems
  Tools/Software Requirements: Apache Hadoop /Oracle
  Processor/Hardware Requirements: Intel Core 2 Duo or above, 2 GB RAM or above
  Operating System: 64 Bit Windows/Linux/Mac OS
  Programming Language: Java

Practical No:12

Aim: Write a simple program in SCALA using Apache Spark

Theory: 1. What is Scala

Scala is an acronym for “Scalable Language”. It is a general-purpose programming language designed for the programmers who want to write programs in a concise, elegant, and type- safe way. Scala enables programmers to be more productive. Scala is developed as an object-oriented and functional programming language.

If you write a code in Scala, you will see that the style is similar to a scripting language. Even though Scala is a new language, it has gained enough users and has a wide community support. It is one of the most user-friendly languages.

2. About Scala

The design of Scala started in 2001 in the programming methods laboratory at EPFL (École Polytechnique Fédérale de Lausanne). Scala made its first public appearance in January 2004 on the JVM platform and a few months later in June 2004, it was released on the .(dot)NET platform. The .(dot)NET support of Scala was officially dropped in 2012. A few more characteristics of Scala are:

2.1 Scala is pure Object-Oriented programming language

Scala is an object-oriented programming language. Everything in Scala is an object and any operations you perform is a method call. Scala, allow you to add new operations to existing classes with the help of implicit classes.

One of the advantages of Scala is that it makes it very easy to interact with Java code. You can also write a Java code inside Scala class. The Scala supports advanced component architectures through classes and traits.

2.2 Scala is a functional language

Scala is a programming language that has implemented major functional programming concepts. In Functional programming, every computation is treated as a mathematical function which avoids states and mutable data. The functional programming exhibits following characteristics:

- Power and flexibility
- Simplicity
- Suitable for parallel processing

Scala is not a pure functional language. Haskell is an example of a pure functional language. If you want to read more about functional programming, please refer to this article.

2.3 Scala is a compiler based language (and not interpreted)

Scala is a compiler based language which makes Scala execution very fast if you compare it with Python (which is an interpreted language). The compiler in Scala works in similar fashion as Java compiler. It gets the source code and generates Java byte-code that can be executed independently on any standard JVM (Java Virtual Machine). If you want to know more about the difference between complied vs interpreted language please refer this article.

There are more important points about Scala which I have not covered. Some of them are:

- Scala has thread based executors
- Scala is statically typed language
- Scala can execute Java code
- You can do concurrent and Synchronized processing in Scala
- Scala is JVM based languages

2.4 Companies using Scala

Scala is now big name. It is used by many companies to develop the commercial software. These are the following notable big companies which are using Scala as a programming alternative.

- LinkedIn
- Twitter
- Foursquare
- Netflix
- Tumblr
- The Guardian
- Precog
- Sony
- AirBnB
- Klout
- Apple

If you want to read more about how and when these companies started using Scala please refer this blog.

3. Installing Scala

Scala can be installed in any Unix or windows based system. Below are the steps to install for Ubuntu (14.04) for scala version 2.11.7. I am showing the steps for installing Scala (2.11.7) with Java version 7. It is necessary to install Java before installing Scala. You can also install latest version of Scala(2.12.1) as well.

Step 0: Open the terminal

Step 1: Install Java

```
$ sudo apt-add-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
```

If you are asked to accept Java license terms, click on “Yes” and proceed. Once finished, let us check whether Java has installed successfully or not. To check the Java version and installation, you can type:

```
$ java -version
```

Step 2: Once Java is installed, we need to install Scala

```
$ cd ~/Downloads
$ wget http://www.scala-lang.org/files/archive/scala-2.11.7.deb
$ sudo dpkg -i scala-2.11.7.deb
$ scala –version
```

This will show you the version of Scala installed

4. Prerequisites for Learning Scala

Scala being an easy to learn language has minimal prerequisites. If you are someone with basic knowledge of C/C++, then you will be easily able to get started with Scala. Since Scala is developed on top of Java. Basic programming function in Scala is similar to Java. So, ifyou have some basic knowledge of Java syntax and OOPs concept, it would be helpful foryou to work in Scala.

5. Choosing a development environment

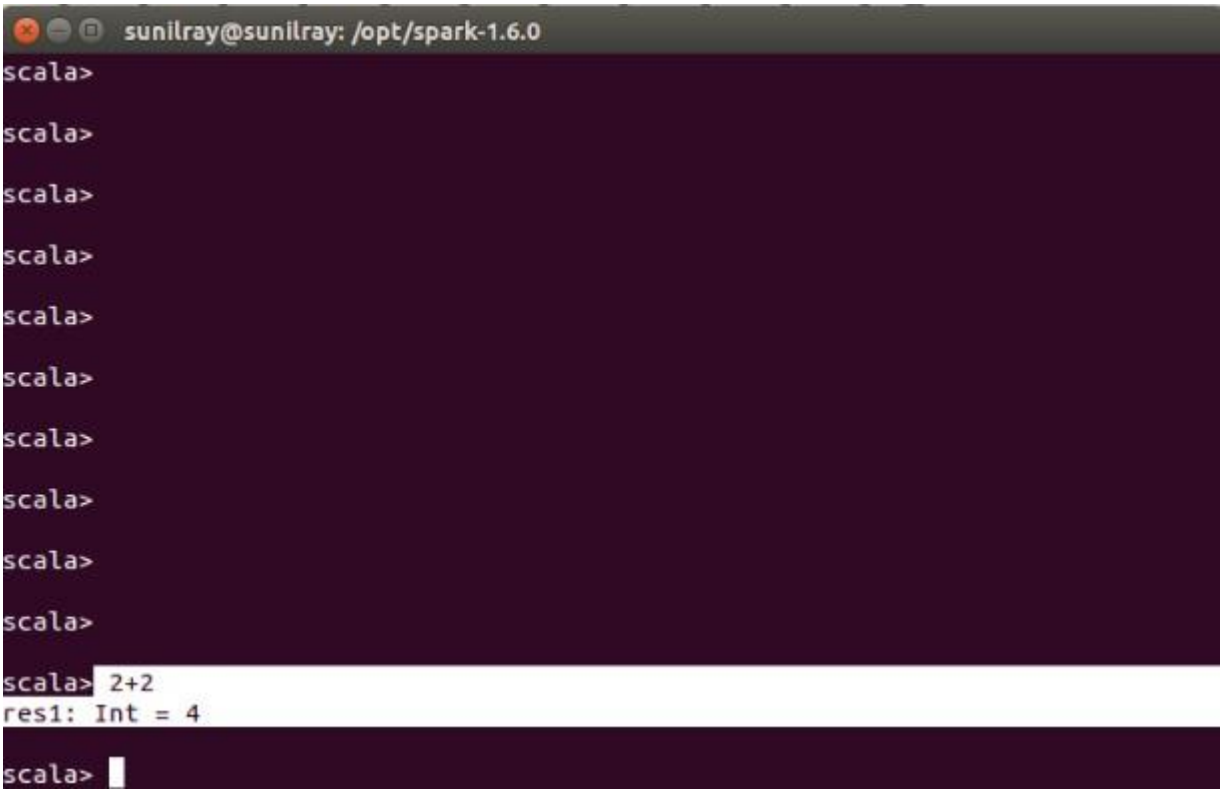
Once you have installed Scala, there are various options for choosing an environment. Here are the 3 most common options:

- Terminal / Shell based
- Notepad / Editor based
- IDE (Integrated development environment)

Choosing right environment depends on your preference and use case. I personally prefer writing a program on shell because it provides a lot of good features like suggestions for method call and you can also run your code while writing line by line.

Warming up: Running your first Scala program in Shell:

Let's write a first program which adds two numbers.



6. Scala Basics Terms

Object: An entity that has state and behavior is known as an object. For example: table, person, car etc.

Class: A class can be defined as a blueprint or a template for creating different objects which defines its properties and behavior.

Method: It is a behavior of a class. A class can contain one or more than one method. For example: deposit can be considered a method of bank class.

Closure: Closure is any function that closes over the environment in which it's defined. A closure returns value depends on the value of one or more variables which is declared outside this closure.

Traits: Traits are used to define object types by specifying the signature of the supported methods. It is like interface in java.

7. Things to note about Scala

- It is case sensitive
- If you are writing a program in Scala, you should save this program using ".scala"
- Scala execution starts from main() methods
- Any identifier name cannot begin with numbers. For example, variable name "123salary" is invalid.
- You can not use Scala reserved keywords for variable declarations or constant or any identifiers.

8. Variable declaration in Scala

In Scala, you can declare a variable using 'var' or 'val' keyword. The decision is based on whether it is a constant or a variable. If you use 'var' keyword, you define a variable as mutable variable. On the other hand, if you use 'val', you define it as immutable. Let's first declare a variable using "var" and then using "val".

8.1 Declare using var

```
var Var1 : String = "Ankit"
```

In the above Scala statement, you declare a mutable variable called "Var1" which takes a string value. You can also write the above statement without specifying the type of variable. Scala will automatically identify it. For example:

var Var1 = "Gupta"

8.2 Declare using val

val Var2 : String = "Ankit"

In the above Scala statement, we have declared an immutable variable “Var2” which takes a string “Ankit”. Try it for without specifying the type of variable. If you want to read about mutable and immutable please refer this link.

9. Operations on variables

You can perform various operations on variables. There are various kinds of operatorsdefined in Scala. For example: Arithmetic Operators, Relational Operators, Logical Operators, Bitwise Operators, Assignment Operators.

Lets see “+” , “==” operators on two variables ‘Var4’, “Var5”. But, before that, let us first assign values to “Var4” and “Var5”.

scala> var Var4 = 2

Output: Var4: Int = 2

scala> var Var5 = 3

Output: Var5: Int = 3

Now, let us apply some operations using operators in Scala.

Apply ‘+’ operator

Var4+Var5

Output:

res1: Int = 5

Apply “==” operator

Var4==Var5

Output:

res2: Boolean = false

If you want to know complete list of operators in Scala refer this link:

10. The if-else expression in Scala

In Scala, if-else expression is used for conditional statements. You can write one or more conditions inside "if". Let's declare a variable called "Var3" with a value 1 and then compare "Var3" using if-else expression.

```
var Var3 =1
if (Var3 ==1){
println("True")}else{
println("False")}
```

Output: True

In the above snippet, the condition evaluates to True and hence True will be printed in the output.

11. Iteration in Scala

Like most languages, Scala also has a FOR-loop which is the most widely used method for iteration. It has a simple syntax too.

```
for( a <- 1 to 10){
 println( "Value of a: " + a );
 }
```

Output:

Value of a: 1

Value of a: 2

Value of a: 3

Value of a: 4

Value of a: 5

Value of a: 6

Value of a: 7

Value of a: 8

Value of a: 9

Value of a: 10

Scala also supports “while” and “do while” loops. If you want to know how both work, please refer this link.

12. Declare a simple function in Scala and call it by passing value

You can define a function in Scala using “def” keyword. Let’s define a function called“mul2” which will take a number and multiply it by 10. You need to define the return type of function, if a function not returning any value you should use the “Unit” keyword.

In the below example, the function returns an integer value. Let’s define the function “mul2”:

```
def mul2(m: Int): Int = m * 10
```

Output: mul2: (m: Int)Int

Now let’s pass a value 2 into mul2

```
mul2(2)
```

Output:

res9: Int = 20

If you want to read more about the function, please refer this tutorial.

13. Few Data Structures in Scala

- Arrays
- Lists
- Sets
- Tuple
- Maps
- Option

13.1 Arrays in Scala

In Scala, an array is a collection of similar elements. It can contain duplicates. Arrays are also immutable in nature. Further, you can access elements of an array using an index:

Declaring Array in Scala

To declare any array in Scala, you can define it either using a new keyword or you can directly assign some values to an array.

Declare an array by assigning it some values

```
var name = Array("Faizan","Swati","Kavya", "Deepak", "Deepak")
```

Output:

```
name: Array[String] = Array(Faizan, Swati, Kavya, Deepak, Deepak)
```

In the above program, we have defined an array called name with 5 string values.

Declaring an array using "new" keywords

The following is the syntax for declaring an array variable using a new keyword.

```
var name:Array[String] = new Array[String](3)
```

or

```
var name = new Array[String](3)
```

Output:

```
name: Array[String] = Array(null, null, null)
```

Here you have declared an array of Strings called "name" that can hold up to three elements. You can also assign values to "name" by using an index.

```
scala> name(0) = "jal"
scala> name(1) = "Faizy"
scala> name(2) = "Expert in deep learning"
```

Let's print contents of "name" array.

```
scala> name
res3: Array[String] = Array(jal, Faizy, Expert in deep learning)
```

Accessing an array

You can access the element of an array by index. Lets access the first element of array "name". By giving index 0. Index in Scala starts from 0.

```
name(0)
```

Output:

```
res11: String = jal
```

13.2 List in Scala

Lists are one of the most versatile data structure in Scala. Lists contain items of different types in Python, but in Scala the items all have the same type. Scala lists are immutable.

Here is a quick example to define a list and then access it.

Declaring List in Scala

You can define list simply by comma separated values inside the "List" method.

```
scala> val numbers = List(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
numbers: List[Int] = List(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

You can also define multi dimensional list in Scala. Lets define a two dimensional list:

```
val number1 = List( List(1, 0, 0), List(0, 1, 0), List(0, 0, 1) )
number1: List[List[Int]] = List(List(1, 0, 0), List(0, 1, 0), List(0, 0, 1))
```

Accessing a list

Let's get the third element of the list "numbers" . The index should 2 because index in Scala start from 0.

```
scala> numbers(2)
res6: Int = 3
```

We have discussed two of the most used data Structures. You can learn more from this link.

14. Writing & Running a program in Scala using an editor

Let us start with a "Hello World!" program. It is a good simple way to understand how to write, compile and run codes in Scala. No prizes for telling the outcome of this code!

