Developing a Flight Delay Prediction Model using Machine Learning

Date	17/11/2022
Team Id	PNT2022TMID15779
Project Name	Developing a flight delay estimation model using machine Learning Maximum
Maximum Marks	4 Marks

1.DATA PREPROCESSING

Data Preprocessing

```
# importing the required libraries
import sys
import numpy as np
import pandas as pd
import seaborn as sns
import pickle
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
import warnings
warnings.filterwarnings('ignore')
```

2.IMPORTING DATA

Importing Data

```
dataset=pd.read_csv('flightdata.csv')
dataset.head()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_ARR_TIME	ARR_TIME	ARR_DELAY
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	2143	2102.0	-41.0
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	1435	1439.0	4.0
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	1215	1142.0	-33.0
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	1335	1345.0	10.0
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	607	615.0	8.0

5 rows × 26 columns

3.ANALYSING THE DATA

Analysing the data

[199…]
```
dataset.info()
```

```
RangeIndex: 11231 entries, 0 to 11230
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   YEAR                 11231 non-null  int64  
 1   QUARTER              11231 non-null  int64  
 2   MONTH                11231 non-null  int64  
 3   DAY_OF_MONTH         11231 non-null  int64  
 4   DAY_OF_WEEK          11231 non-null  int64  
 5   UNIQUE_CARRIER       11231 non-null  object 
 6   TAIL_NUM             11231 non-null  object 
 7   FL_NUM               11231 non-null  int64  
 8   ORIGIN_AIRPORT_ID    11231 non-null  int64  
 9   ORIGIN               11231 non-null  object 
 10  DEST_AIRPORT_ID      11231 non-null  int64  
 11  DEST                 11231 non-null  object 
 12  CRS_DEP_TIME         11231 non-null  int64  
 13  DEP_TIME             11124 non-null  float64
 14  DEP_DELAY            11124 non-null  float64
 15  DEP_DEL15            11124 non-null  float64
 16  CRS_ARR_TIME         11231 non-null  int64  
 17  ARR_TIME             11116 non-null  float64
 18  ARR_DELAY            11043 non-null  float64
 19  ARR_DEL15            11043 non-null  float64
 20  CANCELLED            11231 non-null  float64
 21  DIVERTED             11231 non-null  float64
 22  CRS_ELAPSED_TIME     11231 non-null  float64
 23  ACTUAL_ELAPSED_TIME  11043 non-null  float64
 24  DISTANCE             11231 non-null  float64
 25  Unnamed: 25          0 non-null      float64
dtypes: float64(12), int64(10), object(4)
memory usage: 2.2+ MB
```

[200…]
```
dataset.describe()
```

[200…]

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	CRS_DEP_TIME	DEP_TIME	...	CRS_ARR_TIM
count	11231.0	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11124.000000	...	11231.0000
mean	2016.0	2.544475	6.628973	15.790758	3.960199	1334.325617	12334.516695	12302.274508	1320.798326	1327.189410	...	1537.3127
std	0.0	1.090701	3.354678	8.782056	1.995257	811.875227	1595.026510	1601.988550	490.737845	500.306462	...	502.5124
min	2016.0	1.000000	1.000000	1.000000	1.000000	7.000000	10397.000000	10397.000000	10.000000	1.000000	...	2.0000
25%	2016.0	2.000000	4.000000	8.000000	2.000000	624.000000	10397.000000	10397.000000	905.000000	905.000000	...	1130.0000
50%	2016.0	3.000000	7.000000	16.000000	4.000000	1267.000000	12478.000000	12478.000000	1320.000000	1324.000000	...	1559.0000
