Lab Report

Course Code: CSE 316

Course Name: Artificial Intelligence Lab

Lab Report No: 04

Date of submission : 25-08-2023

Submitted To :

Md. Sadekur Rahman

Assistant Professor

Department of CSE

Daffodil International University

Submitted By:

Name: Rayhan Rafin

ID: 213-15-4278

Section: 60_B

Check data

Checking original data and shape:

```
# imported file and libraries
file_path = "/content/Profit Prediction.csv"

import numpy as np
import pandas as pd

df = pd.read_csv(file_path)
df.head(10)
```

	Marketing Spend	Administration	Transport	Area	Profit
0	114523.61	136897.80	471784.10	Dhaka	192261.83
1	162597.70	151377.59	443898.53	Ctg	191792.06
2	153441.51	101145.55	407934.54	Rangpur	191050.39
3	144372.41	118671.85	383199.62	Dhaka	182901.99
4	142107.34	91391.77	366168.42	Rangpur	166187.94
5	131876.90	99814.71	362861.36	Dhaka	156991.12
6	134615.46	147198.87	127716.82	Ctg	156122.51
7	130298.13	145530.06	323876.68	Rangpur	155752.60
8	120542.52	148718.95	311613.29	Dhaka	152211.77
9	123334.88	108679.17	304981.62	Ctg	149759.96

```
#get row and column of the data
df.shape
```

```
(50, 5)
```

Check null:

```
#get null values
df.isnull().sum()
```

```
Marketing Spend    0
Administration     0
Transport          0
Area               0
Profit             0
dtype: int64
```

Get information of data:

```
#gives a brief description of data
df.describe()
```

	Marketing Spend	Administration	Transport	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000
25%	39936.370000	103730.875000	129300.132500	90138.902500
50%	73051.080000	122699.795000	212716.240000	107978.190000
75%	101602.800000	144842.180000	299469.085000	139765.977500
max	165349.200000	182645.560000	471784.100000	192261.830000

```
#gives information about the data
#we can see that we have an object type data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Marketing Spend  50 non-null     float64
 1   Administration   50 non-null     float64
 2   Transport        50 non-null     float64
 3   Area             50 non-null     object
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

Dividing dataframe

Encoding object:

```
#onehotencoding the object data

areadummies = pd.get_dummies(df['Area'])
areadummies.head()
```

	Ctg	Dhaka	Rangpur
0	0	1	0
1	1	0	0
2	0	0	1
3	0	1	0
4	0	0	1

Combining dataframe:

```
#combining both dataframe to df2
df2 = pd.concat([df,areadummies],axis =1)
df2.head()
```

	Marketing Spend	Administration	Transport	Area	Profit	Ctg	Dhaka	Rangpur
0	114523.61	136897.80	471784.10	Dhaka	192261.83	0	1	0
1	162597.70	151377.59	443898.53	Ctg	191792.06	1	0	0
2	153441.51	101145.55	407934.54	Rangpur	191050.39	0	0	1
3	144372.41	118671.85	383199.62	Dhaka	182901.99	0	1	0
4	142107.34	91391.77	366168.42	Rangpur	166187.94	0	0	1

Dividing input and output:

```
# we need to seperate given data and the value we want to find
# we put given data in x and result data in y
#obj data Area is encoded with onehotencoder
#deleting obj data and result data to get pure given data

x = df2.drop(['Area','Profit'],axis=1)
# result data goes to y
y = df2['Profit']
x.head()
```

	Marketing Spend	Administration	Transport	Ctg	Dhaka	Rangpur
0	114523.61	136897.80	471784.10	0	1	0
1	162597.70	151377.59	443898.53	1	0	0
2	153441.51	101145.55	407934.54	0	0	1
3	144372.41	118671.85	383199.62	0	1	0
4	142107.34	91391.77	366168.42	0	0	1

Splitting data:

```
# splitting into train and test data
# defined the ratio of test:train as 20%:80%

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size = 0.2)
```

Show divided data:

```
#input train data
xtrain.head()
```

	Marketing Spend	Administration	Transport	Ctg	Dhaka	Rangpur
12	93863.75	127320.38	249839.44	0	0	1
31	61136.38	152701.92	88218.23	0	1	0
24	77044.01	99281.34	140574.81	0	1	0
7	130298.13	145530.06	323876.68	0	0	1
41	27892.92	84710.77	164470.71	0	0	1

