Name: Harsh Chheda

Roll Number: 22-15405/31031521005

Class: Msc. Computer Science (Part 2)

Subject: Machine Learning

Year: 2022-23

INDEX

Name: Harsh Chheda
Roll Number: 22-15405/31031521005 MSC COMPUTER SCIENCE Subject: Machine Learning

Practical 1

Q1) Performing the basic data pre-processing steps.

→

1. Importing the libraries

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

2. Importing the dataset

```
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
print(X)
```

```
[17]
print(X)

[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
print(y)
```

```
print(y)
```

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

3. Taking care of missing data

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

4. Encoding categorical data

4.1 Encoding the Independent Variable:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))

print(X)
```

```
print(X)
```

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

4.2 Encoding the Dependent Variable

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

print(y)
```

```
print(y)
```

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

5. Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 1)

print(X_train)
```

```
1 print(X_train)
```

```
[[0.0 0.0 1.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
```

```
print(X_test)
```

```
1 print(X_test)
```

```
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

```
print(y_train)
```

```
1 print(y_train)
```

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

```
print(y_test)
```

```
1 print(y_test)

[0 1]
```

6. Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])

print(X_train)
```

```
1 print(X_train)

[[0.0 0.0 1.0 -0.19159184384578545 -1.0781259408412425]
 [0.0 1.0 0.0 -0.014117293757057777 -0.07013167641635372]
 [1.0 0.0 0.0 0.566708506533324 0.633562432710455]
 [0.0 0.0 1.0 -0.30453019390224867 -0.30786617274297867]
 [0.0 0.0 1.0 -1.9018011447007988 -1.420463615551582]
 [1.0 0.0 0.0 1.1475343068237058 1.232653363453549]
 [0.0 1.0 0.0 1.4379472069688968 1.5749910381638885]
 [1.0 0.0 0.0 -0.7401495441200351 -0.5646194287757332]]
```

```
print(X_test)
```

```
print(X_test)
```

```
[[0.0 1.0 0.0 -1.4661817944830124 -0.9069571034860727]
 [1.0 0.0 0.0 -0.44973664397484414 0.2056403393225306]]
```

Practical 2

Q1) Performing the basic data collection and data modelling steps.

→

Data Collection:

- Data collection is defined as the procedure of collecting, measuring and analysing accurate insights for research using standard validated techniques.
- A researcher can evaluate their hypothesis on the basis of collected data. In most cases, data collection is the primary and most important step for research, irrespective of the field of research.
- The approach of data collection is different for different fields of study, depending on the required information.
- The most critical objective of data collection is ensuring that information-rich and reliable data is collected for statistical analysis so that data-driven decisions can be made for research.

```
my_dict={'Name':["a","b","c","d","e","f","g"],
        'age':[20,27,35,45,55,43,35],
        'designation':["VP","CEO","CFO","VP","VP","CEO","MD"]}

import pandas as pd
import numpy as np
df=pd.DataFrame(my_dict)
df
```

	Name	age	designation
0	a	20	VP
1	b	27	CEO
2	c	35	CFO
3	d	45	VP
4	e	55	VP
5	f	43	CEO
6	g	35	MD

1. Let's first create our own CSV file using the data that is currently present in the DataFrame, we can store the data of this DataFrame in CSV format using the API called to_csv(...) of Pandas DataFrame as:

```
df.to_csv('Csv example')
df
```

	Name	age	designation
0	a	20	VP
1	b	27	CEO
2	c	35	CFO
3	d	45	VP
4	e	55	VP
5	f	43	CEO
6	g	35	MD

```
df_csv=pd.read_csv('Csv example')
df_csv
```

	Unnamed: 0	Name	age	designation
0	0	a	20	VP
1	1	b	27	CEO
2	2	c	35	CFO
3	3	d	45	VP
4	4	e	55	VP
5	5	f	43	CEO
6	6	g	35	MD

```
df.to_csv('CSV Ex',index=False)
df_csv=pd.read_csv('CSV Ex')
df_csv
```

	Name	age	designation
0	a	20	VP
1	b	27	CEO
2	c	35	CFO
3	d	45	VP
4	e	55	VP
5	f	43	CEO
6	g	35	MD

2. Load data from csv file and display data without headers

```
import pandas as pd
Location = "student-mat.csv"
df = pd.read_csv(Location, header=None)
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26	27	28	29	30	31	32
0	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
1	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
2	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
3	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
4	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15

```
import pandas as pd
Location = "student-mat.csv"
df = pd.read_csv(Location)
df.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	6	5	6	6
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	4	5	5	6
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	10	7	8	10
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	2	15	14	15
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	4	6	10	10

3. Loading Data from CSV File and Adding Headers

```
import pandas as pd
Location = "student-mat.csv"
# To add headers as we load the data...
df = pd.read_csv(Location, names=['RollNo','Names','Grades'])
# To add headers to a dataframe
df.columns = ['RollNo','Names','Grades']
df.head()
```

school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian	traveltime	studytime	failures	schoolsup	famsup	paid	activities	nursery	highe
GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother	2	2	0	yes	no	no	no	yes	ye
		17	U	GT3	T	1	1	at_home	other	course	father	1	2	0	no	yes	no	no	no	ye
		15	U	LE3	T	1	1	at_home	other	other	mother	1	2	3	yes	no	yes	no	yes	ye
				GT3	T	4	2	health	services	home	mother	1	3	0	no	yes	yes	yes	yes	ye

```
import pandas as pd
names = ['Bob','Jessica','Mary','John','Mel']
grades = [76,95,77,78,99]
bsdegrees = [1,1,0,0,1]
msdegrees = [2,1,0,0,0]
phddegrees = [0,1,0,0,0]
Degrees = zip(names,grades,bsdegrees,msdegrees,phddegrees)
columns = ['Names','Grades','BS','MS','PhD']
df = pd.DataFrame(data = Degrees, columns=columns)
df
```

	Names	Grades	BS	MS	PhD
0	Bob	76	1	2	0
1	Jessica	95	1	1	1
2	Mary	77	0	0	0
3	John	78	0	0	0
4	Mel	99	1	0	0

```
%pip install openpyxl xlrd  xlwt xlsxwriter
import pandas as pd
Location = "gradedata.xlsx"
```

```
df = pd.read_excel(Location)

#Changing column Names
df.columns = ['first','last','sex','age','exer','hrs','grd','addr']
df.head()
```

	first	last	sex	age	exer	hrs	grd	addr
0	Marcia	Pugh	female	17	3	10	82.4	7379 Highland Rd. , Dublin, GA 31021
1	Kadeem	Morrison	male	18	4	4	78.2	8 Bayport St. , Honolulu, HI 96815
2	Nash	Powell	male	18	5	9	79.3	Encino, CA 91316, 3 Lilac Street
3	Noelani	Wagner	female	14	2	7	83.2	Riverview, FL 33569, 9998 North Smith Dr.
4	Noelani	Cherry	female	18	4	15	87.4	97 SE. Ocean Street , Bethlehem, PA 18015

```
import pandas as pd
names = ['Bob','Jessica','Mary','John','Mel']
grades = [76,95,77,78,99]
GradeList = zip(names,grades)
df = pd.DataFrame(data = GradeList,columns=['Names','Grades'])

writer = pd.ExcelWriter('dataframe.xlsx', engine='xlsxwriter')
df.to_excel(writer, sheet_name='Sheet1')
writer.save()
```

4. Load Data from sqlite

```
import sqlite3
con = sqlite3.connect("portal_mammals.sqlite")
cur = con.cursor()


for row in cur.execute('SELECT * FROM species;'):
    print(row)

con.close()
```

```
Output exceeds the size limit. Open the full output data in a text editor
('AB', 'Amphispiza', 'bilineata', 'Bird')
('AH', 'Ammospermophilus', 'harrisi', 'Rodent')
('AS', 'Ammodramus', 'savannarum', 'Bird')
('BA', 'Baiomys', 'taylori', 'Rodent')
('CB', 'Campylorhynchus', 'brunneicapillus', 'Bird')
('CM', 'Calamospiza', 'melanocorys', 'Bird')
('CQ', 'Callipepla', 'squamata', 'Bird')
('CS', 'Crotalus', 'scutalatus', 'Reptile')
('CT', 'Cnemidophorus', 'tigris', 'Reptile')
('CU', 'Cnemidophorus', 'uniparens', 'Reptile')
('CV', 'Crotalus', 'viridis', 'Reptile')
('DM', 'Dipodomys', 'merriami', 'Rodent')
('DO', 'Dipodomys', 'ordii', 'Rodent')
('DS', 'Dipodomys', 'spectabilis', 'Rodent')
('DX', 'Dipodomys', 'sp.', 'Rodent')
('EO', 'Eumeces', 'obsoletus', 'Reptile')
('GS', 'Gambelia', 'silus', 'Reptile')
('NL', 'Neotoma', 'albigula', 'Rodent')
('NX', 'Neotoma', 'sp.', 'Rodent')
('OL', 'Onychomys', 'leucogaster', 'Rodent')
('OT', 'Onychomys', 'torridus', 'Rodent')
('OX', 'Onychomys', 'sp.', 'Rodent')
('PB', 'Chaetodipus', 'baileyi', 'Rodent')
('PC', 'Pipilo', 'chlorurus', 'Bird')
('PE', 'Peromyscus', 'eremicus', 'Rodent')
```

```python
import sqlite3

# Create a SQL connection to our SQLite database
con = sqlite3.connect("portal_mammals.sqlite")

cur = con.cursor()

# Return all results of query
cur.execute('SELECT plot_id FROM plots WHERE plot_type="Control"')
print(cur.fetchall())

# Return first result of query
cur.execute('SELECT species FROM species WHERE taxa="Bird"')
print(cur.fetchone())

# Be sure to close the connection
con.close()
```

```
[(2,), (4,), (8,), (11,), (12,), (14,), (17,), (22,)]
('bilineata',)
```

```
import pandas as pd
import sqlite3

# Read sqlite query results into a pandas DataFrame
con = sqlite3.connect("portal_mammals.sqlite")
df = pd.read_sql_query("SELECT * from surveys", con)

