PRACTICAL-1

AIM:

Perform the following using Python Pandas and Matplotlib library on given dataset:

i) Deal with missing values in the data either by deleting records or using mean/median/mode imputation.

ii) Detect if Outliers exist and plot the data distribution using Box Plots, Scatter Plots and Histograms of matplotlib library

iii) Create and display the correlation matrix of all features of the data. Record and Analyse Observations.

CODE:

```
#IMPORTING LIBRARIES
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns

#GETTING THE DATA-SET
df = pd.read_csv('cwurData.csv')

#BOX PLOT USING SEABORN
sns.boxplot(df['broad_impact'])

#SCATTERPLOT USING MATPLOTLIB
fig, plot = plt.subplots(figsize = (15,15))
plot.scatter(df['world_rank'], df['quality_of_education'])

#X-LABEL
plot.set_xlabel('WORLD RANKS')

#Y-LABEL
plot.set_ylabel('QUALITY OF EDUCATION')
plt.show()
plt.boxplot(df['score'])


plt.figure(figsize=(12,5))
```

#HISTOGRAM USING MATPLOTLIB
plt.hist(df['country'])
plt.xticks(rotation = 90)
plt.show()

#CORRELATION
df.corr()
corrmat = df.corr()

#HEATMAP
sns.heatmap(corrmat,annot=True)
sns.heatmap(corrmat,annot=False)
sns.pairplot(df)

OUTPUT:

	world_rank	institution	country	national_rank	quality_of_education	alumni_employment	quality_of_faculty	publications	influence	citations	broad_impact	patents	score	year
0	1	Harvard University	USA	1	7	9	1	1	1	1	NaN	5	100.00	2012
1	2	Massachusetts Institute of Technology	USA	2	9	17	3	12	4	4	NaN	1	91.67	2012
2	3	Stanford University	USA	3	17	11	5	4	2	2	NaN	15	89.50	2012
3	4	University of Cambridge	United Kingdom	1	10	24	4	16	16	11	NaN	50	86.17	2012
4	5	California Institute of Technology	USA	4	2	29	7	37	22	22	NaN	18	85.21	2012
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2195	996	University of the Algarve	Portugal	7	367	567	218	926	845	812	969.0	816	44.03	2015
2196	997	Alexandria University	Egypt	4	236	566	218	997	908	645	981.0	871	44.03	2015
2197	998	Federal University of Ceará	Brazil	18	367	549	218	830	823	812	975.0	824	44.03	2015
2198	999	University of A Coruña	Spain	40	367	567	218	886	974	812	975.0	651	44.02	2015
2199	1000	China Pharmaceutical University	China	83	367	567	218	861	991	812	981.0	547	44.02	2015

2200 rows × 14 columns

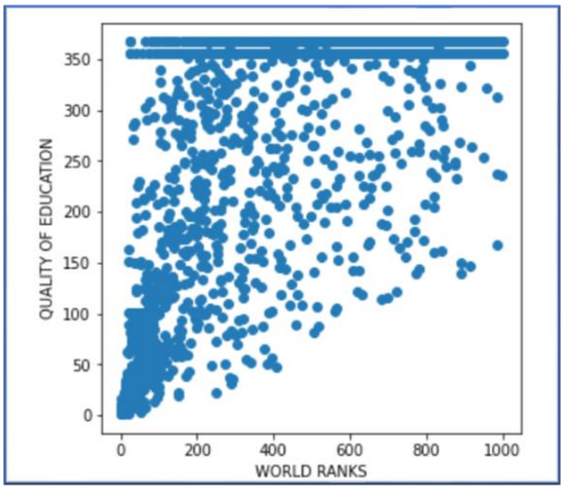
GLIMPSES OF DATASET

```
world_rank               459.590909
national_rank             40.278182
quality_of_education     275.100455
alumni_employment        357.116818
quality_of_faculty       178.888182
publications             459.908636
influence                459.797727
citations                413.417273
broad_impact             496.699500
patents                  433.346364
score                     47.798395
year                    2014.318182
dtype: float64
```

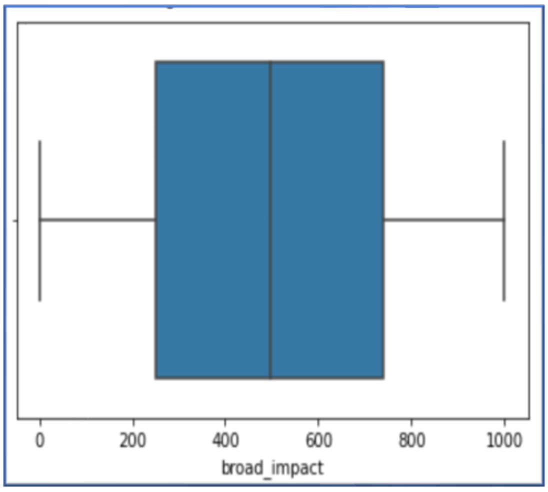
Mean

```
world_rank                450.5
national_rank              21.0
quality_of_education      355.0
alumni_employment         450.5
quality_of_faculty        210.0
publications              450.5
influence                 450.5
citations                 406.0
broad_impact              496.0
patents                   426.0
score                      45.1
year                     2014.0
dtype: float64
```

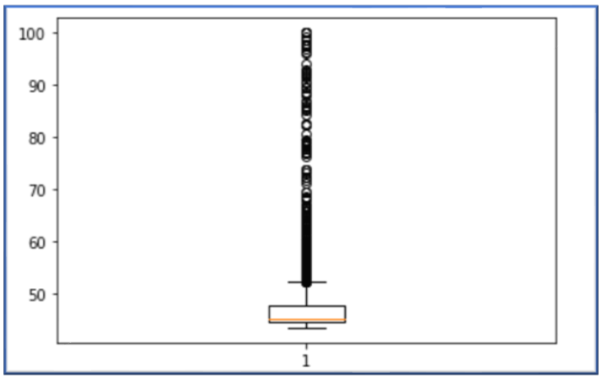
Median



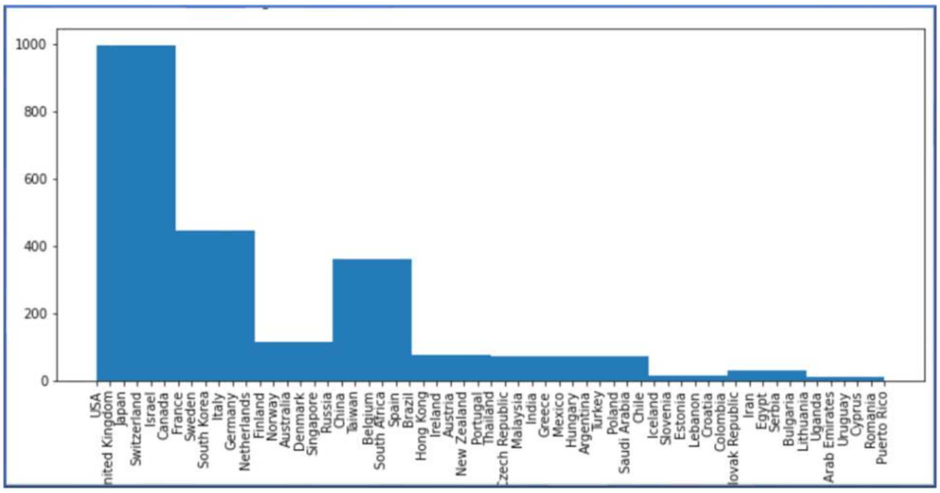
Scatter plot



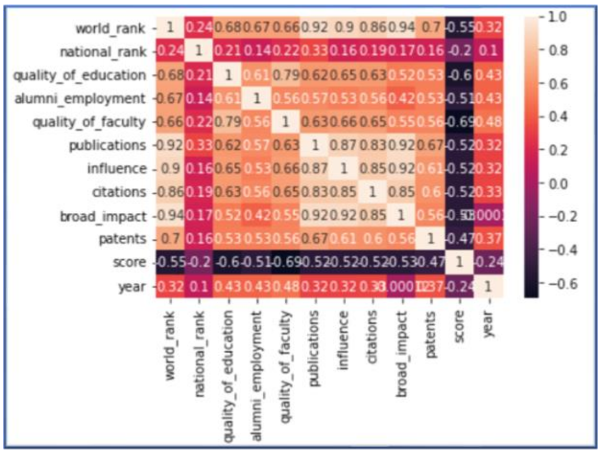
Boxplot



Boxplot of score column



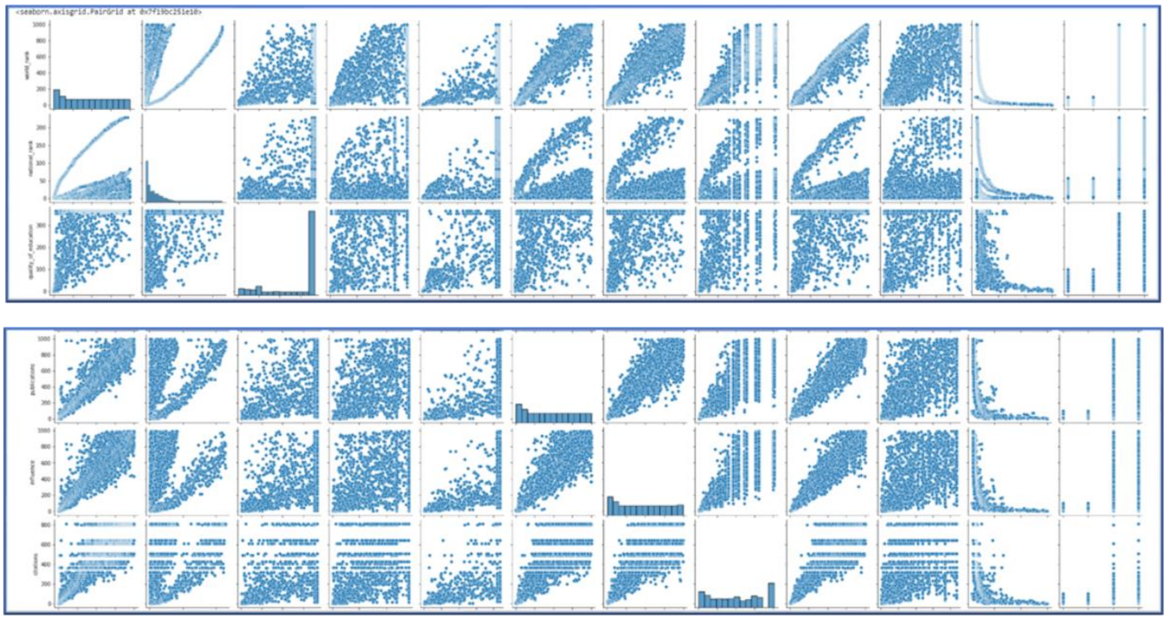
Histogram



Correlation heatmap

	world_rank	national_rank	quality_of_education	alumni_employment	quality_of_faculty	publications	influence	citations	broad_impact	patents	score	year
world_rank	1.000000	0.238553	0.676166	0.668529	0.663864	0.923037	0.895871	0.856573	0.942998	0.698214	-0.549098	0.320844
national_rank	0.238553	1.000000	0.211975	0.135342	0.222833	0.326857	0.161086	0.186797	0.173043	0.159440	-0.199756	0.102951
quality_of_education	0.676166	0.211975	1.000000	0.606421	0.786181	0.624657	0.645641	0.630455	0.521615	0.528120	-0.600541	0.427732
alumni_employment	0.668529	0.135342	0.606421	1.000000	0.558618	0.572096	0.527382	0.559200	0.423619	0.528219	-0.510374	0.427710
quality_of_faculty	0.663864	0.222833	0.786181	0.558618	1.000000	0.634423	0.656406	0.652186	0.549345	0.555107	-0.693540	0.484788
publications	0.923037	0.326857	0.624657	0.572096	0.634423	1.000000	0.874952	0.829912	0.917878	0.671558	-0.522111	0.318245
influence	0.895871	0.161086	0.645641	0.527382	0.656406	0.874952	1.000000	0.845207	0.916040	0.611811	-0.522837	0.318298
citations	0.856573	0.186797	0.630455	0.559200	0.652186	0.829912	0.845207	1.000000	0.852638	0.598728	-0.522438	0.328771
broad_impact	0.942998	0.173043	0.521615	0.423619	0.549345	0.917878	0.916040	0.852638	1.000000	0.562861	-0.531590	-0.000124
patents	0.698214	0.159440	0.528120	0.528219	0.555107	0.671558	0.611811	0.598728	0.562861	1.000000	-0.474810	0.368570
score	-0.549098	-0.199756	-0.600541	-0.510374	-0.693540	-0.522111	-0.522837	-0.522438	-0.531590	-0.474810	1.000000	-0.239136
year	0.320844	0.102951	0.427732	0.427710	0.484788	0.318245	0.318298	0.328771	-0.000124	0.368570	-0.239136	1.000000

Correlation matrix



Pairplot

PRACTICAL-2

AIM:

For given Dataset (you may continue to use the same processed dataset from experiment 1 only for this experiment), perform the following using Python Pandas and scikit-learn library or by writing your own user-defined function:

I. Perform Data Standardization and Normalization
II. Select the 10 best features of the data using different statistical scoring methods. (Hint: Chi-Squared Statistical Test is a good scoring method)
III. Split the data into training and testing sets in a ratio of 80:20.

