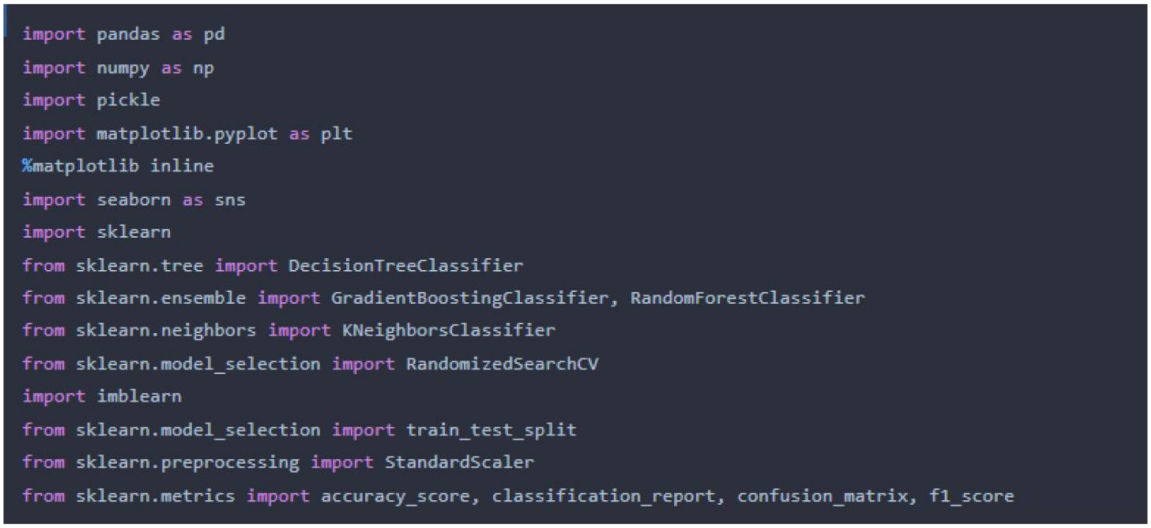
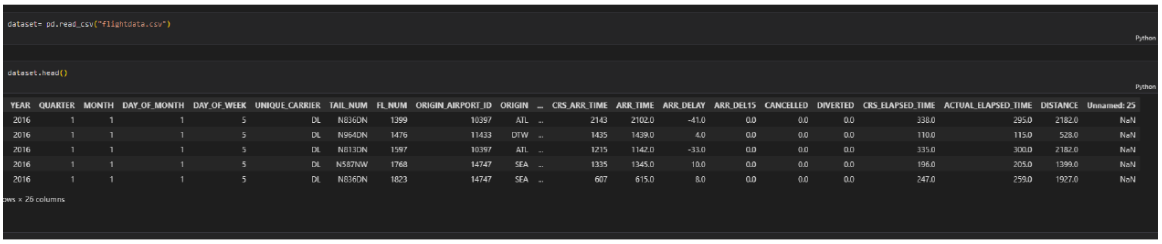


```python
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors 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
```

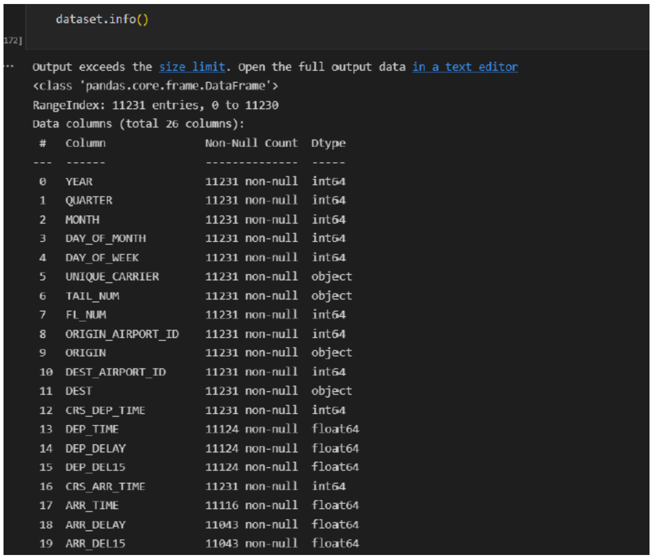


```python
dataset= pd.read_csv("flightdata.csv")
```

```python
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	CANCELLED	DIVERTED	CRS_ELAPSED_TIME	ACTUAL_ELAPSED_TIME	DISTANCE	Unnamed: 25
2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	2143	2102.0	-41.0	0.0	0.0	0.0	338.0	295.0	2182.0	NaN
2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	1435	1439.0	4.0	0.0	0.0	0.0	110.0	115.0	528.0	NaN
2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	1215	1142.0	-33.0	0.0	0.0	0.0	335.0	300.0	2182.0	NaN
2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	1335	1345.0	10.0	0.0	0.0	0.0	196.0	205.0	1399.0	NaN
2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	607	615.0	8.0	0.0	0.0	0.0	247.0	259.0	1927.0	NaN

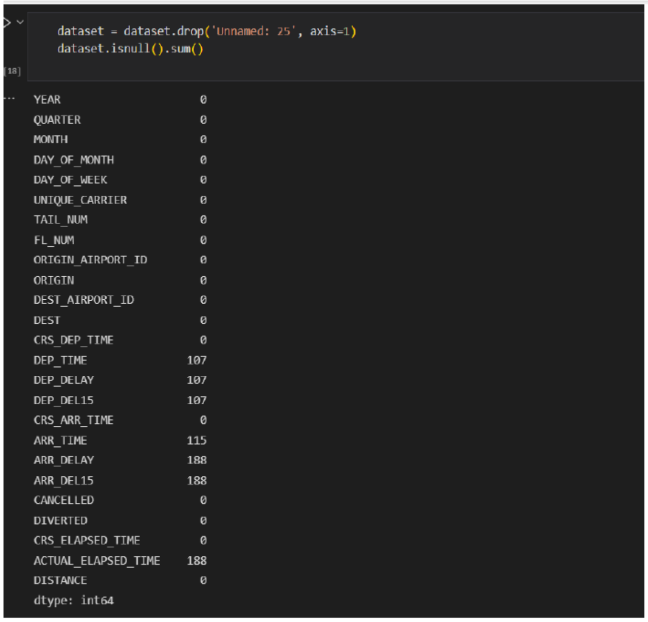
ows × 26 columns



```python
dataset.info()
```

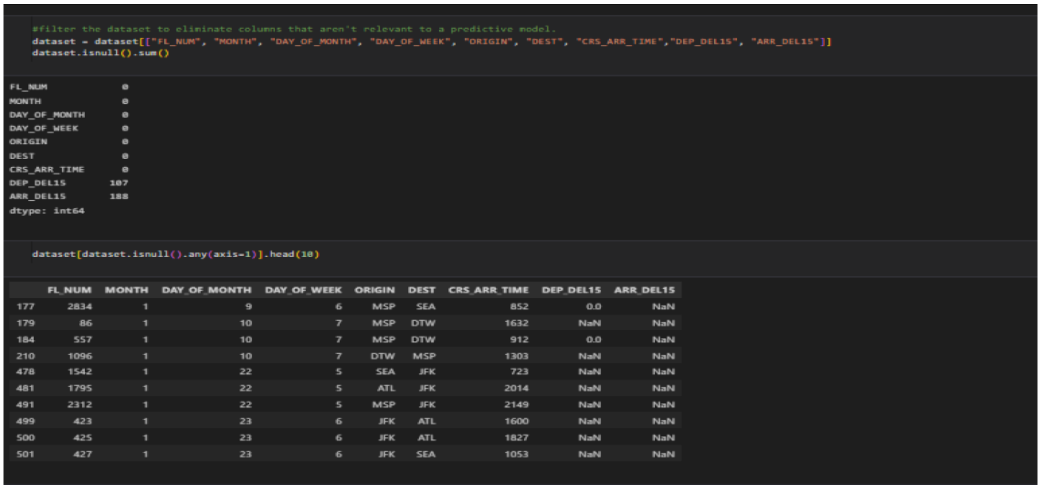
172]