# Verify that result of SQL query is stored in the dataframe
print(df.head())

con.close()
```

```
   record_id  month  day  year  plot_id species_id sex  hindfoot_length  \
0          1      7   16  1977        2         NL   M             32.0
1          2      7   16  1977        3         NL   M             33.0
2          3      7   16  1977        2         DM   F             37.0
3          4      7   16  1977        7         DM   M             36.0
4          5      7   16  1977        3         DM   M             35.0

   weight
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
```

5. Saving data to SQL

```
from pandas import DataFrame
Cars={'Brand':['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
      'Price':[22000,25000,27000,35000]
      }
df=DataFrame(Cars,columns=['Brand','Price'])
print(df)
```

	Brand	Price
0	Honda Civic	22000
1	Toyota Corolla	25000
2	Ford Focus	27000
3	Audi A4	35000

```
import sqlite3
conn=sqlite3.connect('TestDB1.db')
c=conn.cursor()

c.execute('CREATE TABLE CARS2(Brand text, Price number)')
conn.commit()

df.to_sql('CARS2',conn,if_exists='replace',index=False)
df
```

	Brand	Price
0	Honda Civic	22000
1	Toyota Corolla	25000
2	Ford Focus	27000
3	Audi A4	35000

```
c.execute('''
SELECT Brand,max(Price) from CARS2
''')
```

```
<sqlite3.Cursor at 0x1f4e9773a40>
```

```
%pip install sqlalchemy
import pandas as pd
import os
```

```
import sqlite3 as lite
from sqlalchemy import create_engine

studentId=["rj101","rj150","rj134","rj70"]
SName=["Saurabh","Giftson","Vikas","Radha"]
LName=["Chavan","Paul","Bisoi","Rai"]
Department=["Bms","Bcom","BscCS","BScIT"]
Email=["100rabh@gmail.com","gift01@gmail.com","vik21@gmail.com","rad01@gmail.c
om"]

studata = zip(studentId,SName,LName,Department,Email)

df = pd.DataFrame(data =studata,
columns=['StudentId','SName','LName','Department','Email'])
df
```

	StudentId	SName	LName	Department	Email
0	rj101	Saurabh	Chavan	Bms	100rabh@gmail.com
1	rj150	Giftson	Paul	Bcom	gift01@gmail.com
2	rj134	Vikas	Bisoi	BscCS	vik21@gmail.com
3	rj70	Radha	Rai	BScIT	rad01@gmail.com

```
df1=df.to_csv('studentdata.csv',index=False,header=True)
df1

df2=df.to_excel('studentdata2.xlsx',index=False,header=True)

df2

db_filename = r'studentdata.db'
con = lite.connect(db_filename)
df.to_sql('student',
con,
schema=None,
if_exists='replace',
index=True,
index_label=None,
chunksize=None,
dtype=None)
con.close()
```

```
db_file = r'studentdata.db'
engine = create_engine(r"sqlite:///{}" .format(db_file))
sql = 'SELECT * from student '


studf = pd.read_sql(sql, engine)
studf
```

	index	StudentId	SName	LName	Department	Email
0	0	rj101	Saurabh	Chavan	Bms	100rabh@gmail.com
1	1	rj150	Giftson	Paul	Bcom	gift01@gmail.com
2	2	rj134	Vikas	Bisoi	BscCS	vik21@gmail.com
3	3	rj70	Radha	Rai	BScIT	rad01@gmail.com

Data Preprocessing

```
import numpy as np
import pandas as pd

state=pd.read_csv("US_violent_crime.csv")
state.head()
```

	State	Murder	Assault	UrbanPop	Rape
0	Alabama	13.2	236	58	21.2
1	Alaska	10.0	263	48	44.5
2	Arizona	8.1	294	80	31.0
3	Arkansas	8.8	190	50	19.5
4	California	9.0	276	91	40.6

```
def some_func(x):
    return x*2
state.apply(some_func) #update each entry of dataframe without any loop
state.apply(lambda n: n*2) #lambda also works the same
```

	State	Murder	Assault	UrbanPop	Rape
0	AlabamaAlabama	26.4	472	116	42.4
1	AlaskaAlaska	20.0	526	96	89.0
2	ArizonaArizona	16.2	588	160	62.0
3	ArkansasArkansas	17.6	380	100	39.0
4	CaliforniaCalifornia	18.0	552	182	81.2
5	ColoradoColorado	15.8	408	156	77.4
6	ConnecticutConnecticut	6.6	220	154	22.2
7	DelawareDelaware	11.8	476	144	31.6
8	FloridaFlorida	30.8	670	160	63.8
9	GeorgiaGeorgia	34.8	422	120	51.6
10	HawaiiHawaii	10.6	92	166	40.4
11	IdahoIdaho	5.2	240	108	28.4
12	IllinoisIllinois	20.8	498	166	48.0
13	IndianaIndiana	14.4	226	130	42.0
14	IowaIowa	4.4	112	114	22.6
15	KansasKansas	12.0	230	132	36.0
16	KentuckyKentucky	19.4	218	104	32.6
17	LouisianaLouisiana	30.8	498	132	44.4
18	MaineMaine	4.2	166	102	15.6

```
state.transform(func = lambda x : x * 10)
```

	State	Murder	Assault	UrbanPop	Rape
0	AlabamaAlabamaAlabamaAlabamaAlabamaAlabamaAlab...	132.0	2360	580	212.0
1	AlaskaAlaskaAlaskaAlaskaAlaskaAlaskaAlaskaAlas...	100.0	2630	480	445.0
2	ArizonaArizonaArizonaArizonaArizonaArizonaAriz...	81.0	2940	800	310.0
3	ArkansasArkansasArkansasArkansasArkansasArkans...	88.0	1900	500	195.0
4	CaliforniaCaliforniaCaliforniaCaliforniaCalifo...	90.0	2760	910	406.0
5	ColoradoColoradoColoradoColoradoColoradoColora...	79.0	2040	780	387.0
6	ConnecticutConnecticutConnecticutConnecticutCo...	33.0	1100	770	111.0
7	DelawareDelawareDelawareDelawareDelawareDelawa...	59.0	2380	720	158.0
8	FloridaFloridaFloridaFloridaFloridaFloridaFlor...	154.0	3350	800	319.0
9	GeorgiaGeorgiaGeorgiaGeorgiaGeorgiaGeorgiaGeor...	174.0	2110	600	258.0
10	HawaiiHawaiiHawaiiHawaiiHawaiiHawaiiHawaiiHawa...	53.0	460	830	202.0
11	IdahoIdahoIdahoIdahoIdahoIdahoIdahoIdahoIdahoI...	26.0	1200	540	142.0
12	IllinoisIllinoisIllinoisIllinoisIllinoisIllino...	104.0	2490	830	240.0
13	IndianaIndianaIndianaIndianaIndianaIndianaIndi...	72.0	1130	650	210.0
14	IowaIowaIowaIowaIowaIowaIowaIowaIowaIowa	22.0	560	570	113.0
15	KansasKansasKansasKansasKansasKansasKansasKans...	60.0	1150	660	180.0
16	KentuckyKentuckyKentuckyKentuckyKentuckyKentuc...	97.0	1090	520	163.0
17	LouisianaLouisianaLouisianaLouisianaLouisianaL...	154.0	2490	660	222.0
18	MaineMaineMaineMaineMaineMaineMaineMaineMaineM...	21.0	830	510	78.0
19	MarylandMarylandMarylandMarylandMarylandMaryla...	113.0	3000	670	278.0

```
#usinggroupby
mean_purchase
=state.groupby('State')["Murder"].mean().rename("User_mean").reset_index()
print(mean_purchase)
```

```
             State  User_mean
0          Alabama       13.2
1           Alaska       10.0
2          Arizona        8.1
3         Arkansas        8.8
4       California        9.0
5         Colorado        7.9
6      Connecticut        3.3
7         Delaware        5.9
8          Florida       15.4
9          Georgia       17.4
10          Hawaii        5.3
11           Idaho        2.6
12        Illinois       10.4
13         Indiana        7.2
14            Iowa        2.2
15          Kansas        6.0
16        Kentucky        9.7
17       Louisiana       15.4
18           Maine        2.1
19        Maryland       11.3
20   Massachusetts        4.4
21        Michigan       12.1
22       Minnesota        2.7
23     Mississippi       16.1
...
```

```
mer=state.merge(mean_purchase)
mer
```

	State	Murder	Assault	UrbanPop	Rape	User_mean
0	Alabama	13.2	236	58	21.2	13.2
1	Alaska	10.0	263	48	44.5	10.0
2	Arizona	8.1	294	80	31.0	8.1
3	Arkansas	8.8	190	50	19.5	8.8
4	California	9.0	276	91	40.6	9.0
5	Colorado	7.9	204	78	38.7	7.9
6	Connecticut	3.3	110	77	11.1	3.3
7	Delaware	5.9	238	72	15.8	5.9
8	Florida	15.4	335	80	31.9	15.4
9	Georgia	17.4	211	60	25.8	17.4
10	Hawaii	5.3	46	83	20.2	5.3
11	Idaho	2.6	120	54	14.2	2.6
12	Illinois	10.4	249	83	24.0	10.4
13	Indiana	7.2	113	65	21.0	7.2
14	Iowa	2.2	56	57	11.3	2.2
15	Kansas	6.0	115	66	18.0	6.0
16	Kentucky	9.7	109	52	16.3	9.7
17	Louisiana	15.4	249	66	22.2	15.4
18	Maine	2.1	83	51	7.8	2.1

```python
#checking for missing values
print(state.isnull().sum())
```

```
State       0
Murder      0
Assault     0
UrbanPop    0
Rape        0
dtype: int64
```

```
import pandas as pd
import numpy as np
cols=['col0', 'col1', 'col2', 'col3', 'col4']
rows=['row0', 'row1', 'row2', 'row3', 'row4']
data=np.random.randint(0, 100, size=(5,5))
df=pd.DataFrame(data, columns=cols, index=rows)
df.head()
```

	col0	col1	col2	col3	col4
row0	31	35	75	17	4
row1	78	73	48	1	71
row2	75	60	72	48	72
row3	93	41	9	11	74
row4	19	11	82	10	86

```
df.iloc[4,2]
```

```
82
```

Dealing with 0 and NAN values NaN stands for Not A Number and is one of the common ways to represent the missing value in the data.