CODE:

```
#IMPORTING LIBRARIES
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler

#GETTING THE DATASET
df = pd.read_csv('winequality-red.csv',sep=';')

y=df['quality']
X = df.drop('quality',inplace=False,axis=1)

# SPLITTING THE DATASET INTO TESTING AND TRAINING SET
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_state=17)

sc_X = StandardScaler()
sc_X = sc_X.fit_transform(df)

keys = df.head(0).keys()

sc_X = pd.DataFrame(data=sc_X, columns=keys)
```

```
sc_X

#STANDARDIZATION
scaler = StandardScaler().fit(X)
standardized_X_train = scaler.transform(X_train)
standardized_X_test = scaler.transform(X_test)

# Normalisation
scalerscaler = Normalizer()
Normalize_x_test = scalerscaler.transform(X_train)
Normalize_y_test = scalerscaler.transform(X_test)

print (Normalize_x_test)
print(df.isna().sum())

scaler = MinMaxScaler()
scaler.fit(df)
scaled_features = scaler.transform(df)
df_MinMax = pd.DataFrame(data=scaled_features,columns=keys)
df_MinMax

bestfeatures = SelectKBest(score_func=chi2,k=3)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Column Names','Column Scores']
print(featureScores.nlargest(10,'Column Scores'))
```

OUTPUT:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

Glimpses of Dataset

index	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	-0.5283596117263187	0.9618766712454476	-1.3914722776605752	-0.45321840673805114	-0.2437068659232994	-0.4661925171277346	-0.3791326888959121	0.5582744625309729	1.288642916709892	-0.5792065221278776	-0.9602461068947519
1	-0.2985474337867381	1.9674424541409123	-1.3914722776605752	0.0434161447934977	0.22387519619714352	0.8726382318679612	0.6243632277859998	0.028260767740937578	-0.7199332966852555	0.1289504006827337	-0.5847771110397104
2	-0.2985474337867381	1.297065265543936	-1.1860704292930377	-0.16942723443430907	0.09635286452728704	-0.08366944598610641	0.2290465545476709	0.1342635066969329	-0.3311766102216806	-0.04808883001991929	-0.5847771110397104
3	1.6548560786997	-1.3844434888439687	1.4841535994849522	-0.45321840673805114	-0.264960408537306	0.1075920895847071	0.41150045758074577	0.6642772014889682	-0.9791044209943073	-0.4611803683261096	-0.5847771110397104
4	-0.5283596117263187	0.9618766712454476	-1.3914722776605752	-0.45321840673805114	-0.2437068659232994	-0.4661925171277346	-0.3791326888959121	0.5582744625309729	1.288642916709892	-0.5792065221278776	-0.9602461068947519
5	-0.5283596117263187	0.7384176083797893	-1.3914722776605752	-0.5241661998139666	-0.264960408537306	-0.2749309815569199	-0.19667888586283722	0.5582744625309729	1.288642916709892	-0.5792065221278776	-0.9602461068947519
6	-0.24109438930184252	0.40322901408130096	-1.0833695051092689	-0.6660617859658576	-0.3924827402071622	-0.08366944598610641	0.38109149040856666	-0.18374471017511187	-0.07200548591262876	-1.1693372911367206	-0.9602461068947519
7	-0.5858126562112143	0.8825528426633746	-1.3914722776605752	-0.9498529582695998	-0.4774976279870665	-0.08366944598610641	-0.774449262134241	-1.1377693607971284	0.5111295437827393	-1.1103242142358365	-0.3970426131121905
8	-0.2985474337867381	0.2914994826484715	-1.2887713534768064	-0.38227061366211557	-0.30746785242725816	-0.657454052698547	-0.8656761636507785	0.028260767740937578	0.316751200550949	-0.5201934452266994	-0.8663788579309919
9	-0.4709065672414237	-0.1554186430828458	0.4571443576472635	2.526588902451241	-0.3499752963172103	0.1075920895847071	1.688677078812 27	0.5582744625309729	0.2519584194736 8747	0.8371073234933449	0.07229363170661092
10	-0.9305309231205855	0.2914994826484715	-0.9806685809254999	-0.5241661998139666	0.20262147425216745	-0.08366944598610641	0.5635452934416415	-0.4487515575701001	-0.2015910480671547	-0.6972326759296462	-1.1479806048222734
11	-0.4709065672414237	-0.1554186430828458	0.4571443576472635	2.526588902451241	-0.3499752963172103	0.1075920895847071	1.688677078812 27	0.5582744625309729	0.2519584194736 8747	0.8371073234933449	0.07229363170661092
12	-1.5625144124544335	0.487026162655923	-1.3914722776605752	-0.6660617859658576	0.03259169869235881	0.011961321799300349	0.38109149040856666	-1.2967734692341801	1.742192384250734	-0.8152588297314148	-0.49090986207595044
13	-0.2985474337867381	0.45909377979771565	0.0976911230040726	-0.6660617859658576	0.5639347473167605	-0.657454052698547	-0.5311775247568079	0.3462689846149235	-0.3311766102216806	5.32210116796055	-1.2418478537860334
14	0.3334360555471099	0.5149585455141305	-0.4671639600066559	0.8947896617047238	1.881965507905276	3.454668962073944	2.996262687215974	0.962285418363013	-0.9791044209943073	1.3092119387004189	-1.1479806048222734
15	0.3334360555471099	0.5149585455141305	-0.41581349791477146	0.9557374547806594	1.7541431762354 2	3.359038194288537	3.087489568732511	0.962285418363013	-0.9143116399170458	1.6042773232048406	-1.1479806048222734
16	0.10362387760752874	-1.3844434888439687	1.4841535994849522	-0.5241661998139666	0.09635286452728704	1.8289459097220289	1.7190860459844493	0.08126213721993523	-0.07200548591262876	0.5420419389889234	0.07229363170661092
17	-0.12618830033205244	0.1797699512156426	0.046340660912188414	-0.5951139928899222	5.96238012134068	0.011961321799300349	0.2898645888920292	0.028260767740937578	-1.3030683263806235	3.66973501473579	-1.0541133558585118
18	-0.5283596117263187	0.3473642483648862	-0.9806685809254999	1.3204764201603374	-0.03116946714256943	-0.9443463560547672	-0.5311775247568079	0.3462689846149235	0.4463367627054749	-0.9332849835331836	-1.3357151027497933
19	-0.24109438930184252	-1.1609844259783102	1.22740128902553	-0.5241661998139666	5.388529628826326	0.1075920895847071	0.2898645888920292	0.08126213721993523	-1.7566177939214629	2.489473476718105	-1.1479806048222734
20	0.3334360555471099	-1.7196320831424572	1.0733499027498765	-0.5241661998139666	-0.22245296464735387	1.2551613030095883	0.41150045758074577	0.028260767740937578	0.5111295437827393	-0.7562457528305305	-0.9602461068947519

Data after min max scaler implementation

```
[[ 0.21852997  0.96187667 -0.15906119 ...  0.05758008 -0.34315421
  -1.3357151 ]
 [ 0.9079665  -0.99339013  0.55984528 ... -0.33117661  0.95513348
   0.16616088]
 [ 0.04617083  0.59875569  0.45714436 ... -0.0072127  -0.52019345
  -0.02157362]
 ...
 [-1.1603431  -0.76993107 -0.98066858 ...  0.18716564 -0.46118037
  -0.96024611]
 [-0.6432657   0.23563472 -1.13471997 ...  0.44633676 -0.34315421
  -0.11544087]
 [-0.24109439 -0.09955388 -0.10771073 ...  1.15905735  1.78131655
   1.57416962]]
```

Standardisation

```
[[0.42155632 0.03391832 0.01162914 ... 0.16086977 0.02907285 0.43609274]
 [0.16990398 0.00600671 0.00652157 ... 0.05594818 0.01407285 0.18191739]
 [0.1431037  0.01081796 0.00613302 ... 0.05638967 0.00971061 0.177176  ]
 ...
 [0.26800367 0.0165907  0.00340322 ... 0.14208448 0.02467335 0.39987849]
 [0.17318856 0.01371076 0.0012027  ... 0.08130241 0.01443238 0.24775586]
 [0.15210164 0.00981922 0.00481334 ... 0.06719427 0.01848324 0.2329658 ]]
```

Normalisation

	Column Names	Column Scores
6	total sulfur dioxide	2755.557984
5	free sulfur dioxide	161.936036
10	alcohol	46.429892
1	volatile acidity	15.580289
2	citric acid	13.025665
0	fixed acidity	11.260652
9	sulphates	4.558488
3	residual sugar	4.123295
4	chlorides	0.752426
8	pH	0.154655

Best Feature Selection

PRACTICAL-3

AIM:

Implement the linear regression and calculate the different evaluation measure (MAE, RMSE etc.). for the same. Also implement gradient descent and observe the cost with linear regression using gradient descent. Do not use any Python library for linear regression. (Hint: Linear Regression Formula is Y= mX +b where Y is target variable and X is independent variable)

CODE:

```
#PRACTICAL-3
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#GETTING THE DATASET
df = pd.read_csv('fifa_players.csv')
df.head()

#Removing special character, "M" and "K" for the data column
df['Release Clause'] = df['Release Clause'].str.replace(r"€", "")
df['Release Clause'] = df['Release Clause'].str.replace(r"M", "")
df['Release Clause'] = df['Release Clause'].str.replace(r"K", "")
df['Release Clause'] = df['Release Clause'].astype(float)

x = df.Potential[:1000]
df['Release Clause'] = df['Release Clause'].fillna(0)
y = df['Release Clause'][:1000]

#SCATTER PLOT
plt.scatter(x,y, color='violet')
plt.show()

mean_x = x.mean()  mean_y = y.mean()
tempx = [(i - mean_x)**2 for i in x]
var_x = sum(tempx, 0)/(len(tempx))

tempy = [(i - mean_y)**2 for i in y]
var_y = sum(tempy, 0)/(len(tempy))

#PRINTING VARIANCE
```

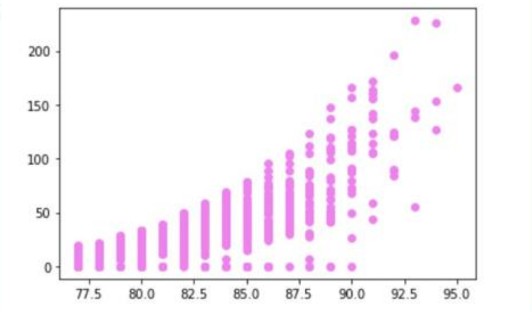
```
print("Variance of x : ", var_x)
print("Variance of y : ", var_y)

temp1x = [(i - mean_x) for i in x]
temp1y = [(i - mean_y) for i in y]
total = 0

for i in range(len(temp1x)):
  for j in range(len(temp1y)):
    if i == j:
      total = total + temp1x[i]*temp1y[j]
cov = total/(len(temp1x) - 1)

#PRINTING CO-VARIANCE
print("Covariance : ", cov)
r = cov/((var_x*var_y)**0.5)
slope = r*((var_y**0.5)/(var_x**0.5))

c = mean_y - slope * mean_x

print("Value of slope : ", slope)
print("Value of intercept : ", c)
predicted_val = []
for i in range(len(x)):
  new_y = slope*x[i] + c
  predicted_val.append(round(new_y,2))
  print(round(new_y, 2), y[i])

new_meanY = sum(predicted_val)
new_diff = [(i - j)**2 for i, j in zip(y, predicted_val)]
new_sum = sum(new_diff)
mse = new_sum/len(new_diff)
rmse = mse**0.5 print("Mean squared error : ", mse)
print("Root mean squared error : ", rmse)
```