```
Output exceeds the size limit. Open the full output data in a text editor
<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
 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
```



```
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
DISTANCE                 0
dtype: int64
```

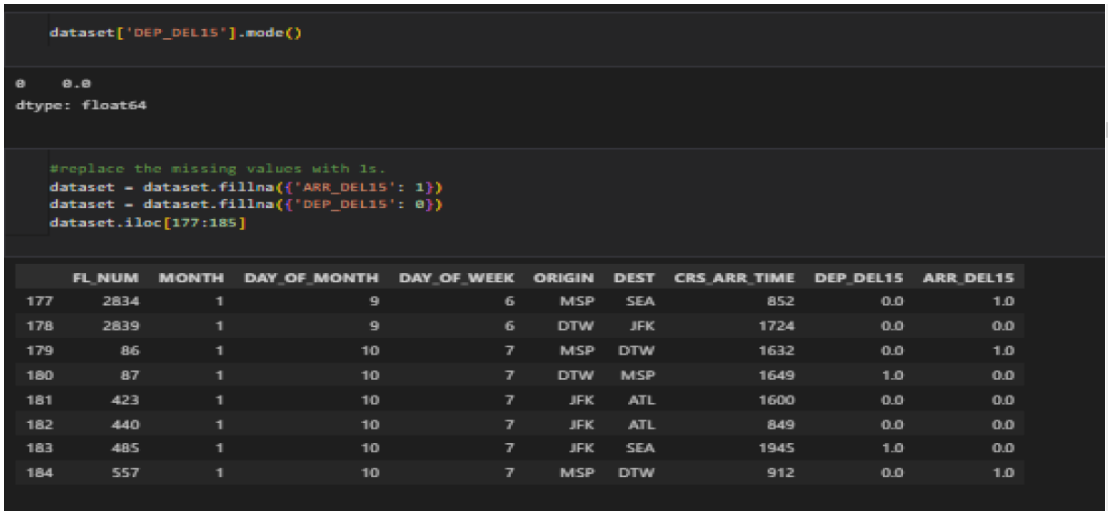


```
#filter the dataset to eliminate columns that aren't relevant to a predictive model.
dataset = dataset[["FL_NUM", "MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_ARR_TIME","DEP_DEL15", "ARR_DEL15"]]
dataset.isnull().sum()
```

```
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
```

```
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

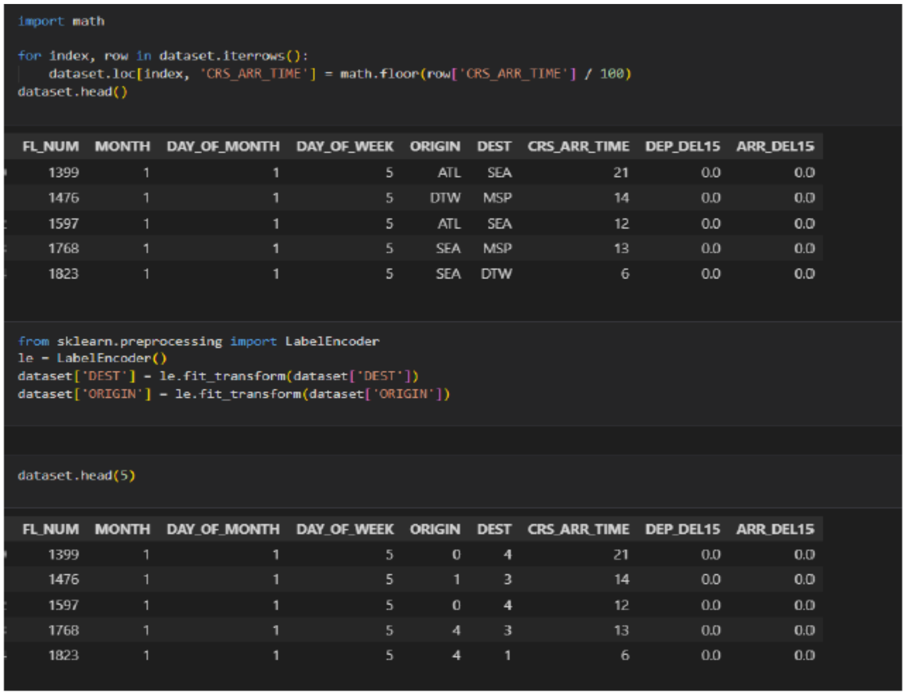


```
dataset['DEP_DEL15'].mode()
```

```
0    0.0
dtype: float64
```

```
#replace the missing values with 1s.
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



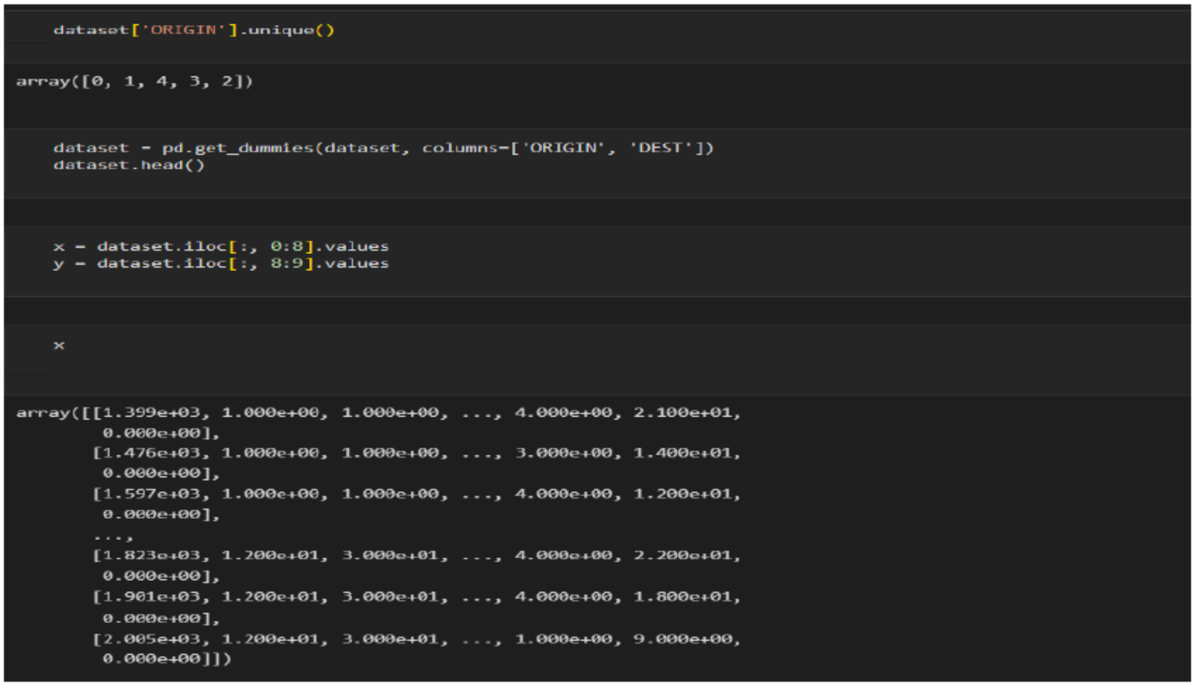
```python
import math