75%	2016.0	3.000000	9.000000	23.000000	6.000000	2032.000000	13487.000000	13487.000000	1735.000000	1739.000000	...	1952.0000
max	2016.0	4.000000	12.000000	31.000000	7.000000	2853.000000	14747.000000	14747.000000	2359.000000	2400.000000	...	2359.0000

8 rows × 22 columns

4. HANDLING MISSING VALUES

Handling missing values

In [201…
```
dataset.isnull().sum()
```

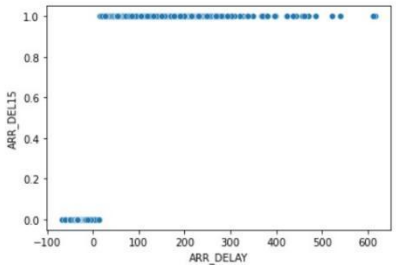
Out[201…
```
YEAR                    0
QUARTER                 0
MONTH                   0
DAY_OF_MONTH            0
DAY_OF_WEEK             0
UNIQUE_CARRIER          0
TAIL_NUM                0
FL_NUM                  0
ORIGIN_AIRPORT_ID       0
ORIGIN                  0
DEST_AIRPORT_ID         0
DEST                    0
CRS_DEP_TIME            0
DEP_TIME              107
DEP_DELAY             107
DEP_DEL15             107
CRS_ARR_TIME            0
ARR_TIME              115
ARR_DELAY             188
ARR_DEL15             188
CANCELLED               0
DIVERTED                0
CRS_ELAPSED_TIME        0
ACTUAL_ELAPSED_TIME   188
DISTANCE                0
Unnamed: 25         11231
dtype: int64
```

In [202…
```
dataset['DEST'].unique()
```

Out[202…
```
array(['SEA', 'MSP', 'DTW', 'ATL', 'JFK'], dtype=object)
```

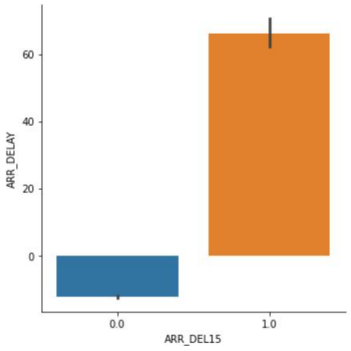
In [203…
```
# Data visualization
sns.scatterplot(dataset['ARR_DELAY'],dataset['ARR_DEL15'])
```

Out[203…



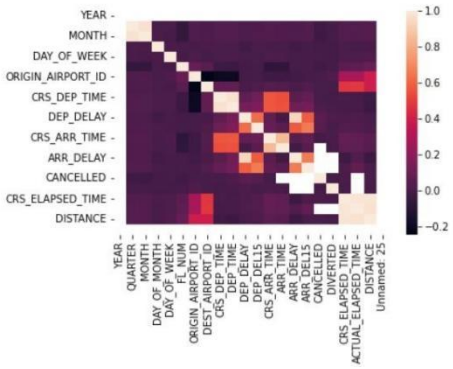
In [204…
```
sns.catplot(x="ARR_DEL15",y="ARR_DELAY",kind='bar',data=dataset)
```

Out[204…



In [205…
```
sns.heatmap(dataset.corr())
```

Out[205…



5. DROPPING UNNECESSARY COLUMNS

Dropping unnecessary columns

In [206…
```
dataset = dataset.drop('Unnamed: 25',axis=1)
dataset.isnull().sum()
```

Out[206…
```
YEAR                    0
QUARTER                 0
MONTH                   0
DAY_OF_MONTH            0
DAY_OF_WEEK             0
UNIQUE_CARRIER          0
TAIL_NUM                0
FL_NUM                  0
ORIGIN_AIRPORT_ID       0
ORIGIN                  0
DEST_AIRPORT_ID         0
DEST                    0
CRS_DEP_TIME            0
DEP_TIME              107
DEP_DELAY             107
DEP_DEL15             107
CRS_ARR_TIME            0
ARR_TIME              115
ARR_DELAY             188
ARR_DEL15             188
CANCELLED               0
DIVERTED                0
CRS_ELAPSED_TIME        0
ACTUAL_ELAPSED_TIME   188
DISTANCE                0
dtype: int64
```

In [207…
```
dataset = dataset[["FL_NUM","MONTH","DAY_OF_MONTH","DAY_OF_WEEK","ORIGIN","DEST","CRS_ARR_TIME","DEP_DEL15","ARR_DEL15"]]
dataset.isnull().sum()
```

Out[207…
```
FL_NUM            0
MONTH             0
DAY_OF_MONTH      0
DAY_OF_WEEK       0
ORIGIN            0
DEST              0
CRS_ARR_TIME      0
DEP_DEL15       107
ARR_DEL15       188
dtype: int64
```

In [208…
```
# replace missing values with 0 and 1
dataset = dataset.fillna({'ARR_DEL15':1})
dataset = dataset.fillna({'DEP_DEL15':0})
dataset.iloc[177:185]
```

Out[208…

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
177	2834	1	9	6	MSP	SEA	852	0.0	1.0
178	2839	1	9	6	DTW	JFK	1724	0.0	0.0
179	86	1	10	7	MSP	DTW	1632	0.0	1.0
180	87	1	10	7	DTW	MSP	1649	1.0	0.0
181	423	1	10	7	JFK	ATL	1600	0.0	0.0
182	440	1	10	7	JFK	ATL	849	0.0	0.0
183	485	1	10	7	JFK	SEA	1945	1.0	0.0
184	557	1	10	7	MSP	DTW	912	0.0	1.0

In [207…