OUTPUT:

index	Unnamed: 0	ID	Name	Age	Photo	Nationality	Flag	Overall	Potential	Club	Club Logo	Value	Wage	Special	Preferred Foot
0	0	158023	L. Messi	31	https://cdn.sofifa.org/players/4/19/158023.png	Argentina	https://cdn.sofifa.org/flags/52.png	94	94	FC Barcelona	https://cdn.sofifa.org/teams/2/light/241.png	€110.5M	€565K	2202	Left
1	1	20801	Cristiano Ronaldo	33	https://cdn.sofifa.org/players/4/19/20801.png	Portugal	https://cdn.sofifa.org/flags/38.png	94	94	Juventus	https://cdn.sofifa.org/teams/2/light/45.png	€77M	€405K	2228	Right
2	2	190871	Neymar Jr	26	https://cdn.sofifa.org/players/4/19/190871.png	Brazil	https://cdn.sofifa.org/flags/54.png	92	93	Paris Saint-Germain	https://cdn.sofifa.org/teams/2/light/73.png	€118.5M	€290K	2143	Right
3	3	193080	De Gea	27	https://cdn.sofifa.org/players/4/19/193080.png	Spain	https://cdn.sofifa.org/flags/45.png	91	93	Manchester United	https://cdn.sofifa.org/teams/2/light/11.png	€72M	€260K	1471	Right
4	4	192985	K. De Bruyne	27	https://cdn.sofifa.org/players/4/19/192985.png	Belgium	https://cdn.sofifa.org/flags/7.png	91	92	Manchester City	https://cdn.sofifa.org/teams/2/light/10.png	€102M	€355K	2281	Right

Glimpses of Dataset

```
plt.scatter(x,y, color='violet')
plt.show()
```



Scatter plot

```
Variance of x :  14.051950999999991
Variance of y :  841.1069849599946
```

Variance

```
Value of slope :  5.857502330867523
Value of intercept :  -448.66014481288676
```

Slope and Intercept

```
Covariance :  82.3093357357357
```

Covariance

```
Mean squared error :  359.9442667000011
Root mean squared error :  18.972197202749108
```

MSE & RMSE

NON-LINEAR REGRESSION:

CODE:

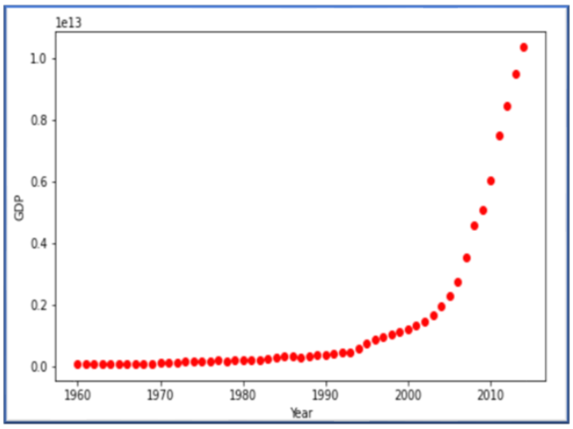
```
import numpy as np
import pandas as pd
df = pd.read_csv("china_gdp.csv")
df.head()
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
x_data, y_data = (df["Year"].values, df["Value"].values)
plt.plot(x_data, y_data, 'ro')
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
X = np.arange(-5.0, 5.0, 0.1)
Y = 1.0 / (1.0 + np.exp(-X))
plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Indepdendent Variable')
plt.show()
def sigmoid(x, Beta_1, Beta_2):
    y = 1 / (1 + np.exp(-Beta_1*(x-Beta_2)))
    return y
beta_1 = 0.10
beta_2 = 1990.0
#logistic function
Y_pred = sigmoid(x_data, beta_1 , beta_2)
#plot initial prediction against datapoints
plt.plot(x_data, Y_pred*15000000000000.)
plt.plot(x_data, y_data, 'ro')
# Lets normalize our data
xdata =x_data/max(x_data)
ydata =y_data/max(y_data)
from scipy.optimize import curve_fit
popt, pcov = curve_fit(sigmoid, xdata, ydata)
# Now we plot our resulting regression model.
x = np.linspace(1960, 2015, 55)
x = x/max(x)
plt.figure(figsize=(8,5))
y = sigmoid(x, *popt)
plt.plot(xdata, ydata, 'ro', label='data')
plt.plot(x,y, linewidth=3.0, label='fit')
plt.legend(loc='best')
plt.ylabel('GDP')
```

plt.xlabel('Year')
plt.show()

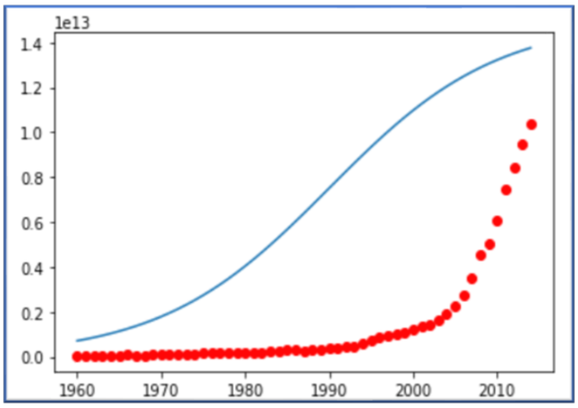
OUTPUT:

	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10

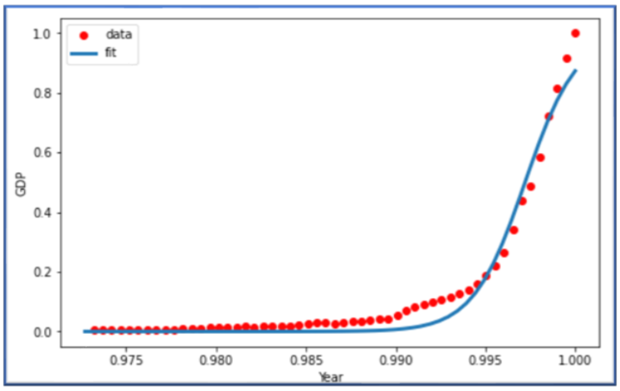
Glimpses of Dataset



Plot of Dataset



Initial Prediction against data points



Graph after normalization

PRACTICAL-4

AIM:

Create Visual analysis for the given data set using MATLAB.

BASIC THEORY:

- The name MATLAB stands for **MATrix LABoratory.**
- It is written in C, C++, and Java.
- Matlab was initially released in 1984.
- MATLAB is a programming platform designed specifically for engineers and scientists to analyze and design systems and products that transform our world.
- Matlab helps to perform mathematical calculation, design, analysis and optimization (structural and mathematical), as well as gives speed, accuracy and precision to results.
- The basic data element of MATLAB as the name suggests is the Matrix or an array.

Common applications of matlab:

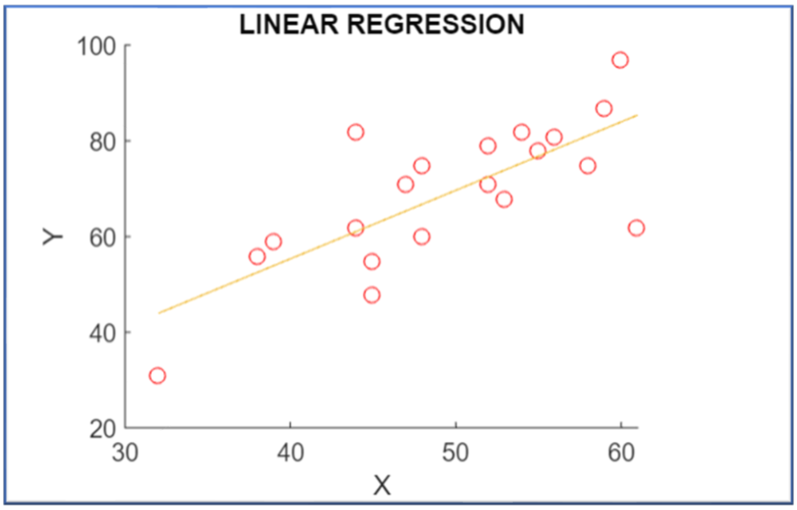
- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

CODE:

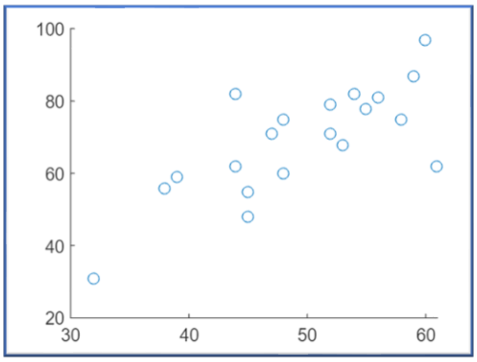
```
X=[32,53,61,47,59,55,52,39,48,52,45,54,44,58,56,48,44,60,45,38]
Y=[31,68,62,71,87,78,79,59,75,71,55,82,62,75,81,60,82,97,48,56]

n=length(X)
denominator =  ( ( n * sum(X .* X)) - sum(X) * sum(X))
b = ( ( sum(Y) * sum(X.*X) ) - ( sum(X) .* sum(X.*Y) ) ) / denominator
m = ( n * sum( X .* Y ) - (sum(X) * sum(Y) ) ) / denominator

yCalc1 = X.*m + b
RGB = [255 0 0]/256

scatter(X,Y,[],RGB)
hold on
plot(X,yCalc1)
```

```
xlabel('X')
ylabel('Y')
title('LINEAR REGRESSION')
```

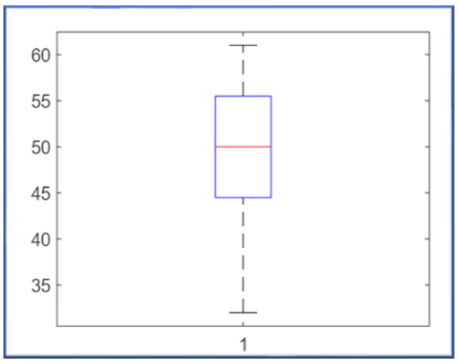
OUTPUT:



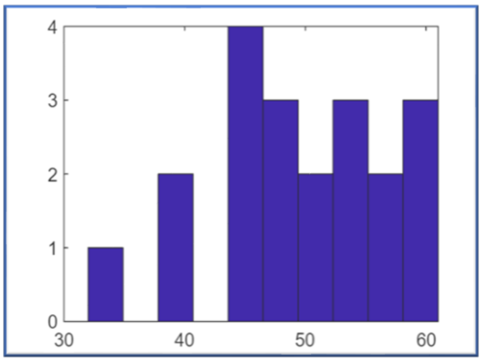
Linear Regression Output



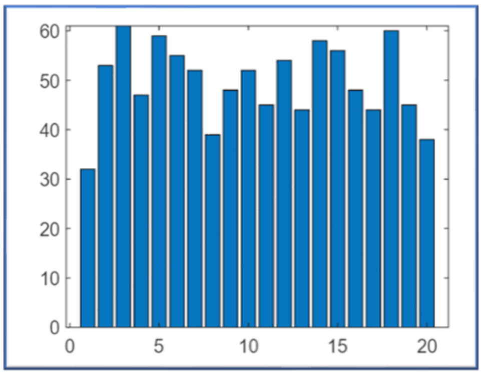
Scatter Plot



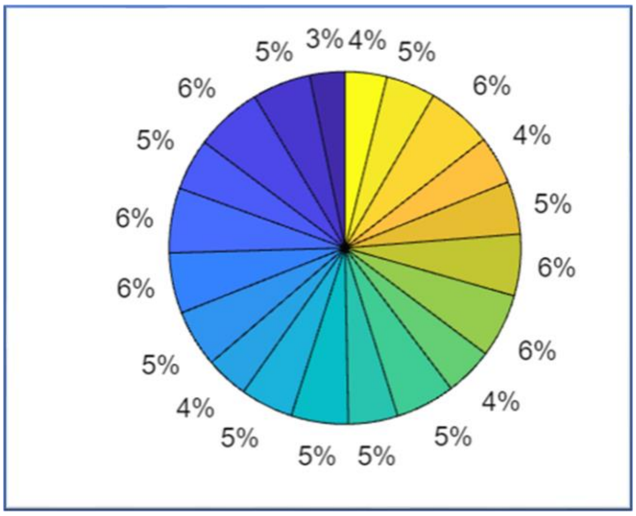
Box Plot



Histogram



Bar plot



Pie Chart

PRACTICAL-5

AIM:

Implement logistic regression and calculate the different evaluation measure (F-measures, Confusion Matrix etc.) for the same. Also implement gradient descent and observe the cost with logistic regression using gradient descent. (Hint: Confusion Matrix and Fmeasures involve use of True Negatives, True Positives, False Negatives and False Positives). Also implement Cross- Validation

CODE:

```
#PRACTICAL-5
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import precision_recall_fscore_support
import matplotlib.pyplot as plt

#GETTING DATASET
df = pd.read_excel('default of credit card clients.xls')
df1 = df.rename(columns=df.iloc[0]).drop(df.index[0])
df1
df1.drop('ID',inplace = True, axis = 1)
df1.head(2)

#DEFINING DEPENDENT AND INDEPENDENT VARIABLE
y = df1['default payment next month']
x = df1.drop('default payment next month',inplace =False,axis = 1)

#IMPLEMENTING LOGISTIC REGRESSION CLASSIFIER
y=y.astype('int')
classifier = LogisticRegression().fit(x, y)
classifier.score(x,y)

#IMPLEMENTING NORMALIZATION CLASSIFIER
transformer = Normalizer().fit(x)
x_norm = transformer.transform(x)
x_norm

classifier = LogisticRegression().fit(x_norm,y)
classifier.score(x_norm,y)
```

DEPSTAR

scaler = StandardScaler().fit(x)
std_x=scaler.transform(x)
std_x
cls=LogisticRegression().fit(std_x,y)
cls.score(std_x,y)
y_pred=cls.predict(std_x)
y_pred
#BUILDING CONFUSION MATRIX
mat = confusion_matrix(y,y_pred)
print(mat)
conf = ConfusionMatrixDisplay(mat)
conf.plot()
#FINDING OUT PRECISION,RECALL AND F1 SCORE
precision, recall, f1, support = precision_recall_fscore_support(y,y_pred)

OUTPUT:

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default payment next month
1	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689	0	0	0	0	689	0	0	0	0	1
2	120000	2	2	2	26	-1	2	0	0	0	2	2682	1725	2682	3272	3455	3261	0	1000	1000	1000	0	2000	1
3	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
4	50000	2	2	1	37	0	0	0	0	0	0	46990	48233	49291	28314	28959	29547	2000	2019	1200	1100	1069	1000	0
5	50000	1	2	1	57	-1	0	-1	0	0	0	8617	5670	35835	20940	19146	19131	2000	36681	10000	9000	689	679	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
996	220000	1	3	1	39	0	0	0	0	0	0	188948	192815	208365	88004	31237	15980	8500	20000	5003	3047	5000	1000	0
997	150000	1	3	2	43	-1	-1	-1	-1	0	0	1683	1828	3502	8979	5190	0	1837	3526	8998	129	0	0	0
998	30000	1	2	2	37	4	3	2	-1	0	0	3565	3356	2758	20878	20582	19357	0	0	22000	4200	2000	3100	1
999	80000	1	3	1	41	1	-1	0	0	0	-1	-1645	78379	76304	52774	11855	48944	85900	3409	1178	1926	52964	1804	1
000	50000	1	2	1	46	0	0	0	0	0	0	47929	48905	49764	36535	32428	15313	2078	1800	1430	1000	1000	1000	1

Glimpses of Dataset

```
y=y.astype('int')
classifier = LogisticRegression().fit(x, y)
classifier.score(x,y)

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.7788333333333334
```

Logistic Regression

```
transformer = Normalizer().fit(x)
x_norm = transformer.transform(x)
x_norm

array([[9.69135103e-01, 9.69135103e-05, 9.69135103e-05, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [9.98004610e-01, 1.66334102e-05, 1.66334102e-05, ...,
        8.31670508e-03, 0.00000000e+00, 1.66334102e-02],
       [8.98264904e-01, 1.99614423e-05, 1.99614423e-05, ...,
        9.98072115e-03, 9.98072115e-03, 4.99036058e-02],
       ...,
       [5.79388907e-01, 1.93129636e-05, 3.86259271e-05, ...,
        8.11144469e-02, 3.86259271e-02, 5.98701870e-02],
       [4.34542691e-01, 5.43178364e-06, 1.62953509e-05, ...,
        1.04616153e-02, 2.87688989e-01, 9.79893768e-03],
       [4.50856476e-01, 9.01712953e-06, 1.80342591e-05, ...,
        9.01712953e-03, 9.01712953e-03, 9.01712953e-03]])
```

Normalizer procedure

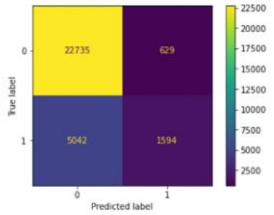
```
classifier = LogisticRegression().fit(x_norm,y)
classifier.score(x_norm,y)

0.7787
```

Logistic Regression

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(x)
std_x=scaler.transform(x)
std_x
cls=LogisticRegression().fit(std_x,y)
cls.score(std_x,y)
y_pred=cls.predict(std_x)
y_pred

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

Standard scaler implementation

```
[[22735   629]
 [ 5042  1594]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fbfd8eff150>
```



Confusion Matrix

```
precision

array([0.81848292, 0.71704903])

recall

array([0.97307824, 0.24020494])

support

array([23364,  6636])

f1

array([0.8891105 , 0.35986003])
```

F1 score, precision, recall

PRACTICAL-6

AIM:

Implement K-Nearest Neighbours, Support Vector Machine (SVM) and Naïve Bayes Classifier with python’s Scikit-Learn on different datasets. Compare the classifiers based on their evaluation measures.

CODE:

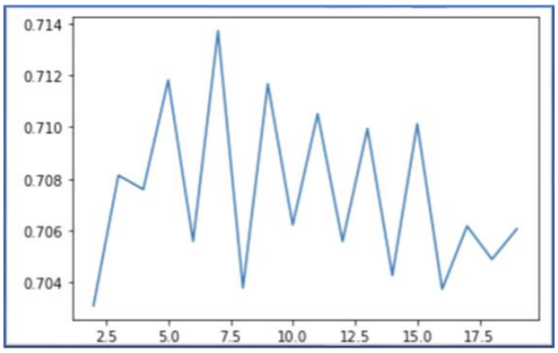
```
#PRACTICAL-6
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB

dft = pd.read_excel('default of credit card clients.xls')
df=dft.rename(columns=dft.iloc[0]).drop(dft.index[0])
df.head()
df['default payment next month'].unique()
y = df['default payment next month']
y=y.astype('int')
y.head()

X=df.drop('default payment next month',axis=1,inplace=False)
X.head()

X_train,X_test,y_train,y_test = train_test_split(X,y)
X_train.head()

f1=[]
n_neighbours = [i for i in range(2,20)]
for i in range(2,20):
  neigh = KNeighborsClassifier(n_neighbors=i,algorithm='auto')
  neigh.fit(X_train, y_train)
  y_pred = neigh.predict(X_test)
  f1.append(f1_score(y_test,y_pred,average='weighted'))
  print(f"The fi score when neighbours is {i} is {f1_score(y_test,y_pred,average='weighted')}
")

plt.plot(n_neighbours,f1)
```

```
svm_clf = LinearSVC()
svm_clf.fit(X_train,y_train)
y_pred = svm_clf.predict(X_test)
svm_clf_f1 = f1_score(y_test,y_pred,average='weighted') print(svm_clf_f1)

gnb = GaussianNB()
gnb.fit(X_train,y_train)
y_pred = gnb.predict(X_test)
gnb_f1 = f1_score(y_test,y_pred,average='weighted')
print(gnb_f1)
```

OUTPUT:

```
The fi score when neighbours is 2 is 0.7031004550353422
The fi score when neighbours is 3 is 0.7081389347726782
The fi score when neighbours is 4 is 0.7075921193966088
The fi score when neighbours is 5 is 0.7118188181744632
The fi score when neighbours is 6 is 0.7055771684729621
The fi score when neighbours is 7 is 0.7137206214458904
The fi score when neighbours is 8 is 0.7037809772332481
The fi score when neighbours is 9 is 0.7116757135140046
The fi score when neighbours is 10 is 0.7062109207254927
The fi score when neighbours is 11 is 0.7105174178177153
The fi score when neighbours is 12 is 0.705569144442591
The fi score when neighbours is 13 is 0.7099455471261585
The fi score when neighbours is 14 is 0.7042637135208477
The fi score when neighbours is 15 is 0.7101312550503249
The fi score when neighbours is 16 is 0.7037339276187846
The fi score when neighbours is 17 is 0.7061655667308234
The fi score when neighbours is 18 is 0.7048826240048939
The fi score when neighbours is 19 is 0.7060573001158837
```

K nearest neighbour



output KNN graph plot

```
svm_clf = LinearSVC()
svm_clf.fit(X_train,y_train)
y_pred = svm_clf.predict(X_test)
svm_clf_f1 = f1_score(y_test,y_pred,average='weighted')
print(svm_clf_f1)

0.679288590876176
```

Linear SVC implementation

```
print(gnb_f1)

0.36591782267034734
```

Gaussian Naïve bayes implementation

PRACTICAL-7

AIM:

Use K-Means Clustering and Hierarchical Clustering algorithm for following datasets

CODE:

```
#PRACTICAL-7
import pandas as pd
from sklearn.cluster import KMeans
from pandas.core.common import random_state
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import numpy as np
from scipy.cluster.hierarchy import dendrogram

df=pd.read_csv('data.csv')
df.head()
df.drop('diagnosis', inplace=True, axis=1)
df.drop('Unnamed: 32', inplace=True, axis=1)
df.drop('id', inplace=True, axis=1)
kmeans = KMeans(n_clusters=2, random_state=0).fit(df)
kmeans.cluster_centers_
kmeans.predict([[1.93799237e+01, 2.16945802e+01, 1.28231298e+02, 1.18592977e+03,
 1.01294580e-01, 1.48612977e-01, 1.76939466e-01, 1.00698779e-01,
 1.91539695e-01, 6.06029008e-02, 7.42803817e-01, 1.22253817e+00,
 5.25058015e+00, 9.56781679e+01, 6.59868702e-03, 3.21766947e-02,
 4.24197710e-02, 1.56739847e-02, 2.03039695e-02, 3.95338931e-03,
 2.37094656e+01, 2.89126718e+01, 1.58496183e+02, 1.75302290e+03,
 1.40424733e-01, 3.57757710e-01, 4.49306107e-01, 1.92431069e-01,
 3.11881679e-01, 8.61654962e-02]])

WCSS=[]
for i in range(2,11):
 kmeans= KMeans(n_clusters=i, random_state=42, init='k-means++').fit(df)
 WCSS.append(kmeans.inertia_);
WCSS
no_of_cluster = [i for i in range(2,11)]

plt.plot(no_of_cluster, WCSS)

clustering = AgglomerativeClustering(distance_threshold=0, n_clusters=None).fit(df)
```

```
clustering.labels_

def plot_dendrogram(model, **kwargs):
 counts = np.zeros(model.children_.shape[0])
 n_samples = len(clustering.labels_)
 for i, merge in enumerate(model.children_):
current_count = 0
 for child_idx in merge:
 if child_idx < n_samples:
  current_count += 1
 else:
  current_count += counts[child_idx - n_samples]
 counts[i] = current_count
 linkage_matrix = np.column_stack(
 [model.children_, clustering.distances_, counts]
 ).astype(float)
 dendrogram(linkage_matrix, **kwargs)
plot_dendrogram(clustering, truncate_mode="level",p=3)
```