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
1399	1	1	5	ATL	SEA	21	0.0	0.0
1476	1	1	5	DTW	MSP	14	0.0	0.0
1597	1	1	5	ATL	SEA	12	0.0	0.0
1768	1	1	5	SEA	MSP	13	0.0	0.0
1823	1	1	5	SEA	DTW	6	0.0	0.0

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

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

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



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

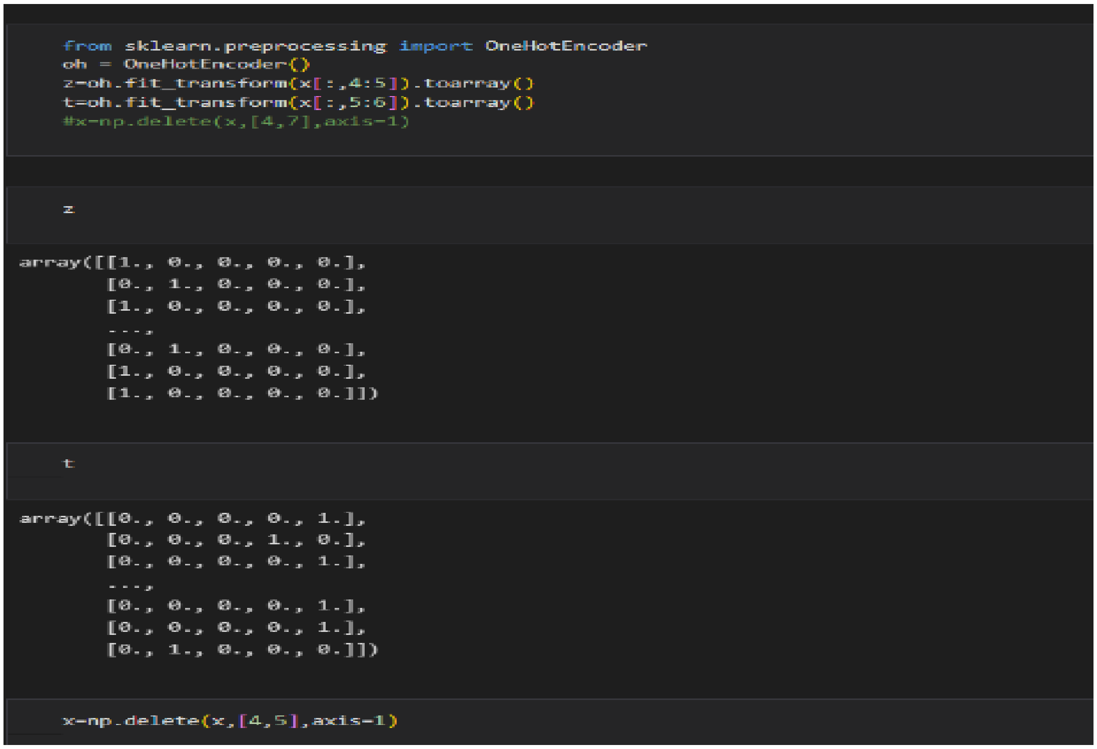
```
array([0, 1, 4, 3, 2])
```

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

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

```python
x
```

```
array([[1.399e+03, 1.000e+00, 1.000e+00, ..., 4.000e+00, 2.100e+01,
        0.000e+00],
       [1.476e+03, 1.000e+00, 1.000e+00, ..., 3.000e+00, 1.400e+01,
        0.000e+00],
       [1.597e+03, 1.000e+00, 1.000e+00, ..., 4.000e+00, 1.200e+01,
        0.000e+00],
       ...,
       [1.823e+03, 1.200e+01, 3.000e+01, ..., 4.000e+00, 2.200e+01,
        0.000e+00],
       [1.901e+03, 1.200e+01, 3.000e+01, ..., 4.000e+00, 1.800e+01,
        0.000e+00],
       [2.005e+03, 1.200e+01, 3.000e+01, ..., 1.000e+00, 9.000e+00,
        0.000e+00]])
```



```
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)
```