```
df.iloc[3, 3]=0
df.iloc[1, 2]=np.nan
df.iloc[4, 0]=np.nan
df['col5']=0
df['col6']=np.nan
df.head()
```

	col0	col1	col2	col3	col4	col5	col6
row0	31.0	35	75.0	17	4	0	NaN
row1	78.0	73	NaN	1	71	0	NaN
row2	75.0	60	72.0	48	72	0	NaN
row3	93.0	41	9.0	0	74	0	NaN
row4	NaN	11	82.0	10	86	0	NaN

```
df.loc[:,df.all()]
```

	col0	col1	col2	col4	col6
row0	31.0	35	75.0	4	NaN
row1	78.0	73	NaN	71	NaN
row2	75.0	60	72.0	72	NaN
row3	93.0	41	9.0	74	NaN
row4	NaN	11	82.0	86	NaN

```
df.loc[:,df.any()]
```

	col0	col1	col2	col3	col4
row0	31.0	35	75.0	17	4
row1	78.0	73	NaN	1	71
row2	75.0	60	72.0	48	72
row3	93.0	41	9.0	0	74
row4	NaN	11	82.0	10	86

```
df.loc[:,df.isnull().any()]
```

	col0	col2	col6
row0	31.0	75.0	NaN
row1	78.0	NaN	NaN
row2	75.0	72.0	NaN
row3	93.0	9.0	NaN
row4	NaN	82.0	NaN

```
df.loc[:,df.notnull().all()]
```

	col1	col3	col4	col5
row0	35	17	4	0
row1	73	1	71	0
row2	60	48	72	0
row3	41	0	74	0
row4	11	10	86	0

```
df.dropna(how="all",axis=0)
```

	col0	col1	col2	col3	col4	col5	col6
row0	31.0	35	75.0	17	4	0	NaN
row1	78.0	73	NaN	1	71	0	NaN
row2	75.0	60	72.0	48	72	0	NaN
row3	93.0	41	9.0	0	74	0	NaN
row4	NaN	11	82.0	10	86	0	NaN

```
df.fillna(df.sum())
```

	col0	col1	col2	col3	col4	col5	col6
row0	31.0	35	75.0	17	4	0	0.0
row1	78.0	73	238.0	1	71	0	0.0
row2	75.0	60	72.0	48	72	0	0.0
row3	93.0	41	9.0	0	74	0	0.0
row4	277.0	11	82.0	10	86	0	0.0

```
#Demonstrate transfomr function using pandas in python
import pandas as pd
import numpy as np
import random
data = pd.DataFrame({
    'C' : [random.choice(('a','b','c')) for i in range(1000000)],
    'A' : [random.randint(1,10) for i in range(1000000)],
    'B' : [random.randint(1,10) for i in range(1000000)]

})
data
```

	C	A	B
0	a	6	9
1	b	7	8
2	a	10	8
3	b	7	9
4	b	2	5
...	...	...	...
999995	c	6	6
999996	b	8	5
999997	b	9	10
999998	a	9	4
999999	a	4	4

1000000 rows × 3 columns

```
v=data.groupby('C')["A"].mean
v

mean=data.groupby('C')["A"].mean().rename("D").reset_index()
mean
```

	C	D
0	a	5.492720
1	b	5.508571
2	c	5.501291

```
df_1=data.merge(mean)
df_1
```

	C	A	B	D
0	a	6	9	5.492720
1	a	10	8	5.492720
2	a	6	6	5.492720
3	a	9	1	5.492720
4	a	2	2	5.492720
...	...	...	...	...
999995	c	10	5	5.501291
999996	c	6	5	5.501291
999997	c	6	4	5.501291
999998	c	6	9	5.501291
999999	c	6	6	5.501291

1000000 rows × 4 columns

Name: Harsh Chheda
Roll Number: 22-15405/31031521005 MSC COMPUTER SCIENCE Subject: Machine Learning

Practical 3

Q1) Performing the basic feature engineerign steps.

→

What is feature engineering?

- All machine learning algorithms use some input data to generate outputs. Input data contains many features which may not be in proper form to be given to the model directly. It needs some kind of processing and here feature engineering helps. Feature engineering fulfils mainly two goals:
- It prepares the input dataset in the form which is required for a specific model or machine learning algorithm.
- Feature engineering helps in improving the performance of machine learning models magically.
- According to some surveys, data scientists spend their time on data preparation

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

```
The main feature engineering techniques that will be discussed are:

1. Missing data imputation

2. Categorical encoding

3. Variable transformation

4. Outlier engineering

5. Date and time engineering
```

Missing Data Imputation for Feature Engineering

- In your input data, there may be some features or columns which will have missing data, missing values. It occurs if there is no data stored for a certain observation in a variable. Missing data is very common and it is an unavoidable problem especially in real-world data sets. If this data containing a missing value is used then you can see the significance in the results. So, imputation is the act of replacing missing data with statistical estimates of the missing values. It helps you to complete your training data which can then be provided to any model or an algorithm for prediction.
- There are multiple techniques for missing data imputation. These are as follows:-

- Complete case analysis
- Mean / Median / Mode imputation
- Missing Value Indicator

Complete Case Analysis for Missing Data Imputation

- Complete case analysis is basically analyzing those observations in the dataset that contains values in all the variables. Or you can say, remove all the observations that contain missing values. But this method can only be used when there are only a few observations which has a missing dataset otherwise it will reduce the dataset size and then it will be of not much use.
- So, it can be used when missing data is small but in real-life datasets, the amount of missing data is always big. So, practically, complete case analysis is never an option to use, although you can use it if the missing data size is small.
- Let's see the use of this on the titanic dataset.

```
titanic = pd.read_csv('train.csv')
# make a copy of titanic dataset
data1 = titanic.copy()
data1.isnull().mean()
```

```
PassengerId    0.000000
Survived       0.000000
Pclass         0.000000
Name           0.000000
Sex            0.000000
Age            0.198653
SibSp          0.000000
Parch          0.000000
Ticket         0.000000
Fare           0.000000
Cabin          0.771044
Embarked       0.002245
dtype: float64
```

```
If we remove all the missing observations, we would end up with a very small
dataset, given that the Cabin is missing for 77% of the observations
```

```
# check how many observations we would drop
print('total passengers with values in all variables: ',
data1.dropna().shape[0])
print('total passengers in the Titanic: ', data1.shape[0])
print('percentage of data without missing values: ', data1.dropna().shape[0]/
np.float(data1.shape[0]))
```

```
total passengers with values in all variables:  183
total passengers in the Titanic:  891
percentage of data without missing values:  0.2053872053872054

C:\Users\dell\AppData\Local\Temp\ipykernel_11760\397274795.py:4: DeprecationWarning: `np.float` is a deprecated alias for the builtin `float`. To silence this warning, use `float` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  print('percentage of data without missing values: ', data1.dropna().shape[0]/ np.float(data1.shape[0]))
```

```
So, we have complete information for only 20% of our observations in the
Titanic dataset. Thus, Complete Case Analysis method would not be an option
for this dataset.
```

Mean/ Median/ Mode for Missing Data Imputation

```
Missing values can also be replaced with the mean, median, or mode of the
variable(feature). It is widely used in data competitions and in almost every
situation. It is suitable to use this technique where data is missing at
random places and in small proportions.
```

```
# impute missing values in age in train and test set
median =data1.Age.median()
data1['Age'].fillna(median, inplace=True)
data1['Age'].isnull().sum()
```

```
0
```

```
0 represents that now the Age feature has no null values.

One important point to consider while doing imputation is that it should be
done over the training set first and then to the test set. All missing values
in the train set and test set should be filled with the value which is
extracted from the train set only. This helps in avoiding overfitting.
```

Missing Value Indicator for Missing Value Indication

```
This technique involves adding a binary variable to indicate whether the value
is missing for a certain observation. This variable takes the value 1 if the
observation is missing, or 0 otherwise. But we still need to replace the
missing values in the original variable, which we tend to do with mean or
median imputation. By using these 2 techniques together, if the missing value
has predictive power, it will be captured by the missing indicator, and if it
doesn't it will be masked by the mean / median imputation.
```

```
data1['Age_NA'] = np.where(data1['Age'].isnull(), 1, 0)

data1.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_NA
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	0

```
data1.Age.mean(), data1.Age.median()
```

```
(29.36158249158249, 28.0)
```

```
Now, since mean and median are the same, let’s replace them with the median.
```

```
data1['Age'].fillna(data1.Age.median(), inplace=True)

data1.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_NA
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	0
5	6	0	3	Moran, Mr. James	male	28.0	0	0	330877	8.4583	NaN	Q	0
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S	0
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S	0
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S	0
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C	0

```
So, the Age_NA variable was created to capture the missingness.
```

Categorical encoding in Feature Engineering

- Categorical data is defined as that data that takes only a number of values. Let’s understand this with an example. Parameter Gender in a dataset will have categorical values like Male, Female. If a survey is done to know which car people own then the result will be categorical (because the answers would be in categories like Honda, Toyota, Hyundai, Maruti, None, etc.). So, the point to notice here is that data falls in a fixed set of categories.
- If you directly give this dataset with categorical variables to a model, you will get an error. Hence, they are required to be encoded. There are multiple techniques to do so:

1. One-Hot encoding (OHE)
2. Ordinal encoding
3. Count and Frequency encoding
4. Target encoding / Mean encoding

One-Hot Encoding

- It is a commonly used technique for encoding categorical variables. It basically creates binary variables for each category present in the categorical variable. These binary variables will have 0 if it is absent in the category or 1 if it is present. Each new variable is called a dummy variable or binary variable.

- Example: If the categorical variable is Gender with labels female and male, two boolean variables can be generated called male and female. Male will take 1 if the person is male or 0 otherwise. Similarly for a female variable. See this code below for the titanic dataset.