OUTPUT:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se	sym
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.40	0.006399	0.04904	0.05373	0.01587	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308	0.01860	0.01340	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04006	0.03832	0.02058	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.23	0.009110	0.07458	0.05661	0.01867	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461	0.05688	0.01885	

Glimpses of Brest cancer dataset

```
array([1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 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, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0],
      dtype=int32)
```

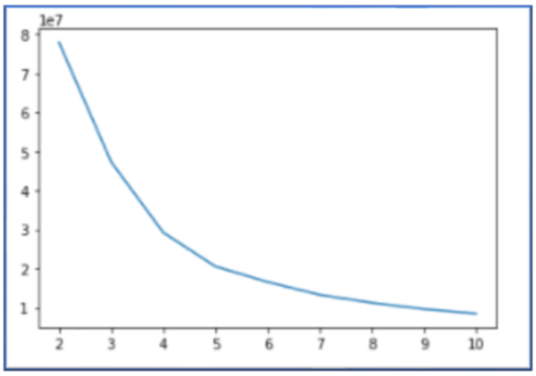
K means labels

```
array([[1.25562991e+01, 1.85703653e+01, 8.11234703e+01, 4.96061872e+02,
        9.48844977e-02, 9.10998174e-02, 6.24377642e-02, 3.34325434e-02,
        1.78057991e-01, 6.34540183e-02, 3.04190868e-01, 1.21515320e+00,
        2.15288059e+00, 2.37852922e+01, 7.17326256e-03, 2.34746895e-02,
        2.87455128e-02, 1.06363242e-02, 2.06135799e-02, 3.74750297e-03,
        1.40439018e+01, 2.47095434e+01, 9.19375114e+01, 6.19647945e+02,
        1.29959110e-01, 2.23311758e-01, 2.19214947e-01, 9.13298425e-02,
        2.83553653e-01, 8.32819406e-02],
       [1.93799237e+01, 2.16945802e+01, 1.28231298e+02, 1.18592977e+03,
        1.01294580e-01, 1.48612977e-01, 1.76939466e-01, 1.00698779e-01,
        1.91539695e-01, 6.06029008e-02, 7.42803817e-01, 1.22253817e+00,
        5.25058015e+00, 9.56781679e+01, 6.59868702e-03, 3.21766947e-02,
        4.24197710e-02, 1.56739847e-02, 2.03039695e-02, 3.95338931e-03,
        2.37094656e+01, 2.89126718e+01, 1.58496183e+02, 1.75302290e+03,
        1.40424733e-01, 3.57757710e-01, 4.49306107e-01, 1.92431069e-01,
        3.11881679e-01, 8.61654962e-02]])
```

cluster centers

```
[77943099.8782988,
 47336610.42199052,
 29226541.65197979,
 20539877.622102883,
 16562261.629261006,
 13279328.977890484,
 11226538.859168805,
 9609383.579294574,
 8471165.823852414]
```

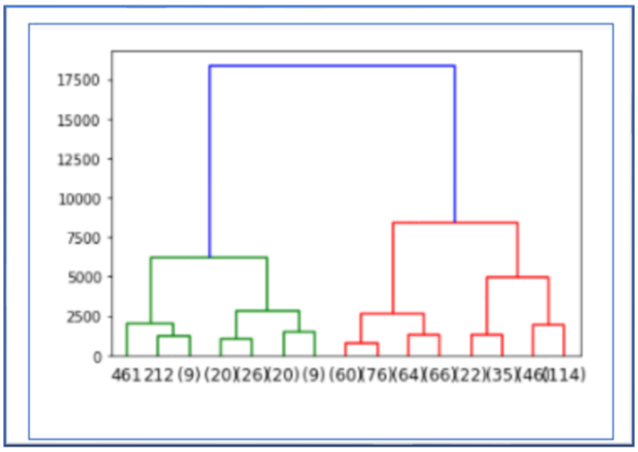
WCSS sum of squared distance



graph plot (elbow)

```
array([439, 418, 335, 333, 459, 527, 467, 423, 427, 563, 292, 323, 287,
       288, 394, 336, 431, 381, 507, 538, 510, 441, 342, 359, 337, 487,
       428, 523, 547, 318, 285, 167, 531, 483, 433, 453, 312, 417, 295,
       370, 341, 468, 170, 415, 481, 313, 488, 316, 540, 511, 213, 463,
       534, 559, 348, 566, 416, 509, 499, 564, 294, 365, 357, 363, 314,
       395, 406, 546, 520, 479, 465, 408, 289, 505, 526, 482, 366, 424,
       301, 537, 486, 489, 500, 421, 519, 389, 390, 315, 369, 501, 401,
       302, 372, 384, 449, 332, 377, 545, 249, 432, 182, 391, 548, 411,
       557, 565, 567, 282, 515, 529, 450, 385, 320, 512, 215, 469, 399,
       402, 325, 386, 521, 455, 495, 556, 407, 560, 517, 327, 360, 194,
       493, 297, 329, 309, 293, 321, 549, 273, 259, 207, 233, 203, 442,
       518, 473, 434, 286, 353, 328, 343, 543, 163, 460, 464, 443, 456,
       387, 305, 444, 475, 506, 419, 291, 496, 383, 497, 354, 535, 176,
       435, 562, 470, 525, 466, 516, 345, 346, 290, 400, 227, 420, 388,
       403, 472, 351, 192, 447, 536, 396, 438, 422, 508, 143, 355, 304,
       558, 334, 409, 356, 221, 374, 146, 255, 551, 232, 513, 392, 436,
       197, 445, 382, 462, 429, 491, 347, 373, 454, 193, 405, 299, 522,
       404, 528, 364, 430, 338, 530, 504, 190, 371, 361, 330, 310, 413,
       156, 226, 257, 367, 231, 494, 262, 555, 502, 410, 498, 452, 553,
       425, 490, 554, 331, 476, 296, 426, 281, 253, 115, 306, 514, 179,
       349, 145, 376, 172, 165, 303, 378, 173, 542, 181, 208, 204, 216,
       248, 550, 352, 324, 368, 448, 260, 256, 340, 480, 457, 267, 239,
       103, 568, 220, 166, 283, 150, 127, 541, 393, 533, 243, 152, 266,
       244, 175, 147, 234, 322,  82, 485, 107, 379, 116, 164, 561, 380,
       458, 437,  96, 319, 229, 272, 246, 183, 247, 212, 539, 269, 268,
       129, 492, 414, 412, 237, 307, 317, 263, 471, 532, 277, 308, 161,
       184,  81, 191, 326, 552, 235, 524, 477, 280, 274, 136, 210, 544,
       478,  57, 461, 254, 177, 242, 121, 228, 311, 503, 218, 168, 300,
       270, 250, 188, 240, 264, 271, 440, 151, 375, 275, 474,  87, 154,
       217, 265, 284, 258, 108, 451, 209, 261, 134, 201, 484, 200, 211,
       144, 155, 189, 251, 446, 128,  72, 205, 230, 196,  40, 104, 238,
        95,  63, 245, 153, 339, 114, 362, 206, 358, 225, 397, 236, 350,
        71, 199, 122, 180, 178, 279, 222, 252, 149, 126, 398,  89,  76,
       142, 125, 133, 187,  99,  47, 186, 344, 202,  60, 110, 102,  90,
       195, 101, 162, 241, 159,  53, 135,  62, 124, 169,  88, 120, 130,
       174,  43, 140,  93,  74, 185,  64, 223, 224, 118, 160, 111, 139,
        66, 278,  44,  21,  91, 117,  67, 100,  92,  75,  31,  84, 198,
        51, 219, 298,  97, 214,  98, 113, 132, 171,  48, 123,  23, 158,
       157, 138, 109,  45,  80,  54, 141, 148, 112,  69, 276,  61, 106,
       137,  55,  83,  94,  26,  77,  86,  37,  58, 119,  30,  35,  46,
       131,  50,  28,  59,  56,  68,  29,  22, 105,  41,  70,  78,  73,
        79,  36,  17,  49,  52,  10,  27,  65,  39,  25,  13,  85,  33,
        42,  34,  32,  24,  12,  38,  16,  19,  15,  20,  11,  14,  18,
         7,   9,   4,   8,   6,   3,   5,   2,   1,   0])
```

Hierachical cluster label



Dendrogram

PRACTICAL-8

AIM:

Implement following using Tensorflow:
Constants, Variables, Placeholder, and operations, creating Graph and executing graph.

CODE:

```
import tensorflow as tf
print("TensorFlow version:", tf._version_)
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

OUTPUT:

```
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2920 - accuracy: 0.9149
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1416 - accuracy: 0.9573
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1072 - accuracy: 0.9672
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0883 - accuracy: 0.9724
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0742 - accuracy: 0.9768
<keras.callbacks.History at 0x7f296cc5d410>
```

```
313/313 - 0s - loss: 0.0737 - accuracy: 0.9765 - 490ms/epoch - 2ms/step
[0.07372540980577469, 0.9764999747276306]
```

PRACTICAL-9

AIM:

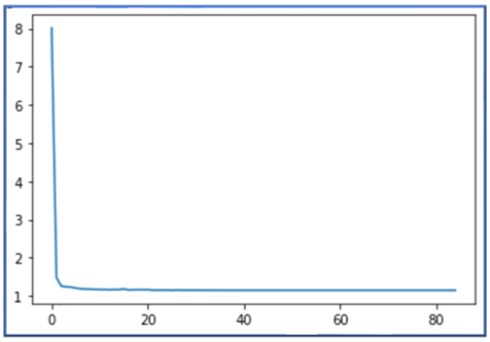
Implement the Multi-Layer Perceptron from scratch with at least 3 layers for a classification or a regression problem of your choice, implement Backpropogation and observe Underfitting, Overfitting and Regularization.

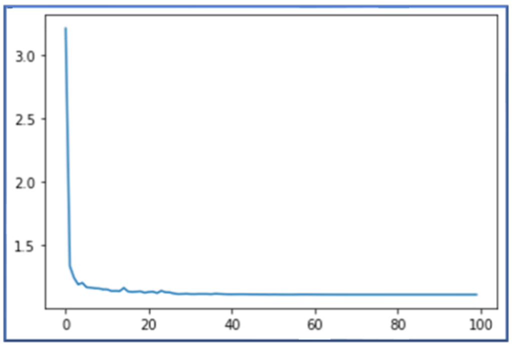
CODE:

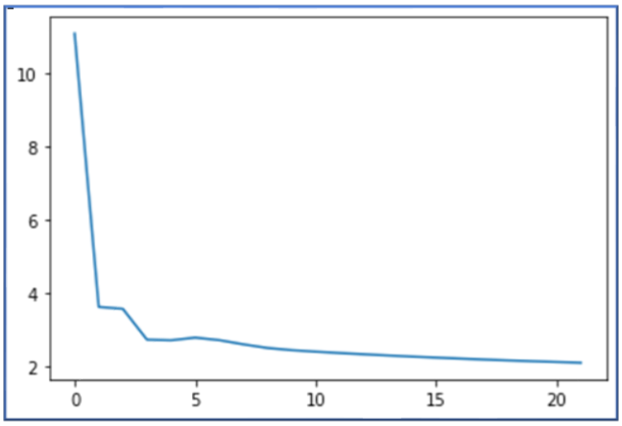
```
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
%cd MyDrive/Colab\ Notebooks
%ls
%cd
%cd /gdrive/MyDrive/Datasets
%ls
import pandas as pd
df = pd.read_csv('/content/wineQT.csv')
df.head(5)
df.drop('Id', axis=1, inplace=True)
df.head()
from sklearn.neural_network import MLPClassifier
y = df['quality']
y.head()
X = df.drop('quality', axis=1, inplace=False)
X.head()
X.describe()
model = MLPClassifier(solver='sgd', hidden_layer_sizes=(16, 8), random_state=1,
learning_rate_init=0.005, learning_rate='adaptive', verbose=True, validation_fraction=0.1,
early_stopping=True)
model.fit(X,y)
model2 = MLPClassifier(solver='sgd', hidden_layer_sizes=(24, 8), random_state=1,
learning_rate_init=0.005, learning_rate='adaptive', verbose=True, validation_fraction=0.1,
early_stopping=True)
model2.fit(X,y)
import matplotlib.pyplot as plt
plt.plot(model.loss_curve_)
plt.plot(model2.loss_curve_)
```

