Files

..
.config
sample_data
dataset.h5
flight.pkl
flightdata.csv

Disk 84.54 GB available

+ Code + Text

RAM
Disk

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,f1_score
```

```
[2] dataset=pd.read_csv("flightdata.csv")
```

```
[3] dataset.head()
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_AR

0s completed at 1:47 PM

```
[2] dataset=pd.read_csv("flightdata.csv")
```

```
[3] 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	ARR_DEL15
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	2143	2102.0	-41.0	0.0
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	1435	1439.0	4.0	0.0
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	1215	1142.0	-33.0	0.0
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	1335	1345.0	10.0	0.0
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	607	615.0	8.0	0.0

5 rows × 26 columns

[4] `dataset.info()`

```
<class 'pandas.core.frame.DataFrame'>
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
```

[4]

```
 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
```

```
[5]  dataset=dataset.drop('Unnamed: 25', axis=1)
     dataset.isnull().sum()
```

```
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
```

```
CANCELLED                0
DIVERTED                 0
CRS_ELAPSED_TIME         0
ACTUAL_ELAPSED_TIME    188
DISTANCE                 0
dtype: int64
```

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

```
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
```

```
[7]  dataset[dataset.isnull().any(axis=1)].head(10)
```

	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	NaN
179	86	1	10	7	MSP	DTW	1632	NaN	NaN
184	557	1	10	7	MSP	DTW	912	0.0	NaN
210	1096	1	10	7	DTW	MSP	1303	NaN	NaN
478	1542	1	22	5	SEA	JFK	723	NaN	NaN
481	1795	1	22	5	ATL	JFK	2014	NaN	NaN
491	2312	1	22	5	MSP	JFK	2149	NaN	NaN
499	423	1	23	6	JFK	ATL	1600	NaN	NaN
500	425	1	23	6	JFK	ATL	1827	NaN	NaN
501	427	1	23	6	JFK	SEA	1053	NaN	NaN

[8] `dataset['DEP_DEL15'].mode()`

```
0    0.0
Name: DEP_DEL15, dtype: float64
```

[9]
```
dataset=dataset.fillna({'ARR_DEL15':1})
dataset=dataset.fillna({'DEP_DEL15':0})
dataset.iloc[177:185]
```

	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

```
[10] import math
```

```
for index,row in dataset.iterrows():
  dataset.loc[index,'CRS_ARR_TIME']=math.floor(row['CRS_ARR_TIME']/100)
dataset.head()
```

	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	21	0.0	0.0
1	1476	1	1	5	DTW	MSP	14	0.0	0.0
2	1597	1	1	5	ATL	SEA	12	0.0	0.0
3	1768	1	1	5	SEA	MSP	13	0.0	0.0
4	1823	1	1	5	SEA	DTW	6	0.0	0.0

```
[12] from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset['DEST']=le.fit_transform(dataset['DEST'])
```

[12]
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset['DEST']=le.fit_transform(dataset['DEST'])
dataset['ORIGIN']=le.fit_transform(dataset['ORIGIN'])
```

```
dataset.head(5)
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	5	0	4	21	0.0	0.0
1	1476	1	1	5	1	3	14	0.0	0.0
2	1597	1	1	5	0	4	12	0.0	0.0
3	1768	1	1	5	4	3	13	0.0	0.0
4	1823	1	1	5	4	1	6	0.0	0.0

[14]
```
dataset['ORIGIN'].unique()
```

```
array([0, 1, 4, 3, 2])
```

```
[15] dataset=pd.get_dummies(dataset, columns=['ORIGIN','DEST'])
     dataset.head()
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15	ORIGIN_0	ORIGIN_1	ORIGIN_2	ORIGIN_3	ORIGIN_4	DEST_0	DEST_1	DEST_2	DEST_3	D
0	1399	1	1	5	21	0.0	0.0	1	0	0	0	0	0	0	0	0	
1	1476	1	1	5	14	0.0	0.0	0	1	0	0	0	0	0	0	1	
2	1597	1	1	5	12	0.0	0.0	1	0	0	0	0	0	0	0	0	
3	1768	1	1	5	13	0.0	0.0	0	0	0	0	1	0	0	0	1	
4	1823	1	1	5	6	0.0	0.0	0	0	0	0	1	0	1	0	0	

```
x=dataset.iloc[:,0:8].values
y=dataset.iloc[:,8:9].values
```

```
x
```

```
array([[1.399e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.476e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.597e+03, 1.000e+00, 1.000e+00, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       ...,
       [1.823e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        0.000e+00],
       [1.901e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [2.005e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00,
        1.000e+00]])
```

```
[18] from sklearn.preprocessing import OneHotEncoder
     oh=OneHotEncoder()
     z=oh.fit_transform(x[:,4:5]).toarray()
     t=oh.fit_transform(x[:,5:6]).toarray()
     #x=np.delete(x,[4,7],axis=1)
```