```python
#output train data
ytrain.head()
```

```
12    141585.52
31     97483.56
24    108552.04
7     155752.60
41     77798.83
Name: Profit, dtype: float64
```

```python
#input test data
xtest.head()
```

	Marketing Spend	Administration	Transport	Ctg	Dhaka	Rangpur
4	142107.34	91391.77	366168.42	0	0	1
21	78389.47	153773.43	299737.29	0	1	0
36	28663.76	127056.21	201126.82	0	0	1
46	1315.46	115816.21	297114.46	0	0	1
40	28754.33	118546.05	172795.67	1	0	0

```python
#ouput test data
ytest.head()
```

```
4     166187.94
21    111313.02
36     90708.19
46     49490.75
40     78239.91
Name: Profit, dtype: float64
```

Scaling data

Scaling from 0 to 1:

```
#scaled the input data (0 to 1)

from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
mms.fit(xtrain)
xtrain_mms = mms.transform(xtrain)
xtest_mms = mms.transform(xtest)

print(xtrain_mms)
```

```
[[0.577276   0.57883556 0.52956308 0.         0.         1.        ]
 [0.37599782 0.77205322 0.18698856 0.         1.         0.        ]
 [0.4738321  0.3653876  0.29796428 0.         1.         0.        ]
 [0.80135285 0.71745725 0.68649342 0.         0.         1.        ]
 [0.1715456  0.25446874 0.34861436 0.         0.         1.        ]
 [0.94368807 0.37957895 0.8646636  0.         0.         1.        ]
 [0.6191475  0.30836422 0.52936195 1.         0.         0.        ]
 [0.00615156 0.5547241  0.0040356  0.         1.         0.        ]
 [0.23714056 0.24130912 0.3709309  1.         0.         0.        ]
 [0.53149399 0.77823604 0.         0.         1.         0.        ]
 [0.46328374 0.70684477 0.28413435 0.         0.         1.        ]
 [0.45507753 0.54429273 0.64291963 0.         0.         1.        ]
 [0.40348344 0.77456642 0.22709197 0.         1.         0.        ]
 [0.58215559 0.71401273 0.59894835 0.         1.         0.        ]
 [0.28552722 0.81005496 0.44680961 1.         0.         0.        ]
 [0.81106252 0.3694479  0.76912588 0.         1.         0.        ]
 [0.75852783 0.43692884 0.64644319 1.         0.         0.        ]
 [0.47979221 0.53527036 0.56031151 1.         0.         0.        ]
 [0.88791176 0.51299839 0.81223513 0.         1.         0.        ]
 [0.56576686 0.64106561 0.53555202 1.         0.         0.        ]
 [0.44347245 0.58297807 0.74861321 0.         1.         0.        ]
 [0.82790507 0.73016111 0.27071031 1.         0.         0.        ]
 [1.         0.76197173 0.94089337 1.         0.         0.        ]
 [0.13639639 0.78807166 0.06005866 1.         0.         0.        ]
 [0.40622666 1.         0.25042853 0.         0.         1.        ]
 [0.14539523 0.34185188 0.31370517 1.         0.         0.        ]
 [0.         0.50014806 0.09574943 1.         0.         0.        ]
 [0.73766874 0.8013272  0.54370828 0.         0.         1.        ]
 [0.00333369 0.00350184 0.         0.         1.         0.        ]
 [0.70433721 0.65174393 1.         0.         1.         0.        ]
```

Creating new DataFrame:

```
#added columns to create dataframe

xtrain_mms_df = pd.DataFrame(xtrain_mms,columns=xtrain.columns)
xtest_mms_df = pd.DataFrame(xtest_mms,columns=xtest.columns)
xtrain_mms_df
```

	Marketing Spend	Administration	Transport	Ctg	Dhaka	Rangpur
0	0.577276	0.578836	0.529563	0.0	0.0	1.0
1	0.375998	0.772053	0.186989	0.0	1.0	0.0
2	0.473832	0.365388	0.297964	0.0	1.0	0.0
3	0.801353	0.717457	0.686493	0.0	0.0	1.0
4	0.171546	0.254469	0.348614	0.0	0.0	1.0
5	0.943688	0.379579	0.864664	0.0	0.0	1.0
6	0.619148	0.308364	0.529362	1.0	0.0	0.0
7	0.006152	0.554724	0.004036	0.0	1.0	0.0
8	0.237141	0.241309	0.370931	1.0	0.0	0.0
9	0.531494	0.778236	0.000000	0.0	1.0	0.0
10	0.463284	0.706845	0.284134	0.0	0.0	1.0
11	0.455078	0.544293	0.642920	0.0	0.0	1.0
12	0.403483	0.774566	0.227092	0.0	1.0	0.0
13	0.582156	0.714013	0.598948	0.0	1.0	0.0
14	0.285527	0.810055	0.446810	1.0	0.0	0.0
15	0.811063	0.369448	0.769126	0.0	1.0	0.0
16	0.758528	0.436929	0.646443	1.0	0.0	0.0

LInearRegression Model

Insert in model:

```
#inserting the data in linear regression model

from sklearn.linear_model import LinearRegression
model_linear = LinearRegression()
model_linear.fit(xtrain_mms_df,ytrain)
```

```
▾ LinearRegression
LinearRegression()
```

Check accuracy:

```
#predicted result is 62% close to real result
model_linear.score(xtest_mms_df,ytest)
```

```
0.6216500003260639
```

Predict data:

```
#show first test
xtest.head(1)
```

	Marketing Spend	Administration	Transport	Ctg	Dhaka	Rangpur
4	142107.34	91391.77	366168.42	0	0	1

```
#predicting the result of first test
model_linear.predict(xtest_mms_df.head(1))
```

```
array([171551.76658782])
```

```
#predict for all test
model_linear.predict(xtest_mms_df)
```

```
array([171551.76658782, 126765.31390141,  75160.80566426,  59618.80934177,
        75459.08817672, 132824.28709055,  96023.04190476,  94415.22177388,
       190626.87325536, 159803.9315159 ])
```

Error calculation

Error :

```
#finding error

from sklearn import metrics
test_pred = model_linear.predict(xtest_mms_df)
mae = metrics.mean_absolute_error(ytest,test_pred)
mse = metrics.mean_squared_error(ytest,test_pred)
mse = np.sqrt(metrics.mean_squared_error(ytest,test_pred))
print ('MAE ',mae)
print ('RMSE ',mse)
```

```
MAE  13369.220999207924
RMSE  21317.97594757009
```

Ridge Regression model

```
# inserting data into ridge regression model

from sklearn.linear_model import Ridge
model_Ridge = Ridge()
model_Ridge.fit(xtrain_mms_df,ytrain)
test_pred = model_Ridge.predict(xtest_mms_df)
mae = metrics.mean_absolute_error(ytest, test_pred)
mse = metrics.mean_squared_error(ytest, test_pred)
mse = np.sqrt(metrics.mean_squared_error(ytest, test_pred))
print('MAE:', mae)
print('MSE:', mse)
```

```
MAE: 31800.027104984445
MSE: 43308.800218347846
```

Lasso Regression model

```
#inserting data into lasso regression model
from sklearn.linear_model import Lasso
model_Lasso = Lasso()
model_Lasso.fit(xtrain_mms_df,ytrain)
test_pred = model_Lasso.predict(xtest_mms_df)
mae = metrics.mean_absolute_error(ytest, test_pred)
mse = metrics.mean_squared_error(ytest, test_pred)
mse = np.sqrt(metrics.mean_squared_error(ytest, test_pred))
print('MAE:  ', mae)
print('MSE:', mse)
```

```
MAE:   33650.65420351333
MSE: 47032.709140628154
```

Check second data

Checking original data and shape:

```
#import libraries and show data
file_path = "/content/breast_cancer_dataframe.csv"

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv(file_path)
df.head(10)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...
5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2087	0.07613	...
6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794	0.05742	...
7	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196	0.07451	...
8	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	0.2350	0.07389	...
9	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	0.2030	0.08243	...

10 rows × 31 columns

```
#get row and column of the data
df.shape
```

```
(569, 31)
```

Check null:

```
#get null values
df.isnull().sum()
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

Get information of data:

```
#gives a brief description of data
df.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304

8 rows × 31 columns

```python
#gives information about the data
#we can see that we dont have any object data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

Dividing dataframe

Dividing input and output:

```
#input data
x=df.drop ([ 'target' ], axis = 1)
x.head (10)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...
5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2087	0.07613	...
6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794	0.05742	...
7	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196	0.07451	...
8	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	0.2350	0.07389	...
9	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	0.2030	0.08243	...