```
dataset = dataset[["FL_NUM","MONTH","DAY_OF_MONTH","DAY_OF_WEEK","ORIGIN","DEST","CRS_ARR_TIME","DEP_DEL15","ARR_DEL15"]]
dataset.isnull().sum()
```

Out[207…

```
FL_NUM            0
MONTH             0
DAY_OF_MONTH      0
DAY_OF_WEEK       0
ORIGIN            0
DEST              0
CRS_ARR_TIME      0
DEP_DEL15       107
ARR_DEL15       188
dtype: int64
```

In [208…

```
# replace missing values with 0 and 1
dataset = dataset.fillna({'ARR_DEL15':1})
dataset = dataset.fillna({'DEP_DEL15':0})
dataset.iloc[177:185]
```

Out[208…

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
177	2834	1	9	6	MSP	SEA	852	0.0	1.0
178	2839	1	9	6	DTW	JFK	1724	0.0	0.0
179	86	1	10	7	MSP	DTW	1632	0.0	1.0
180	87	1	10	7	DTW	MSP	1649	1.0	0.0
181	423	1	10	7	JFK	ATL	1600	0.0	0.0
182	440	1	10	7	JFK	ATL	849	0.0	0.0
183	485	1	10	7	JFK	SEA	1945	1.0	0.0
184	557	1	10	7	MSP	DTW	912	0.0	1.0

In [209…

```
#count no of values with respect to unique values of each columns
for i in dataset.columns:
    print(dataset[i].value_counts())
```

```
888     98
589     96
1991    96
588     95
902     94
        ..
2849     1
1531     1
1493     1
1507     1
2640     1
Name: FL_NUM, Length: 690, dtype: int64
8     1127
7     1078
6      979
9      962
10     955
5      916
12     899
11     891
3      885
4      882
1      860
2      797
Name: MONTH, dtype: int64
8     390
23    381
28    379
11    378
14    378
21    378
22    376
26    375
18    373
9     372
2     371
10    370
15    370
12    369
19    368
```