```
model3 = MLPClassifier(solver='sgd', hidden_layer_sizes=(16, 12), random_state=1, learning_rate_init=0.001, learning_rate='invscaling', verbose=True, validation_fraction=0.1, early_stopping=True)
model3.fit(X,y)
plt.plot(model3.loss_curve_)
model3.score(X,y)
import pickle
filename = 'finalized_sklearn_classification_model.sav'
pickle.dump(model3, open(filename, 'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.score(X,y)
```

OUTPUT:







The loss curves

PRACTICAL-10

AIM:

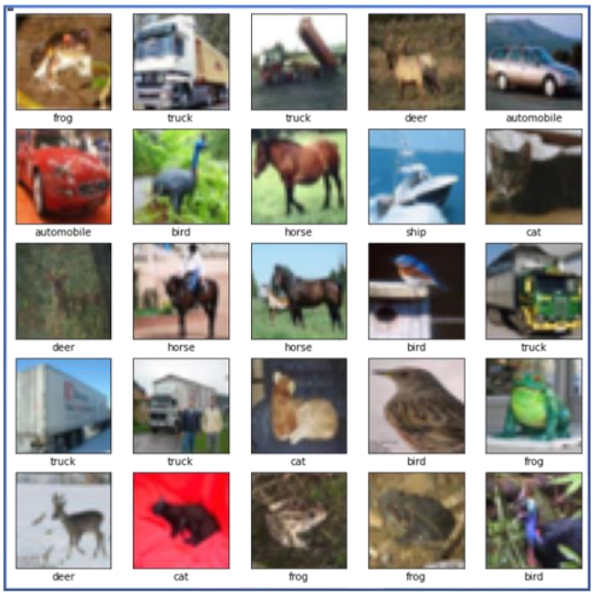
Implement a Convolutional Neural Network (CNN) using Keras library for a face classification problem. Create dataset of faces of your 5 friends. Also use data augmentation technique to increase dataset.

CODE:

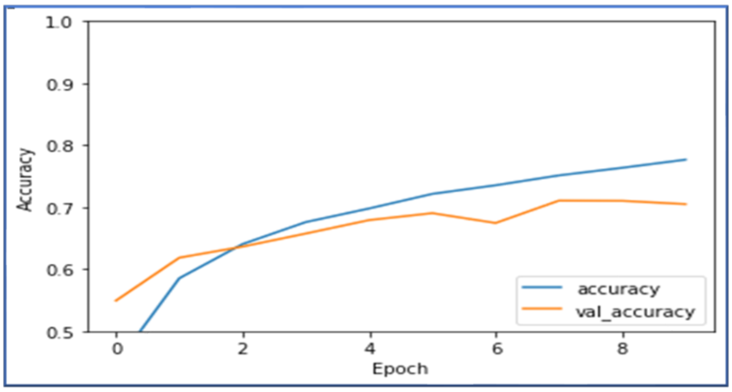
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(test_acc)
```

OUTPUT:



Glimpses of the images used



PRACTICAL-11

AIM:

Train a Reinforcement Learning Agent for the Multi-Armed Bandit Problem and visualize the results using matplotlib or seaborn libraries in Python. Consider at least 15 arms (n=15).

CODE:

```
!pip install tf-agents
import abc
import numpy as np
import tensorflow as tf
from tf_agents.agents import tf_agent
from tf_agents.drivers import driver
from tf_agents.environments import py_environment
from tf_agents.environments import tf_environment
from tf_agents.environments import tf_py_environment
from tf_agents.policies import tf_policy
from tf_agents.specs import array_spec
from tf_agents.specs import tensor_spec
from tf_agents.trajectories import time_step as ts
from tf_agents.trajectories import trajectory
from tf_agents.trajectories import policy_step
nest = tf.nest
class BanditPyEnvironment(py_environment.PyEnvironment):
  def _init_(self, observation_spec, action_spec):
    self._observation_spec = observation_spec
    self._action_spec = action_spec
    super(BanditPyEnvironment, self)._init_()
  # Helper functions.
  def action_spec(self):
    return self._action_spec
  def observation_spec(self):
    return self._observation_spec
  def _empty_observation(self):
    return tf.nest.map_structure(lambda x: np.zeros(x.shape, x.dtype),
                              self.observation_spec())
  # These two functions below should not be overridden by subclasses.
  def _reset(self):
    """Returns a time step containing an observation."""
```

```
    return ts.restart(self._observe(), batch_size=self.batch_size)
  def _step(self, action):
    """Returns a time step containing the reward for the action taken."""
    reward = self._apply_action(action)
    return ts.termination(self._observe(), reward)
  # These two functions below are to be implemented in subclasses.
  @abc.abstractmethod
  def _observe(self):
    """Returns an observation."""
  @abc.abstractmethod
  def _apply_action(self, action):
    """Applies `action` to the Environment and returns the corresponding reward.
    """
class SimplePyEnvironment(BanditPyEnvironment):
  def _init_(self):
    action_spec = array_spec.BoundedArraySpec(
        shape=(), dtype=np.int32, minimum=0, maximum=2, name='action')
    observation_spec = array_spec.BoundedArraySpec(
        shape=(1,), dtype=np.int32, minimum=-2, maximum=2, name='observation')
    super(SimplePyEnvironment, self)._init_(observation_spec, action_spec)
  def _observe(self):
    self._observation = np.random.randint(-2, 3, (1,), dtype='int32')
    return self._observation
  def _apply_action(self, action):
    return action * self._observation
environment = SimplePyEnvironment()
observation = environment.reset().observation
print("observation: %d" % observation)
action = 2 #@param
print("action: %d" % action)
reward = environment.step(action).reward
print("reward: %f" % reward)
tf_environment = tf_py_environment.TFPyEnvironment(environment)
class SignPolicy(tf_policy.TFPolicy):
  def _init_(self):
    observation_spec = tensor_spec.BoundedTensorSpec(
        shape=(1,), dtype=tf.int32, minimum=-2, maximum=2)
    time_step_spec = ts.time_step_spec(observation_spec)
    action_spec = tensor_spec.BoundedTensorSpec(
        shape=(), dtype=tf.int32, minimum=0, maximum=2)
    super(SignPolicy, self)._init_(time_step_spec=time_step_spec,
                                    action_spec=action_spec)
  def _distribution(self, time_step):
    pass
```

```
  def _variables(self):
    return ()
  def _action(self, time_step, policy_state, seed):
    observation_sign = tf.cast(tf.sign(time_step.observation[0]), dtype=tf.int32)
    action = observation_sign + 1
    return policy_step.PolicyStep(action, policy_state)
sign_policy = SignPolicy()
current_time_step = tf_environment.reset()
print('Observation:')
print (current_time_step.observation)
action = sign_policy.action(current_time_step).action
print('Action:')
print (action)
reward = tf_environment.step(action).reward
print('Reward:')
print(reward)
step = tf_environment.reset()
action = 1
next_step = tf_environment.step(action)
reward = next_step.reward
next_observation = next_step.observation
print("Reward: ")
print(reward)
print("Next observation:")
print(next_observation)
class TwoWayPyEnvironment(BanditPyEnvironment):
  def _init_(self):
    action_spec = array_spec.BoundedArraySpec(
        shape=(), dtype=np.int32, minimum=0, maximum=2, name='action')
    observation_spec = array_spec.BoundedArraySpec(
        shape=(1,), dtype=np.int32, minimum=-2, maximum=2, name='observation')
    # Flipping the sign with probability 1/2.
    self._reward_sign = 2 * np.random.randint(2) - 1
    print("reward sign:")
    print(self._reward_sign)
    super(TwoWayPyEnvironment, self)._init_(observation_spec, action_spec)
  def _observe(self):
    self._observation = np.random.randint(-2, 3, (1,), dtype='int32')
    return self._observation
  def _apply_action(self, action):
    return self._reward_sign * action * self._observation[0]
two_way_tf_environment =
tf_py_environment.TFPyEnvironment(TwoWayPyEnvironment())
class TwoWaySignPolicy(tf_policy.TFPolicy):
  def _init_(self, situation):
```