10 rows × 30 columns

```
#output data
y=df['target']
y.head (10)
```

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
9    0
Name: target, dtype: int64
```

Splitting data:

```
# splitting into train and test data
# defined the ratio of test:train as 20%:80%

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size = 0.2)
```

Standard Scaling

Scaling:

```
# standard scaling input data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
xtrain_sc = sc.fit_transform (xtrain )
xtest_sc = sc.transform (xtest )
```

Non-Scaled Accuracy:

```
#accuracy without standard scaling
#used classification model
from sklearn.metrics import confusion_matrix, classification_report , accuracy_score
from sklearn.svm import SVC
svc_classifier = SVC ()
svc_classifier.fit (xtrain,ytrain )
y_pred_scv = svc_classifier.predict (xtest )
accuracy_score (ytest,y_pred_scv)
```

```
0.9122807017543859
```

Scaled Accuracy:

```
#accuracy with standard scaling
svc_classifier2 = SVC ()
svc_classifier2.fit (xtrain_sc,ytrain )
y_pred_svc_sc = svc_classifier2.predict (xtest_sc )
accuracy_score (ytest , y_pred_svc_sc )
```

```
0.9824561403508771
```

KNeighborsClassifier Model

```
#accuracy with KNeighborsClassifier model
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier ()
knn_classifier.fit (xtrain ,ytrain )
y_pred_knn = knn_classifier.predict (xtest )
accuracy_score (ytest , y_pred_knn )
```

```
0.956140350877193
```

GaussianNB Model

```
#accuracy with GaussianNB model
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB ()
nb_classifier.fit (xtrain ,ytrain )
y_pred_nb = nb_classifier.predict (xtest )
accuracy_score (ytest, y_pred_nb )
```

```
0.9385964912280702
```

DecisionTreeClassifier Model

```
#accuracy with DecisionTreeClassifier model
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit (xtrain , ytrain )
y_pred_dt = dt_classifier.predict (xtest )
accuracy_score (ytest, y_pred_dt )
```

```
0.9122807017543859
```

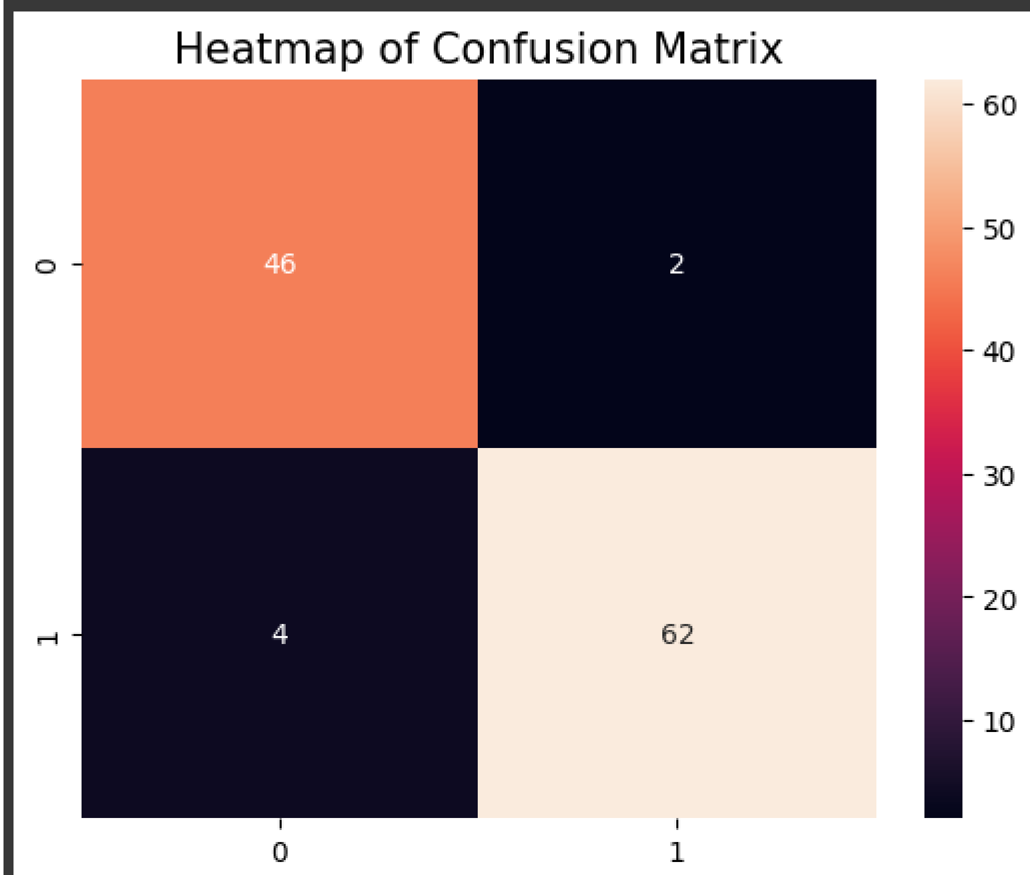
RandomForestClassifier Model