[19] z

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

[20] t

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

[21] x=np.delete(x,[4,5],axis=1)

```
[22] dataset.describe()
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15	ORIGIN_0	ORIGIN_1	ORIGIN_2	ORIGIN_3	ORI
count	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.0
mean	1334.325617	6.628973	15.790758	3.960199	15.067314	0.141483	0.139168	0.276022	0.195975	0.122340	0.225982	0.1
std	811.875227	3.354678	8.782056	1.995257	5.023534	0.348535	0.346138	0.447048	0.396967	0.327693	0.418246	0.3
min	7.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	624.000000	4.000000	8.000000	2.000000	11.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	1267.000000	7.000000	16.000000	4.000000	15.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	2032.000000	9.000000	23.000000	6.000000	19.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.0
max	2853.000000	12.000000	31.000000	7.000000	23.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

```
[23] sns.distplot(dataset.MONTH)
```

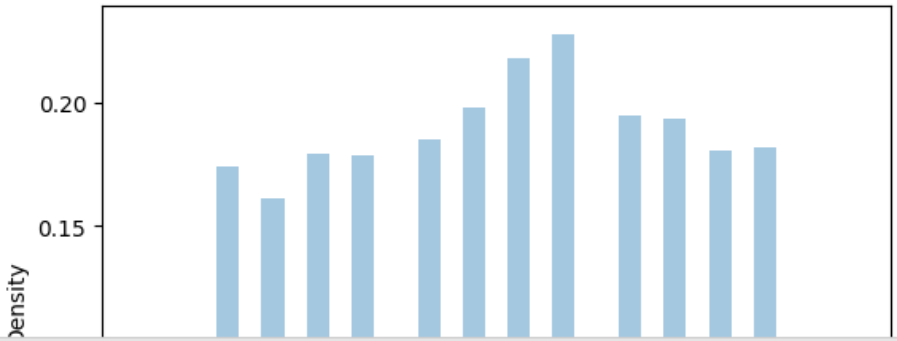
```
dataset.describe()
```

ME	DEP_DEL15	ARR_DEL15	ORIGIN_0	ORIGIN_1	ORIGIN_2	ORIGIN_3	ORIGIN_4	DEST_0	DEST_1	DEST_2	DEST_3	DEST_4
00	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000
14	0.141483	0.139168	0.276022	0.195975	0.122340	0.225982	0.179681	0.286795	0.196866	0.116820	0.221975	0.177544
34	0.348535	0.346138	0.447048	0.396967	0.327693	0.418246	0.383939	0.452285	0.397648	0.321219	0.415593	0.382146
00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
00	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
00	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

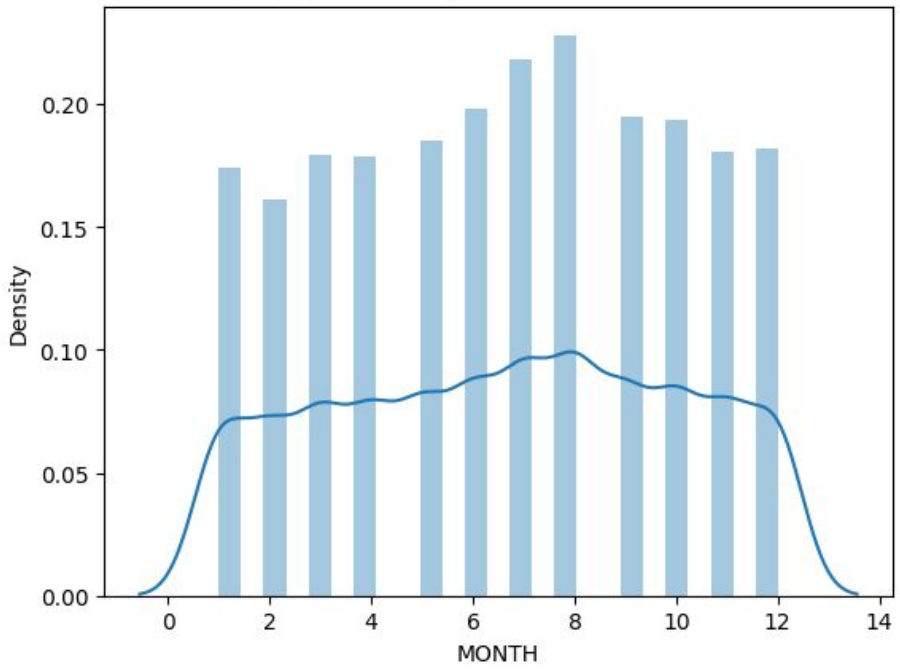
```
[23] sns.distplot(dataset.MONTH)
```