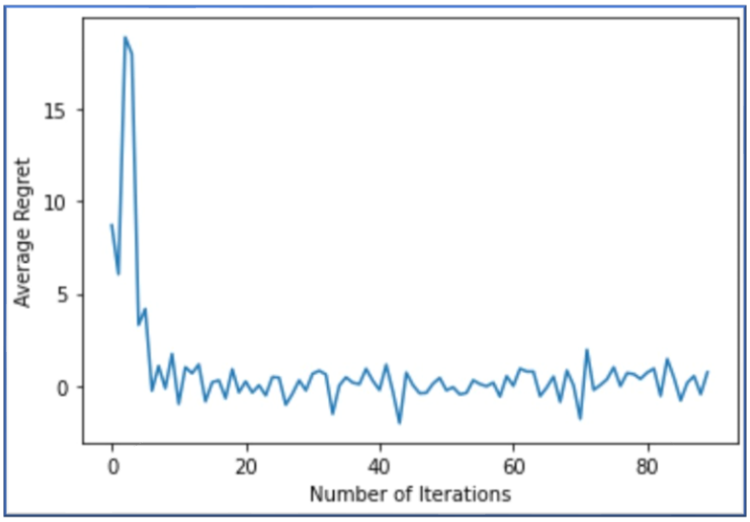
```
    observation_spec = tensor_spec.BoundedTensorSpec(
        shape=(1,), dtype=tf.int32, minimum=-2, maximum=2)
    action_spec = tensor_spec.BoundedTensorSpec(
        shape=(), dtype=tf.int32, minimum=0, maximum=2)
    time_step_spec = ts.time_step_spec(observation_spec)
    self._situation = situation
    super(TwoWaySignPolicy, self)._init_(time_step_spec=time_step_spec,
                                          action_spec=action_spec)
  def _distribution(self, time_step):
    pass
  def _variables(self):
    return [self._situation]
  def _action(self, time_step, policy_state, seed):
    sign = tf.cast(tf.sign(time_step.observation[0, 0]), dtype=tf.int32)
    def case_unknown_fn():
      # Choose 1 so that we get information on the sign.
      return tf.constant(1, shape=(1,))
    # Choose 0 or 2, depending on the situation and the sign of the observation.
    def case_normal_fn():
      return tf.constant(sign + 1, shape=(1,))
    def case_flipped_fn():
      return tf.constant(1 - sign, shape=(1,))
    cases = [(tf.equal(self._situation, 0), case_unknown_fn),
             (tf.equal(self._situation, 1), case_normal_fn),
             (tf.equal(self._situation, 2), case_flipped_fn)]
    action = tf.case(cases, exclusive=True)
    return policy_step.PolicyStep(action, policy_state)
class SignAgent(tf_agent.TFAgent):
  def _init_(self):
    self._situation = tf.Variable(0, dtype=tf.int32)
    policy = TwoWaySignPolicy(self._situation)
    time_step_spec = policy.time_step_spec
    action_spec = policy.action_spec
    super(SignAgent, self)._init_(time_step_spec=time_step_spec,
                                  action_spec=action_spec,
                                  policy=policy,
                                  collect_policy=policy,
                                  train_sequence_length=None)
  def _initialize(self):
    return tf.compat.v1.variables_initializer(self.variables)
  def _train(self, experience, weights=None):
    observation = experience.observation
    action = experience.action
    reward = experience.reward
```

```
    # We only need to change the value of the situation variable if it is
    # unknown (0) right now, and we can infer the situation only if the
    # observation is not 0.
    needs_action = tf.logical_and(tf.equal(self._situation, 0),
                                  tf.not_equal(reward, 0))

    def new_situation_fn():
      """This returns either 1 or 2, depending on the signs."""
      return (3 - tf.sign(tf.cast(observation[0, 0, 0], dtype=tf.int32) *
                          tf.cast(action[0, 0], dtype=tf.int32) *
                          tf.cast(reward[0, 0], dtype=tf.int32))) / 2
    new_situation = tf.cond(needs_action,
                            new_situation_fn,
                            lambda: self._situation)
    new_situation = tf.cast(new_situation, tf.int32)
    tf.compat.v1.assign(self._situation, new_situation)
    return tf_agent.LossInfo((), ())
sign_agent = SignAgent()
# We need to add another dimension here because the agent expects the
# trajectory of shape [batch_size, time, ...], but in this tutorial we assume
# that both batch size and time are 1. Hence all the expand_dims.
def trajectory_for_bandit(initial_step, action_step, final_step):
  return trajectory.Trajectory(observation=tf.expand_dims(initial_step.observation, 0),
                               action=tf.expand_dims(action_step.action, 0),
                               policy_info=action_step.info,
                               reward=tf.expand_dims(final_step.reward, 0),
                               discount=tf.expand_dims(final_step.discount, 0),
                               step_type=tf.expand_dims(initial_step.step_type, 0),
                               next_step_type=tf.expand_dims(final_step.step_type, 0))
step = two_way_tf_environment.reset()
for _ in range(10):
  action_step = sign_agent.collect_policy.action(step)
  next_step = two_way_tf_environment.step(action_step.action)
  experience = trajectory_for_bandit(step, action_step, next_step)
  print(experience)
  sign_agent.train(experience)
  step = next_step
# Imports for example.
from tf_agents.bandits.agents import lin_ucb_agent
from tf_agents.bandits.environments import stationary_stochastic_py_environment as sspe
from tf_agents.bandits.metrics import tf_metrics
from tf_agents.drivers import dynamic_step_driver
from tf_agents.replay_buffers import tf_uniform_replay_buffer
import matplotlib.pyplot as plt
batch_size = 2 # @param
```

```
arm0_param = [-3, 0, 1, -2] # @param
arm1_param = [1, -2, 3, 0] # @param
arm2_param = [0, 0, 1, 1] # @param
def context_sampling_fn(batch_size):
  """Contexts from [-10, 10]^4."""
  def _context_sampling_fn():
    return np.random.randint(-10, 10, [batch_size, 4]).astype(np.float32)
  return _context_sampling_fn
class LinearNormalReward(object):
  """A class that acts as linear reward function when called."""
  def _init_(self, theta, sigma):
    self.theta = theta
    self.sigma = sigma
  def _call_(self, x):
    mu = np.dot(x, self.theta)
    return np.random.normal(mu, self.sigma)
arm0_reward_fn = LinearNormalReward(arm0_param, 1)
arm1_reward_fn = LinearNormalReward(arm1_param, 1)
arm2_reward_fn = LinearNormalReward(arm2_param, 1)
environment = tf_py_environment.TFPyEnvironment(
    sspe.StationaryStochasticPyEnvironment(
        context_sampling_fn(batch_size),
        [arm0_reward_fn, arm1_reward_fn, arm2_reward_fn],
        batch_size=batch_size))
observation_spec = tensor_spec.TensorSpec([4], tf.float32)
time_step_spec = ts.time_step_spec(observation_spec)
action_spec = tensor_spec.BoundedTensorSpec(
    dtype=tf.int32, shape=(), minimum=0, maximum=2)
agent = lin_ucb_agent.LinearUCBAgent(time_step_spec=time_step_spec,
                                     action_spec=action_spec)
def compute_optimal_reward(observation):
  expected_reward_for_arms = [
      tf.linalg.matvec(observation, tf.cast(arm0_param, dtype=tf.float32)),
      tf.linalg.matvec(observation, tf.cast(arm1_param, dtype=tf.float32)),
      tf.linalg.matvec(observation, tf.cast(arm2_param, dtype=tf.float32))]
  optimal_action_reward = tf.reduce_max(expected_reward_for_arms, axis=0)
  return optimal_action_reward
regret_metric = tf_metrics.RegretMetric(compute_optimal_reward)
num_iterations = 90 # @param
steps_per_loop = 1 # @param
replay_buffer = tf_uniform_replay_buffer.TFUniformReplayBuffer(
    data_spec=agent.policy.trajectory_spec,
    batch_size=batch_size,
    max_length=steps_per_loop)
observers = [replay_buffer.add_batch, regret_metric]
```

```
driver = dynamic_step_driver.DynamicStepDriver(
    env=environment,
    policy=agent.collect_policy,
    num_steps=steps_per_loop * batch_size,
    observers=observers)
regret_values = []
for _ in range(num_iterations):
  driver.run()
  loss_info = agent.train(replay_buffer.gather_all())
  replay_buffer.clear()
  regret_values.append(regret_metric.result())

plt.plot(regret_values)
plt.ylabel('Average Regret')
plt.xlabel('Number of Iterations')
```

OUTPUT:



PRACTICAL-12

AIM:

Implement a Deep Learning Algorithm/Method to Predict stock prices based on past price variation.

CODE:

```
pip install yahoo-fin
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from yahoo_fin import stock_info as si
from collections import deque
import os
import numpy as np
import pandas as pd
import random
# set seed, so we can get the same results after rerunning several times
np.random.seed(314)
tf.random.set_seed(314)
random.seed(314)
import os
import time
from tensorflow.keras.layers import LSTM
# Window size or the sequence length
N_STEPS = 50
# Lookup step, 1 is the next day
LOOKUP_STEP = 15
# whether to scale feature columns & output price as well
SCALE = True
scale_str = f"sc-{int(SCALE)}"
# whether to shuffle the dataset
SHUFFLE = True
shuffle_str = f"sh-{int(SHUFFLE)}"
# whether to split the training/testing set by date
SPLIT_BY_DATE = False
split_by_date_str = f"sbd-{int(SPLIT_BY_DATE)}"
```

```
# test ratio size, 0.2 is 20%
TEST_SIZE = 0.2
# features to use
FEATURE_COLUMNS = ["adjclose", "volume", "open", "high", "low"]
# date now
date_now = time.strftime("%Y-%m-%d")
### model parameters
N_LAYERS = 2
# LSTM cell
CELL = LSTM
# 256 LSTM neurons
UNITS = 256
# 40% dropout
DROPOUT = 0.4
# whether to use bidirectional RNNs
BIDIRECTIONAL = False
### training parameters
# mean absolute error loss
# LOSS = "mae"
# huber loss
LOSS = "huber_loss"
OPTIMIZER = "adam"
BATCH_SIZE = 64
EPOCHS = 500
# Amazon stock market
ticker = "AMZN"
ticker_data_filename = os.path.join("data", f"{ticker}_{date_now}.csv")
# model name to save, making it as unique as possible based on parameters
model_name = f"{date_now}_{ticker}-{shuffle_str}-{scale_str}-{split_by_date_str}-\
{LOSS}-{OPTIMIZER}-{CELL.__name__}-seq-{N_STEPS}-step-{LOOKUP_STEP}-
layers-{N_LAYERS}-units-{UNITS}"
if BIDIRECTIONAL:
    model_name += "-b"
def shuffle_in_unison(a, b):
    # shuffle two arrays in the same way
    state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(state)
    np.random.shuffle(b)
def load_data(ticker, n_steps=50, scale=True, shuffle=True, lookup_step=1,
split_by_date=True,
                test_size=0.2, feature_columns=['adjclose', 'volume', 'open', 'high', 'low']):

    # see if ticker is already a loaded stock from yahoo finance
    if isinstance(ticker, str):
```

```
        # load it from yahoo_fin library
        df = si.get_data(ticker)
    elif isinstance(ticker, pd.DataFrame):
        # already loaded, use it directly
        df = ticker
    else:
        raise TypeError("ticker can be either a str or a `pd.DataFrame` instances")

    # this will contain all the elements we want to return from this function
    result = {}
    # we will also return the original dataframe itself
    result['df'] = df.copy()
    # make sure that the passed feature_columns exist in the dataframe
    for col in feature_columns:
        assert col in df.columns, f"'{col}' does not exist in the dataframe."
    # add date as a column
    if "date" not in df.columns:
        df["date"] = df.index
    if scale:
        column_scaler = {}
        # scale the data (prices) from 0 to 1
        for column in feature_columns:
            scaler = preprocessing.MinMaxScaler()
            df[column] = scaler.fit_transform(np.expand_dims(df[column].values, axis=1))
            column_scaler[column] = scaler
        # add the MinMaxScaler instances to the result returned
        result["column_scaler"] = column_scaler
    # add the target column (label) by shifting by `lookup_step`
    df['future'] = df['adjclose'].shift(-lookup_step)
    # last `lookup_step` columns contains NaN in future column
    # get them before droping NaNs
    last_sequence = np.array(df[feature_columns].tail(lookup_step))
    # drop NaNs
    df.dropna(inplace=True)
    sequence_data = []
    sequences = deque(maxlen=n_steps)
    for entry, target in zip(df[feature_columns + ["date"]].values, df['future'].values):
        sequences.append(entry)
        if len(sequences) == n_steps:
            sequence_data.append([np.array(sequences), target])
    # get the last sequence by appending the last `n_step` sequence with `lookup_step` 
sequence
    # for instance, if n_steps=50 and lookup_step=10, last_sequence should be of 60 (that is 
50+10) length
```

```
    # this last_sequence will be used to predict future stock prices that are not available in the
dataset
    last_sequence = list([s[:len(feature_columns)] for s in sequences]) + list(last_sequence)
    last_sequence = np.array(last_sequence).astype(np.float32)
    # add to result
    result['last_sequence'] = last_sequence
    # construct the X's and y's
    X, y = [], []
    for seq, target in sequence_data:
        X.append(seq)
        y.append(target)
    # convert to numpy arrays
    X = np.array(X)
    y = np.array(y)
    if split_by_date:
        # split the dataset into training & testing sets by date (not randomly splitting)
        train_samples = int((1 - test_size) * len(X))
        result["X_train"] = X[:train_samples]
        result["y_train"] = y[:train_samples]
        result["X_test"]  = X[train_samples:]
        result["y_test"]  = y[train_samples:]
        if shuffle:
            # shuffle the datasets for training (if shuffle parameter is set)
            shuffle_in_unison(result["X_train"], result["y_train"])
            shuffle_in_unison(result["X_test"], result["y_test"])
    else:
        # split the dataset randomly
        result["X_train"], result["X_test"], result["y_train"], result["y_test"] = train_test_split(X,
y, test_size=test_size, shuffle=shuffle)
    # get the list of test set dates
    dates = result["X_test"][:, -1, -1]
    # retrieve test features from the original dataframe
    result["test_df"] = result["df"].loc[dates]
    # remove duplicated dates in the testing dataframe
    result["test_df"] = result["test_df"][~result["test_df"].index.duplicated(keep='first')]
    # remove dates from the training/testing sets & convert to float32
    result["X_train"] = result["X_train"][:, :, :len(feature_columns)].astype(np.float32)
    result["X_test"] = result["X_test"][:, :, :len(feature_columns)].astype(np.float32)
    return result
def create_model(sequence_length, n_features, units=256, cell=LSTM, n_layers=2,
dropout=0.3,
                loss="mean_absolute_error", optimizer="rmsprop", bidirectional=False):
    model = Sequential()
    for i in range(n_layers):
        if i == 0:
```

```
            # first layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True),
batch_input_shape=(None, sequence_length, n_features)))
            else:
                model.add(cell(units, return_sequences=True, batch_input_shape=(None,
sequence_length, n_features)))
        elif i == n_layers - 1:
            # last layer
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=False)))
            else:
                model.add(cell(units, return_sequences=False))
        else:
            # hidden layers
            if bidirectional:
                model.add(Bidirectional(cell(units, return_sequences=True)))
            else:
                model.add(cell(units, return_sequences=True))
        # add dropout after each layer
        model.add(Dropout(dropout))
    model.add(Dense(1, activation="linear"))
    model.compile(loss=loss, metrics=["mean_absolute_error"], optimizer=optimizer)
    return model
# create these folders if they does not exist
if not os.path.isdir("results"):
    os.mkdir("results")
if not os.path.isdir("logs"):
    os.mkdir("logs")
if not os.path.isdir("data"):
    os.mkdir("data")
# load the data
data = load_data(ticker, N_STEPS, scale=SCALE, split_by_date=SPLIT_BY_DATE,
                shuffle=SHUFFLE, lookup_step=LOOKUP_STEP, test_size=TEST_SIZE,
                feature_columns=FEATURE_COLUMNS)
# save the dataframe
data["df"].to_csv(ticker_data_filename)
# construct the model
model = create_model(N_STEPS, len(FEATURE_COLUMNS), loss=LOSS, units=UNITS,
cell=CELL, n_layers=N_LAYERS,
                    dropout=DROPOUT, optimizer=OPTIMIZER,
bidirectional=BIDIRECTIONAL)
# some tensorflow callbacks
checkpointer = ModelCheckpoint(os.path.join("results", model_name + ".h5"),
save_weights_only=True, save_best_only=True, verbose=1)
```

```
tensorboard = TensorBoard(log_dir=os.path.join("logs", model_name))
# train the model and save the weights whenever we see
# a new optimal model using ModelCheckpoint
history = model.fit(data["X_train"], data["y_train"],
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    validation_data=(data["X_test"], data["y_test"]),
                    callbacks=[checkpointer, tensorboard],
                    verbose=1)
import matplotlib.pyplot as plt
def plot_graph(test_df):
    """
    This function plots true close price along with predicted close price
    with blue and red colors respectively
    """
    plt.plot(test_df[f'true_adjclose_{LOOKUP_STEP}'], c='b')
    plt.plot(test_df[f'adjclose_{LOOKUP_STEP}'], c='r')
    plt.xlabel("Days")
    plt.ylabel("Price")
    plt.legend(["Actual Price", "Predicted Price"])
    plt.show()
def get_final_df(model, data):
    """
    This function takes the `model` and `data` dict to
    construct a final dataframe that includes the features along
    with true and predicted prices of the testing dataset
    """
    # if predicted future price is higher than the current,
    # then calculate the true future price minus the current price, to get the buy profit
    buy_profit  = lambda current, pred_future, true_future: true_future - current if pred_future
> current else 0
    # if the predicted future price is lower than the current price,
    # then subtract the true future price from the current price
    sell_profit = lambda current, pred_future, true_future: current - true_future if pred_future <
current else 0
    X_test = data["X_test"]
    y_test = data["y_test"]
    # perform prediction and get prices
    y_pred = model.predict(X_test)
    if SCALE:
        y_test =
np.squeeze(data["column_scaler"]["adjclose"].inverse_transform(np.expand_dims(y_test,
axis=0)))
        y_pred = np.squeeze(data["column_scaler"]["adjclose"].inverse_transform(y_pred))
    test_df = data["test_df"]
```

```
    # add predicted future prices to the dataframe
    test_df[f"adjclose_{LOOKUP_STEP}"] = y_pred
    # add true future prices to the dataframe
    test_df[f"true_adjclose_{LOOKUP_STEP}"] = y_test
    # sort the dataframe by date
    test_df.sort_index(inplace=True)
    final_df = test_df
    # add the buy profit column
    final_df["buy_profit"] = list(map(buy_profit,
                                    final_df["adjclose"],
                                    final_df[f"adjclose_{LOOKUP_STEP}"],
                                    final_df[f"true_adjclose_{LOOKUP_STEP}"])
                                    # since we don't have profit for last sequence, add 0's
                                    )
    # add the sell profit column
    final_df["sell_profit"] = list(map(sell_profit,
                                    final_df["adjclose"],
                                    final_df[f"adjclose_{LOOKUP_STEP}"],
                                    final_df[f"true_adjclose_{LOOKUP_STEP}"])
                                    # since we don't have profit for last sequence, add 0's
                                    )
    return final_df
def predict(model, data):
    # retrieve the last sequence from data
    last_sequence = data["last_sequence"][-N_STEPS:]
    # expand dimension
    last_sequence = np.expand_dims(last_sequence, axis=0)
    # get the prediction (scaled from 0 to 1)
    prediction = model.predict(last_sequence)
    # get the price (by inverting the scaling)
    if SCALE:
        predicted_price = data["column_scaler"]["adjclose"].inverse_transform(prediction)[0][0]
    else:
        predicted_price = prediction[0][0]
    return predicted_price
# load optimal model weights from results folder
model_path = os.path.join("results", model_name) + ".h5"
model.load_weights(model_path)
# evaluate the model
loss, mae = model.evaluate(data["X_test"], data["y_test"], verbose=0)
# calculate the mean absolute error (inverse scaling)
if SCALE:
    mean_absolute_error =
data["column_scaler"]["adjclose"].inverse_transform([[mae]])[0][0]
else:
```

```
    mean_absolute_error = mae
# get the final dataframe for the testing set
final_df = get_final_df(model, data)
# predict the future price
future_price = predict(model, data)
# we calculate the accuracy by counting the number of positive profits
accuracy_score = (len(final_df[final_df['sell_profit'] > 0]) +
len(final_df[final_df['buy_profit'] > 0])) / len(final_df)
# calculating total buy & sell profit
total_buy_profit  = final_df["buy_profit"].sum()
total_sell_profit = final_df["sell_profit"].sum()
# total profit by adding sell & buy together
total_profit = total_buy_profit + total_sell_profit
# dividing total profit by number of testing samples (number of trades)
profit_per_trade = total_profit / len(final_df)
# printing metrics
print(f"Future price after {LOOKUP_STEP} days is {future_price:.2f}$")
print(f"{LOSS} loss:", loss)
print("Mean Absolute Error:", mean_absolute_error)
print("Accuracy score:", accuracy_score)
print("Total buy profit:", total_buy_profit)
print("Total sell profit:", total_sell_profit)
print("Total profit:", total_profit)
print("Profit per trade:", profit_per_trade)
# plot true/pred prices graph
plot_graph(final_df)
final_df.head(20)
final_df.tail(20)
# save the final dataframe to csv-results folder
csv_results_folder = "csv-results"
if not os.path.isdir(csv_results_folder):
    os.mkdir(csv_results_folder)
csv_filename = os.path.join(csv_results_folder, model_name + ".csv")
final_df.to_csv(csv_filename)
```