```
3     351
30    335
31    209
Name: DAY_OF_MONTH, dtype: int64
5    1668
1    1652
4    1637
3    1624
2    1607
7    1593
6    1450
Name: DAY_OF_WEEK, dtype: int64
ATL    3100
MSP    2538
DTW    2201
SEA    2018
JFK    1374
Name: ORIGIN, dtype: int64
ATL    3221
MSP    2493
DTW    2211
SEA    1994
JFK    1312
Name: DEST, dtype: int64
1655    81
1840    72
1945    64
1345    54
1051    54
        ..
2141     1
1933     1
7        1
1757     1
1821     1
Name: CRS_ARR_TIME, Length: 958, dtype: int64
0.0    9642
1.0    1589
Name: DEP_DEL15, dtype: int64
0.0    9668
1.0    1563
Name: ARR_DEL15, dtype: int64
```

In [210…
```
dataset.head(5)
```

Out[210…

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	5	ATL	SEA	2143	0.0	0.0
1	1476	1	1	5	DTW	MSP	1435	0.0	0.0
2	1597	1	1	5	ATL	SEA	1215	0.0	0.0
3	1768	1	1	5	SEA	MSP	1335	0.0	0.0
4	1823	1	1	5	SEA	DTW	607	0.0	0.0

In [211…
```
dataset.duplicated().sum()
```

Out[211… `0`

In [212…
```
dataset.iloc[179,:].isnull(),dataset.iloc[179,:]
```

Out[212…
```
(FL_NUM          False
 MONTH           False
 DAY_OF_MONTH    False
 DAY_OF_WEEK     False
 ORIGIN          False
 DEST            False
 CRS_ARR_TIME    False
 DEP_DEL15       False
 ARR_DEL15       False
 Name: 179, dtype: bool,
 FL_NUM            86
 MONTH              1
 DAY_OF_MONTH      10
 DAY_OF_WEEK        7
 ORIGIN           MSP
 DEST             DTW
 CRS_ARR_TIME    1632
 DEP_DEL15        0.0
 ARR_DEL15        1.0
 Name: 179, dtype: object)
```

In [213…
```
dataset[(dataset["DEP_DEL15"]==0)&(dataset["ARR_DEL15"]==1)]
```

Out[213…

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
34	1770	1	4	1	SEA	MSP	2032	0.0	1.0
128	744	1	7	4	MSP	ATL	1334	0.0	1.0
146	8	1	8	5	MSP	ATL	2105	0.0	1.0
166	1473	1	8	5	SEA	JFK	1930	0.0	1.0
167	1598	1	8	5	SEA	ATL	1401	0.0	1.0
…	…	…	…	…	…	…	…	…	…
11120	811	12	29	4	ATL	MSP	1532	0.0	1.0
11168	984	12	30	5	ATL	JFK	2315	0.0	1.0
11173	2610	12	31	6	ATL	MSP	900	0.0	1.0
11187	95	12	3	6	ATL	DTW	1436	0.0	1.0
11203	2221	12	9	5	MSP	SEA	1311	0.0	1.0

512 rows × 9 columns

In [214…
```
dataset.describe()
```

Out[214…

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
count	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000
mean	1334.325617	6.628973	15.790758	3.960199	1537.312795	0.141483	0.139168
std	811.875227	3.354678	8.782056	1.995257	502.512494	0.348535	0.346138
min	7.000000	1.000000	1.000000	1.000000	2.000000	0.000000	0.000000
25%	624.000000	4.000000	8.000000	2.000000	1130.000000	0.000000	0.000000
50%	1267.000000	7.000000	16.000000	4.000000	1559.000000	0.000000	0.000000
75%	2032.000000	9.000000	23.000000	6.000000	1952.000000	0.000000	0.000000

In [215…
```
oh = OneHotEncoder()
z = oh.fit_transform(dataset[['ORIGIN','DEST']]).toarray()
t = oh.fit_transform(dataset[['DEST','CRS_ARR_TIME']]).toarray()
```

In [216…
```
z
```

Out[216…
```
array([[1., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 1., 0., ..., 0., 0., 1.],
       [1., 0., 0., ..., 0., 0., 1.],
       [1., 0., 0., ..., 0., 0., 0.]])
```

In [217…
```
t
```

Out[217…
```
array([[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., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]])
```

In [218…
```
#Splitting into dependent and independent variables
dataset = pd.get_dummies(dataset , columns=['ORIGIN','DEST'])
dataset.head()
```

Out[218…

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15	ORIGIN_ATL	ORIGIN_DTW	ORIGIN_JFK	ORIGIN_MSP	ORIGIN_SEA	DEST_ATL	DE
0	1399	1	1	5	2143	0.0	0.0	1	0	0	0	0	0	
1	1476	1	1	5	1435	0.0	0.0	0	1	0	0	0	0	
2	1597	1	1	5	1215	0.0	0.0	1	0	0	0	0	0	
3	1768	1	1	5	1335	0.0	0.0	0	0	0	0	1	0	
4	1823	1	1	5	607	0.0	0.0	0	0	0	0	1	0	