```
#accuracy with RandomForestClassifier model
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier ()
rf_classifier.fit (xtrain, ytrain )
y_pred_rf = rf_classifier.predict (xtest )
accuracy_score (ytest, y_pred_rf)
```

```
0.9473684210526315
```

Confusion Matrix

```
#Confusion Matrix; shows graphical performance
cm = confusion_matrix (ytest, y_pred_rf)
plt . title ( 'Heatmap of Confusion Matrix' , fontsize = 15)
sns . heatmap ( cm, annot = True )
plt . show()
```



Evaluation

Report:

```
#report on the prediction
print (classification_report (ytest , y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94        48
           1       0.97      0.94      0.95        66

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.95       114
weighted avg       0.95      0.95      0.95       114
```

Cross Validation:

```
# Cross validating model
from sklearn.model_selection import cross_val_score
cross_validation = cross_val_score (estimator = rf_classifier, X = x, y = y, cv = 10)
print ("Cross validation accuracy of XGBoost model=" , cross_validation )
print ("Cross validation mean accuracy of XGBoost model=" , cross_validation . mean ())
```

```
Cross validation accuracy of XGBoost model= [0.98245614 0.89473684 0.94736842 0.96491228 0.98245614 0.9824561
 0.96491228 0.98245614 0.94736842 1.        ]
Cross validation mean accuracy of XGBoost model= 0.9649122807017545
```

Other evaluation:

```
#evaluation of the model
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve , roc_auc_score , auc
print ("Fl Score :" , f1_score (ytest , rf_classifier . predict (xtest ), average='macro' ))
print ("Precision :" , precision_score (ytest , rf_classifier . predict (xtest ), average='macro' ))
print ("Recall :" , recall_score (ytest , rf_classifier . predict (xtest ), average='macro' ))
```