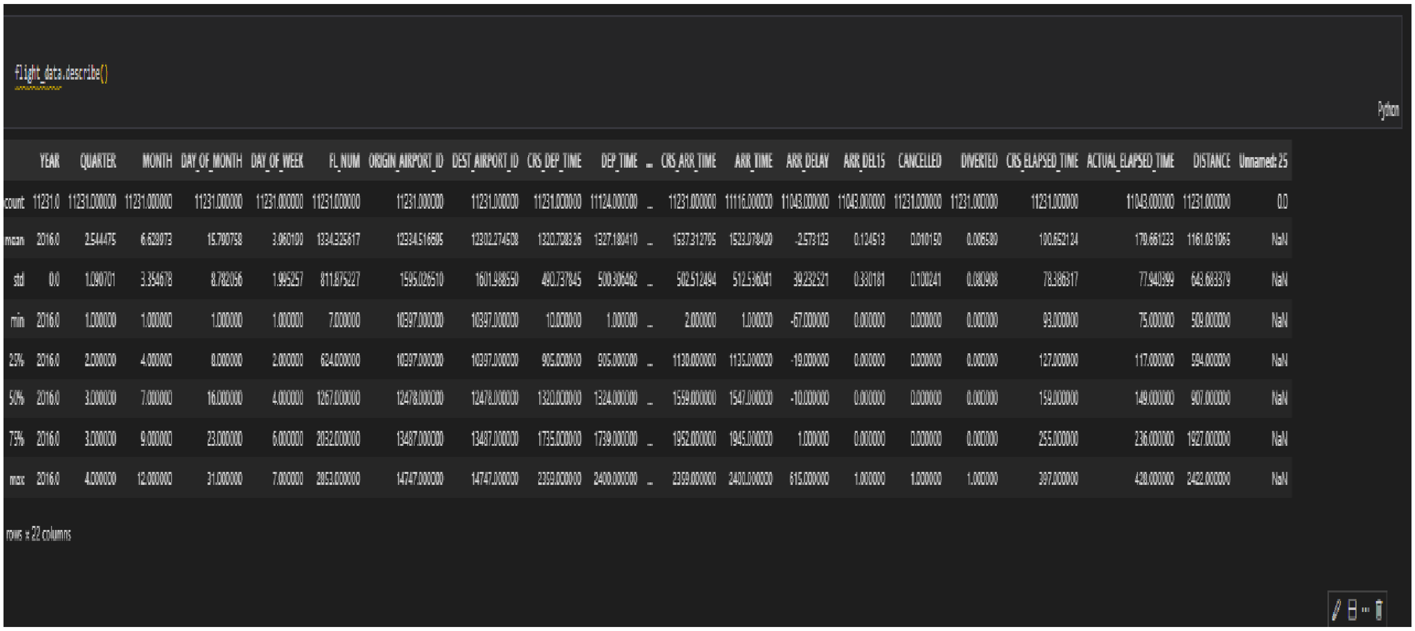
```
z
```

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

```
t
```

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

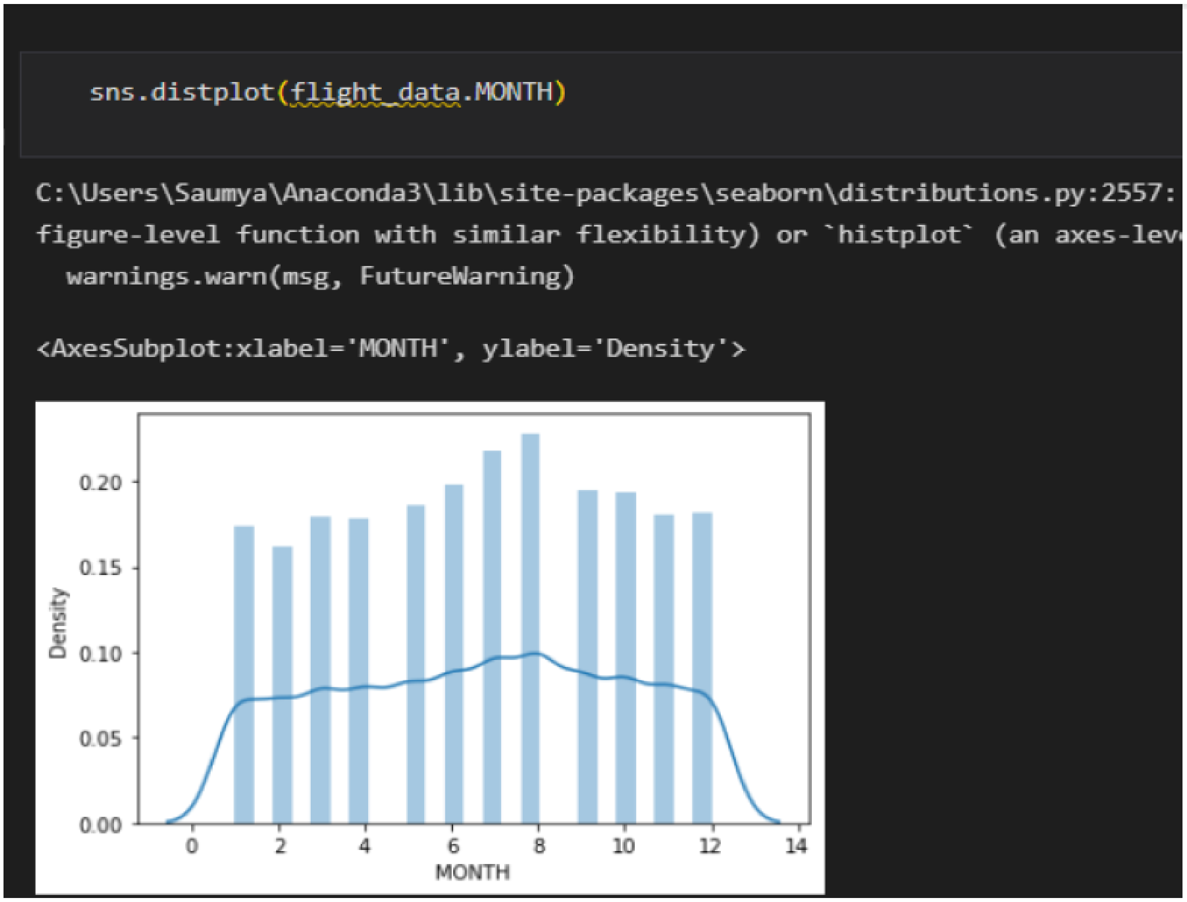
```
x=np.delete(x,[4,5],axis=1)
```



```
flight_data.describe()
```

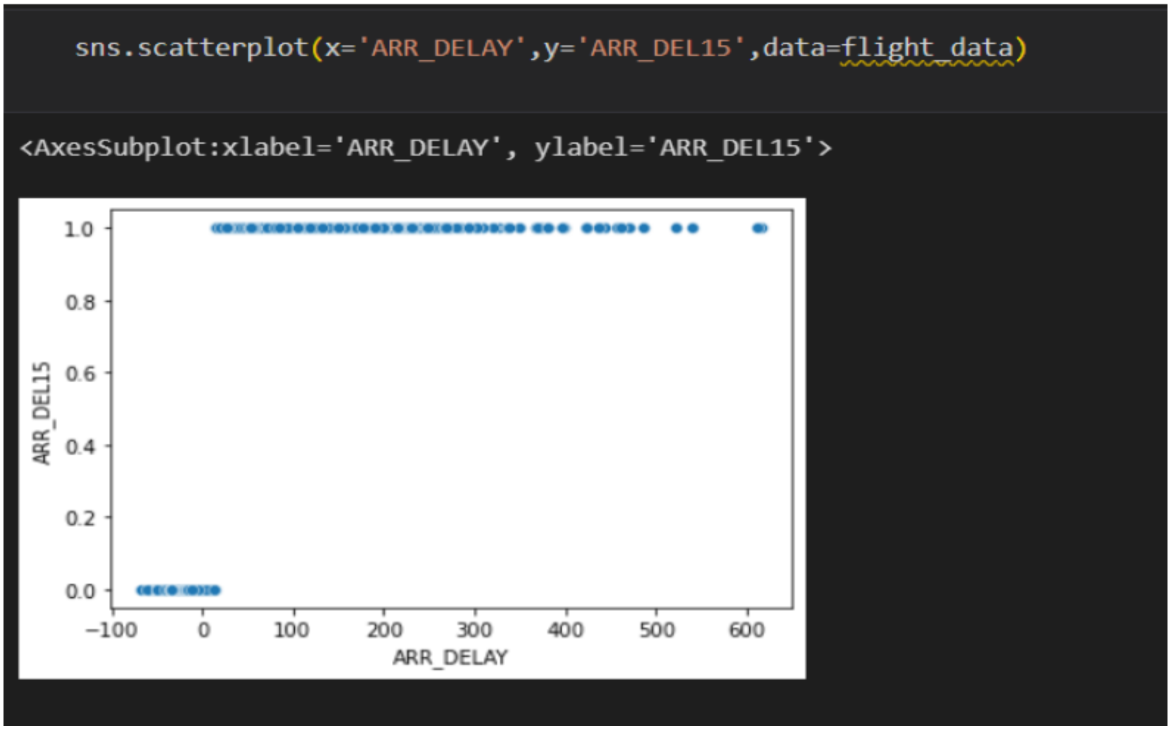
	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	CRS_DEP_TIME	DEP_TIME	...	CRS_ARR_TIME	ARR_TIME	ARR_DELAY	ARR_DEL15	CANCELLED	DIVERTED	CRS_ELAPSED_TIME	ACTUAL_ELAPSED_TIME	DISTANCE	Unnamed: 25
count	11231.0	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11124.000000	...	11231.000000	11116.000000	11043.000000	11043.000000	11231.000000	11231.000000	11231.000000	11043.000000	11231.000000	0.0
mean	2016.0	2.544475	6.628073	15.790798	3.960100	1334.325617	12334.516605	12302.274508	1320.708326	1327.189410	...	1537.312705	1523.078409	-2.573123	0.124513	0.010150	0.006589	180.652124	179.661233	1161.031066	NaN
std	0.0	1.090701	3.354678	8.782056	1.995257	811.875227	1595.026510	1601.988550	490.737845	500.306462	...	502.512494	512.536041	39.232521	0.330181	0.100241	0.080908	78.386317	77.940399	643.683379	NaN
min	2016.0	1.000000	1.000000	1.000000	1.000000	7.000000	10397.000000	10397.000000	10.000000	1.000000	...	2.000000	1.000000	-67.000000	0.000000	0.000000	0.000000	93.000000	75.000000	509.000000	NaN
25%	2016.0	2.000000	4.000000	8.000000	2.000000	624.000000	10397.000000	10397.000000	905.000000	905.000000	...	1130.000000	1135.000000	-19.000000	0.000000	0.000000	0.000000	127.000000	117.000000	594.000000	NaN
50%	2016.0	3.000000	7.000000	16.000000	4.000000	1267.000000	12478.000000	12478.000000	1320.000000	1324.000000	...	1559.000000	1547.000000	-10.000000	0.000000	0.000000	0.000000	159.000000	149.000000	907.000000	NaN
75%	2016.0	3.000000	9.000000	23.000000	6.000000	2032.000000	13487.000000	13487.000000	1735.000000	1739.000000	...	1952.000000	1945.000000	1.000000	0.000000	0.000000	0.000000	255.000000	236.000000	1927.000000	NaN
max	2016.0	4.000000	12.000000	31.000000	7.000000	2853.000000	14747.000000	14747.000000	2359.000000	2400.000000	...	2359.000000	2400.000000	615.000000	1.000000	1.000000	1.000000	397.000000	428.000000	2422.000000	NaN