```
x=dataset.drop(columns=['ARR_DEL15']).values

y=dataset['ARR_DEL15'].values
```

6.SPLITTING INTO TRAIN AND TEST DATA

Splitting into train and test data

In [220…
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

In [221…
```
x_test.shape
```

Out[221… (2247, 16)

In [222…
```
x_train.shape
```

Out[222… (8984, 16)

In [223…
```
y_test.shape
```

Out[223… (2247,)

In [224…
```
y_train.shape
```

Out[224… (8984,)

7.MODEL BUILDING

MODEL BUILDING

DecisionTree

In [225…
```
from sklearn.tree import DecisionTreeClassifier
dc=DecisionTreeClassifier()
dc.fit(x_train,y_train)
dc.score(x_test,y_test)
```

Out[225… 0.872719181130396

8. RANDOM FOREST

RandomForest

In [226…
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=50,random_state=42)
rf.fit(x_train,y_train)
rf.score(x_test,y_test)
```

Out[226… 0.9101023587004895

In [227…
```
pd.DataFrame(rf.predict(x_test)).value_counts()
```

Out[227…
```
0.0    2010
1.0     237
dtype: int64
```

9.LOGISTIC REGRESSION

LogisticRegression

In [228…
```
from sklearn.linear_model import LogisticRegression
lr1=LogisticRegression(solver='sag')
lr1.fit(x_train,y_train)
lr1.score(x_test,y_test)
```

Out[228… 0.8615932354250111

In [229…
```
lr1.predict(x_test).sum()
```

Out[229… 0.0

10. SVM

SVM

In [230…
```
from sklearn.svm import SVC
svm=SVC(kernel='sigmoid')
svm.fit(x_train,y_train)
svm.score(x_test,y_test)
```