```
pd.get_dummies(data1['Sex']).head()
```

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

```
pd.concat([data1['Sex'], pd.get_dummies(data1['Sex'])], axis=1).head()
```

	Sex	female	male
0	male	0	1
1	female	1	0
2	female	1	0
3	female	1	0
4	male	0	1

```
But you can see that we only need 1 dummy variable to represent Sex
categorical variable. So, you can take it as a general formula where if there
are n categories, you only need an n-1 dummy variable. So you can easily drop
anyone dummy variable. To get n-1 dummy variables simply use this:
```

```
pd.get_dummies(data1['Sex'], drop_first=True).head()
```

	male
0	1
1	0
2	0
3	0
4	1

Count and Frequency Encoding

- In this encoding technique, categories are replaced by the count of the observations that show that category in the dataset. Replacement can also be done with the frequency of the percentage of observations in the dataset. Suppose, if 30 of 100 genders are male we can replace male with 30 or by 0.3.
- This approach is popularly used in data science competitions, so basically it represents how many times each label appears in the dataset.

Target / Mean Encoding

- In target encoding, also called mean encoding, we replace each category of a variable with the mean value of the target for the observations that show a certain category.
- For example, there is a categorical variable “city”, and we want to predict if the customer will buy a TV provided we send a letter. If 30 percent of the people in the city “London” buy the TV, we would replace London with 0.3.
- So it helps in capturing some information regarding the target at the time of encoding the category and it also does not expands the feature space.
- Hence, it also can be considered as an option for encoding. But it may cause over-fitting to the model, so be careful. Look at this code for implementation:

```
import pandas as pd
# creating dataset
data={'CarName':['C1','C2','C3','C1','C4','C3','C2','C1','C2','C4','C1'],
      'Target':[1,0,1,1,1,0,0,1,1,1,0]}
df = pd.DataFrame(data)
print(df)
```

```
   CarName  Target
0       C1       1
1       C2       0
2       C3       1
3       C1       1
4       C4       1
5       C3       0
6       C2       0
7       C1       1
8       C2       1
9       C4       1
10      C1       0
```

```
df.groupby(['CarName'])['Target'].count()
```

```
CarName
C1    4
C2    3
C3    2
C4    2
Name: Target, dtype: int64
```

```
df.groupby(['CarName'])['Target'].mean()
```

```
CarName
C1    0.750000
C2    0.333333
C3    0.500000
C4    1.000000
Name: Target, dtype: float64
```

```
Mean_encoded = df.groupby(['CarName'])['Target'].mean().to_dict()
```

```
df['CarName'] = df['CarName'].map(Mean_encoded)
print(df)
```

```
     CarName  Target
0   0.750000       1
1   0.333333       0
2   0.500000       1
3   0.750000       1
4   1.000000       1
5   0.500000       0
6   0.333333       0
7   0.750000       1
8   0.333333       1
9   1.000000       1
10  0.750000       0
```

Variable Transformation

- Machine learning algorithms like linear and logistic regression assume that the variables are normally distributed. If a variable is not normally distributed, sometimes it is possible to find a mathematical transformation so that the transformed variable is Gaussian. Gaussian distributed variables many times boost the machine learning algorithm performance.
- Commonly used mathematical transformations are:

1. Logarithm transformation – log(x)
2. Square root transformation – sqrt(x)
3. Reciprocal transformation – 1 / x
4. Exponential transformation – exp(x)

- Let's check these out on the titanic dataset.

```
cols_reqiuired = ['Age', 'Fare', 'Survived']
data1[cols_reqiuired].head()
```

	Age	Fare	Survived
0	22.0	7.2500	0
1	38.0	71.2833	1
2	26.0	7.9250	1
3	35.0	53.1000	1
4	35.0	8.0500	0

```
First, we need to fill in missing data. We will start with filling missing data with a random sample.
```

```
def impute(data1, variable):
    df = data1.copy()
    df[variable+'_random'] = df[variable]
    # extract the random sample to fill the na
    random_sample = df[variable].dropna().sample(df[variable].isnull().sum(),
random_state=0)
    random_sample.index = df[df[variable].isnull()].index
    df.loc[df[variable].isnull(), variable+'_random'] = random_sample
    return df[variable+'_random']
# fill na
data1['Age'] = impute(data1, 'Age')
```

Now, to visualize the distribution of the age variable we will plot histogram and Q-Q-plot.

Date and Time Feature Engineering

- Date variables are considered a special type of categorical variable and if they are processed well they can enrich the dataset to a great extent. From the date we can extract various important information like: Month, Semester, Quarter, Day, Day of the week, Is it a weekend or not, hours, minutes, and many more. Let's use some dataset and do some coding around it.
- For this, we will use the Lending club dataset.
- We will use only two columns from the dataset: issue_d and last_pymnt_d.

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

```
use_cols = ['issue_d', 'last_pymnt_d']
data = pd.read_csv('loan.csv', usecols=use_cols, nrows=10000)
data.head()
```

	issue_d	last_pymnt_d
0	Dec-2018	Feb-2019
1	Dec-2018	Feb-2019
2	Dec-2018	Feb-2019
3	Dec-2018	Feb-2019
4	Dec-2018	Feb-2019

```
Now, parse dates into DateTime format as they are coded in strings currently.
```

```
data['issue_dt'] = pd.to_datetime(data.issue_d)
data['last_pymnt_dt'] = pd.to_datetime(data.last_pymnt_d)
data[['issue_d','issue_dt','last_pymnt_d', 'last_pymnt_dt']].head()
```

	issue_d	issue_dt	last_pymnt_d	last_pymnt_dt
0	Dec-2018	2018-12-01	Feb-2019	2019-02-01
1	Dec-2018	2018-12-01	Feb-2019	2019-02-01
2	Dec-2018	2018-12-01	Feb-2019	2019-02-01
3	Dec-2018	2018-12-01	Feb-2019	2019-02-01
4	Dec-2018	2018-12-01	Feb-2019	2019-02-01

```
data['issue_dt_month'] = data['issue_dt'].dt.month
data[['issue_dt', 'issue_dt_month']].head()
```

	issue_dt	issue_dt_month
0	2018-12-01	12
1	2018-12-01	12
2	2018-12-01	12
3	2018-12-01	12
4	2018-12-01	12

```
data['issue_dt_quarter'] = data['issue_dt'].dt.quarter
data[['issue_dt', 'issue_dt_quarter']].head()
```

	issue_dt	issue_dt_quarter
0	2018-12-01	4
1	2018-12-01	4
2	2018-12-01	4
3	2018-12-01	4
4	2018-12-01	4

```
data['issue_dt_dayofweek'] = data['issue_dt'].dt.dayofweek
data[['issue_dt', 'issue_dt_dayofweek']].head()
```

	issue_dt	issue_dt_dayofweek
0	2018-12-01	5
1	2018-12-01	5
2	2018-12-01	5
3	2018-12-01	5
4	2018-12-01	5

Outlier engineering

```
Outliers are defined as those values that are unusually high or low with
respect to the rest of the observations of the variable. Some of the
techniques to handle outliers are:

1. Outlier removal

2. Treating outliers as missing values

3. Outlier capping

How to identify outliers?

For that, the basic form of detection is an extreme value analysis of data. If
the distribution of the variable is Gaussian then outliers will lie outside
the mean plus or minus three times the standard deviation of the variable. But
if the variable is not normally distributed, then quantiles can be used.
Calculate the quantiles and then inter quartile range:

Inter quantile is 75th quantile-25quantile.

upper boundary: 75th quantile + (IQR * 1.5)

lower boundary: 25th quantile – (IQR * 1.5)

So, the outlier will sit outside these boundaries. Outlier removal

In this technique, simply remove outlier observations from the dataset. In
datasets if outliers are not abundant, then dropping the outliers will not
affect the data much. But if multiple variables have outliers then we may end
up removing a big chunk of data from our dataset. So, this point has to be
kept in mind whenever dropping the outliers. Treating outliers as missing
values

You can also treat outliers as missing values. But then these missing values
also have to be filled. So to fill missing values you can use any of the
methods as discussed above in this article. Outlier capping

This procedure involves capping the maximum and minimum values at a predefined
value. This value can be derived from the variable distribution. If a variable
is normally distributed we can cap the maximum and minimum values at the mean
plus or minus three times the standard deviation. But if the variable is
skewed, we can use the inter-quantile range proximity rule or cap at the
bottom percentiles.
```

```
import pandas as pd
```

```
df = pd.read_csv('height.csv')
df.head()
```

	name	height
0	mohan	5.9
1	maria	5.2
2	sakib	5.1
3	tao	5.5
4	virat	4.9

```
max_thresold = df['height'].quantile(0.95)
max_thresold
```

```
9.689999999999998
```

```
df[df['height']>max_thresold]
```

	name	height
9	imran	14.5

```
min_thresold = df['height'].quantile(0.05)
min_thresold
```

```
3.6050000000000004
```

```
df[df['height']<min_thresold]
```

	name	height
12	yoseph	1.2

Remove outliers

```
df[(df['height']<max_thresold) & (df['height']>min_thresold)]
```

	name	height
0	mohan	5.9
1	maria	5.2
2	sakib	5.1
3	tao	5.5
4	virat	4.9
5	khusbu	5.4
6	dmitry	6.2
7	selena	6.5
8	john	7.1
10	jose	6.1
11	deepika	5.6
13	binod	5.5

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

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