```
object HelloWorld {
 def main(args: Array[String]) {
 println("Hello, world!")
 }
 }
```

As mentioned before, if you are familiar with Java, it will be easier for you to understand Scala. If you know Java, you can easily see that the structure of above "HelloWorld" programis very similar to Java program.

This program contains a method "main" (not returning any value) which takes an argument – a string array through command line. Next, it calls a predefined method called "Println" and passes the argument "Hello, world!".

You can define the main method as static in Java but in Scala, the static method is no longer available. Scala programmer can't use static methods because they use singleton objects. To read more about singleton object you can refer this article.

14.1 Compile a Scala Program

To run any Scala program, you first need to compile it. "Scalac" is the compiler which takes source program as an argument and generates object files as output.

Let's start compiling your "HelloWorld" program using the following steps:

1. For compiling it, you first need to paste this program into a text file then you need to save this program as HelloWorld.scala

2. Now you need change your working directory to the directory where your program is saved

3. After changing the directory you can compile the program by issuing the command.

scalac HelloWorld.scala

4. After compiling, you will get Helloworld.class as an output in the same directory. If you can see the file, you have successfully compiled the above program.

14.2 Running Scala Program

After compiling, you can now run the program using following command:

scala HelloWorld

You will get an output if the above command runs successfully. The program will print "Hello, world!"

15. Advantages of using Scala for Apache Spark

If you are working with Apache Spark then you would know that it has 4 different APIs support for different languages: **Scala, Java, Python and R**.

Each of these languages have their own unique advantages. But using Scala is more advantageous than other languages. These are the following reasons why Scala is taking over big data world.

- Working with Scala is more productive than working with Java
- Scala is faster than Python and R because it is compiled language
- Scala is a functional language

16. Comparing Scala, Java, Python and R APIs in Apache Spark

Let's compare 4 major languages which are supported by Apache Spark API.

Metrics	S	J	Python		R
Type	Compiled	Compiled	Interpreted	In	rpret
JVM based	Y	Y	No		No
Verbosity	L	M	Less		Less
Code Length	L	M	Less		ss
Productivity	H	L	High		High
Scalability	H	H	Less		Less
OOPS Support	Y	Y	Yes		es

17. Install Apache Spark & some basic concepts about Apache Spark

To know the basics of Apache Spark and installation, please refer to my first article on Pyspark. I have introduced basic terminologies used in Apache Spark like big data, cluster computing, driver, worker, spark context, In-memory computation, lazy evaluation, DAG, memory hierarchy and Apache Spark architecture in the previous article.

As a quick refresher, I will be explaining some of the topics which are very useful to proceed further. If you are a beginner, then I strongly recommend you to go through my first article before proceeding further.

- **Lazy operation:** Operations which do not execute until we require results.
- **Spark Context:** holds a connection with Spark cluster manager.
- **Driver and Worker:** A driver is in charge of the process of running the main() function ofan application and creating the SparkContext.
- **In-memory computation:** Keeping the data in RAM instead of Hard Disk for fast processing.

Spark has three data representations viz RDD, Dataframe, Dataset. To use Apache Spark functionality, we must use one of them for data manipulation. Let's discuss each of them briefly:

- **RDD:** RDD (Resilient Distributed Database) is a collection of elements, that can be divided across multiple nodes in a cluster for parallel processing. It is also fault tolerant collection of elements, which means it can automatically recover from failures. RDD is immutable, we can create RDD once but can't change it.
- **Dataset:** It is also a distributed collection of data. A Dataset can be constructed from JVM objects and then manipulated using functional transformations (map, flatMap, filter, etc.). As I have already discussed in my previous articles, dataset API is only available in Scala and Java. It is not available in Python and R.
- **DataFrame:** In Spark, a DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame. It is mostly used for structured data processing. In Scala, a DataFrame is represented by a Dataset of Rows. A DataFrame can be constructed by wide range of arrays for example, existing RDDs, Hive tables, database tables.
- **Transformation:** Transformation refers to the operation applied on a RDD to create new RDD.
- **Action:** Actions refer to an operation which also apply on RDD that perform computation and send the result back to driver.
- **Broadcast:** We can use the Broadcast variable to save the copy of data across all node.
- **Accumulator:** In Accumulator, variables are used for aggregating the information.

18. Working with RDD in Apache Spark using Scala

First step to use RDD functionality is to create a RDD. In Apache Spark, RDD can be created by two different ways. One is from existing Source and second is from an external source.

So before moving further let's open the Apache Spark Shell with Scala. Type the following command after switching into the home directory of Spark. It will also load the spark context as sc.

```
$ ./bin/spark-shell
```

After typing above command you can start programming of Apache Spark in Scala.

18.1 Creating a RDD from existing source

When you want to create a RDD from existing storage in driver program (which we would like to be parallelized). For example, converting an array to RDD, which is already created in a driver program.

```
val data = Array(1, 2, 3, 4, 5,6,7,8,9,10)
val distData = sc.parallelize(data)
```

In the above program, I first created an array for 10 elements and then I created a distributed data called RDD from that array using “parallelize” method. SparkContext has a parallelize method, which is used for creating the Spark RDD from an iterable already present in driver program.

To see the content of any RDD we can use “collect” method. Let’s see the content of distData.

```
scala> distData.collect()
Output: res1: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

18.2 Creating a RDD from External sources

You can create a RDD through external sources such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format. So let’s create a RDD from the text file:

The name of the text file is text.txt. and it has only 4 lines given below.

I love solving data mining problems.

I don’t like solving data mining problems.

I love solving data science problems.

I don’t like solving data science problems.

Let’s create the RDD by loading it.

```
val lines = sc.textFile("text.txt");
```

Now let’s see first two lines in it.

lines.take(2)

The output is received is as below:

Output: Array(I love solving data mining problems., I don't like solving data mining problems)

18.3 Transformations and Actions on RDD

18.3.1. Map Transformations

A map transformation is useful when we need to transform a RDD by applying a function to each element. So how can we use map transformation on ‘rdd’ in our case? Let’s calculate the length (number of characters) of each line in “text.txt”

```
val Lenght = lines.map(s => s.length)
Length.collect()
```

After applying above map operation, we get the following output:

Output: res6: Array[Int] = Array(36, 42, 37, 43)

18.3.2 Count Action

Let’s count the number of lines in RDD “lines”.

```
lines.count()
res1: Long = 4
```

The above action on “lines1” will give 4 as the output.

18.3.3 Reduce Action

Let’s take the sum of total number of characters in text.txt.

val totalLength = Length.reduce((a, b) => a + b)

totalLength: Int = 158

18.3.4 flatMap transformation and reduceByKey Action

Let's calculate frequency of each word in "text.txt"

```
val counts = lines.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)
```

counts.collect()

Output:

res6: Array[(String, Int)] = Array((solving,4), (mining,2), (don't,2), (love,2), (problems.,4), (data,4), (science,2), (I,4), (like,2))

18.3.5 filter Transformation

Let's filter out the words in "text.txt" whose length is more than 5.

val lg5 = lines.flatMap(line => line.split(" ")).filter(_.length > 5)

Output:

res7: Array[String] = Array(solving, mining, problems., solving, mining, problems., solving, science, problems., solving, science, problems.)

19. Working with DataFrame in Apache Spark using Scala

A DataFrame in Apache Spark can be created in multiple ways:

- It can be created using different data formats. For example, by loading the data from JSON, CSV
- Loading data from Existing RDD
- Programmatically specifying schema

Let's create a DataFrame using a csv file and perform some analysis on that.

For reading a csv file in Apache Spark, we need to specify a new library in our Scala shell. To perform this action, first, we need to download Spark-csv package (Latest version) and extract this package into the home directory of Spark. Then, we need to open a PySpark shell and include the package (I am using "spark-csv_2.10:1.3.0").

$ ./bin/spark-shell --packages com.databricks:spark-csv_2.10:1.3.0

Now let's load the csv file into a DataFrame df. You can download the file(train) from this link.

```
val df = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("train.csv")
```

19.1 Name of columns

Let's see the name of columns in df by using "columns" method.

```
df.columns
```

Output:

```
res0: Array[String] = Array(User_ID, Product_ID, Gender, Age, Occupation, City_Category,
Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2,
Product_Category_3, Purchase)
```

19.2 Number of observations

To see the number of observation in df you can apply "count" method.

```
df.count()
```

Output:

```
res1: Long = 550068
```

19.3 Print the columns datatype

You can use "printSchema" method on df. Let's print the schema of df.

```
df.printSchema()
```

Output:

root

|-- User_ID: integer (nullable = true)

|-- Product_ID: string (nullable = true)

|-- Gender: string (nullable = true)

|-- Age: string (nullable = true)

|-- Occupation: integer (nullable = true)

|-- City_Category: string (nullable = true)

|-- Stay_In_Current_City_Years: string (nullable = true)

|-- Marital_Status: integer (nullable = true)

|-- Product_Category_1: integer (nullable = true)

|-- Product_Category_2: integer (nullable = true)

|-- Product_Category_3: integer (nullable = true)

|-- Purchase: integer (nullable = true)

19.4 Show first n rows

You can use “show” method on DataFrame. Let’s print the first 2 rows of df.

df.show(2)

Output:

+-------+----------+------+----+----------+-------------+--------------------------+-----------------+-----------------+-----------------+--------+

|User_ID|Product_ID|Gender|Age|Occupation|City_Category|Stay_In_Current_City_Years|Marital_Status|Product_Category_1|Product_Category_2|Product_Category_3|Purchase|

+-------+----------+------+----+----------+-------------+--------------------------+-----------------+-----------------+-----------------+--------+

|1000001| P00069042| F|0-17| 10| A| 2| 0| 3| null| null| 8370|

|1000001| P00248942| F|0-17| 10| A| 2| 0| 1| 6| 14| 15200|

+-------+----------+------+----+----------+-------------+--------------------------+-----------------+-----------------+-----------------+--------+

only showing top 2 rows

19.5 Subsetting or select columns

To select columns you can use “select” method. Let’s apply select on df for “Age” columns.

```
df.select("Age").show(10)
```

Output:

```
+.....+
| Age|
+-----+
| 0-17|
| 0-17|
| 0-17|
| 0-17|
| 55+|
|26-35|
|46-50|
|46-50|
|46-50|
|26-35|
+.....+
only showing top 10 rows
```

19.6 Filter rows

To filter the rows you can use “filter” method. Let’s apply filter on “Purchase” column of df and get the purchase which is greater than 10000.

```
df.filter(df("Purchase") >= 10000).select("Purchase").show(10)
+.....+
|Purchase|
+.....+
| 15200|
| 15227|
| 19215|
```

| 15854|

| 15686|

| 15665|

| 13055|

| 11788|

| 19614|

| 11927|

+.....+

only showing top 10 rows

19.7 Group DataFrame

To groupby columns, you can use groupBy method on DataFrame. Let's see the distribution on "Age" columns in df.

df.groupBy("Age").count().show()

Output:

+.....+.....+

| Age| count|

+.....+.....+

|51-55| 38501|

|46-50| 45701|

| 0-17| 15102|

|36-45|110013|

|26-35|219587|

| 55+| 21504|

|18-25| 996 |

+.....+.....+

19.8 Apply SQL queries on DataFrame

To apply queries on DataFrame You need to register DataFrame(df) as table. Let's first register df as temporary table called (B_friday).

df.registerTempTable("B_friday")

Now you can apply SQL queries on “B_friday” table using sqlContext.sql. Lets select columns “Age” from the “B_friday” using SQL statement.

```
sqlContext.sql("select Age from B_friday").show(5)
+-----+
| Age|
+-----+
|0-17|
|0-17|
|0-17|
|0-17|
| 55+|
+----+
```

20. Building a machine learning model

If you have come this far, you are in for a treat! I’ll complete this tutorial by building a machine learning model.

I will use only three dependent features and the independent variable in df1. Let’s create a DataFrame df1 which has only 4 columns (3 dependent and 1 target).

```
val df1 = df.select("User_ID","Occupation","Marital_Status","Purchase")
```

In above DataFrame df1 “User_ID”,”Occupation” and “Marital_Status” are features and “Purchase” is target column.

Let’s try to create a formula for Machine learning model like we do in R. First, we need to import RFormula. Then we need to specify the dependent and independent column inside this formula. We also have to specify the names for features column and label column.

```
import org.apache.spark.ml.feature.RFormula
```

```
val formula = new RFormula().setFormula("Purchase ~
User_ID+Occupation+Marital_Status").setFeaturesCol("features").setLabelCol("label")
```

After creating the formula, we need to fit this formula on df1 and transform df1 through this formula. Let's fit this formula.

```
val train = formula.fit(df1).transform(df1)
```

After applying the formula we can see that train dataset has 2 extra columns called features and label. These are the ones we have specified in the formula (featuresCol="features" and labelCol="label")

20.1 Applying Linear Regression on train

After applying the RFormula and transforming the DataFrame, we now need to develop the machine learning model on this data. I want to apply a Linear Regression for this task. Let us import a Linear regression and apply on train. Before fitting the model, I am setting the hyperparameters.

```
import org.apache.spark.ml.regression.LinearRegression
val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(train)
```

You can also make predictions on unseen data. But I am not showing this here. Let's print the coefficient and intercept for linear regression.

```
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
```

Output:

```
Coefficients: [0.015092115630330033,16.12117786898672,-10.520580986444338]
Intercept: -5999.754797883323
```

Let's summarize the model over the training set and print out some metrics.

```
val trainingSummary = lrModel.summary
```

Now, See the residuals for train's first 10 rows.

```
trainingSummary.residuals.show(10)
```

```
+-------------------+
| residuals|
+-------------------+
| -883.5877032522076|
| 5946.412296747792|
| -7831.587703252208|
| -8196.587703252208|
|-1381.3298625817588|
| 5892.776223171599|
| 10020.251134994305|
| 6659.251134994305|
| 6491.251134994305|
|-1533.3392694181512|
+-------------------+
only showing top 10 rows
```

Now, let's see RMSE on train.

```
println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
```

Output:

```
RMSE: 5021.899441991144
```

Let's repeat above procedure for taking the prediction on cross-validation set. Let's read the train dataset again.

```
val train = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").load("train.csv")
```

Now, randomly divide the train in two part train_cv and test_cv

```
val splits = train.randomSplit(Array(0.7, 0.3))
val (train_cv,test_cv) = (splits(0), splits(1))
```

Now, Transform train_cv and test_cv using RFormula.

```
import org.apache.spark.ml.feature.RFormula
```

```
val formula = new RFormula().setFormula("Purchase ~
User_ID+Occupation+Marital_Status").setFeaturesCol("features").setLabelCol("label")


val train_cv1 = formula.fit(train_cv).transform(train_cv)
val test_cv1 = formula.fit(train_cv).transform(test_cv)
```

After transforming using RFormula, we can build a machine learning model and take the predictions. Let's apply Linear Regression on training and testing data.

```
import org.apache.spark.ml.regression.LinearRegression
val lr = new LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
val lrModel = lr.fit(train_cv1)
val train_cv_pred = lrModel.transform(train_cv1)
val test_cv_pred = lrModel.transform(test_cv1)
```

In train_cv_pred and test_cv_pred, you will find a new column for prediction.