Out[230… 0.7739207832665776

In [231…
```
pd.DataFrame(svm.predict(x_test)).value_counts()
```

Out[231…
```
0.0    1948
1.0     299
dtype: int64
```

In [232…
```
pd.DataFrame(y_test).value_counts()
```

Out[232…
```
0.0    1936
1.0     311
dtype: int64
```

11. KNEAREST NEIGHBOUR CLASSIFIER

KNearestNeighbourClassifier

In [233…
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
knn.score(x_test,y_test)
```

Out[233… 0.8531375166889186

In [234…
```
pd.DataFrame(knn.predict(x_test)).value_counts()
```

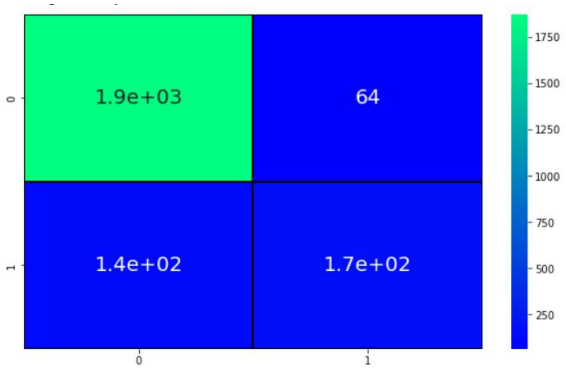
Out[234…
```
0.0    2158
1.0      89
dtype: int64
```

12. EVALUATION OF RANDOM FOREST

Evaluation Of Random Forest

In [235…
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
pred=rf.predict(x_test)
cm=confusion_matrix(y_test, pred)
plt.figure(figsize=(10,6))
sns.heatmap(cm, annot=True,cmap='winter',linewidths=0.3, linecolor='black',annot_kws={"size": 20})
TP=cm[0][0]
TN=cm[1][1]
FN=cm[1][0]
FP=cm[0][1]
#print(round(accuracy_score(prediction3,y_test)*100,2))
#print('Testing Accuracy for knn',(TP+TN)/(TP+TN+FN+FP))
print('Testing Sensitivity for Random Forest',(TP/(TP+FN)))
print('Testing Specificity for Random Forest',(TN/(TN+FP)))
print('Testing Precision for Random Forest',(TP/(TP+FP)))
print('Testing accuracy for Random Forest',accuracy_score(y_test, pred))
```

```
Testing Sensitivity for Random Forest 0.9313432835820895
Testing Specificity for Random Forest 0.729957805907173
Testing Precision for Random Forest 0.9669421487603306
Testing accuracy for Random Forest 0.9101023587004895
```



In [236…
```
print(classification_report(y_test,pred))#RandomForest
```

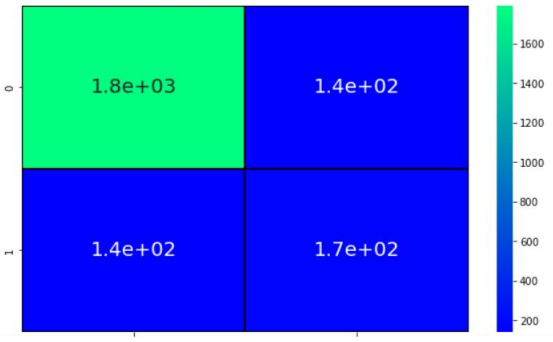
```
              precision    recall  f1-score   support

         0.0       0.93      0.97      0.95      1936
         1.0       0.73      0.56      0.63       311

    accuracy                           0.91      2247
   macro avg       0.83      0.76      0.79      2247
weighted avg       0.90      0.91      0.90      2247
```

13. EVALUATION OF DECISION TREE AND SAVING THE MODEL

Evaluation Of Decission Tree

In [237…
```
pred1=dc.predict(x_test)
cm1=confusion_matrix(y_test, pred1)
plt.figure(figsize=(10,6))
sns.heatmap(cm1, annot=True,cmap='winter',linewidths=0.3, linecolor='black',annot_kws={"size": 20})
TP=cm1[0][0]
TN=cm1[1][1]
FN=cm1[1][0]
FP=cm1[0][1]
#print(round(accuracy_score(prediction3,y_test)*100,2))
print('Testing Accuracy for Decision Tree',(TP+TN)/(TP+TN+FN+FP))
print('Testing Sensitivity for Decision Tree',(TP/(TP+FN)))
print('Testing Specificity for Decision Tree',(TN/(TN+FP)))
print('Testing Precision for Decision Tree',(TP/(TP+FP)))
print('Testing accuracy for Decision Tree',accuracy_score(y_test, pred1))
```

```
Testing Accuracy for Decision Tree 0.872719181130396
Testing Sensitivity for Decision Tree 0.9265770423991727
Testing Specificity for Decision Tree 0.5399361022364217
Testing Precision for Decision Tree 0.925619834710438
Testing accuracy for Decision Tree 0.872719181130396
```



In [238…
```
print(classification_report(y_test,pred1))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.93      0.93      1936
         1.0       0.54      0.54      0.54       311

    accuracy                           0.87      2247
   macro avg       0.73      0.73      0.73      2247
weighted avg       0.87      0.87      0.87      2247
```

In [239…
```
import pickle
```

In [240…
```
pickle.dump(rf,open("rfmodel.pkl",'wb'))
```