OUTPUT:

	open	high	low	close	adjclose	volume	ticker	adjclose_15	true_adjclose_15	buy_profit	sell_profit
1997-08-07	2.250000	2.260417	2.125000	2.177083	2.177083	2034000	AMZN	5.142529	2.375000	0.197917	0.0
1997-08-18	2.052083	2.052083	1.968750	2.041667	2.041667	1784400	AMZN	6.876284	3.239583	1.197916	0.0
1997-08-21	2.135417	2.171875	2.072917	2.114583	2.114583	624000	AMZN	7.502244	3.687500	1.572917	0.0
1997-09-05	2.583333	2.666667	2.458333	2.500000	2.500000	1908000	AMZN	6.734718	4.166667	1.666667	0.0
1997-09-08	2.531250	3.020833	2.500000	3.000000	3.000000	5648400	AMZN	7.318435	4.041667	1.041667	0.0
1997-09-10	3.312500	3.328125	3.125000	3.302083	3.302083	3866400	AMZN	8.576524	4.020833	0.718750	0.0
1997-09-12	3.187500	3.697917	3.156250	3.687500	3.687500	3333600	AMZN	7.957679	4.015625	0.328125	0.0
1997-09-15	3.666667	3.677083	3.052083	3.093750	3.093750	5583600	AMZN	7.694222	4.125000	1.031250	0.0
1997-09-17	3.458333	3.500000	3.333333	3.406250	3.406250	2607600	AMZN	7.246346	4.005208	0.598958	0.0
1997-09-29	4.145833	4.187500	3.958333	4.041667	4.041667	2371200	AMZN	8.693190	3.822917	-0.218750	0.0
1997-10-06	4.000000	4.125000	3.942708	4.125000	4.125000	2028000	AMZN	9.905589	4.270833	0.145833	0.0
1997-10-17	3.614583	3.656250	3.520833	3.625000	3.625000	2534400	AMZN	8.898285	4.479167	0.854167	0.0
1997-10-20	3.677083	3.875000	3.666667	3.822917	3.822917	4912800	AMZN	8.644722	4.208333	0.385416	0.0
1997-10-21	3.958333	4.437500	3.854167	4.427083	4.427083	12096000	AMZN	9.539251	3.947917	-0.479166	0.0
1997-10-28	3.916667	5.000000	3.875000	4.947917	4.947917	11719200	AMZN	9.641577	4.416667	-0.531250	0.0
1997-10-31	5.364583	5.458333	4.968750	5.083333	5.083333	5026800	AMZN	9.026456	4.489583	-0.593750	0.0
1997-11-05	4.979167	5.119792	4.875000	4.875000	4.875000	3093600	AMZN	9.216101	4.260417	-0.614583	0.0

Output of Head()

	open	high	low	close	adjclose	volume	ticker	adjclose_15	true_adjclose_15	buy_profit	sell_profit
2021-11-02	3315.010010	3331.120117	3283.550049	3312.750000	3312.750000	2627600	AMZN	3237.732910	3580.040039	0.000000	-267.290039
2021-11-09	3515.250000	3593.770020	3501.429932	3576.229980	3576.229980	4294900	AMZN	3268.910400	3443.719971	0.000000	132.510010
2021-11-12	3485.000000	3540.729980	3447.050049	3525.149902	3525.149902	2689400	AMZN	3278.257080	3427.370117	0.000000	97.779785
2021-11-16	3539.000000	3576.500000	3525.149902	3540.699951	3540.699951	2217100	AMZN	3285.008301	3523.159912	0.000000	17.540039
2021-12-01	3545.000000	3559.879883	3441.600098	3443.719971	3443.719971	3745800	AMZN	3296.915527	3420.739990	0.000000	22.979980
2022-01-03	3351.000000	3414.070068	3323.209961	3408.090088	3408.090088	3176000	AMZN	3255.872314	2799.719971	0.000000	608.370117
2022-01-04	3408.760010	3428.000000	3326.989990	3350.439941	3350.439941	3536300	AMZN	3254.081787	2777.449951	0.000000	572.989990
2022-01-06	3269.010010	3296.000000	3238.739990	3265.080078	3265.080078	2597900	AMZN	3242.699951	2879.560059	0.000000	385.520020
2022-01-10	3211.709961	3233.229980	3126.090088	3229.719971	3229.719971	4389900	AMZN	3226.582275	3023.870117	0.000000	205.849854
2022-01-13	3305.010010	3324.429932	3221.820068	3224.280029	3224.280029	2609400	AMZN	3222.180664	3152.790039	0.000000	71.489990
2022-01-14	3203.000000	3245.000000	3196.010010	3242.760010	3242.760010	2298700	AMZN	3218.104492	3158.709961	0.000000	84.050049
2022-01-18	3182.100098	3194.689941	3153.290039	3178.350098	3178.350098	3364600	AMZN	3210.176270	3228.270020	49.919922	0.000000
2022-01-19	3175.239990	3185.000000	3125.000000	3125.979980	3125.979980	2662100	AMZN	3201.065430	3223.790039	97.810059	0.000000
2022-01-20	3135.320068	3160.000000	3027.020020	3033.350098	3033.350098	3598700	AMZN	3188.230225	3180.070068	146.719971	0.000000
2022-01-24	2780.000000	2898.899902	2707.040039	2890.879883	2890.879883	7781200	AMZN	3133.450195	3103.340088	212.460205	0.000000
2022-01-25	2844.850098	2872.000000	2762.899902	2799.719971	2799.719971	4541200	AMZN	3105.869873	3130.209961	330.489990	0.000000
2022-02-01	3000.000000	3034.159912	2952.550049	3023.870117	3023.870117	2961000	AMZN	3058.072266	2896.540039	-127.330078	0.000000

Output of Tail()

```
Future price after 15 days is 3184.45$
huber_loss loss: 0.00020447642600062948
Mean Absolute Error: 36.33039132104265
Accuracy score: 0.5753424657534246
Total buy profit: 5760.4542326927185
Total sell profit: 2591.721192836761
Total profit: 8352.17542552948
Profit per trade: 6.7301977643267366
```

Metrics

Actual vs predicted price