Conclusion: Hence, We have thoroughly studied how to **program in SCALA using Apache Spark.**

Practical No.13	Group B
Title	Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 13

Title: Data Analytics

- **Problem Statement:**

  Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.

- **Prerequisites:**

  Subjects: Discrete Mathematics, Database Management Systems
  Tools/Software Requirements: Apache Hadoop
  Processor/Hardware Requirements: Intel Core 2 Duo or above, 2 GB RAM or above
  Operating System: 64 Bit Windows/Linux/Mac OS
  Programming Language: SCALA/Python3

Practical No: 13

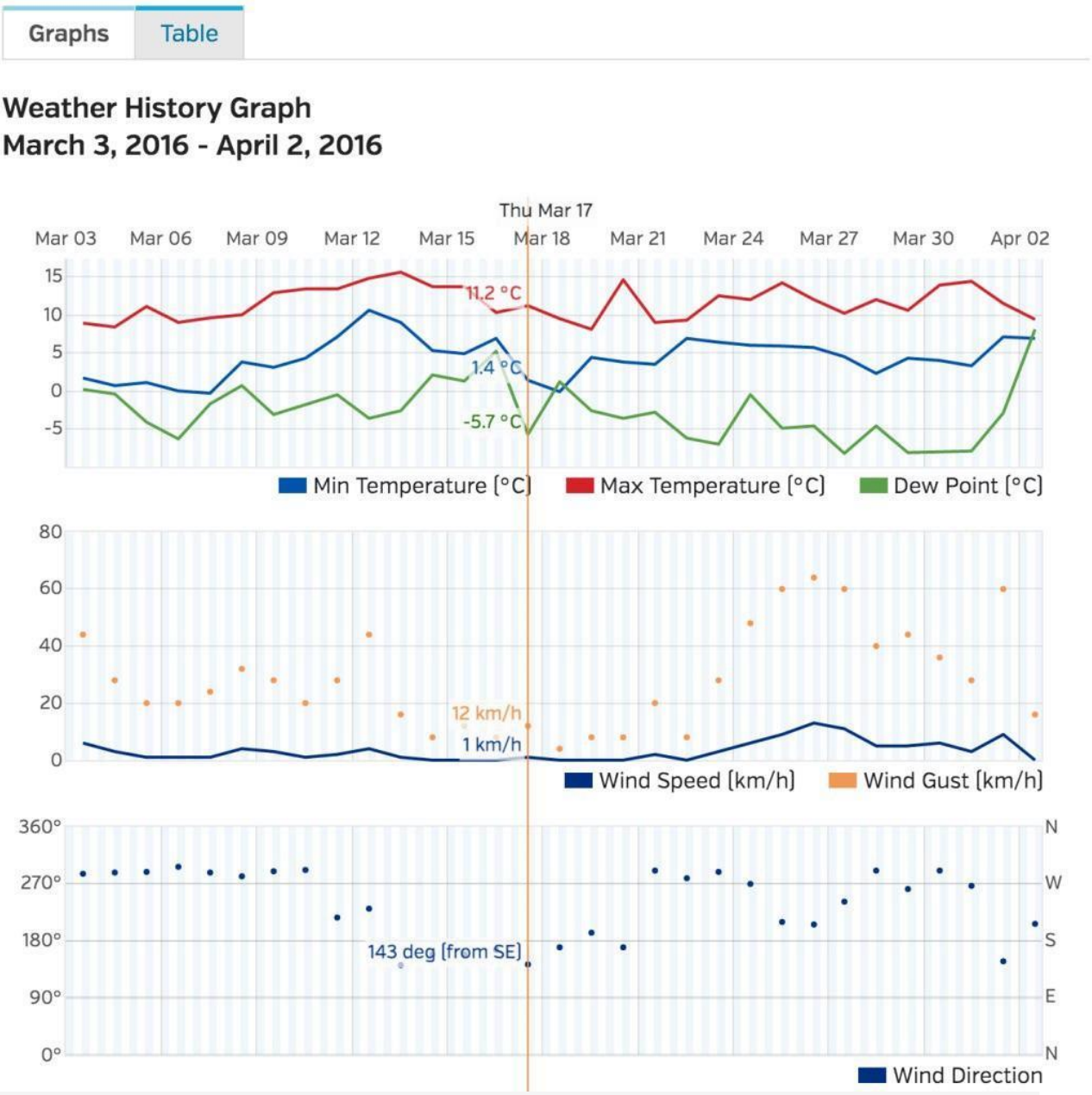
Aim: Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.

Theory: Analysis of Weather data using Pandas, Python, and Seaborn:

The most recent post on this site was an analysis of how often people cycling to work actually get rained on in different cities around the world. You can check it out here.
The analysis was completed using data from the Wunderground weather website, Python, specifically the Pandas and Seaborn libraries. In this post, I will provide the Python code to replicate the work and analyse information for your own city. During the analysis, I used Python Jupyter notebooks to interactively explore and cleanse data; there's a simple setup if you elect to use something like the Anaconda Python distribution to install everything you need.
If you want to skip data downloading and scraping, all of the data I used is available to download here.

Scraping Weather Data

Wunderground.com has a "Personal Weather Station (PWS)" network for which fantastic historical weather data is available – covering temperature, pressure, wind speed and direction, and of course rainfall in mm – all available on a per-minute level. Individual stations can be examined at specific URLS, for example here for station "IDUBLIND35". There's no official API for the PWS stations that I could see, but there is a very good API for forecast data. However, CSV format data with hourly rainfall, temperature, and pressure information can be downloaded from the website with some simple Python scripts.
The hardest part here is to actually find stations that contain enough information for your analysis – you'll need to switch to "yearly view" on the website to find stations that have been around more than a few months, and that record all of the information you want. If you're looking for temperature info – you're laughing, but precipitation records are more sparse.



Wunderground have an excellent site with interactive graphs to look at weather data on a daily, monthly, and yearly level. Data is also available to download in CSV format, which is great for data science purposes.

```
import requests

import pandas as pd

from dateutil import parser, rrule

from datetime import datetime, time, date

import time
```

```
def getRainfallData(station, day, month, year):
"""
Function to return a data frame of minute-level weather data for a single Wunderground
PWS station.
Args:
station (string): Station code from the Wunderground website
day (int): Day of month for which data is requested
month (int): Month for which data is requested
year (int): Year for which data is requested
Returns:
Pandas Dataframe with weather data for specified station and date.
"""
url =
"http://www.wunderground.com/weatherstation/WXDailyHistory.asp?ID={station}&amp;amp;day={day}&amp;amp;month={month}&amp;amp;year={year}&amp;amp;graphspan=day&amp;amp;format=1"
full_url = url.format(station=station, day=day, month=month, year=year)
# Request data from wunderground data
response = requests.get(full_url, headers={'User-agent': 'Mozilla/5.0 (Windows NT 6.1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36'})
data = response.text
# remove the excess &amp;lt;br&amp;gt; from the text data
data = data.replace('&amp;lt;br&amp;gt;', '')
# Convert to pandas dataframe (fails if issues with weather station)
try:
dataframe = pd.read_csv(io.StringIO(data), index_col=False)
```

```
dataframe['station'] = station

except Exception as e:

print("Issue with date: {}-{}-{} for station {}".format(day,month,year, station))

return None

return dataframe

# Generate a list of all of the dates we want data for

start_date = "2015-01-01"

end_date = "2015-12-31"

start = parser.parse(start_date)

end = parser.parse(end_date)

dates = list(rrule.rrule(rrule.DAILY, dtstart=start, until=end))

# Create a list of stations here to download data for

stations = ["IDUBLINF3", "IDUBLINF2", "ICARRAIG2", "IGALWAYR2", "IBELFAST4",
"ILONDON59", "IILEDEFR28"]

# Set a backoff time in seconds if a request fails

backoff_time = 10

data = {}

# Gather data for each station in turn and save to CSV.

for station in stations:

print("Working on {}".format(station))

data[station] = []

for date in dates:

# Print period status update messages

if date.day % 10 == 0:
```

```
print("Working on date: {} for station {}".format(date, station))

done = False

while done == False:

try:

weather_data = getRainfallData(station, date.day, date.month, date.year)

done = True

except ConnectionError as e:

# May get rate limited by Wunderground.com, backoff if so.

print("Got connection error on {}".format(date))

print("Will retry in {} seconds".format(backoff_time))

time.sleep(10)

# Add each processed date to the overall data

data[station].append(weather_data)

# Finally combine all of the individual days and output to CSV for analysis.

pd.concat(data[station]).to_csv("data/{}_weather.csv".format(station))
```

Cleansing and Data Processing

The data downloaded from Wunderground needs a little bit of work. Again, if you want the raw data, it's here. Ultimately, we want to work out when its raining at certain times of the day and aggregate this result to daily, monthly, and yearly levels. As such, we use Pandas to add month, year, and date columns. Simple stuff in preparation, and we can then output plots as required.

```
station = 'IEDINBUR6' # Edinburgh

data_raw = pd.read_csv('data/{}_weather.csv'.format(station))

# Give the variables some friendlier names and convert types as necessary.

data_raw['temp'] = data_raw['TemperatureC'].astype(float)

data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
```

```
data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)

data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)

data_raw['humidity'] = data_raw['Humidity'].astype(float)

data_raw['wind_direction'] = data_raw['WindDirectionDegrees']

data_raw['wind'] = data_raw['WindSpeedKMH']

# Extract out only the data we need.

data = data_raw.loc[:, ['date', 'station', 'temp', 'rain', 'total_rain', 'humidity', 'wind']]

data = data[(data['date'] &gt;= datetime(2015,1,1)) &amp; (data['date'] &lt;=
datetime(2015,12,31))]

# There's an issue with some stations that record rainfall ~-2500 where data is missing.

if (data['rain'] &lt; -500).sum() &gt; 10:

print("There's more than 10 messed up days for {}".format(station))

# remove the bad samples

data = data[data['rain'] &gt; -500]

# Assign the "day" to every date entry

data['day'] = data['date'].apply(lambda x: x.date())

# Get the time, day, and hour of each timestamp in the dataset

data['time_of_day'] = data['date'].apply(lambda x: x.time())

data['day_of_week'] = data['date'].apply(lambda x: x.weekday())

data['hour_of_day'] = data['time_of_day'].apply(lambda x: x.hour)

# Mark the month for each entry so we can look at monthly patterns

data['month'] = data['date'].apply(lambda x: x.month)

# Is each time stamp on a working day (Mon-Fri)

data['working_day'] = (data['day_of_week'] &gt;= 0) &amp; (data['day_of_week'] &lt;= 4)
```

```
# Classify into morning or evening times (assuming travel between 8.15-9am and 5.15-6pm)

data['morning'] = (data['time_of_day'] &gt;= time(8,15)) &amp; (data['time_of_day'] &lt;= time(9,0))

data['evening'] = (data['time_of_day'] &gt;= time(17,15)) &amp; (data['time_of_day'] &lt;= time(18,0))

# If there's any rain at all, mark that!

data['raining'] = data['rain'] &gt; 0.0

# You get wet cycling if its a working day, and its raining at the travel times!

data['get_wet_cycling'] = (data['working_day']) &amp; ((data['morning'] &amp; data['rain']) |

(data['evening'] &amp; data['rain']))
```

At this point, the dataset is relatively clean, and ready for analysis. If you are not familiar with grouping and aggregation procedures in Python and Pandas, here is another blog post on the topic.

Data after cleansing from Wunderground.com. This data is now in good format for grouping and visualisation using Pandas.

Data summarisation and aggregation

With the data cleansed, we now have non-uniform samples of the weather at a given station throughout the year, at a sub-hour level. To make meaningful plots on this data, we can aggregate over the days and months to gain an overall view and to compare across stations.

```
# Looking at the working days only and create a daily data set of working days:

wet_cycling = data[data['working_day'] == True].groupby('day')['get_wet_cycling'].any()

wet_cycling = pd.DataFrame(wet_cycling).reset_index()

# Group by month for display - monthly data set for plots.

wet_cycling['month'] = wet_cycling['day'].apply(lambda x: x.month)

monthly = wet_cycling.groupby('month')['get_wet_cycling'].value_counts().reset_index()

monthly.rename(columns={"get_wet_cycling":"Rainy", 0:"Days"}, inplace=True)

monthly.replace({"Rainy": {True: "Wet", False:"Dry"}}, inplace=True)

monthly['month_name'] = monthly['month'].apply(lambda x: calendar.month_abbr[x])

# Get aggregate stats for each day in the dataset on rain in general - for heatmaps.

rainy_days = data.groupby(['day']).agg({

"rain": {"rain": lambda x: (x &gt; 0.0).any(),

"rain_amount": "sum"},

"total_rain": {"total_rain": "max"},

"get_wet_cycling": {"get_wet_cycling": "any"}

})

# clean up the aggregated data to a more easily analysed set:

rainy_days.reset_index(drop=False, inplace=True) # remove the 'day' as the index

rainy_days.rename(columns={"":"date"}, inplace=True) # The old index column didn't have a name - add "date" as name
```

```
rainy_days.columns = rainy_days.columns.droplevel(level=0) # The aggregation left us with a multi-index

# Remove the top level of this index.

rainy_days['rain'] = rainy_days['rain'].astype(bool) # Change the "rain" column to True/False values

# Add the number of rainy hours per day this to the rainy_days dataset.

temp = data.groupby(["day", "hour_of_day"])['raining'].any()

temp = temp.groupby(level=[0]).sum().reset_index()

temp.rename(columns={'raining': 'hours_raining'}, inplace=True)

temp['day'] = temp['day'].apply(lambda x: x.to_datetime().date())

rainy_days = rainy_days.merge(temp, left_on='date', right_on='day', how='left')

rainy_days.drop('day', axis=1, inplace=True)

print "In the year, there were {} rainy days of {} at {}".format(rainy_days['rain'].sum(), len(rainy_days), station)

print "It was wet while cycling {} working days of {} at {}".format(wet_cycling['get_wet_cycling'].sum(),

len(wet_cycling),

station)

print "You get wet cycling {} % of the time!!".format(wet_cycling['get_wet_cycling'].sum()*1.0*100/len(wet_cycling))
```

At this point, we have two basic data frames which we can use to visualise patterns for the city being analysed.