```
Fl Score : 0.9463108320251178
Precision : 0.944375
Recall : 0.9488636363636365
```

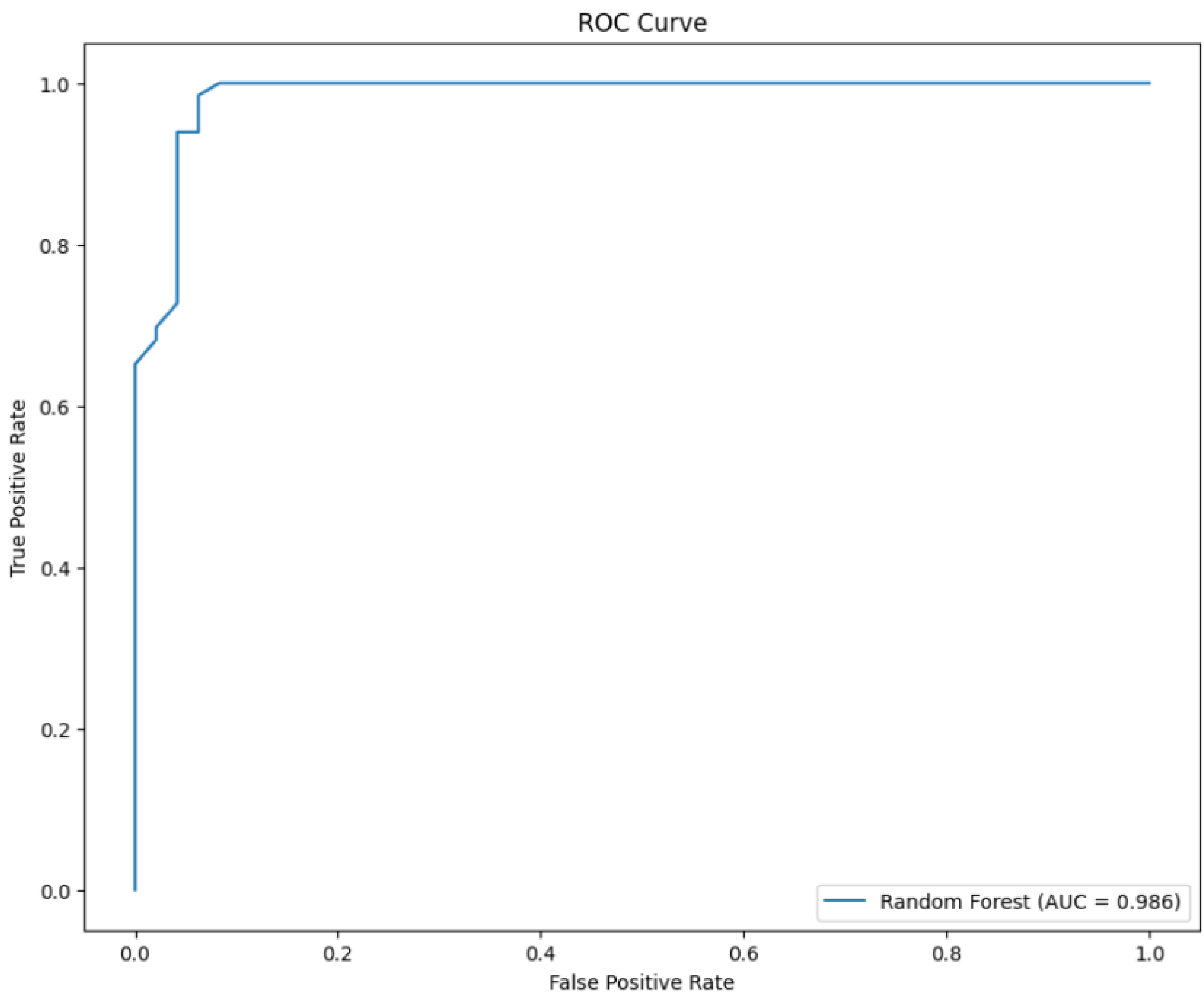
AUC evaluation:

```
#AUC evaluation
from sklearn import metrics
probs = rf_classifier.predict_proba (xtest )
probs = probs [ :, 1]
auc = roc_auc_score (ytest, probs )
fpr , tpr, _ = roc_curve (ytest, probs )
aucl= metrics . auc (fpr , tpr )
print ( "AUC :" , aucl )
```

```
AUC : 0.9856376262626262
```

ROC Curve

```
#Showing ROC curve
import matplotlib.pyplot as plt
plt . figure (figsize=(10, 8))
plt . plot (fpr , tpr, label='Random Forest (AUC = %0.3f)' % aucl )
plt . title ( 'ROC Curve' )
plt . xlabel ( 'False Positive Rate' )
plt . ylabel ( 'True Positive Rate' )
plt . legend ()
plt . grid (False)
plt . show()
```


ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Random Forest (AUC = 0.986)

Saving

Model saving:

```
#saving model

import pickle
pickle . dump ( rf_classifier, open ( 'breast_cancer_detector.pickle' , 'wb' ))
```

Using model:

```
# using the model

breast_cancer_detector_model = pickle.load (open ( 'breast_cancer_detector.pickle' , 'rb' ))
y_pred = breast_cancer_detector_model . predict (xtest )
print ( 'Confusion matrix of XGBoost model: \n' , confusion_matrix (ytest , y_pred ), '\n' )
print ( 'Accuracy of XGBoost model= ' , accuracy_score (ytest , y_pred ))
```

```
Confusion matrix of XGBoost model:
 [[46  2]
 [ 4 62]]

Accuracy of XGBoost model=  0.9473684210526315
```