```
[23] sns.distplot(dataset.MONTH)
```

```
<ipython-input-23-43f5c122a6ef>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset.MONTH)
<Axes: xlabel='MONTH', ylabel='Density'>
```

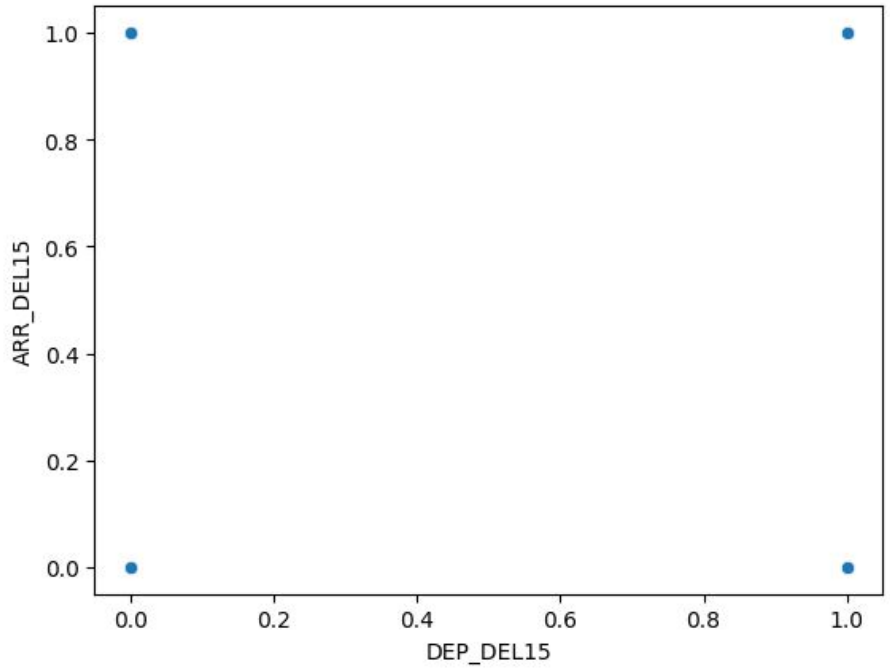


[23] <Axes: xlabel='MONTH', ylabel='Density'>



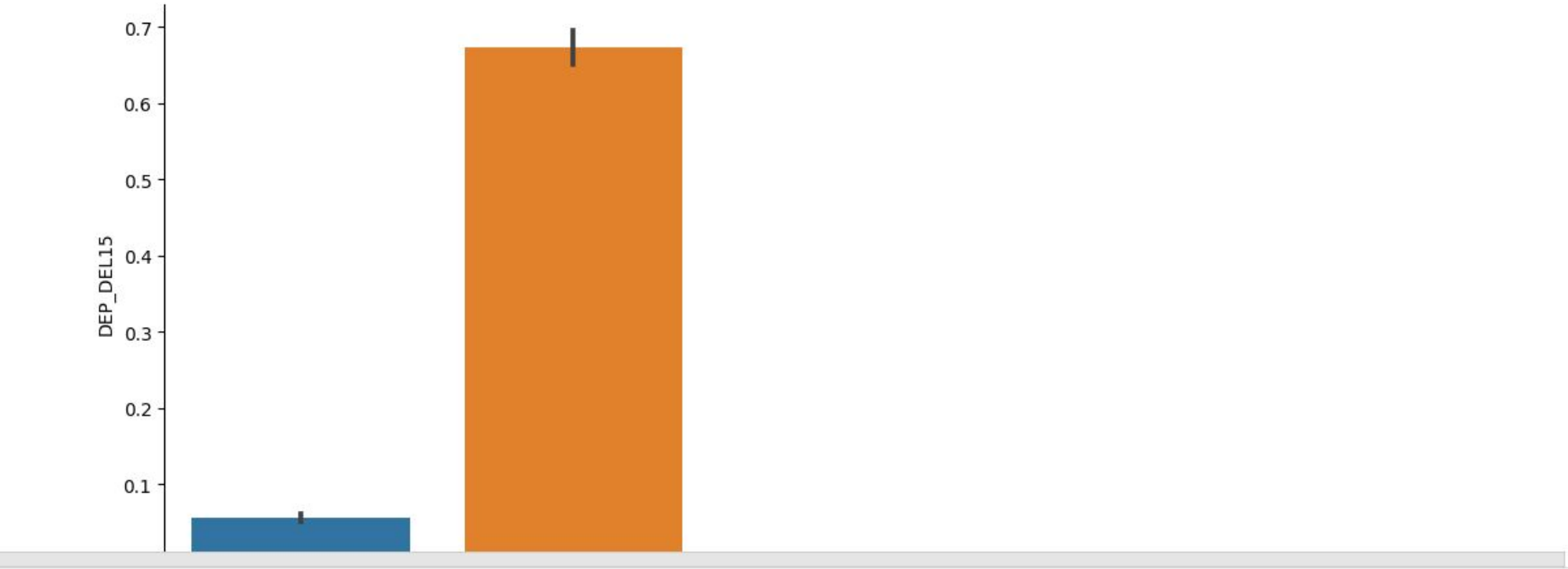
```
[24] sns.scatterplot(x='DEP_DEL15',y='ARR_DEL15',data=dataset)
```