- **Visualisation using Pandas and Seaborn**

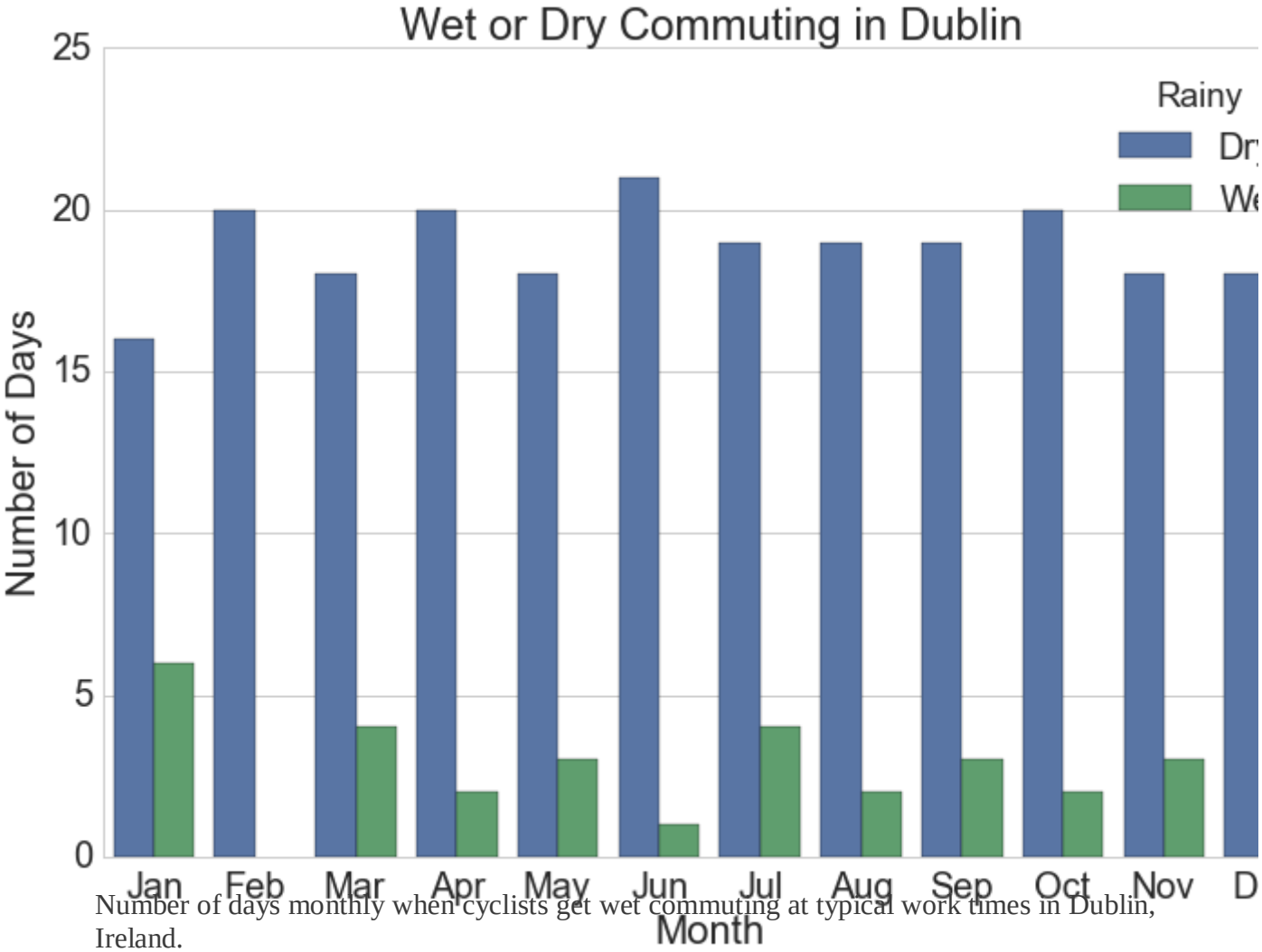
  At this point, we can start to plot the data. It's well worth reading the documentation on plotting with Pandas, and looking over the API of Seaborn, a high-level data visualisation library that is a level above matplotlib.
  This is not a tutorial on how to plot with seaborn or pandas – that'll be a seperate blog post, but rather instructions on how to reproduce the plots shown on this blog post.

  Barchart of Monthly Rainy Cycles

  The monthly summarised rainfall data is the source for this chart.

```
# Monthly plot of rainy days

plt.figure(figsize=(12,8))
```

```
sns.set_style("whitegrid")

sns.set_context("notebook", font_scale=2)

sns.barplot(x="month_name", y="Days", hue="Rainy", data=monthly.sort_values(['month',
'Rainy'])) plt.xlabel("Month")

plt.ylabel("Number of Days")

plt.title("Wet or Dry Commuting in {}".format(station))
```



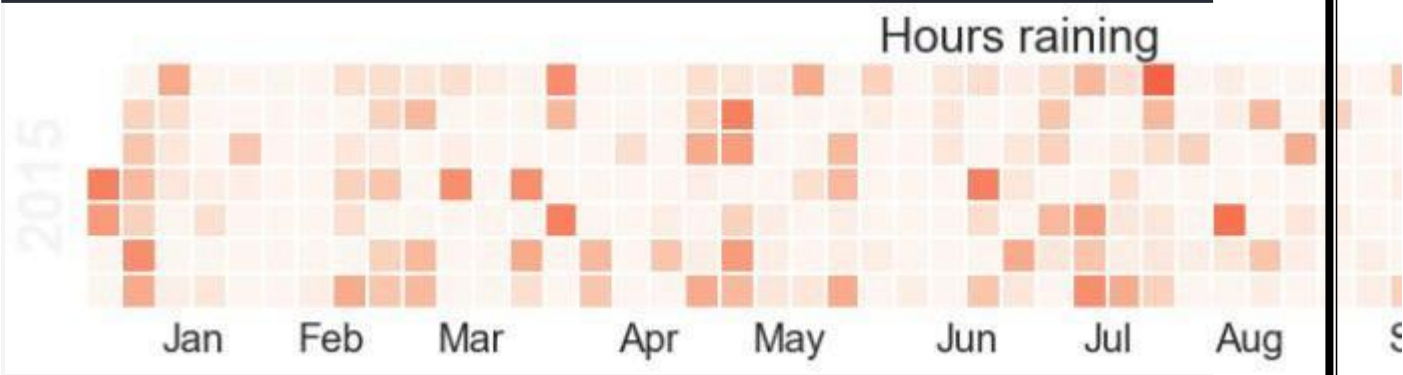
Number of days monthly when cyclists get wet commuting at typical work times in Dublin, Ireland.

Heatmaps of Rainfall and Rainy Hours per day

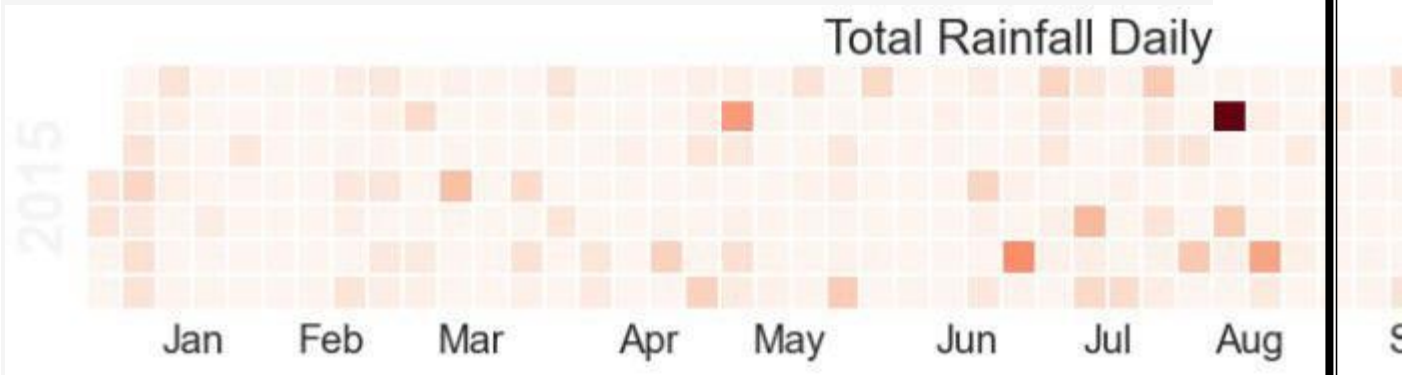
The heatmaps shown on the blog post are generated using the “calmap” python library, installable using pip. Simply import the library, and form a Pandas series with a

DateTimeIndex and the library takes care of the rest. I had some difficulty here with font sizes, so had to increase the size of the plot overall to counter.

```
import calmap

temp = rainy_days.copy().set_index(pd.DatetimeIndex(analysis['rainy_days']['date']))

#temp.set_index('date', inplace=True)

fig, ax = calmap.calendarplot(temp['hours_raining'], fig_kws={"figsize":(15,4)})

plt.title("Hours raining")

fig, ax = calmap.calendarplot(temp['total_rain'], fig_kws={"figsize":(15,4)})

plt.title("Total Rainfall Daily")
```



The Calmap package is very useful for generating heatmaps. Note that if you have highly outlying points of data, these will skew your color mapping considerably – I'd advise removing or reducing them for visualisation purposes.



Heatmap of total rainfall daily over 2015. Note that if you are looking at rainfall data like this, outlying values such as that in August in this example will skew the overall visualisation and reduce the colour-resolution of smaller values. Its best to normalise the data or reduce the outliers prior to plotting.

Exploratory Line Plots

Remember that Pandas can be used on its own for quick visualisations of the data – this is really useful for error checking and sense checking your results. For example:

```
temp[['get_wet_cycling', 'total_rain', 'hours_raining']].plot()
```

Quickly view and analyse your data with Pandas straight out of the box. The .plot() command will plot against the axis, but you can specify x and y variables as required.

Comparison of Every City in Dataset

To compare every city in the dataset, summary stats for each city were calculated in advance and then the plot was generated using the seaborn library. To achieve this as quickly as possible, I wrapped the entire data preparation and cleansing phase described above into a single function called "analyse data", used this function on each city's dataset, and extracted out the pieces of information needed for the plot

.