```
df.shape
```

```
(13320, 9)
```

```
df.describe()
```

	bath	balcony	price
count	13247.000000	12711.000000	13320.000000
mean	2.692610	1.584376	112.565627
std	1.341458	0.817263	148.971674
min	1.000000	0.000000	8.000000
25%	2.000000	1.000000	50.000000
50%	2.000000	2.000000	72.000000
75%	3.000000	2.000000	120.000000
max	40.000000	3.000000	3600.000000

```
min_thresold, max_thresold = df.price.quantile([0.001, 0.999])
min_thresold, max_thresold
```

```
(11.159500000000001, 2000.0)
```

```
df[df.price < min_thresold]
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
171	Super built-up Area	Ready To Move	Attibele	1 BHK	Jae 1hu	450	1.0	1.0	11.00
942	Built-up Area	Ready To Move	Attibele	1 BHK	Jae 2hu	400	1.0	1.0	11.00
1471	Built-up Area	18-Mar	Kengeri	1 BHK	NaN	340	1.0	1.0	10.00
2437	Built-up Area	Ready To Move	Attibele	1 BHK	Jae 1hu	395	1.0	1.0	10.25
4113	Super built-up Area	18-Jan	BTM Layout	3 BHK	NaN	167Sq. Meter	3.0	2.0	10.00
5410	Super built-up Area	Ready To Move	Attibele	1 BHK	Jae 1hu	400	1.0	1.0	10.00
7482	Super built-up Area	Ready To Move	Alur	1 BHK	NaN	470	2.0	1.0	10.00
8594	Built-up Area	Ready To Move	Chandapura	1 BHK	NaN	450	1.0	1.0	9.00
8653	Plot Area	Ready To Move	Doddaballapur	2 Bedroom	NaN	640	1.0	0.0	10.50
10526	Super built-up Area	Ready To Move	Yelahanka New Town	1 BHK	KHatsFl	284	1.0	1.0	8.00
11091	Built-up Area	Ready To Move	Attibele	1 BHK	NaN	410	1.0	1.0	10.00
11569	Plot Area	Immediate Possession	Hosur Road	NaN	AVeldun	1350	NaN	NaN	8.44
11945	Super built-up Area	Ready To Move	Attibele	1 BHK	Jae 2hu	400	1.0	1.0	10.25
12579	Super built-up Area	Ready To Move	Chandapura	1 BHK	NaN	410	1.0	1.0	10.00

```
df[df.price > max_thresold]
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
408	Super built-up Area	19-Jan	Rajaji Nagar	7 BHK	NaN	12000	6.0	3.0	2200.0
605	Super built-up Area	19-Jan	Malleshwaram	7 BHK	NaN	12000	7.0	3.0	2200.0
2623	Plot Area	18-Jul	Dodsworth Layout	4 Bedroom	NaN	30000	4.0	NaN	2100.0
3180	Super built-up Area	Ready To Move	Shanthala Nagar	5 BHK	Kierser	8321	5.0	3.0	2700.0
4162	Built-up Area	Ready To Move	Yemlur	4 Bedroom	Epllan	7000	5.0	NaN	2050.0
6421	Plot Area	18-Sep	Bommenahalli	4 Bedroom	Prood G	2940	3.0	2.0	2250.0
10304	Plot Area	Ready To Move	5th Block Jayanagar	4 Bedroom	NaN	10624	4.0	2.0	2340.0
11080	Super built-up Area	18-Jan	Ashok Nagar	4 BHK	NaN	8321	5.0	2.0	2912.0
11763	Plot Area	Ready To Move	Sadashiva Nagar	5 Bedroom	NaN	9600	7.0	2.0	2736.0
12443	Plot Area	Ready To Move	Dollars Colony	4 Bedroom	NaN	4350	8.0	NaN	2600.0
13067	Plot Area	Ready To Move	Defence Colony	10 Bedroom	NaN	7150	13.0	NaN	3600.0
13197	Plot Area	Ready To Move	Ramakrishnappa Layout	4 Bedroom	NaN	9200	4.0	NaN	2600.0
13200	Plot Area	Ready To Move	Defence Colony	6 Bedroom	NaN	8000	6.0	3.0	2800.0

```
df2 = df[(df.price<max_thresold) & (df.price>min_thresold)]
df2.shape
```

```
(13291, 9)
```

```
df2.describe()
```

	bath	balcony	price
count	13219.000000	12688.000000	13291.000000
mean	2.690673	1.584253	110.010361
std	1.335757	0.817169	125.434347
min	1.000000	0.000000	11.500000
25%	2.000000	1.000000	50.000000
50%	2.000000	2.000000	72.000000
75%	3.000000	2.000000	120.000000
max	40.000000	3.000000	1950.000000

Practical 4

Q1) Performing the Probability Operations.

→

```
# probability of getting 3 when a die is rolled
ns = 6 #n(S) = {1,2,3,4,5,6}
na = 1 #n(A) = {3}
pa = na/ns # P(A)
print("probability of getting 3 is:",pa)
```

```
probability of getting 3 is: 0.16666666666666666
```

```
# probability of atleast getting one head when a coin is tossed thrice
ns = 8 #n(S) = {HHH, HHT, HTH, THH, TTH, THT, HTT, TTT}
na = 7 #n(A) = {HHH, HHT, HTH, THH, TTH, THT, HTT}
pa = na/ns # P(A)
print("probability of getting atleast one head is:",pa)
```

```
probability of getting atleast one head is: 0.875
```

```
# A glass jar contains 5 red, 3 blue and 2 green jelly beans. If a jelly bean
is chosen at random from the jar,
#  mwhat is the probability that it is not blue?
ns = 10 #n(S) = {5red,3blue,2green}
na = 7 #n(A) = {5red, 2green}
pa = na/ns # P(A)
print("probability of getting not blue jellybean is:",pa)
```

```
probability of getting not blue jellybean is: 0.7
```

Independent and Dependent Events

```
# If the probability that person A will be alive in 20 years
#is 0.7 and the probability that person B will be alive in
# 20 years is 0.5, what is the probability that they will
#both be alive in 20 years?

#These are independent events, so
P = 0.7*0.5
print("probability that they will be alive after 20 years is:",P)
```

```
probability that they will be alive after 20 years is: 0.35
```

```
def event_probability(n,s):
  return n/s
```

```
#A fair die is tossed twice. Find the probability of getting a 4 or 5 on the first toss and a 1,2, or 3 in the second toss.
pa = event_probability(2,6) # probability of getting a 4 or 5 on the first toss
pb = event_probability(3,6) # probability of getting 1,2,3 in second toss
P = pa*pb
print("probability of getting a 4 or 5 on the first toss and a 1,2, or 3 in the second toss is:",P)
```

```
probability of getting a 4 or 5 on the first toss and a 1,2, or 3 in the second toss is: 0.16666666666666666
```

```
# A bag contains 5 white marbles, 3 black marbles and 2 green marbles. In each draw, a marble is drawn from the bag
# and not replaced. In three draws, find the probability of obtaining white, black and green in that order.
pw = event_probability(5,10)
```

```
pb = event_probability(3,9)
pg = event_probability(2,8)
print("the probability of obtaining white, black and green in that order is
",(pw*pb*pg))
```

```
the probability of obtaining white, black and green in that order is  0.041666666666666664
```

```
# Sample Space
cards = 52

# Calculate the probability of drawing a heart or a club
hearts = 13
clubs = 13
heart_or_club = event_probability(hearts, cards) + event_probability(clubs,
cards)
print(heart_or_club )
```

```
0.5
```

```
# Calculate the probability of drawing an ace, king, or a queen
aces = 4
kings = 4
queens = 4
ace_king_or_queen = event_probability(aces, cards) + event_probability(kings,
cards) + event_probability(queens, cards)

print(heart_or_club)
print(ace_king_or_queen)
```

```
0.5
0.23076923076923078
```

```
# Calculate the probability of drawing a heart or an ace
hearts = 13
aces = 4
ace_of_hearts = 1
heart_or_ace = event_probability(hearts, cards) + event_probability(aces,
cards) - event_probability(ace_of_hearts, cards)
print(round(heart_or_ace, 1))
```

```
0.3
```

```
red_cards = 26
face_cards = 12
red_face_cards = 6
red_or_face_cards = event_probability(red_cards, cards) +
event_probability(face_cards, cards) - event_probability(red_face_cards,
cards)

print(round(heart_or_ace, 1))
print(round(red_or_face_cards, 1))
```

```
0.3
0.6
```

Complementary Events

```
#probabiltiy of not getting 5 when a fair die is rolled
ns = 6 #n(S) = {1,2,3,4,5,6}
na = 1 #n(A) = {5}
pa = na/ns # P(A)
print("probabilty of not getting 5 is:",1-pa)
```

```
probabilty of not getting 5 is: 0.8333333333333334
```

```
import pandas as pd
import numpy as np
df = pd.read_csv('student-mat.csv')
df.head(3)
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	he
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	

3 rows × 33 columns

```
len(df)
```

```
395
```

```
df['grade_A'] = np.where(df['G3']*5 >= 80, 1, 0)
```

```
df['high_absenses'] = np.where(df['absences'] >= 10, 1, 0)
```

```
df['count'] = 1
```

```
df = df[['grade_A','high_absenses','count']]
df.head()
```

	grade_A	high_absenses	count
0	0	0	1
1	0	0	1
2	0	1	1
3	0	0	1
4	0	0	1

```
final= pd.pivot_table(
    df,
    values='count',
    index=['grade_A'],
    columns=['high_absenses'],
    aggfunc=np.size,
    fill_value=0
)
```

```
print(final)
```

```
high_absenses    0   1
grade_A
0              277  78
1               35   5
```

Practical 5

Q1) Bayes Theorem.