```
<Axes: xlabel='DEP_DEL15', ylabel='ARR_DEL15'>
```

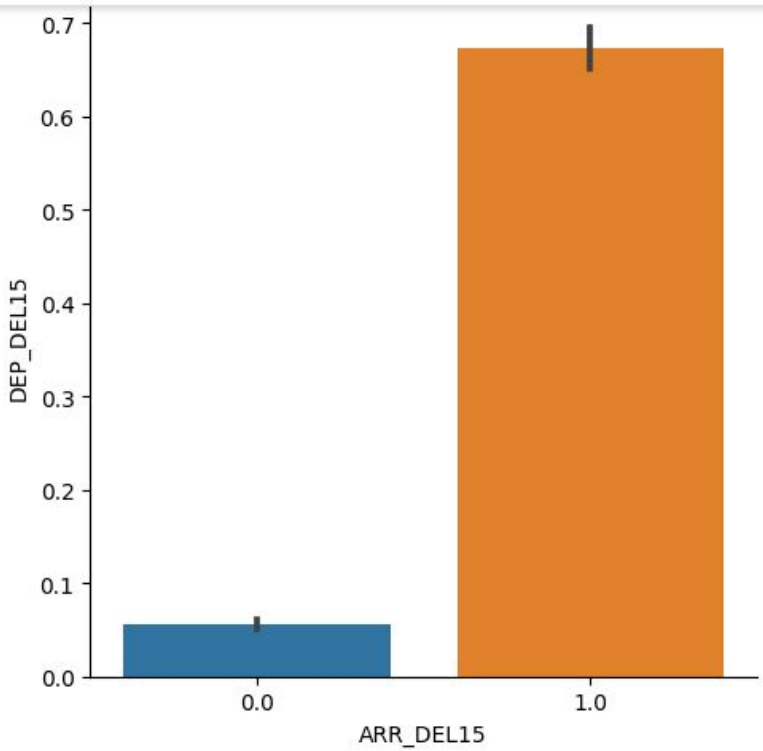