Here's the wrapped analyse_data function:

```
def analyse_station(data_raw, station):
"""
Function to analyse weather data for a period from one weather station.
Args:
data_raw (pd.DataFrame): Pandas Dataframe made from CSV downloaded from wunderground.com
station (String): Name of station being analysed (for comments)
Returns:
dict: Dictionary with analysis in keys:
data: Processed and cleansed data
monthly: Monthly aggregated statistics on rainfall etc.
wet_cycling: Data on working days and whether you get wet or not commuting
```

```
rainy_days: Daily total rainfall for each day in dataset.
"""
# Give the variables some friendlier names and convert types as necessary.
data_raw['temp'] = data_raw['TemperatureC'].astype(float)
data_raw['rain'] = data_raw['HourlyPrecipMM'].astype(float)
data_raw['total_rain'] = data_raw['dailyrainMM'].astype(float)
data_raw['date'] = data_raw['DateUTC'].apply(parser.parse)
data_raw['humidity'] = data_raw['Humidity'].astype(float)
data_raw['wind_direction'] = data_raw['WindDirectionDegrees']
data_raw['wind'] = data_raw['WindSpeedKMH']
# Extract out only the data we need.
data = data_raw.loc[:, ['date', 'station', 'temp', 'rain', 'total_rain', 'humidity', 'wind']]
data = data[(data['date'] &gt;= datetime(2015,1,1)) &amp; (data['date'] &lt;=
datetime(2015,12,31))]
# There's an issue with some stations that record rainfall ~-2500 where data is missing.
if (data['rain'] &lt; -500).sum() &gt; 10:
print("There's more than 10 messed up days for {}".format(station))
# remove the bad samples
data = data[data['rain'] &gt; -500]
# Assign the "day" to every date entry
data['day'] = data['date'].apply(lambda x: x.date())
# Get the time, day, and hour of each timestamp in the dataset
data['time_of_day'] = data['date'].apply(lambda x: x.time())
data['day_of_week'] = data['date'].apply(lambda x: x.weekday())
```

```
data['hour_of_day'] = data['time_of_day'].apply(lambda x: x.hour)

# Mark the month for each entry so we can look at monthly patterns

data['month'] = data['date'].apply(lambda x: x.month)

# Is each time stamp on a working day (Mon-Fri)

data['working_day'] = (data['day_of_week'] &gt;= 0) &amp; (data['day_of_week'] &lt;= 4)

# Classify into morning or evening times (assuming travel between 8.15-9am and 5.15-6pm)

data['morning'] = (data['time_of_day'] &gt;= time(8,15)) &amp; (data['time_of_day'] &lt;=
time(9,0))

data['evening'] = (data['time_of_day'] &gt;= time(17,15)) &amp; (data['time_of_day'] &lt;=
time(18,0))

# If there's any rain at all, mark that!

data['raining'] = data['rain'] &gt; 0.0

# You get wet cycling if its a working day, and its raining at the travel times!

data['get_wet_cycling'] = (data['working_day']) &amp; ((data['morning'] &amp; data['rain']) |

(data['evening'] &amp; data['rain']))

# Looking at the working days only:

wet_cycling = data[data['working_day'] == True].groupby('day')['get_wet_cycling'].any()

wet_cycling = pd.DataFrame(wet_cycling).reset_index()

# Group by month for display

wet_cycling['month'] = wet_cycling['day'].apply(lambda x: x.month)

monthly = wet_cycling.groupby('month')['get_wet_cycling'].value_counts().reset_index()

monthly.rename(columns={"get_wet_cycling":"Rainy", 0:"Days"}, inplace=True)

monthly.replace({"Rainy": {True: "Wet", False:"Dry"}}, inplace=True)

monthly['month_name'] = monthly['month'].apply(lambda x: calendar.month_abbr[x])
```

```
# Get aggregate stats for each day in the dataset.

rainy_days = data.groupby(['day']).agg({

"rain": {"rain": lambda x: (x &gt; 0.0).any(),

"rain_amount": "sum"},

"total_rain": {"total_rain": "max"},

"get_wet_cycling": {"get_wet_cycling": "any"}

})

rainy_days.reset_index(drop=False, inplace=True)

rainy_days.columns = rainy_days.columns.droplevel(level=0)

rainy_days['rain'] = rainy_days['rain'].astype(bool)

rainy_days.rename(columns={"":"date"}, inplace=True)

# Also get the number of hours per day where its raining, and add this to the rainy_days
dataset.

temp = data.groupby(["day", "hour_of_day"])['raining'].any()

temp = temp.groupby(level=[0]).sum().reset_index()

temp.rename(columns={'raining': 'hours_raining'}, inplace=True)

temp['day'] = temp['day'].apply(lambda x: x.to_datetime().date())

rainy_days = rainy_days.merge(temp, left_on='date', right_on='day', how='left')

rainy_days.drop('day', axis=1, inplace=True)

print "In the year, there were {} rainy days of {} at {}".format(rainy_days['rain'].sum(),
len(rainy_days), station)

print "It was wet while cycling {} working days of {} at
{}".format(wet_cycling['get_wet_cycling'].sum(),

len(wet_cycling),

station)
```

```
print "You get wet cycling {} % of the
time!!".format(wet_cycling['get_wet_cycling'].sum()*1.0*100/len(wet_cycling))

return {"data":data, 'monthly':monthly, "wet_cycling":wet_cycling, 'rainy_days': rainy_days}
```

The following code was used to individually analyse the raw data for each city in turn. Note that this could be done in a more memory efficient manner by simply saving the aggregate statistics for each city at first rather than loading all into memory. I would recommend that approach if you are dealing with more cities etc.

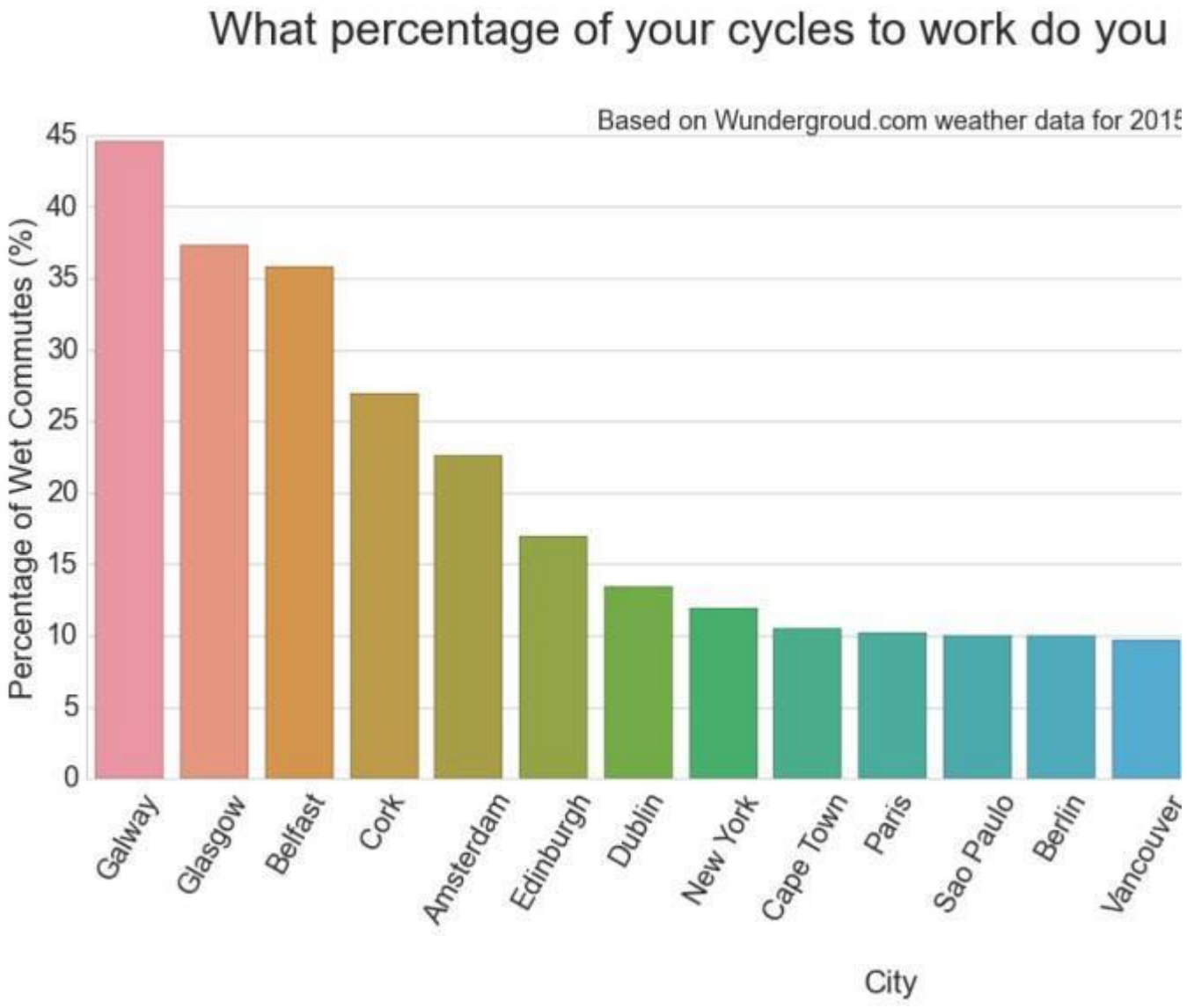
```
# Load up each of the stations into memory.

stations = [

("IAMSTERD55", "Amsterdam"),

("IBCNORTH17", "Vancouver"),

("IBELFAST4", "Belfast"),

("IBERLINB54", "Berlin"),

("ICOGALWA4", "Galway"),

("ICOMUNID56", "Madrid"),

("IDUBLIND35", "Dublin"),

("ILAZIORO71", "Rome"),

("ILEDEFRA6", "Paris"),

("ILONDONL28", "London"),

("IMUNSTER11", "Cork"),

("INEWSOUT455", "Sydney"),

("ISOPAULO61", "Sao Paulo"),

("IWESTERN99", "Cape Town"),

("KCASANFR148", "San Francisco"),

("KNYBROOK40", "New York"),
```

```
("IRENFREW4", "Glasgow"),

("IENGLAND64", "Liverpool"),

('IEDINBUR6', 'Edinburgh')

]

data = []

for station in stations:

weather = {}

print "Loading data for station: {}".format(station[1])

weather['data'] = pd.DataFrame.from_csv("data/{}_weather.csv".format(station[0]))

weather['station'] = station[0]

weather['name'] = station[1]

data.append(weather)

for ii in range(len(data)):

print "Processing data for {}".format(data[ii]['name'])

data[ii]['result'] = analyse_station(data[ii]['data'], data[ii]['station'])

# Now extract the number of wet days, the number of wet cycling days, and the number of wet commutes for a single chart.

output = []

for ii in range(len(data)):

temp = {

"total_wet_days": data[ii]['result']['rainy_days']['rain'].sum(),

"wet_commutes": data[ii]['result']['wet_cycling']['get_wet_cycling'].sum(),

"commutes": len(data[ii]['result']['wet_cycling']),

"city": data[ii]['name']
```

```
}

temp['percent_wet_commute'] = (temp['wet_commutes'] *1.0 / temp['commutes'])*100

output.append(temp)

output = pd.DataFrame(output)
```

The final step in the process is to actually create the diagram using Seaborn.

```
# Generate plot of percentage of wet commutes

plt.figure(figsize=(20,8))

sns.set_style("whitegrid") # Set style for seaborn output

sns.set_context("notebook", font_scale=2)

sns.barplot(x="city", y="percent_wet_commute",
data=output.sort_values('percent_wet_commute', ascending=False))

plt.xlabel("City")

plt.ylabel("Percentage of Wet Commutes (%)")

plt.suptitle("What percentage of your cycles to work do you need a raincoat?", y=1.05,
fontsize=32)

plt.title("Based on Wundergroud.com weather data for 2015", fontsize=18)

plt.xticks(rotation=  )

plt.savefig("images/city_comparison_wet_commutes.png", bbox_inches='tight')
```



Percentage of times you got wet cycling to work in 2015 for cities globally. Galway comes out consistently as one of the wettest places for a cycling commute in the data available, but 2015 was a particularly bad year for Irish weather. Here's hoping for 2016.

Conclusion: Hence we have thourouly studied how to Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.

Practical No.14	Group C
Title	Write a simple program in SCALA using Apache Spark framework.
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Practical No. 14

Title: Scala Apache Framework

- **Problem Statement:**
  Write a simple program in SCALA using Apache Spark framework.

- **Prerequisites:**

  Subjects: Discrete Mathematics, Database Management Systems
  Tools/Software Requirements: Apache Hadoop
  Processor/Hardware Requirements: Intel Core 2 Duo or above, 2 GB RAM or above
  Operating System: 64 Bit Windows/Linux/Mac OS
  Programming Language: Scala

- **Objectives:**

  To learn concepts of Data Wrangling, import /read/scrap meaningful insights from dataset.

Practical No: 14

Aim: Design a distributed application using Map Reduce which processes a log file of a system.

Theory: In this tutorial, you will learn to use Hadoop with MapReduce Examples. The input data used is SalesJan2009.csv. It contains Sales related information like Product name, price, payment mode, city, country of client etc. The goal is to **Find out Number of Products Sold in Each Country.**

First Hadoop MapReduce Program

Now in this MapReduce tutorial, we will create our first Java MapReduce program:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Transaction_date	Product	Price	Payment_	Name	City	State	Country	Account_Created	Last_Login	Latitude	Longitude
2	01-02-2009 06:17	Product1	1200	Mastercar	carolina	Basildon	England	United Kir	01-02-2009 06:00	01-02-2009 06:08	51.5	-1.11667
3	01-02-2009 04:53	Product1	1200	Visa	Betina	Parkville	MO	United Sta	01-02-2009 04:42	01-02-2009 07:49	39.195	-94.6819
4	01-02-2009 13:08	Product1	1200	Mastercar	Federica e	Astoria	OR	United Sta	01-01-2009 16:21	01-03-2009 12:32	46.18806	-123.83
5	01-03-2009 14:44	Product1	1200	Visa	Gouya	Echuca	Victoria	Australia	9/25/05 21:13	01-03-2009 14:22	-36.1333	144.75
6	01-04-2009 12:56	Product2	3600	Visa	Gerd W	Cahaba He	AL	United Sta	11/15/08 15:47	01-04-2009 12:45	33.52056	-86.8025
7	01-04-2009 13:19	Product1	1200	Visa	LAURENCE	Mickleton	NJ	United Sta	9/24/08 15:19	01-04-2009 13:04	39.79	-75.2381
8	01-04-2009 20:11	Product1	1200	Mastercar	Fleur	Peoria	IL	United Sta	01-03-2009 09:38	01-04-2009 19:45	40.69361	-89.5889
9	01-02-2009 20:09	Product1	1200	Mastercar	adam	Martin	TN	United Sta	01-02-2009 17:43	01-04-2009 20:01	36.34333	-88.8503
10	01-04-2009 13:17	Product1	1200	Mastercar	Renee Elis	Tel Aviv	Tel Aviv	Israel	01-04-2009 13:03	01-04-2009 22:10	32.06667	34.76667
11	01-04-2009 14:11	Product1	1200	Visa	Aidan	Chatou	Ile-de-Fra	France	06-03-2008 04:22	01-05-2009 01:17	48.88333	2.15
12	01-05-2009 02:42	Product1	1200	Diners	Stacy	New York	NY	United Sta	01-05-2009 02:23	01-05-2009 04:59	40.71417	-74.0064
13	01-05-2009 05:39	Product1	1200	Amex	Heidi	Eindhover	Noord-Bra	Netherlan	01-05-2009 04:55	01-05-2009 08:15	51.45	5.466667
14	01-02-2009 09:16	Product1	1200	Mastercar	Sean	Shavano P	TX	United Sta	01-02-2009 08:32	01-05-2009 09:05	29.42389	-98.4933
15	01-05-2009 10:08	Product1	1200	Visa	Georgia	Eagle	ID	United Sta	11-11-2008 15:53	01-05-2009 10:05	43.69556	-116.353
16	01-02-2009 14:18	Product1	1200	Visa	Richard	Riverside	NJ	United Sta	12-09-2008 12:07	01-05-2009 11:01	40.03222	-74.9578
17	01-04-2009 01:05	Product1	1200	Diners	Leanne	Julianstov	Meath	Ireland	01-04-2009 00:00	01-05-2009 13:36	53.67722	-6.31917