rows x 22 columns



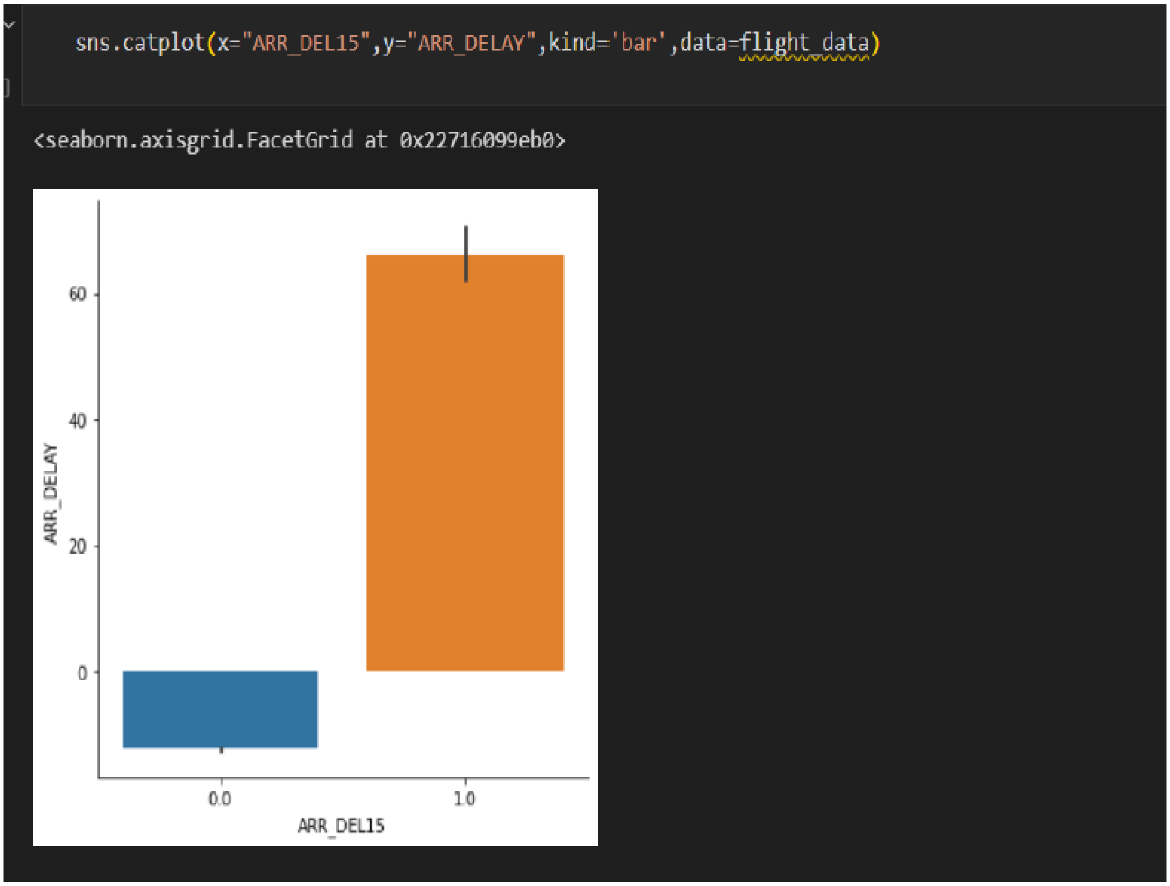
```
sns.distplot(flight_data.MONTH)
```

```
C:\Users\Saumya\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
figure-level function with similar flexibility) or `histplot` (an axes-leve
  warnings.warn(msg, FutureWarning)

<AxesSubplot:xlabel='MONTH', ylabel='Density'>
```



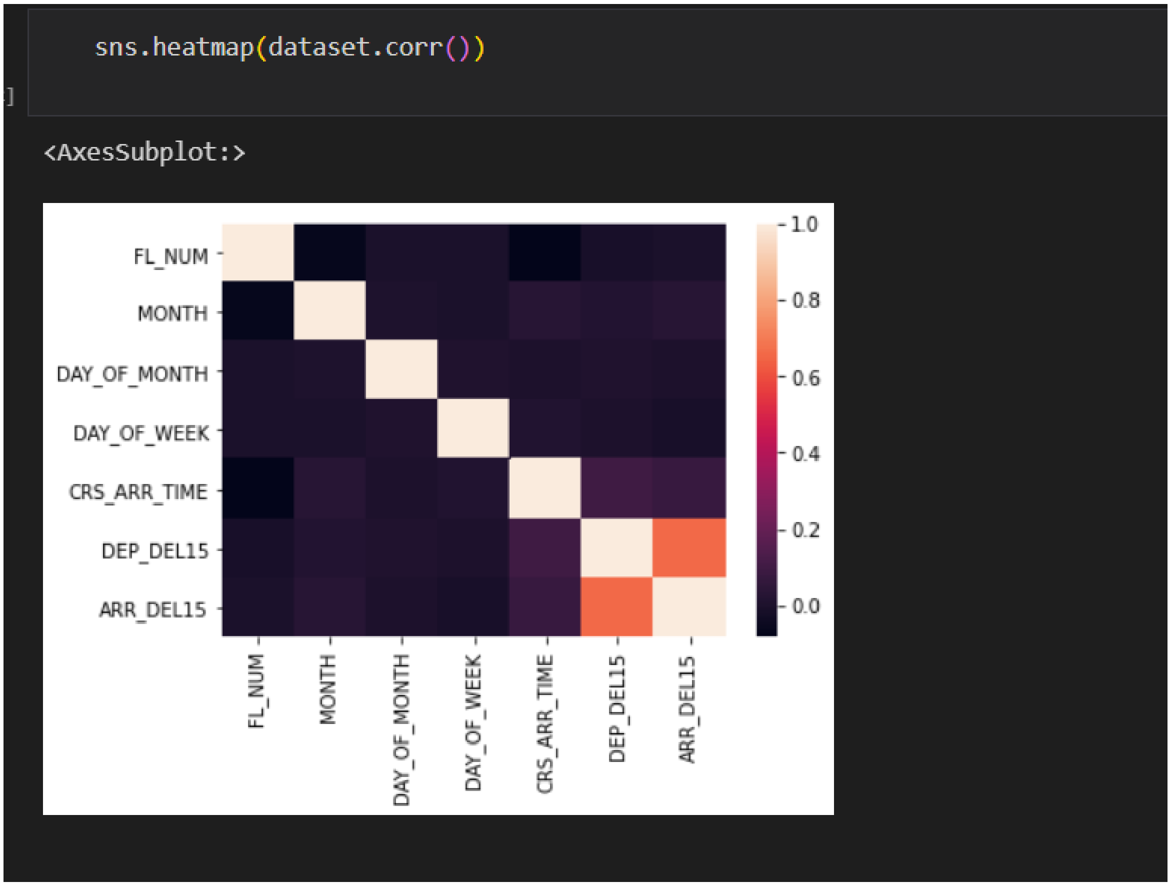
```
sns.scatterplot(x='ARR_DELAY',y='ARR_DEL15',data=flight_data)
```

```
<AxesSubplot:xlabel='ARR_DELAY', ylabel='ARR_DEL15'>
```