```
[25] sns.catplot(x="ARR_DEL15",y="DEP_DEL15",kind='bar',data=dataset)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5a9c449a90>
```

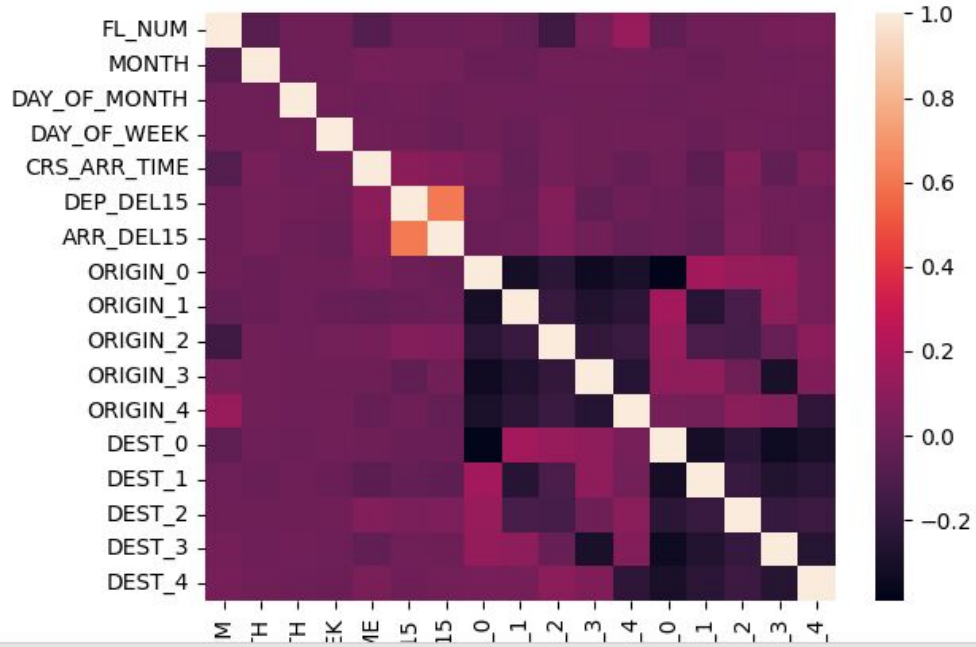


✓ [25]
0s

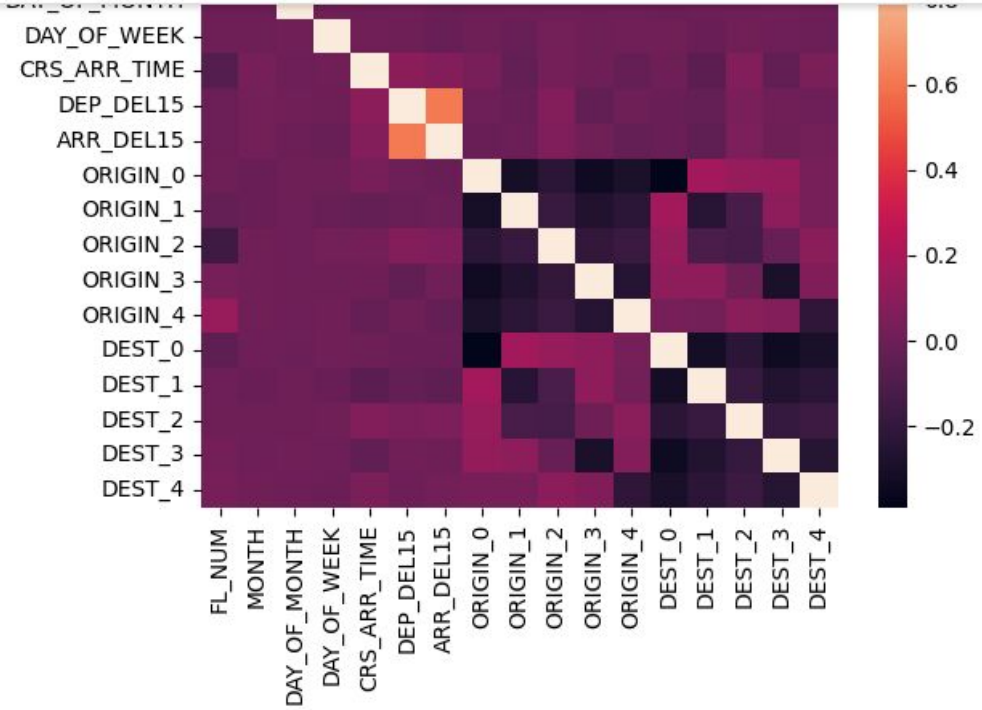


```
[26] sns.heatmap(dataset.corr())
```

```
<Axes: >
```



[26]


DAY_OF_WEEK
CRS_ARR_TIME
DEP_DEL15
ARR_DEL15
ORIGIN_0
ORIGIN_1
ORIGIN_2
ORIGIN_3
ORIGIN_4
DEST_0
DEST_1
DEST_2
DEST_3
DEST_4
FL_NUM
MONTH
DAY_OF_MONTH
DAY_OF_WEEK
CRS_ARR_TIME
DEP_DEL15
ARR_DEL15
ORIGIN_0
ORIGIN_1
ORIGIN_2
ORIGIN_3
ORIGIN_4
DEST_0
DEST_1
DEST_2
DEST_3
DEST_4
0.6
0.4
0.2
0.0
−0.2

```
[27] from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
[28] from sklearn.model_selection import train_test_split
     train_x,test_x,train_y,test_y=train_test_split(dataset.drop('ARR_DEL15',axis=1),dataset['ARR_DEL15'],test_size=0.2,random_state=0)
```

```
[29] x_test.shape
```

```
(2247, 6)
```

```
[30] x_train.shape
```

```
(8984, 6)
```

```
[31] y_test.shape
```

```
(2247, 1)
```

```
[32] y_train.shape
```

```
[32] y_train.shape
```

```
(8984, 1)
```

```
[33] from sklearn.preprocessing import StandardScaler
     sc=StandardScaler()
     x_train=sc.fit_transform(x_train)
     x_test=sc.transform(x_test)
```

```
[34] from sklearn.tree import DecisionTreeClassifier
     classifier=DecisionTreeClassifier(random_state=0)
     classifier.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
[35] decisiontree=classifier.predict(x_test)
```

```
[36] decisiontree
```

[36]
```
decisiontree
```

```
array([1, 0, 0, ..., 1, 0, 0], dtype=uint8)
```

[37]
```
from sklearn.metrics import accuracy_score
desacc=accuracy_score(y_test,decisiontree)
```

[38]
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=10,criterion='entropy')
```