Data of SalesJan2009

Ensure you have Hadoop installed. Before you start with the actual process, change user to 'hduser' (id used while Hadoop configuration, you can switch to the userid used during your Hadoop programming config).

su - hduser_

```
guru99@guru99-VirtualBox:~$ su - hduser_
Password:
hduser_@guru99-VirtualBox:~$
```

Step 1)

Create a new directory with name **MapReduceTutorial** as shwon in the below MapReduce example

sudo mkdir MapReduceTutorial

```
hduser_@guru99-VirtualBox:~$ sudo mkdir MapReduceTutorial
```

Give permissions

sudo chmod -R 777 MapReduceTutorial

```
hduser_@guru99-VirtualBox:~$ sudo chmod -R 777 MapReduceTutorial
```

SalesMapper.java package

SalesCountry; import

java.io.IOException;

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesMapper extends MapReduceBase implements Mapper <LongWritable,
Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector <Text,
IntWritable> output, Reporter reporter) throws IOException {

                String valueString = value.toString();
                String[] SingleCountryData = valueString.split(",");
                output.collect(new Text(SingleCountryData[7]), one);
        }
}
```

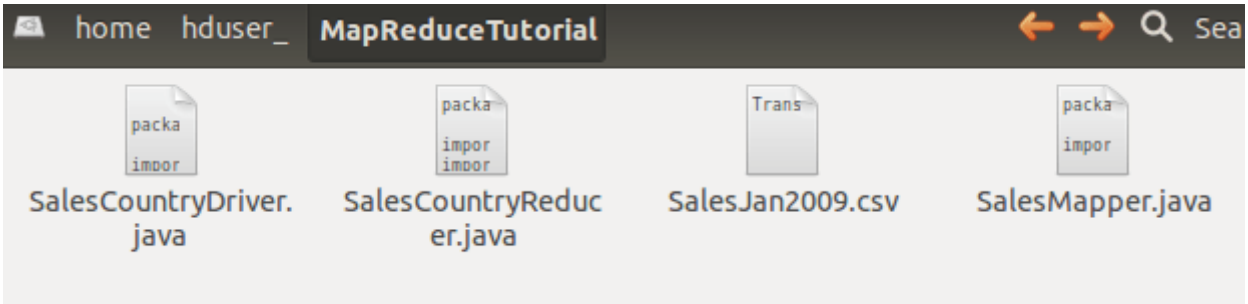
SalesCountryReducer.java

```
package SalesCountry;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import  org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {

        public void reduce(Text t_key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException {
                Text key = t_key;
                int frequencyForCountry = 0;
                while (values.hasNext()) {
                        // replace type of value with the actual type of our value
                        IntWritable value = (IntWritable) values.next();
                        frequencyForCountry += value.get();

                }
                output.collect(key, new IntWritable(frequencyForCountry));
        }
}
```

SalesCountryDriver.java

```
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class SalesCountryDriver {
   public static void main(String[] args) {
      JobClient my_client = new JobClient();
      // Create a configuration object for the job
      JobConf job_conf = new JobConf(SalesCountryDriver.class);

      // Set a name of the Job
      job_conf.setJobName("SalePerCountry");

      // Specify data type of output key and value
      job_conf.setOutputKeyClass(Text.class);
      job_conf.setOutputValueClass(IntWritable.class);

      // Specify names of Mapper and Reducer Class
      job_conf.setMapperClass(SalesCountry.SalesMapper.class);
      job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

      // Specify formats of the data type of Input and output
      job_conf.setInputFormat(TextInputFormat.class);
      job_conf.setOutputFormat(TextOutputFormat.class);

      // Set input and output directories using command line arguments,
      //arg[0] = name of input directory on HDFS, and arg[1] = name of output directory to
be created to store the output file.

      FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
      FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

      my_client.setConf(job_conf);
      try {
         // Run the job
         JobClient.runJob(job_conf);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Download Files Here



Check the file permissions of all these files

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ ls -al
total 144
drwxrwxrwx 2 root    root      4096 May  5 15:00 .
drwxr-xr-x 6 hduser_ hadoop_   4096 May  5 14:53 ..
-rw-rw-r-- 1 guru99  guru99    1367 May  5 02:28 SalesCountryDriver.java
-rw-rw-r-- 1 guru99  guru99     749 May  5 02:28 SalesCountryReducer.jav
-rw-rw-r-- 1 guru99  guru99  123637 May  5 02:28 SalesJan2009.csv
-rw-rw-r-- 1 guru99  guru99     659 May  5 02:28 SalesMapper.java
```

and if 'read' permissions are missing then grant the same-

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ sudo chmod +r *.*
```

Step 2)

Export classpath as shown in the below Hadoop example

export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_HOME/lib/*"

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ export CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoo
p-mapreduce-client-core-2.2.0.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.2.0
.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-2.2.0.jar:~/MapReduceTutorial/SalesCountry/*:$HADOOP_H
OME/lib/*"
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

Step 3)

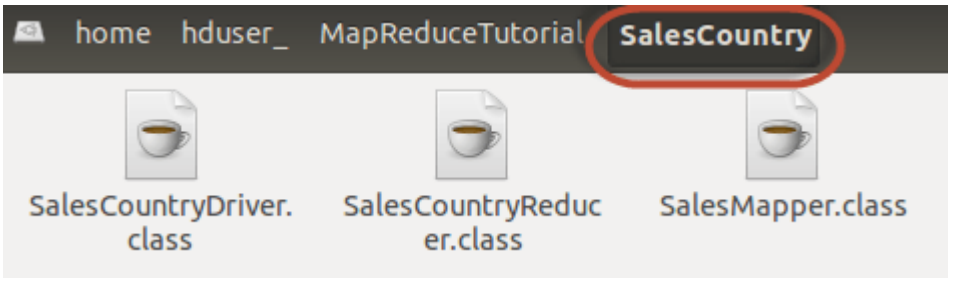
Compile Java files (these files are present in directory **Final-MapReduceHandsOn**). Its class files will be put in the package directory

javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.java

```
hduser_@guru99-VirtualBox:~/MapReduceTutorial$ javac -d . SalesMapper.java SalesCountryReducer.java SalesC
ountryDriver.java
/home/guru99/Downloads/hadoop/share/hadoop/common/hadoop-common-2.2.0.jar(org/apache/hadoop/fs/Path.class)
: warning: Cannot find annotation method 'value()' in type 'LimitedPrivate': class file for org.apache.had
oop.classification.InterfaceAudience not found
1 warning
hduser_@guru99-VirtualBox:~/MapReduceTutorial$
```

This warning can be safely ignored.

This compilation will create a directory in a current directory named with package name specified in the java source file (i.e. **SalesCountry** in our case) and put all compiled class files in it.

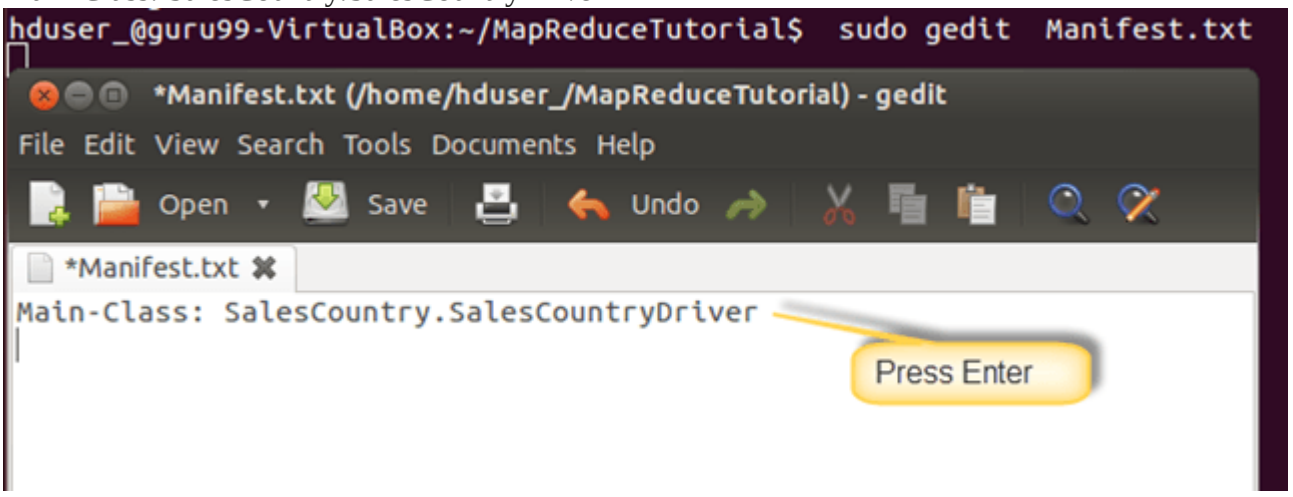


Step 4)

Create a new file **Manifest.txt**

sudo gedit Manifest.txt
add following lines to it,

Main-Class: SalesCountry.SalesCountryDriver

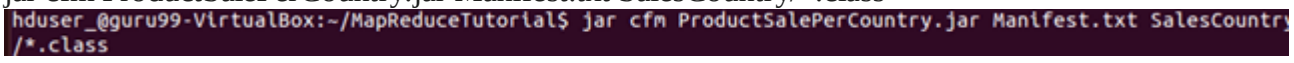


SalesCountry.SalesCountryDriver is the name of main class. Please note that you have to hit enter key at end of this line.

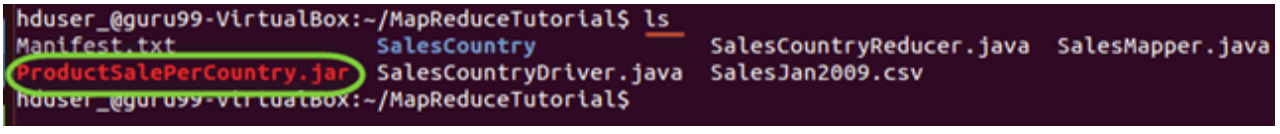
Step 5)

Create a Jar file

jar cfm ProductSalePerCountry.jar Manifest.txt SalesCountry/*.class



Check that the jar file is created



Step 6)

Start Hadoop

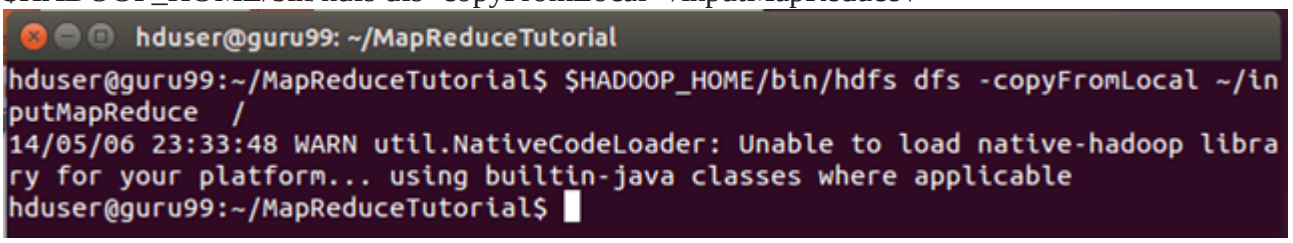
$HADOOP_HOME/sbin/start-dfs.sh
$HADOOP_HOME/sbin/start-yarn.sh

Step 7)

Copy the File **SalesJan2009.csv** into **~/inputMapReduce**

Now Use below command to copy **~/inputMapReduce** to HDFS.

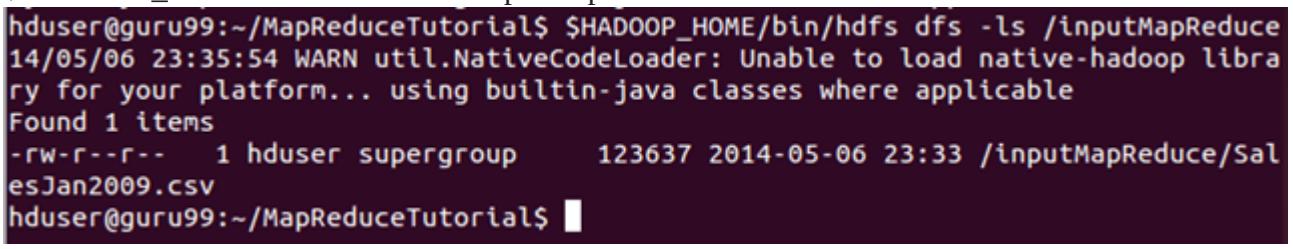
$HADOOP_HOME/bin/hdfs dfs -copyFromLocal ~/inputMapReduce /



We can safely ignore this warning.

Verify whether a file is actually copied or not.

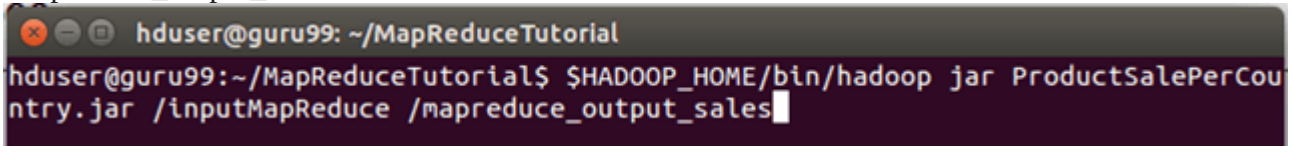
$HADOOP_HOME/bin/hdfs dfs -ls /inputMapReduce



Step 8)

Run MapReduce job

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales



This will create an output directory named mapreduce_output_sales on HDFS. Contents of this directory will be a file containing product sales per country.