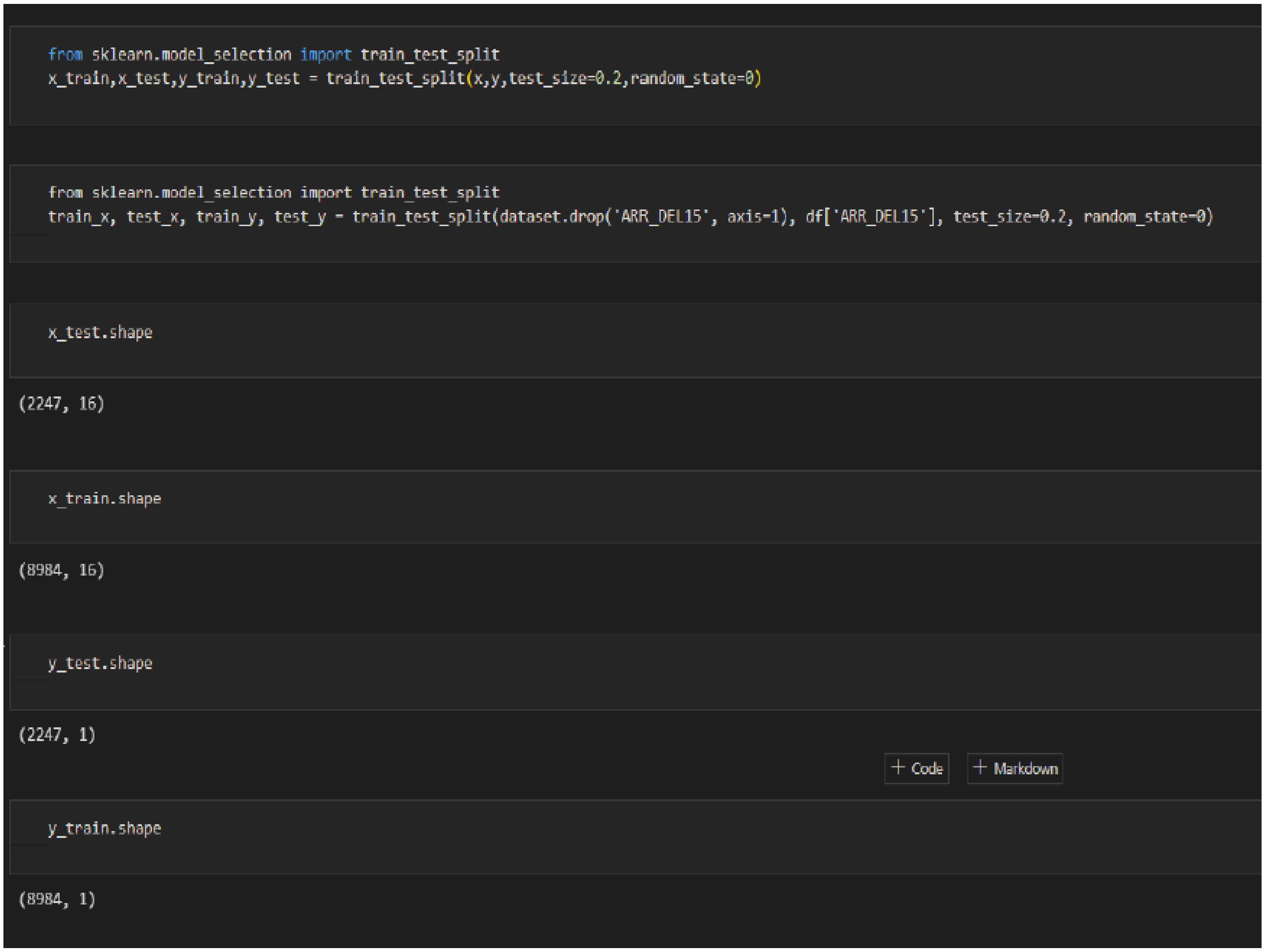
```
sns.catplot(x="ARR_DEL15",y="ARR_DELAY",kind='bar',data=flight_data)
```

```
<seaborn.axisgrid.FacetGrid at 0x22716099eb0>
```



```
sns.heatmap(dataset.corr())
```

```
<AxesSubplot:>
```



```
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)
```

```
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), df['ARR_DEL15'], test_size=0.2, random_state=0)
```

```
x_test.shape
```

```
(2247, 16)
```

```
x_train.shape
```

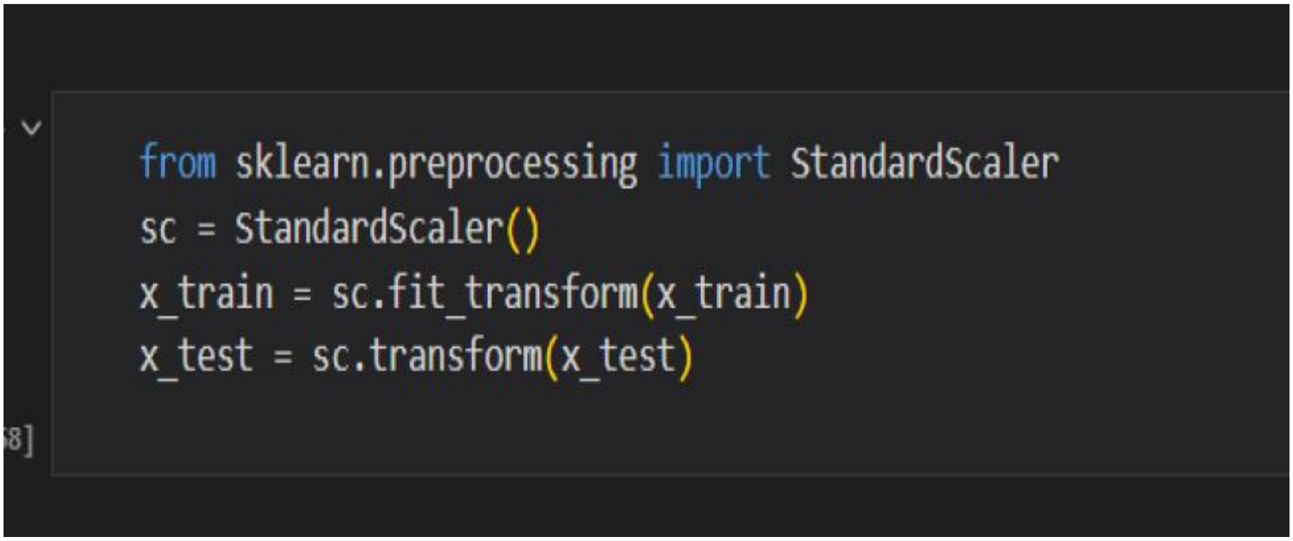
```
(8984, 16)
```

```
y_test.shape
```