[39]
```
rfc.fit(x_train,y_train)
```

```
<ipython-input-39-b87bb2ba9825>:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,),
  rfc.fit(x_train,y_train)
```

RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

[40]
```
y_predict=rfc.predict(x_test)
```

```
[40] y_predict=rfc.predict(x_test)
```

```
[41] import tensorflow
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
```

```
[42] classification=Sequential()
     classification.add(Dense(30,activation='relu'))
     classification.add(Dense(128,activation='relu'))
     classification.add(Dense(64,activation='relu'))
     classification.add(Dense(32,activation='relu'))
     classification.add(Dense(1,activation='sigmoid'))
```

```
[43] classification.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
[44] classification.fit(x_train,y_train,batch_size=4,validation_split=0.2,epochs=100)
```

```
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3381 - accuracy: 0.8296 - val_loss: 0.4408 - val_accuracy: 0.7924
Epoch 73/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3369 - accuracy: 0.8336 - val_loss: 0.4505 - val_accuracy: 0.7880
Epoch 74/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3332 - accuracy: 0.8368 - val_loss: 0.4450 - val_accuracy: 0.8002
Epoch 75/100
```

[44]

```
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3369 - accuracy: 0.8336 - val_loss: 0.4505 - val_accuracy: 0.7880
Epoch 74/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3332 - accuracy: 0.8368 - val_loss: 0.4450 - val_accuracy: 0.8002
Epoch 75/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3338 - accuracy: 0.8325 - val_loss: 0.4331 - val_accuracy: 0.8024
Epoch 76/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3294 - accuracy: 0.8383 - val_loss: 0.4692 - val_accuracy: 0.7958
Epoch 77/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3314 - accuracy: 0.8379 - val_loss: 0.4633 - val_accuracy: 0.7930
Epoch 78/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3300 - accuracy: 0.8376 - val_loss: 0.4431 - val_accuracy: 0.7941
Epoch 79/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3281 - accuracy: 0.8403 - val_loss: 0.4974 - val_accuracy: 0.7902
Epoch 80/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3273 - accuracy: 0.8422 - val_loss: 0.4437 - val_accuracy: 0.7913
Epoch 81/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3295 - accuracy: 0.8393 - val_loss: 0.4616 - val_accuracy: 0.7908
Epoch 82/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3267 - accuracy: 0.8387 - val_loss: 0.4830 - val_accuracy: 0.7757
Epoch 83/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3280 - accuracy: 0.8418 - val_loss: 0.4725 - val_accuracy: 0.7941
Epoch 84/100
1797/1797 [==============================] - 6s 3ms/step - loss: 0.3240 - accuracy: 0.8426 - val_loss: 0.4439 - val_accuracy: 0.7958
Epoch 85/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3239 - accuracy: 0.8426 - val_loss: 0.4776 - val_accuracy: 0.7958
Epoch 86/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3210 - accuracy: 0.8411 - val_loss: 0.5136 - val_accuracy: 0.7896
Epoch 87/100
```

[44]

```
Epoch 88/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3202 - accuracy: 0.8421 - val_loss: 0.4749 - val_accuracy: 0.7880
Epoch 89/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3193 - accuracy: 0.8426 - val_loss: 0.4679 - val_accuracy: 0.7874
Epoch 90/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3188 - accuracy: 0.8457 - val_loss: 0.5036 - val_accuracy: 0.7852
Epoch 91/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3197 - accuracy: 0.8414 - val_loss: 0.4996 - val_accuracy: 0.7785
Epoch 92/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3171 - accuracy: 0.8479 - val_loss: 0.4820 - val_accuracy: 0.7813
Epoch 93/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3163 - accuracy: 0.8476 - val_loss: 0.5020 - val_accuracy: 0.7796
Epoch 94/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3116 - accuracy: 0.8467 - val_loss: 0.5187 - val_accuracy: 0.7757
Epoch 95/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3153 - accuracy: 0.8454 - val_loss: 0.4764 - val_accuracy: 0.7869
Epoch 96/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3161 - accuracy: 0.8472 - val_loss: 0.4893 - val_accuracy: 0.7863
Epoch 97/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3157 - accuracy: 0.8481 - val_loss: 0.4708 - val_accuracy: 0.7969
Epoch 98/100
1797/1797 [==============================] - 5s 3ms/step - loss: 0.3101 - accuracy: 0.8500 - val_loss: 0.5104 - val_accuracy: 0.7885
Epoch 99/100
1797/1797 [==============================] - 4s 2ms/step - loss: 0.3089 - accuracy: 0.8514 - val_loss: 0.4985 - val_accuracy: 0.7835
Epoch 100/100
1797/1797 [==============================] - 6s 3ms/step - loss: 0.3121 - accuracy: 0.8476 - val_loss: 0.5028 - val_accuracy: 0.7947
<keras.callbacks.History at 0x7f5a0cbbcb50>
```

```
[45] y_pred=classifier.predict([[129,99,1,0,0,1]])
     print(y_pred)
     (y_pred)
```

```
[0]
array([0], dtype=uint8)
```

```
[46] y_pred=rfc.predict([[129,99,1,0,0,1]])
     print(y_pred)
     (y_pred)
```

```
[0]
array([0], dtype=uint8)
```

```
[47] classification.save('dataset.h5')
```

```
[48] y_pred=classification.predict(x_test)
```

```
71/71 [==============================] - 0s 1ms/step
```