→

```
def bayes_theorem(p_a, p_b_given_a, p_b_given_not_a):
        # calculate P(not A)
    not_a = 1 - p_a
    # calculate P(B)
    p_b = p_b_given_a * p_a + p_b_given_not_a * not_a
    # calculate P(A|B)
    p_a_given_b = (p_b_given_a * p_a) / p_b
    return p_a_given_b

# P(A)
p_a = 0.0002
# P(B|A)
p_b_given_a = 0.85
# P(B|not A)
p_b_given_not_a = 0.05
# calculate P(A|B)
result = bayes_theorem(p_a, p_b_given_a, p_b_given_not_a)
# summarize
print('P(A|B) = %.3f%%' % (result * 100))
```

```
P(A|B) = 0.339%
```

Practical 6

Q1) Hypothesis Testing.

→

```
%pip install statsmodels
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import ttest_1samp
from statsmodels.stats.power import tt_ind_solve_power
```

T test

```
ages=[10,20,35,50,28,40,55,18,16,55,30,25,43,18,30,28,14,24,16,17,32,35,26,27,
65,18,43,23,21,20,19,70]
ages_mean=np.mean(ages)
print(ages_mean)
```

```
30.34375
```

```
#Lets take sample
sample_size=10
age_sample=np.random.choice(ages,sample_size)
age_sample
```

```
array([14, 40, 55, 16, 16, 35, 14, 35, 28, 21])
```

```
from scipy.stats import ttest_1samp
```

```
ttest,p_value=ttest_1samp(age_sample,30)
```

```
print(p_value)
```

```
0.5640289663170721
```

```
if p_value < 0.05:
    print("We are rejecting null hypothesis")
else:
    print("We are accepting null hypothesis")
```

```
We are accepting null hypothesis
```

```
df=pd.read_excel('result.xlsx')
df
```

	Roll No	Name	Sub1	Sub2	Sub3	Total	Result
0	101	Akash	45	45	45	135	P
1	102	Manoj	35	45	42	122	P
2	103	Saurabh	29	26	30	85	P
3	104	Ashish	38	35	29	102	P
4	105	Sudhir	41	40	34	115	P
5	106	Ria	46	62	41	149	P
6	107	Prathana	29	48	27	104	P
7	108	Mihika	43	33	33	109	P
8	109	Shaurya	37	30	38	105	P
9	110	Mrunal	33	31	41	105	P

```
df.describe()
```

	Roll No	Sub1	Sub2	Sub3	Total
count	10.00000	10.000000	10.000000	10.000000	10.000000
mean	105.50000	37.600000	39.500000	36.000000	113.100000
std	3.02765	6.168018	10.783217	6.236096	18.241893
min	101.00000	29.000000	26.000000	27.000000	85.000000
25%	103.25000	33.500000	31.500000	30.750000	104.250000
50%	105.50000	37.500000	37.500000	36.000000	107.000000
75%	107.75000	42.500000	45.000000	41.000000	120.250000
max	110.00000	46.000000	62.000000	45.000000	149.000000

One way hypothesis

```
Ho = "mu <= 113"
# alt hyp
Ha = "mu > 113"
# alpha
al = 0.05
# mu -> mean
```

```
mu = 113
# tail type
tt = 1
# data
marks = df['Total'].values
print("Ho:", Ho)
print("Ha:", Ha)
print("al:", al)
print("mu:", mu)
print(marks)
print("")
```

```
Ho: mu <= 113
Ha: mu > 113
al: 0.05
mu: 113
[135 122  85 102 115 149 104 109 105 105]
```

```
ts, pv = ttest_1samp(marks, mu)
print("t-stat",ts)
print("p-vals",pv)
t2pv = pv
t1pv = pv*2
print("1t pv",t1pv)
print("2t pv",t2pv)
```

```
t-stat 0.017335249305284756
p-vals 0.9865473848679749
1t pv 1.9730947697359498
2t pv 0.9865473848679749
```

```
if tt == 1:
    if t1pv < al:
        print("Null Hypothesis: Rejected")
        print("Conclusion:",Ha)
    else:
        print("Null Hypothesis: Not Rejected")
        print("Conclusion:",Ho)
```

```
else:
    if t2pv < al/2:
        print("Null Hypothesis: Rejected")
        print("Conclusion:",Ha)
    else:
        print("Null Hypothesis: Not Rejected")
        print("Conclusion:",Ho)
```

```
Null Hypothesis: Not Rejected
Conclusion: mu <= 113
```

Two way hypothesis

```
# Problem: Check if the total mean marks is  equal to 113

#Ho:    m = 113
#Ha:    m != 113
#Tail: Two
#Test: One Sample Mean without std

# null hyp
Ho = "mu = 113"
# alt hyp
Ha = "mu != 113"
# alpha
al = 0.05
# mu - mean
mu = 113
# tail type
tt = 2
# data
marks = df['Total'].values
# print
print("Ho:", Ho)
print("Ha:", Ha)
print("al:", al)
print("mu:", mu)
print(marks)
print("")
```

```
Ho: mu = 113
Ha: mu != 113
al: 0.05
mu: 113
[135 122  85 102 115 149 104 109 105 105]
```

```
ts, pv = ttest_1samp(marks, mu)
print("t-stat",ts)
print("p-vals",pv)
t2pv = pv
t1pv = pv*2
print("1t pv",t1pv)
print("2t pv",t2pv)
```

```
t-stat 0.017335249305284756
p-vals 0.9865473848679749
1t pv 1.9730947697359498
2t pv 0.9865473848679749
```

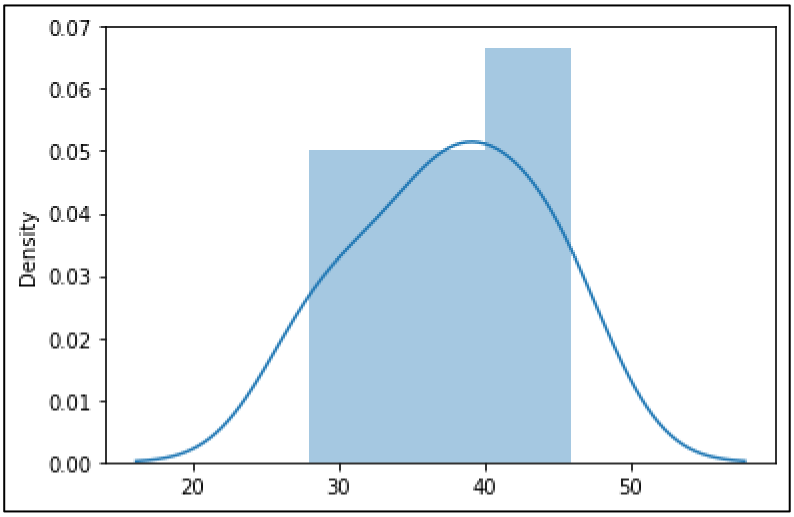
```
if tt == 1:
    if t1pv < al:
        print("Null Hypothesis: Rejected")
        print("Conclusion:",Ha)
    else:
        print("Null Hypothesis: Not Rejected")
        print("Conclusion:",Ho)
else:
    if t2pv < al/2:
        print("Null Hypothesis: Rejected")
        print("Conclusion:",Ha)
    else:
        print("Null Hypothesis: Not Rejected")
        print("Conclusion:",Ho)
```

```
Null Hypothesis: Not Rejected
Conclusion: mu = 113
```

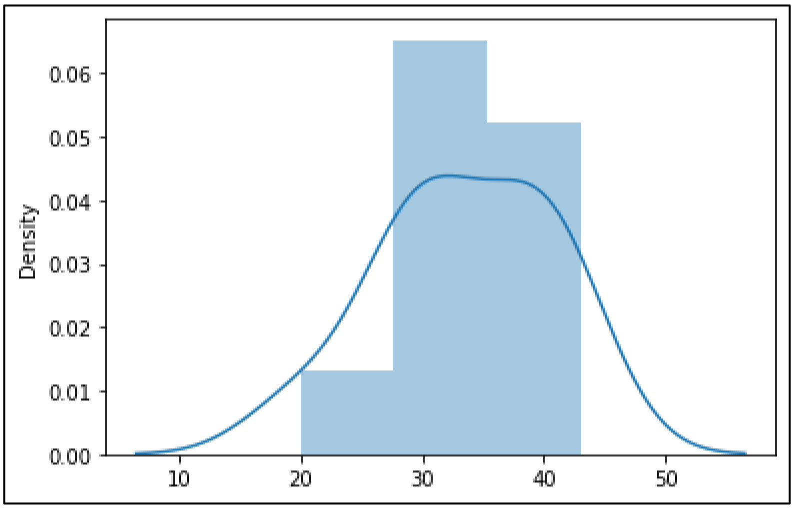
AB Testing

```
subj1 = np.array([45,36,29,40,46,37,43,39,28,33])
subj2 = np.array([40,20,30,35,29,43,40,39,28,31])
```

```
sns.distplot(subj1)
```




```
sns.distplot(subj2)
```




```
t_stat, p_val= stats.ttest_ind(subj1,subj2)
t_stat , p_val
```

```
(1.3659080395381 78, 0.18879292981719703)
```

```
#perform two sample t-test with equal variances
stats.ttest_ind(subj1, subj2, equal_var=True)
```

```
Ttest_indResult(statistic=1.365908039538178, pvalue=0.18879292981719703)
```

```
effect_size=abs((subj1.mean()-subj2.mean())/(subj1.std()-subj2.std()))
sample_size=10
alpha=0.05
ratio=1.0

statistical_power = tt_ind_solve_power(effect_size=effect_size,
nobs1=sample_size, alpha=alpha, ratio=1.0, alternative='two-sided')
print(statistical_power)
```

```
1.0
```

```
type_2_error=1-statistical_power
type_2_error
```

```
0.0
```

Practical 7

Q1) Simple Linear Regression

→

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

```
dataset = pd.read_csv('Salary_Data.csv')
dataset
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0

```
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
x
```

```
Output exceeds the size limit. Open the full output data in a text editor
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
```