Step 9)

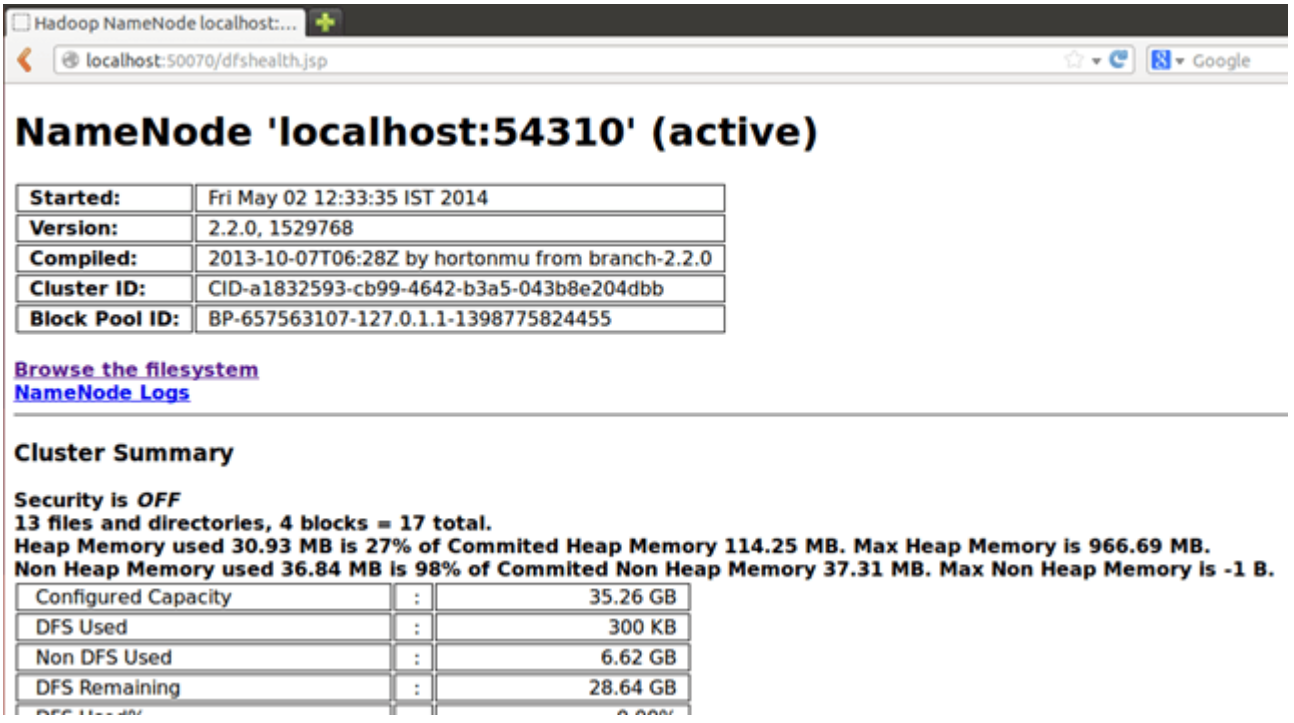
The result can be seen through command interface as,

$HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_output_sales/part-00000

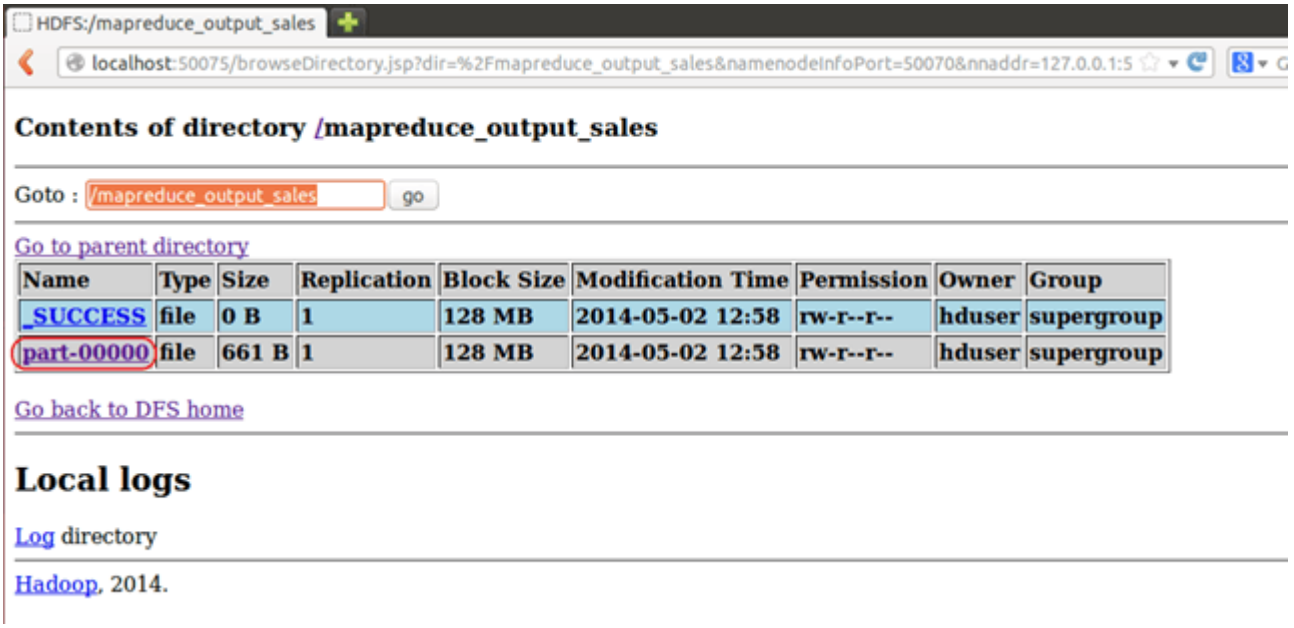
```
hduser@guru99: ~/MapReduceTutorial
hduser@guru99:~/MapReduceTutorial$ $HADOOP_HOME/bin/hdfs dfs -cat /mapreduce_out
put_sales/part-00000
14/05/02 13:03:46 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Argentina       1
Australia       38
Austria 7
Bahrain 1
Belgium 8
Bermuda 1
Brazil  5
Bulgaria        1
CO      1
Canada  76
Cayman Isls     1
```

Results can also be seen via a web interface as-

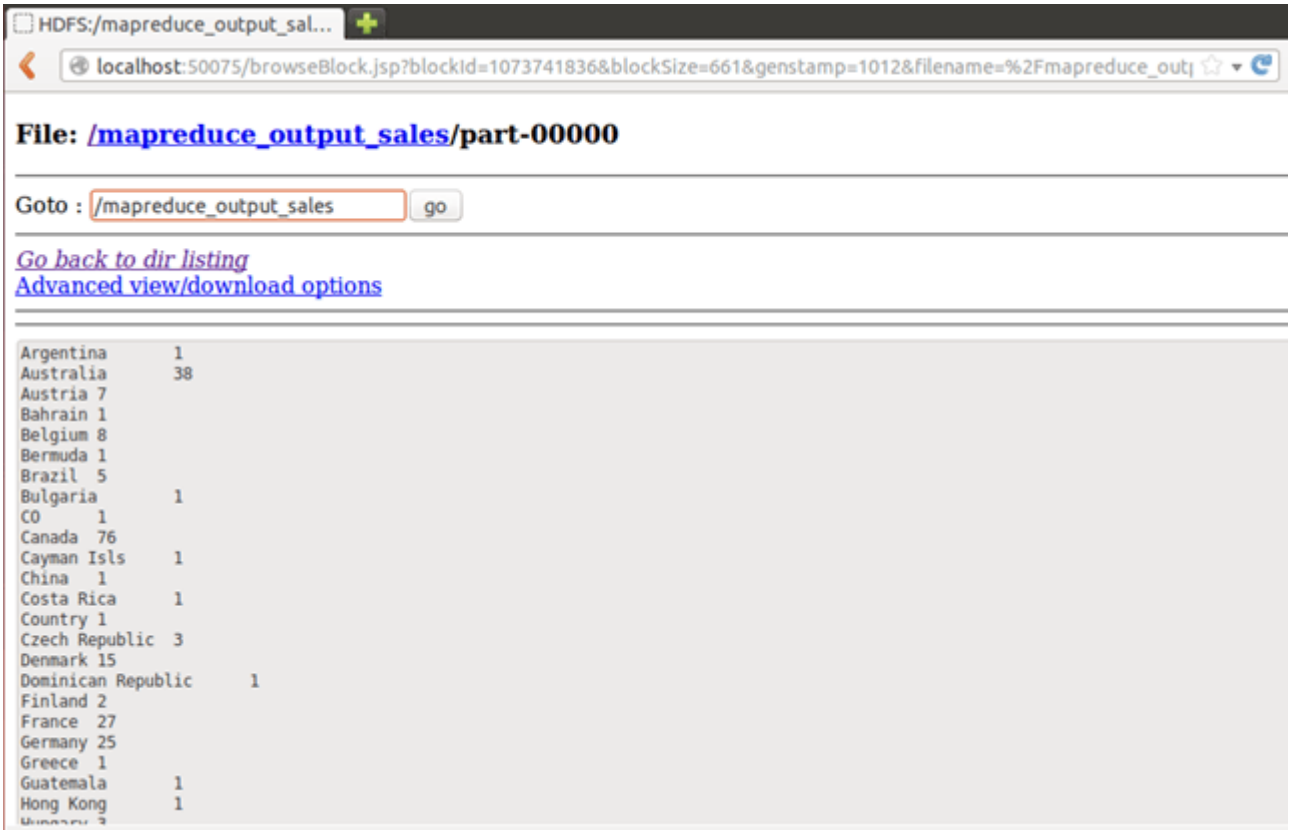
Open r in a web browser.



Now select **'Browse the filesystem'** and navigate to **/mapreduce_output_sales**



Open **part-r-00000**



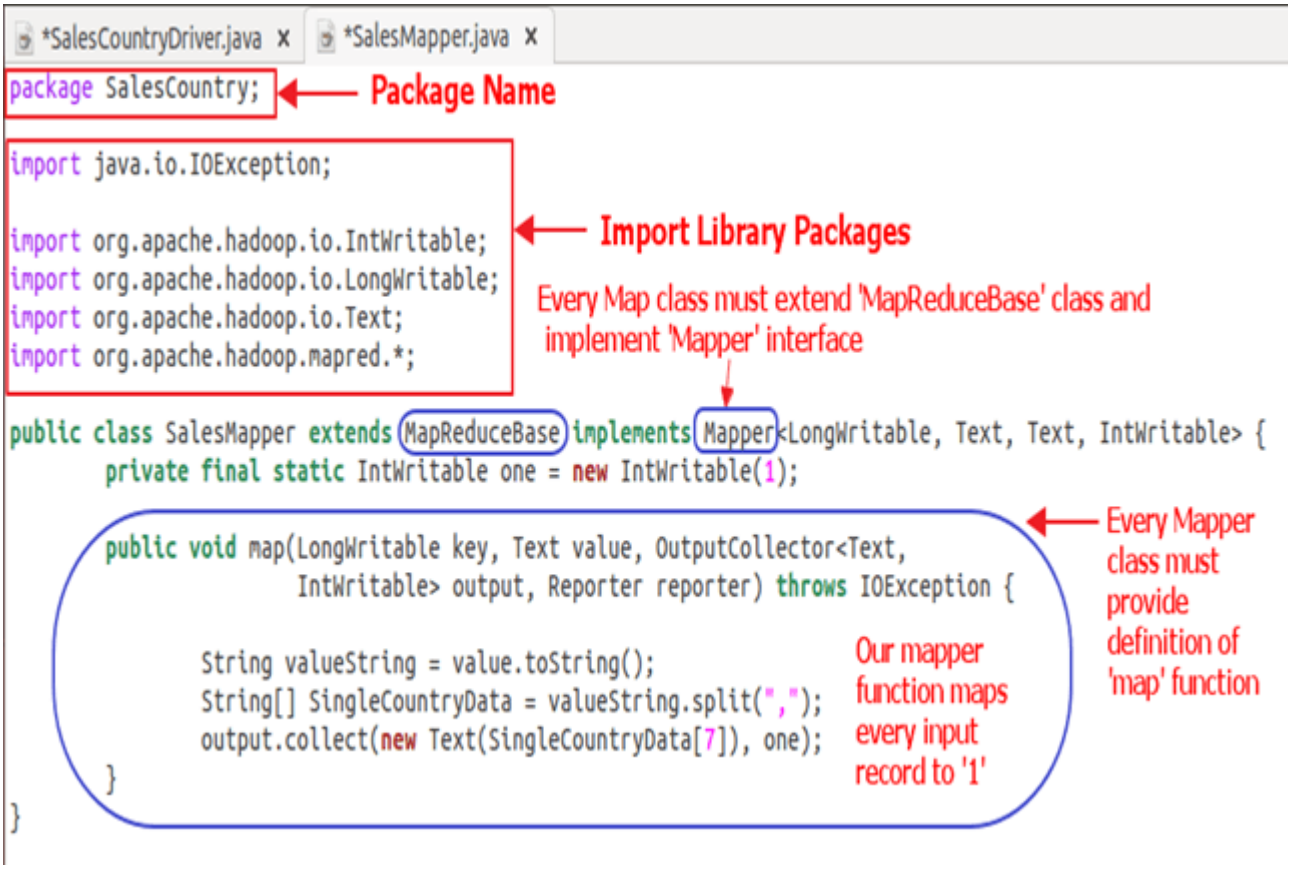
Explanation of SalesMapper Class

In this section, we will understand the implementation of **SalesMapper** class.

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of our package. Please note that output of compilation, **SalesMapper.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesMapper** class-



Sample Code Explanation:

1. SalesMapper Class Definition-

public class SalesMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

Every mapper class must be extended from **MapReduceBase** class and it must implement **Mapper** interface.

2. Defining 'map' function-

public void map(LongWritable key,
Text value,
OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException
The main part of Mapper class is a **'map()'** method which accepts four arguments.

At every call to **'map()'** method, a **key-value** pair (**'key'** and **'value'** in this code) is passed.

'map()' method begins by splitting input text which is received as an argument. It uses the tokenizer to split these lines into words.

String valueString = value.toString();
String[] SingleCountryData = valueString.split(",");
Here, **','** is used as a delimiter.