[49] y_pred

```
array([[0.04850658],
       [0.14575092],
       [0.        ],
       ...,
       [0.7245959 ],
       [0.        ],
       [0.7086633 ]], dtype=float32)
```

[50] y_pred=(y_pred>0.5)
y_pred

```
array([[False],
       [False],
       [False],
       ...,
       [ True],
       [False],
       [ True]])
```

```
[51] def predict_exit(sample_value):
       sample_value=np.array(sample_value)
       sample_value=sample_value.reshape(1,-1)
       sample_value=sc.transform(sample_value)
       return classifier.predict(sample_value)
```

```
[52] test=classification.predict([[1,1,121.000000,36.0,0,0]])
     if test==1:
       print('prediction: Chance of delay')
     else:
       print('prediction: No chanceof delay')
```

```
1/1 [==============================] - 0s 47ms/step
prediction: No chanceof delay
```

```
[53] from sklearn import model_selection
     from sklearn.neural_network import MLPClassifier
```

```
dfs=[]
models=[('RF',RandomForestClassifier()),('DecisionTree',DecisionTreeClassifier()),('ANN',MLPClassifier())]
results=[]
names=[]
scoring=['accuracy','precision_weighted','recall_weighted','f1_weighted','roc_auc']
target_names=['no delay','delay']
for name,model in models:
  kfold=model_selection.KFold(n_splits=5,shuffle=True,random_state=90210)
  cv_results=model_selection.cross_validate(model,x_train,y_train,cv=kfold,scoring=scoring)
  clf=model.fit(x_train,y_train)
  y_pred=clf.predict(x_test)
  print(name)
  print(classification_report(y_test,y_pred,target_names=target_names))
  results.append(cv_results)
  names.append(name)
  this_df=pd.DataFrame(cv_results)
  this_df['model']=name
  dfs.append(this_df)
final=pd.concat(dfs,ignore_index=True)
return final
```

```
RF
              precision    recall  f1-score   support

    no delay       0.86      0.93      0.89      1802
       delay       0.57      0.36      0.44       445

    accuracy                           0.82      2247
   macro avg       0.72      0.65      0.67      2247
weighted avg       0.80      0.82      0.80      2247

DecisionTree
              precision    recall  f1-score   support

    no delay       0.99      0.99      0.99      1802
       delay       0.96      0.96      0.96       445

    accuracy                           0.98      2247
   macro avg       0.98      0.98      0.98      2247
weighted avg       0.98      0.98      0.98      2247
```

```
ANN
              precision    recall  f1-score   support

    no delay       0.81      0.99      0.89      1802
       delay       0.52      0.03      0.06       445

    accuracy                           0.80      2247
   macro avg       0.66      0.51      0.47      2247
weighted avg       0.75      0.80      0.73      2247
```

[56]
```
print('Testing accuracy:',accuracy_score(y_test,y_predict)*100)
```

```
Testing accuracy: 83.75611927013796
```

[57]
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predict)
cm
```

```
array([[1714,   88],
       [ 277,  168]])
```

[58]
```
from sklearn.metrics import accuracy_score
desacc=accuracy_score(y_test,decisiontree)
```

[59] desacc

```
0.9839786381842457
```

[60]
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,decisiontree)
```

[61] cm

```
array([[1786,   16],
       [  20,  425]])
```

[62]
```
from sklearn.metrics import accuracy_score,classification_report
score=accuracy_score(y_pred,y_test)
print('The accuracy for ANN model is:{}%'.format(score*100))
```

```
The accuracy for ANN model is:80.24032042723631%
```

[63]
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

[63]
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[1789,   13],
       [ 431,   14]])
```

[64]
```
parameters={'n_estimators':[1,20,30,55,68,74,90,120,115],'criterion':['gini','entropy'],'max_features':["auto","sqrt","log2"],'max_depth':[2,5,8,10],'verbose':[1
```

[65]
```
RCV=RandomizedSearchCV(estimator=rfc,param_distributions=parameters,cv=10,n_iter=4)
```