```
Y
```

```
array([ 39343.,  46205.,  37731.,  43525.,  39891.,  56642.,  60150.,
        54445.,  64445.,  57189.,  63218.,  55794.,  56957.,  57081.,
        61111.,  67938.,  66029.,  83088.,  81363.,  93940.,  91738.,
        98273., 101302., 113812., 109431., 105582., 116969., 112635.,
       122391., 121872.])
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 1/3,
random_state = 0)
```

```
x_train, x_test, y_train, y_test
```

```
Output exceeds the size limit. Open the full output data in a text editor
(array([[ 2.9],
        [ 5.1],
        [ 3.2],
        [ 4.5],
        [ 8.2],
        [ 6.8],
        [ 1.3],
        [10.5],
        [ 3. ],
        [ 2.2],
        [ 5.9],
        [ 6. ],
        [ 3.7],
        [ 3.2],
        [ 9. ],
        [ 2. ],
        [ 1.1],
        [ 7.1],
        [ 4.9],
        [ 4. ]]),
 array([[ 1.5],
        [10.3],
        [ 4.1],
        [ 3.9],
        [ 9.5],
...
 array([ 56642.,  66029.,  64445.,  61111., 113812.,  91738.,  46205.,
        121872.,  60150.,  39891.,  81363.,  93940.,  57189.,  54445.,
        105582.,  43525.,  39343.,  98273.,  67938.,  56957.]),
 array([ 37731., 122391.,  57081.,  63218., 116969., 109431., 112635.,
         55794.,  83088., 101302.]))
```

```python
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)
```

```python
plt.scatter(x_train, y_train, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```


```python
plt.scatter(x_test, y_test, color = 'red')
```

```
plt.plot(x_train, regressor.predict(x_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Q2) Multiple Linear Regression

→

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

```
dataset = pd.read_csv('50_Startups.csv')
dataset.head()
dataset = pd.read_csv('50_Startups.csv')
dataset.head()
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

print(x)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']
 [67532.53 105751.03 304768.73 'Florida']
 [77044.01 99281.34 140574.81 'New York']
...
 [1315.46 115816.21 297114.46 'Florida']
 [0.0 135426.92 0.0 'California']
 [542.05 51743.15 0.0 'New York']
 [0.0 116983.8 45173.06 'California']]
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
x = np.array(ct.fit_transform(x))
```

```
print(x)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]
 [1.0 0.0 0.0 134615.46 147198.87 127716.82]
 [0.0 1.0 0.0 130298.13 145530.06 323876.68]
 [0.0 0.0 1.0 120542.52 148718.95 311613.29]
 [1.0 0.0 0.0 123334.88 108679.17 304981.62]
 [0.0 1.0 0.0 101913.08 110594.11 229160.95]
 [1.0 0.0 0.0 100671.96 91790.61 249744.55]
 [0.0 1.0 0.0 93863.75 127320.38 249839.44]
 [1.0 0.0 0.0 91992.39 135495.07 252664.93]
 [0.0 1.0 0.0 119943.24 156547.42 256512.92]
 [0.0 0.0 1.0 114523.61 122616.84 261776.23]
 [1.0 0.0 0.0 78013.11 121597.55 264346.06]
 [0.0 0.0 1.0 94657.16 145077.58 282574.31]
 [0.0 1.0 0.0 91749.16 114175.79 294919.57]
 [0.0 0.0 1.0 86419.7 153514.11 0.0]
 [1.0 0.0 0.0 76253.86 113867.3 298664.47]
 [0.0 0.0 1.0 78389.47 153773.43 299737.29]
 [0.0 1.0 0.0 73994.56 122782.75 303319.26]
 [0.0 1.0 0.0 67532.53 105751.03 304768.73]
 [0.0 0.0 1.0 77044.01 99281.34 140574.81]
...
 [0.0 1.0 0.0 1315.46 115816.21 297114.46]
 [1.0 0.0 0.0 0.0 135426.92 0.0]
 [0.0 0.0 1.0 542.05 51743.15 0.0]
 [1.0 0.0 0.0 0.0 116983.8 45173.06]]
```

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,
random_state = 0)
```

```python
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
y_pred = regressor.predict(X_test)
```

```
y_test
```

```
array([103282.38, 144259.4 , 146121.95,  77798.83, 191050.39, 105008.31,
        81229.06,  97483.56, 110352.25, 166187.94])
```

```
y_pred
```

```
array([103015.20159796, 132582.27760815, 132447.73845174,  71976.09851258,
       178537.48221056, 116161.24230167,  67851.69209676,  98791.73374687,
       113969.43533014, 167921.06569551])
```

```
print('Train Score: ', regressor.score(X_train, y_train))
print('Test Score: ', regressor.score(X_test, y_test))
```

```
Train Score:  0.9501847627493607
Test Score:  0.934706847328201
```

Practical 8

Q1) K-Nearest Neighbors

→

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

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
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)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
 [    47  49000]
 [    59  29000]
 [    49  65000]
 [    45 131000]
 [    31  89000]
 [    46  82000]
 [    47  51000]
 [    26  15000]
 [    60 102000]
 [    38 112000]
 [    40 107000]
 [    42  53000]
...
 [    29  43000]
```

```
print(y_train)
```

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

```python
print(X_test)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]
 [    27  20000]
 [    31  15000]
 [    36 144000]
 [    18  68000]
 [    47  43000]
 [    30  49000]
 [    28  55000]
 [    37  55000]
 [    39  77000]
 [    20  86000]
 [    32 117000]
 [    37  77000]
 [    19  85000]
 [    55 130000]
 [    35  22000]
 [    35  47000]
 [    47 144000]
 [    41  51000]
 [    47 105000]
 [    23  28000]
...
 [    23  63000]
 [    48  33000]
 [    48  90000]
 [    42 104000]]
```

```python
print(y_test)
```

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

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

```
print(X_train)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
...
 [-0.90383437 -0.77073441]
 [-0.21060859 -0.50979612]
 [-1.10189888 -0.45180983]
 [-1.20093113  1.40375139]]
```

```
print(X_test)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[-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]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
...
 [-1.49802789 -0.19087153]
 [ 0.97777845 -1.06066585]
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 
2)
classifier.fit(X_train, y_train)
```

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
...
 [0 0]
 [1 1]
 [1 1]
 [1 1]]
```

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

```
accuracy_score(y_test, y_pred)
```

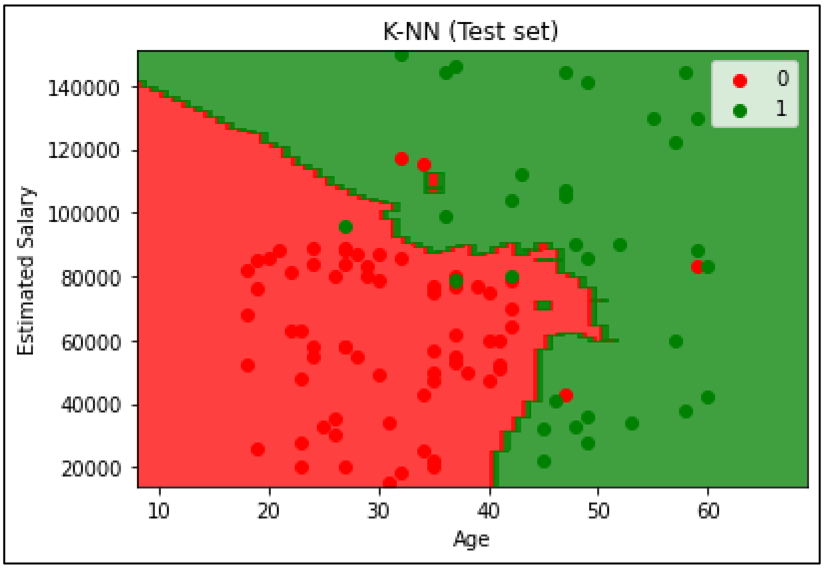
```
[[64  4]
 [ 3 29]]

0.93
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, 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('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, 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('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Practical 9

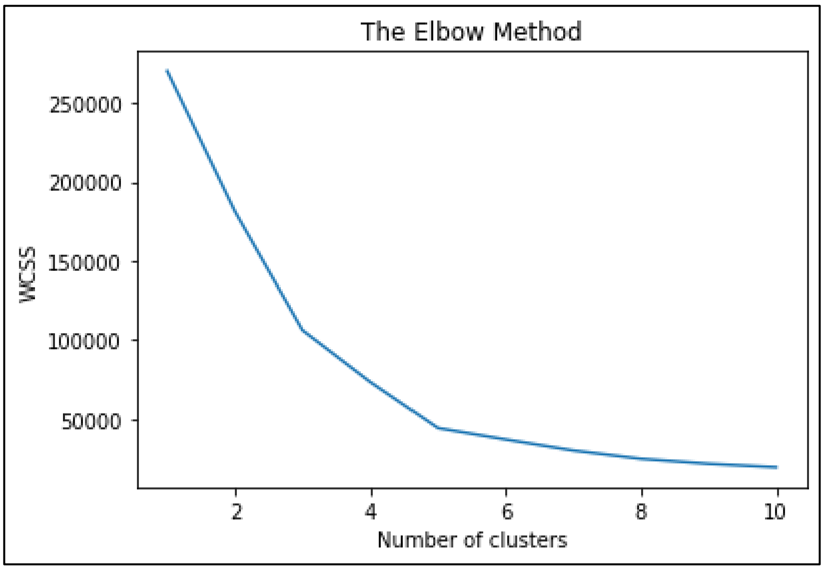
Q1) K-Means Clustering

→

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

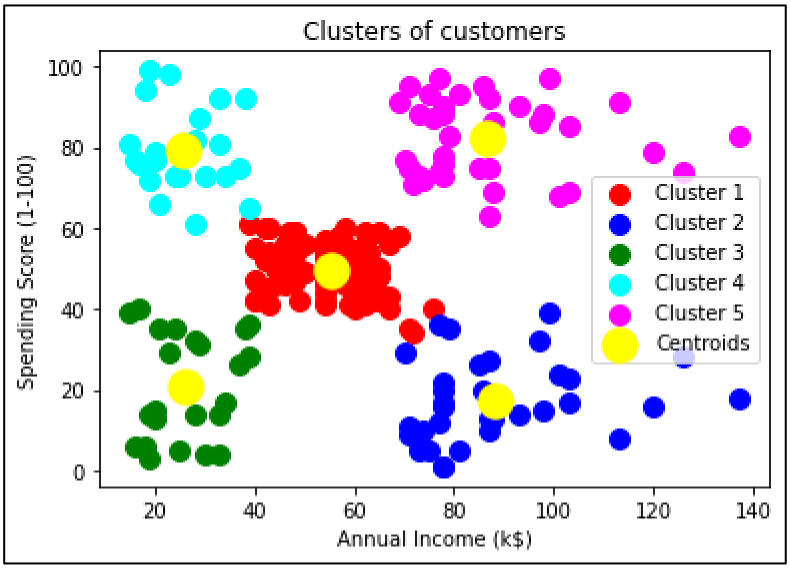
```
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green',
label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta',
label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