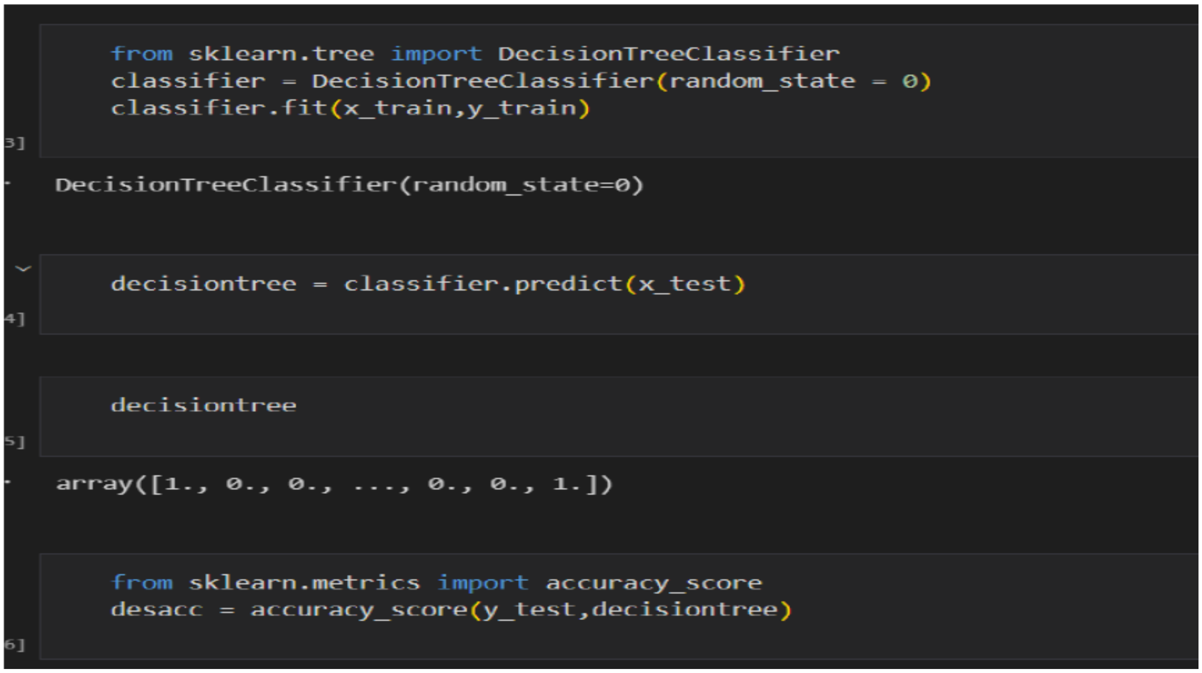
```
(2247, 1)
```

```
y_train.shape
```

```
(8984, 1)
```



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



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

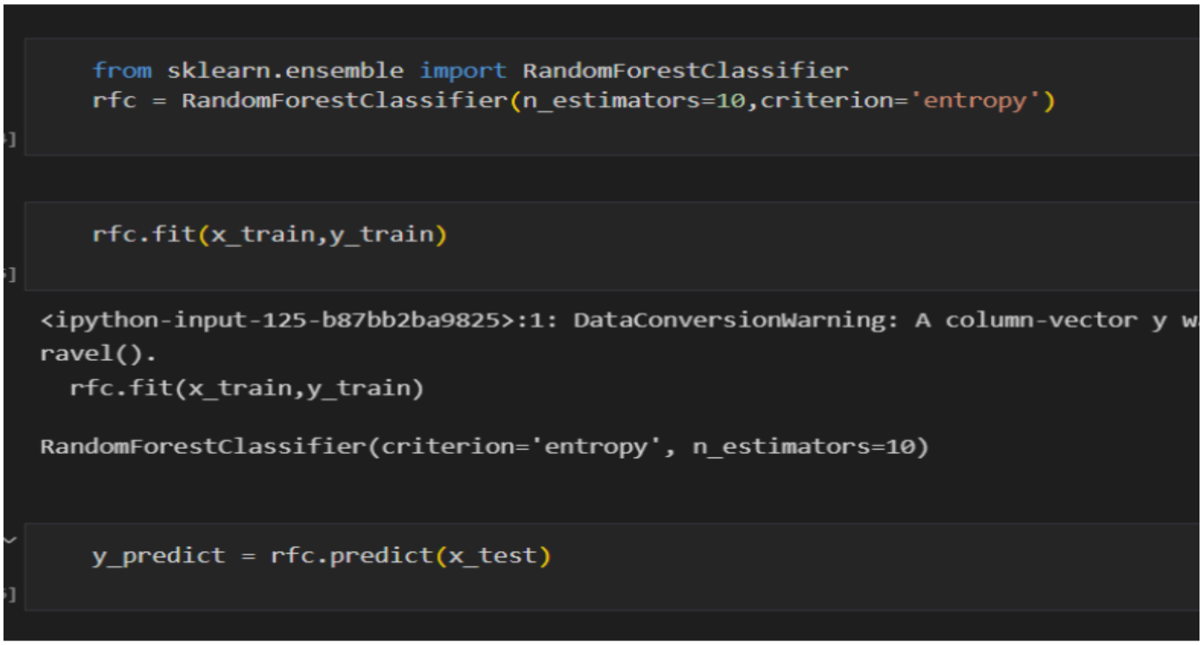
```
DecisionTreeClassifier(random_state=0)
```

```
decisiontree = classifier.predict(x_test)
```

```
decisiontree
```