```
[65] RCV=RandomizedSearchCV(estimator=rfc,param_distributions=parameters,cv=10,n_iter=4)
```

0s

```
[66] RCV.fit(x_train,y_train)
```

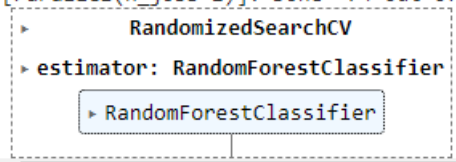
19s

```
building tree 25 of 74
building tree 26 of 74
building tree 27 of 74
building tree 28 of 74
building tree 29 of 74
building tree 30 of 74
building tree 31 of 74
building tree 32 of 74
building tree 33 of 74
building tree 34 of 74
building tree 35 of 74
building tree 36 of 74
```

[66] 19s

```
building tree 34 of 74
building tree 35 of 74
building tree 36 of 74
building tree 37 of 74
building tree 38 of 74
building tree 39 of 74
building tree 40 of 74
building tree 41 of 74
building tree 42 of 74
building tree 43 of 74
building tree 44 of 74
building tree 45 of 74
building tree 46 of 74
building tree 47 of 74
building tree 48 of 74
building tree 49 of 74
building tree 50 of 74
building tree 51 of 74
building tree 52 of 74
building tree 53 of 74
building tree 54 of 74
building tree 55 of 74
building tree 56 of 74
building tree 57 of 74
building tree 58 of 74
building tree 59 of 74
building tree 60 of 74
building tree 61 of 74
```

```
building tree 58 of 74
building tree 59 of 74
building tree 60 of 74
building tree 61 of 74
building tree 62 of 74
building tree 63 of 74
building tree 64 of 74
building tree 65 of 74
building tree 66 of 74
building tree 67 of 74
building tree 68 of 74
building tree 69 of 74
building tree 70 of 74
building tree 71 of 74
building tree 72 of 74
building tree 73 of 74
building tree 74 of 74
[Parallel(n_jobs=1)]: Done  74 out of  74 | elapsed:    0.5s finished
```




```
bt_params=RCV.best_params_
bt_score=RCV.best_score_
```

```
[67] bt_params=RCV.best_params_
     bt_score=RCV.best_score_
```

```
[68] bt_params
```

```
{'verbose': 3,
 'n_estimators': 74,
 'max_features': 'sqrt',
 'max_depth': 10,
 'criterion': 'entropy'}
```

```
[69] bt_score
```

```
0.823573953737263
```

```
[70] model=RandomForestClassifier(verbose=10,n_estimators=120,max_features='log2',max_depth=10,criterion='entropy')
     RCV.fit(x_train,y_train)
```

```
building tree 19 of 68
building tree 20 of 68
building tree 21 of 68
building tree 22 of 68
building tree 23 of 68
building tree 24 of 68
```

✓ 25s [70]

```
building tree 22 of 68
building tree 23 of 68
building tree 24 of 68
building tree 25 of 68
building tree 26 of 68
building tree 27 of 68
building tree 28 of 68
building tree 29 of 68
building tree 30 of 68
building tree 31 of 68
building tree 32 of 68
building tree 33 of 68
building tree 34 of 68
building tree 35 of 68
building tree 36 of 68
building tree 37 of 68
building tree 38 of 68
building tree 39 of 68
building tree 40 of 68
building tree 41 of 68
building tree 42 of 68
building tree 43 of 68
building tree 44 of 68
building tree 45 of 68
building tree 46 of 68
building tree 47 of 68
building tree 48 of 68
building tree 49 of 68
```

✓ 25s [70]

```
building tree 50 of 68
building tree 51 of 68
building tree 52 of 68
building tree 53 of 68
building tree 54 of 68
building tree 55 of 68
building tree 56 of 68
building tree 57 of 68
building tree 58 of 68
building tree 59 of 68
building tree 60 of 68
building tree 61 of 68
building tree 62 of 68
building tree 63 of 68
building tree 64 of 68
building tree 65 of 68
building tree 66 of 68
building tree 67 of 68
building tree 68 of 68
[Parallel(n_jobs=1)]: Done  68 out of  68 | elapsed:    0.5s finished
```

▸ RandomizedSearchCV

▸ estimator: RandomForestClassifier

▸ RandomForestClassifier

[71]
```
y_predict_rf=RCV.predict(x_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  68 out of  68 | elapsed:    0.0s finished
```

[72]
```
RFC=accuracy_score(y_test,y_predict_rf)
RFC
```

```
0.8264352469959947
```

[73]
```
import pickle
pickle.dump(RCV,open('flight.pkl','wb'))
```