Name: Harsh Chheda
Roll Number: 22-15405/31031521005 MSC COMPUTER SCIENCE Subject: Machine Learning

Practical 10

Q1) Random Forest Classification

→

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

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
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)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
 [    47  49000]
 [    59  29000]
 [    49  65000]
 [    45 131000]
 [    31  89000]
 [    46  82000]
 [    47  51000]
 [    26  15000]
 [    60 102000]
 [    38 112000]
 [    40 107000]
 [    42  53000]
...
 [    29  43000]
 [    36  52000]
 [    27  54000]
```

```python
print(y_train)
```

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

```
print(X_test)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]
 [    27  20000]
 [    31  15000]
 [    36 144000]
 [    18  68000]
 [    47  43000]
 [    30  49000]
 [    28  55000]
 [    37  55000]
 [    39  77000]
 [    20  86000]
 [    32 117000]
 [    37  77000]
 [    19  85000]
 [    55 130000]
 [    35  22000]
 [    35  47000]
 [    47 144000]
 [    41  51000]
 [    47 105000]
 [    23  28000]
...
 [    23  63000]
 [    48  33000]
 [    48  90000]
 [    42 104000]]
```

```
print(y_test)
```

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

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

```
print(X_train)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
...
 [-0.90383437 -0.77073441]
 [-0.21060859 -0.50979612]
```

```
print(X_test)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[-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]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
...
 [-1.49802789 -0.19087153]
 [ 0.97777845 -1.06066585]
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
random_state = 0)
classifier.fit(X_train, y_train)

print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [1 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
...
 [0 0]
 [1 1]
 [1 1]
 [1 1]]
```

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

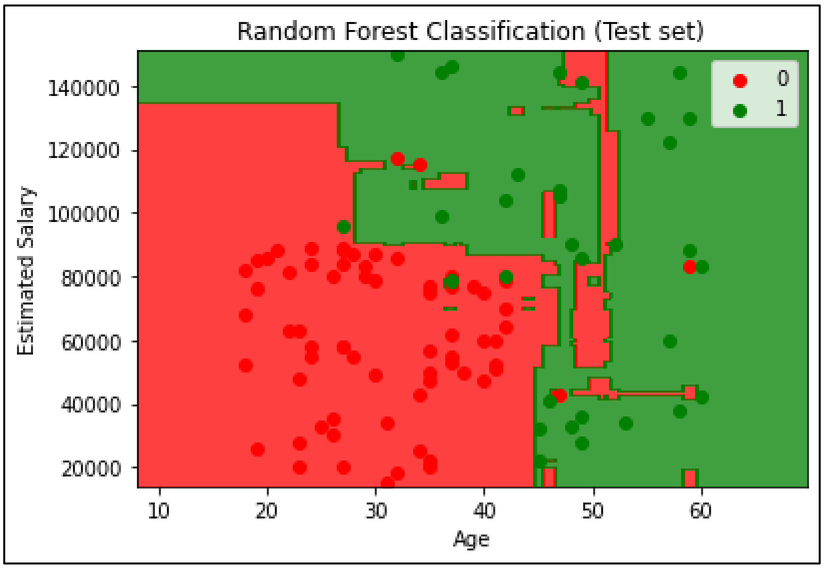
```
[[63  5]
 [ 4 28]]

0.91
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, 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('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, 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('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
```

```
plt.show()
```


Random Forest Classification (Test set)
Estimated Salary
140000
120000
100000
80000
60000
40000
20000
10
20
30
40
50
60
Age
0
1

Name: Harsh Chheda
Roll Number: 22-15405/31031521005 MSC COMPUTER SCIENCE Subject: Machine Learning

Practical 11

Q1) Support Vector Machine

→

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

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
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)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
 [    47  49000]
 [    59  29000]
 [    49  65000]
 [    45 131000]
 [    31  89000]
 [    46  82000]
 [    47  51000]
 [    26  15000]
 [    60 102000]
 [    38 112000]
 [    40 107000]
 [    42  53000]
...
 [    29  43000]
 [    36  52000]
 [    27  54000]
 [    26 118000]]
```

```
print(y_train)
```

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

```
print(X_test)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]
 [    27  20000]
 [    31  15000]
 [    36 144000]
 [    18  68000]
 [    47  43000]
 [    30  49000]
 [    28  55000]
 [    37  55000]
 [    39  77000]
 [    20  86000]
 [    32 117000]
 [    37  77000]
 [    19  85000]
 [    55 130000]
 [    35  22000]
 [    35  47000]
 [    47 144000]
 [    41  51000]
 [    47 105000]
 [    23  28000]
...
 [    23  63000]
 [    48  33000]
 [    48  90000]
 [    42 104000]]
```

```
print(y_test)
```

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

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

```
print(X_train)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
 [ 0.8787462  -0.53878926]
 [-1.20093113 -1.58254245]
 [ 2.1661655   0.93986109]
 [-0.01254409  1.22979253]
 [ 0.18552042  1.08482681]
 [ 0.38358493 -0.48080297]
...
 [-0.90383437 -0.77073441]
 [-0.21060859 -0.50979612]
 [-1.10189888 -0.45180983]
 [-1.20093113  1.40375139]]
```

```
print(X_test)
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[-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]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
...
 [-1.49802789 -0.19087153]
 [ 0.97777845 -1.06066585]
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
```

```
Output exceeds the size limit. Open the full output data in a text editor
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
...
 [0 0]
 [0 1]
 [1 1]
 [1 1]]
```

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

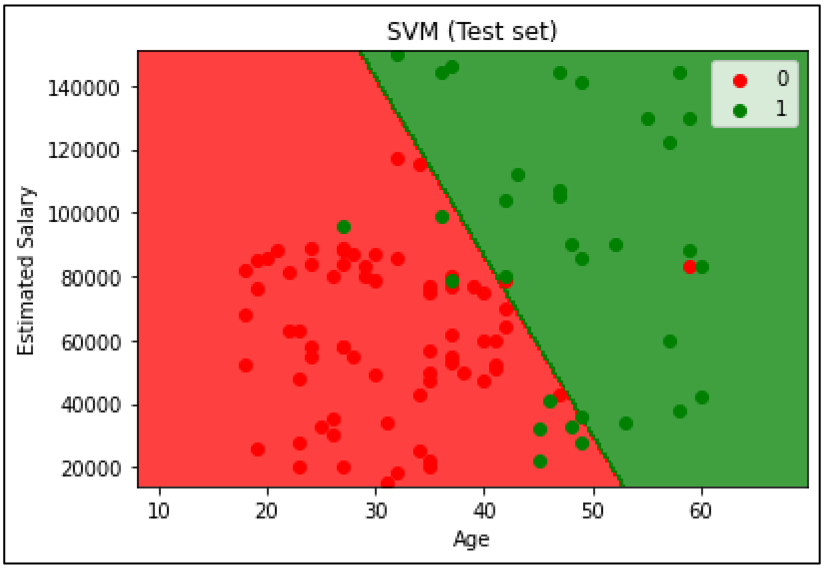
```
[[66  2]
 [ 8 24]]

0.9
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, 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('SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, 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('SVM (Test set)')
plt.xlabel('Age')
```

```
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Practical 12

Q1) ANN

→

```
import numpy as np
```

```
#assign input values
input_value = np.array([[0,0],[0,1],[1,1],[1,0]])
input_value.shape
input_value
```

```
array([[0, 0],
       [0, 1],
       [1, 1],
       [1, 0]])
```

```
#assign output values
output=np.array([0,1,1,0])
output=output.reshape(4,1)
output.shape
```

```
(4, 1)
```

```
#assign weights
weigths=np.array([[0.1],[0.2]])
weigths
```

```
array([[0.1],
       [0.2]])
```

```
#assign bias
bias=0.3


#Activation function
def sigmoid_func(x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid function
def der(x):
    return sigmoid_func(x) * (1- sigmoid_func(x))


#Updating weigths
for epochs in range(10000):
    input_arr = input_value

    weighted_sum=np.dot(input_arr, weigths) + bias
    first_output=sigmoid_func(weighted_sum)

    error =first_output - output
    total_error=np.square(np.subtract(first_output,output)).mean()

    first_der = error
    second_der = der(first_output)
    derivative = first_der * second_der

    t_input = input_value.T
    final_derivative = np.dot(t_input, derivative)

    #update weights
    weigths = weigths - 0.05 * final_derivative

    #update bias
    for i in derivative:
        bias = bias - 0.05 * i

print(bias)
```

```
[-4.19706344]
```

```
pred = np.array([0,1])
result = np.dot(pred, weigths) + bias

res = sigmoid_func(result)

print(res)
```

```
[0.99177089]
```