After this, a pair is formed using a record at 7th index of array **'SingleCountryData'** and a value **'1'**.

output.collect(new Text(SingleCountryData[7]), one);

We are choosing record at 7th index because we need **Country** data and it is located at 7th index in array **'SingleCountryData'**.

Please note that our input data is in the below format (where **Country** is at 7th index, with 0 as a starting index)-

Transaction_date,Product,Price,Payment_Type,Name,City,State,**Country**,Account_Created, Last_Login,Latitude,Longitude

An output of mapper is again a **key-value** pair which is outputted using **'collect()'** method of **'OutputCollector'**.

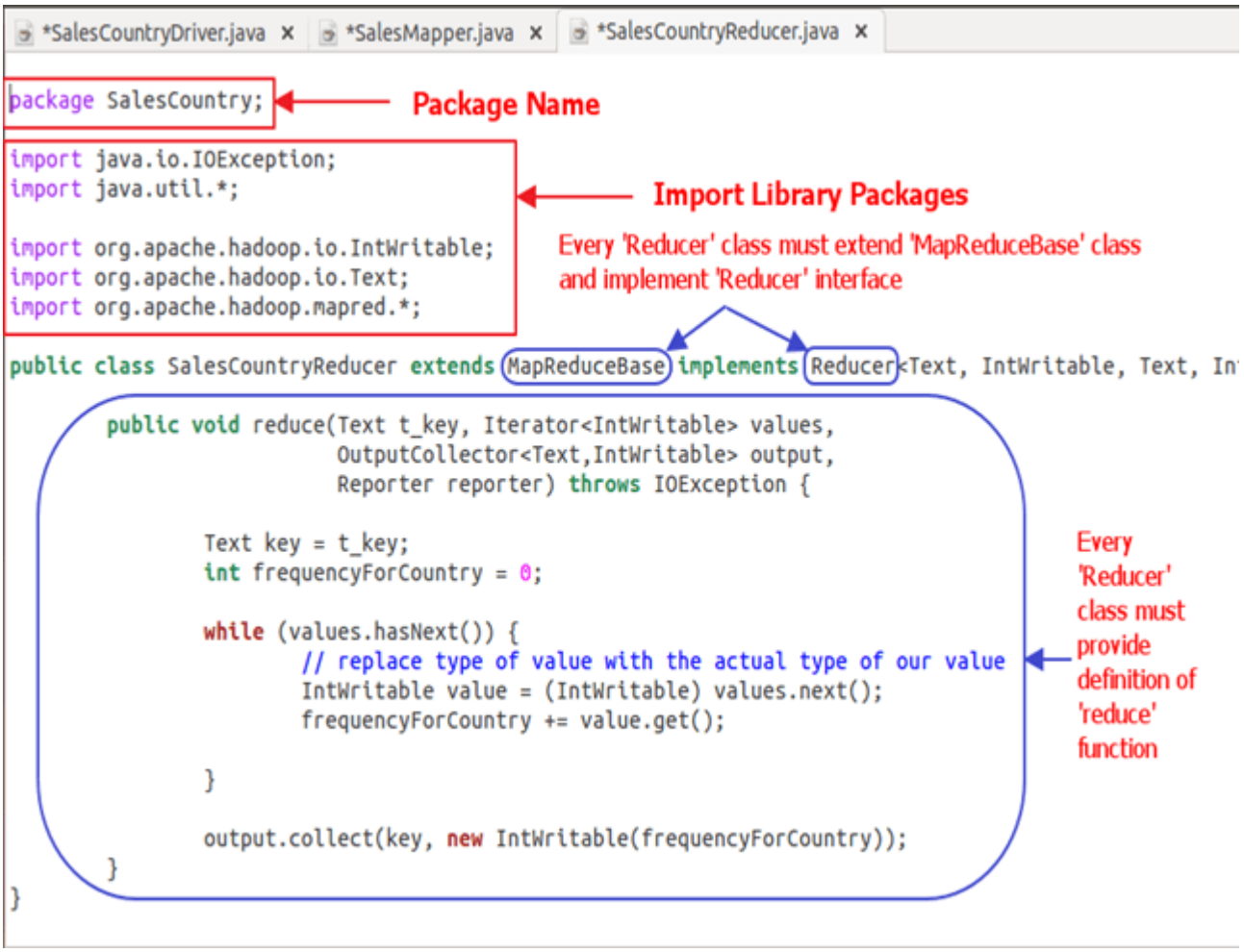
Explanation of SalesCountryReducer Class

In this section, we will understand the implementation of **SalesCountryReducer** class.

1. We begin by specifying a name of the package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryReducer.class** will go into a directory named by this package name: **SalesCountry**.

Followed by this, we import library packages.

Below snapshot shows an implementation of **SalesCountryReducer** class-



Code Explanation:

1. SalesCountryReducer Class Definition-

public class SalesCountryReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

Here, the first two data types, **'Text'** and **'IntWritable'** are data type of input key-value to the reducer.

Output of mapper is in the form of <CountryName1, 1>, <CountryName2, 1>. This output of mapper becomes input to the reducer. So, to align with its data type, **Text** and **IntWritable** are used as data type here.

The last two data types, 'Text' and 'IntWritable' are data type of output generated by reducer in the form of key-value pair.

Every reducer class must be extended from **MapReduceBase** class and it must implement **Reducer** interface.

2. Defining 'reduce' function-

```
public void reduce( Text t_key,
        Iterator<IntWritable> values,
        OutputCollector<Text,IntWritable> output,
        Reporter reporter) throws IOException {
```

An input to the **reduce()** method is a key with a list of multiple values.

For example, in our case, it will be-

<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>,<United Arab Emirates, 1>, <United Arab Emirates, 1>, <United Arab Emirates, 1>.

This is given to reducer as **<United Arab Emirates, {1,1,1,1,1,1}>**

So, to accept arguments of this form, first two data types are used, viz., **Text** and **Iterator<IntWritable>**. **Text** is a data type of key and **Iterator<IntWritable>** is a data type for list of values for that key.

The next argument is of type **OutputCollector<Text,IntWritable>** which collects the output of reducer phase.

reduce() method begins by copying key value and initializing frequency count to 0.

```
Text key = t_key;
int frequencyForCountry = 0;
```

Then, using '**while'** loop, we iterate through the list of values associated with the key and calculate the final frequency by summing up all the values.

```
 while (values.hasNext()) {
        // replace type of value with the actual type of our value
        IntWritable value = (IntWritable) values.next();
        frequencyForCountry += value.get();

    }
```

Now, we push the result to the output collector in the form of **key** and obtained **frequency count**.

Below code does this-

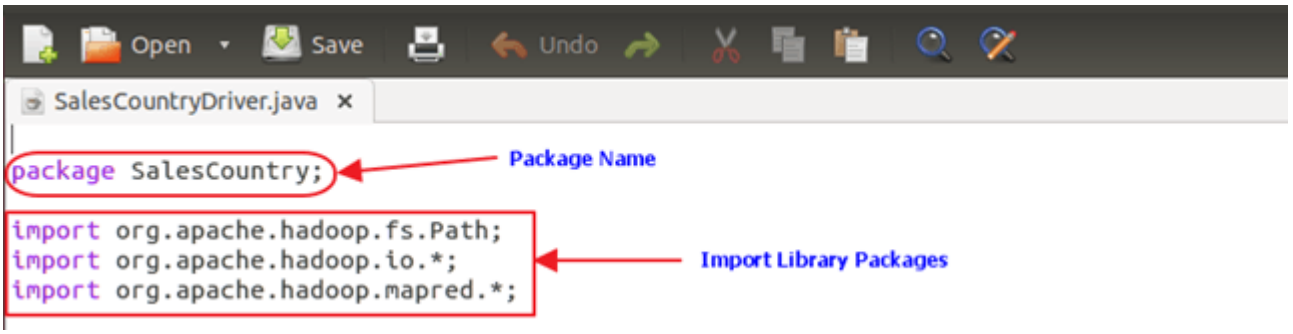
```
output.collect(key, new IntWritable(frequencyForCountry));
```

Explanation of SalesCountryDriver Class

In this section, we will understand the implementation of **SalesCountryDriver** class

1. We begin by specifying a name of package for our class. **SalesCountry** is a name of out package. Please note that output of compilation, **SalesCountryDriver.class** will go into directory named by this package name: **SalesCountry**.

Here is a line specifying package name followed by code to import library packages.



2. Define a driver class which will create a new client job, configuration object and advertise Mapper and Reducer classes.

The driver class is responsible for setting our MapReduce job to run in Hadoop. In this class, we specify **job name, data type of input/output and names of mapper and reducer classes**.

SalesCountryDriver.java

```
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class SalesCountryDriver {
        public static void main(String[] args) {
                JobClient my_client = new JobClient();
                // Create a configuration object for the job
                JobConf job_conf = new JobConf(SalesCountryDriver.class);

                // Set a name of the Job
                job_conf.setJobName("SalePerCountry");

                // Specify data type of output key and value
                job_conf.setOutputKeyClass(Text.class);
                job_conf.setOutputValueClass(IntWritable.class);

                // Specify names of Mapper and Reducer Class
                job_conf.setMapperClass(SalesCountry.SalesMapper.class);
                job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);

                // Specify formats of the data type of Input and output
                job_conf.setInputFormat(TextInputFormat.class);
                job_conf.setOutputFormat(TextOutputFormat.class);
```

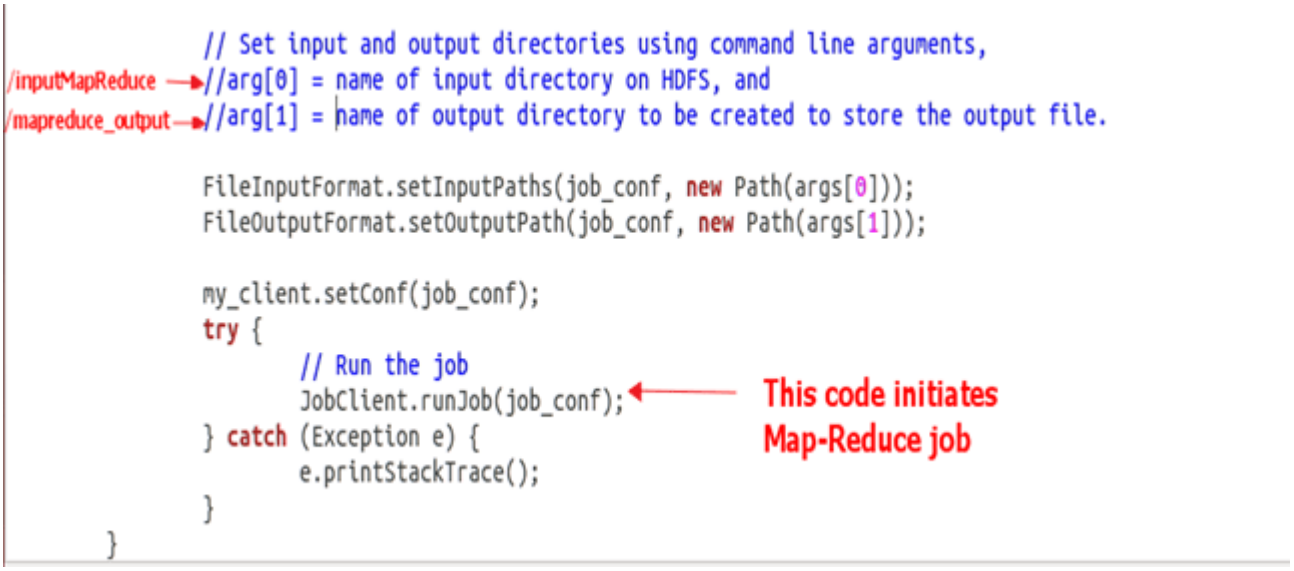
Start of definition of SalesCountryDriver class

Entry point to the application

3. In below code snippet, we set input and output directories which are used to consume input dataset and produce output, respectively.

arg[0] and **arg[1]** are the command-line arguments passed with a command given in MapReduce hands-on, i.e.,

$HADOOP_HOME/bin/hadoop jar ProductSalePerCountry.jar /inputMapReduce /mapreduce_output_sales



4. Trigger our job

Below code start execution of MapReduce job-

```
try {
   // Run the job
   JobClient.runJob(job_conf);
} catch (Exception e) {
   e.printStackTrace();
}
```

Conclusion: Hence we have thourouly studied how to design a distributed application using Map Reduce which processes a log file of a system.

Mini Project No.1	Group C
Title	Develop a movie recommendation model using the scikit-learn library in python. Refer dataset https://github.com/rashida048/Some-NLP-Projects/blob/master/movie_dataset.csv
Subject	Data Science and Big Data Analytics
Student Name	Pokharkar Swapnali Sahebrao
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

Mini Project No.2	Group C
Title	Use the following covid_vaccine_statewise.csv dataset and perform following analytics on the given dataset https://www.kaggle.com/sudalairajkumar/covid19-in-india?select=covid_vaccine_statewise.csv a. Describe the dataset b. Number of persons state wise vaccinated for first dose in India c. Number of persons state wise vaccinated for second dose in India d. Number of Males vaccinated d. Number of females vaccinated
Subject	Data Science and Big Data Analytics
Student Name	
Class	Third Year. (Computer Engineering.)
Division	A
Roll No	
Exam Seat No	
Date of Completion	
Assessment grade/marks	
Signature	

JCEI's JAIHIND COLLEGE OF ENGINEERING, KURAN

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the Data Science and Big Data Analytics Lab Manual under the subject

"Data Science and Big Data Analytics (310256)"

SUBMITTED BY

Mr./ Miss.

_____

Exam Seat No.

Is a Bonafede work carried out by student under the supervision of **Dr. A. A. Khatri** and it is submitted towards the partial fulfilment of the requirement of Second Year of Computer Engineering.

Dr. A. A. Khatri
Subject Teacher
Dept. of Computer Engineering.

Dr. A. A. Khatri
Head of Department
Dept. of Computer Engineering.

Dr. D. J. Garkal
Principal
JCEI's JAIHIND COLLEGE OF ENGINEERING, KURAN