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

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

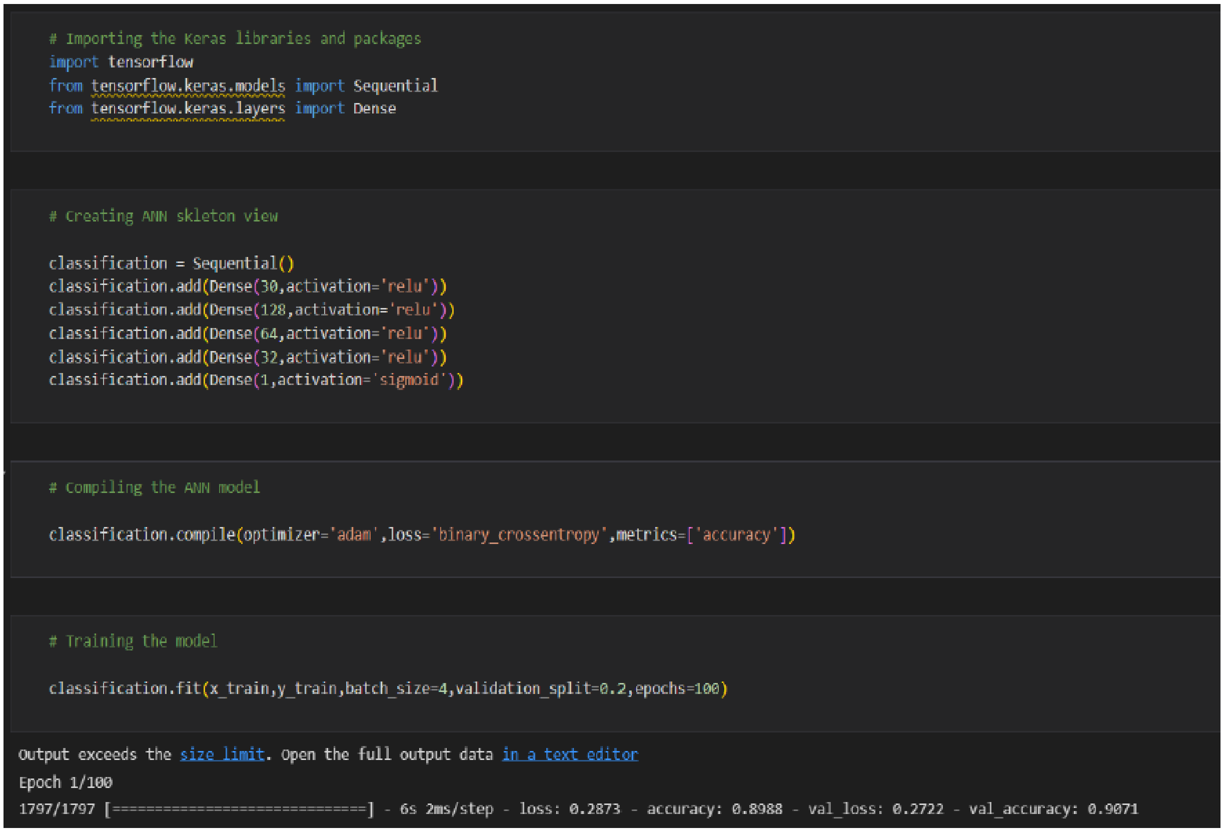


```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10,criterion='entropy')
```

```
rfc.fit(x_train,y_train)
```

```
<ipython-input-125-b87bb2ba9825>:1: DataConversionWarning: A column-vector y w
ravel().
  rfc.fit(x_train,y_train)

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

```
y_predict = rfc.predict(x_test)
```



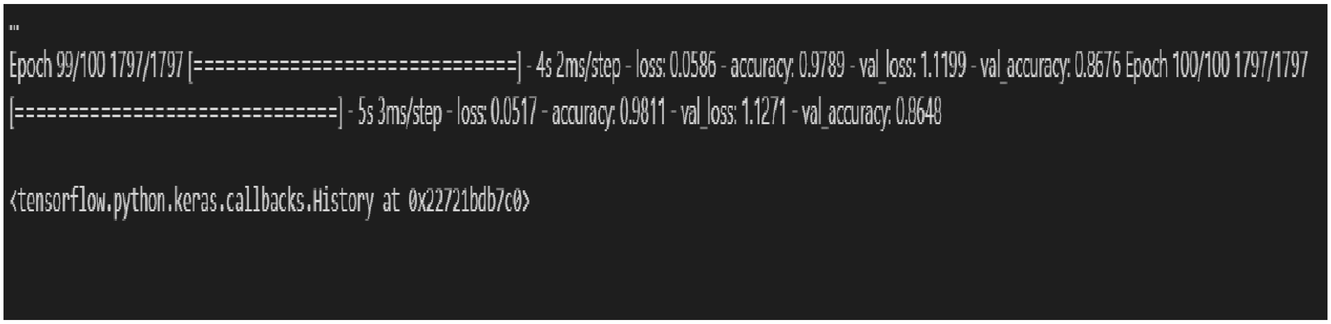
```
# Importing the Keras libraries and packages
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Creating ANN skleton view

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'))
```

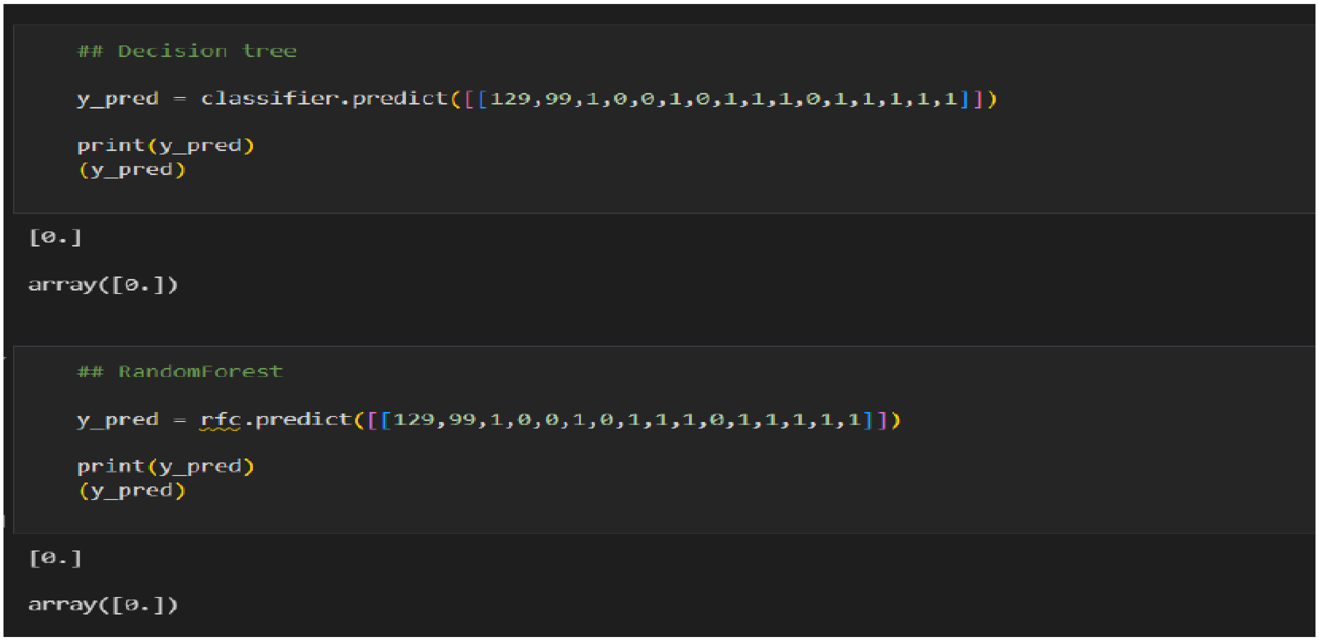
```
# Compiling the ANN model

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

```
# Training the model

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

```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/100
1797/1797 [==============================] - 6s 2ms/step - loss: 0.2873 - accuracy: 0.8988 - val_loss: 0.2722 - val_accuracy: 0.9071
```



```
...
Epoch 99/100 1797/1797 [==============================] - 4s 2ms/step - loss: 0.0586 - accuracy: 0.9789 - val_loss: 1.1199 - val_accuracy: 0.8676 Epoch 100/100 1797/1797
[==============================] - 5s 3ms/step - loss: 0.0517 - accuracy: 0.9811 - val_loss: 1.1271 - val_accuracy: 0.8648

<tensorflow.python.keras.callbacks.History at 0x22721bdb7c0>
```

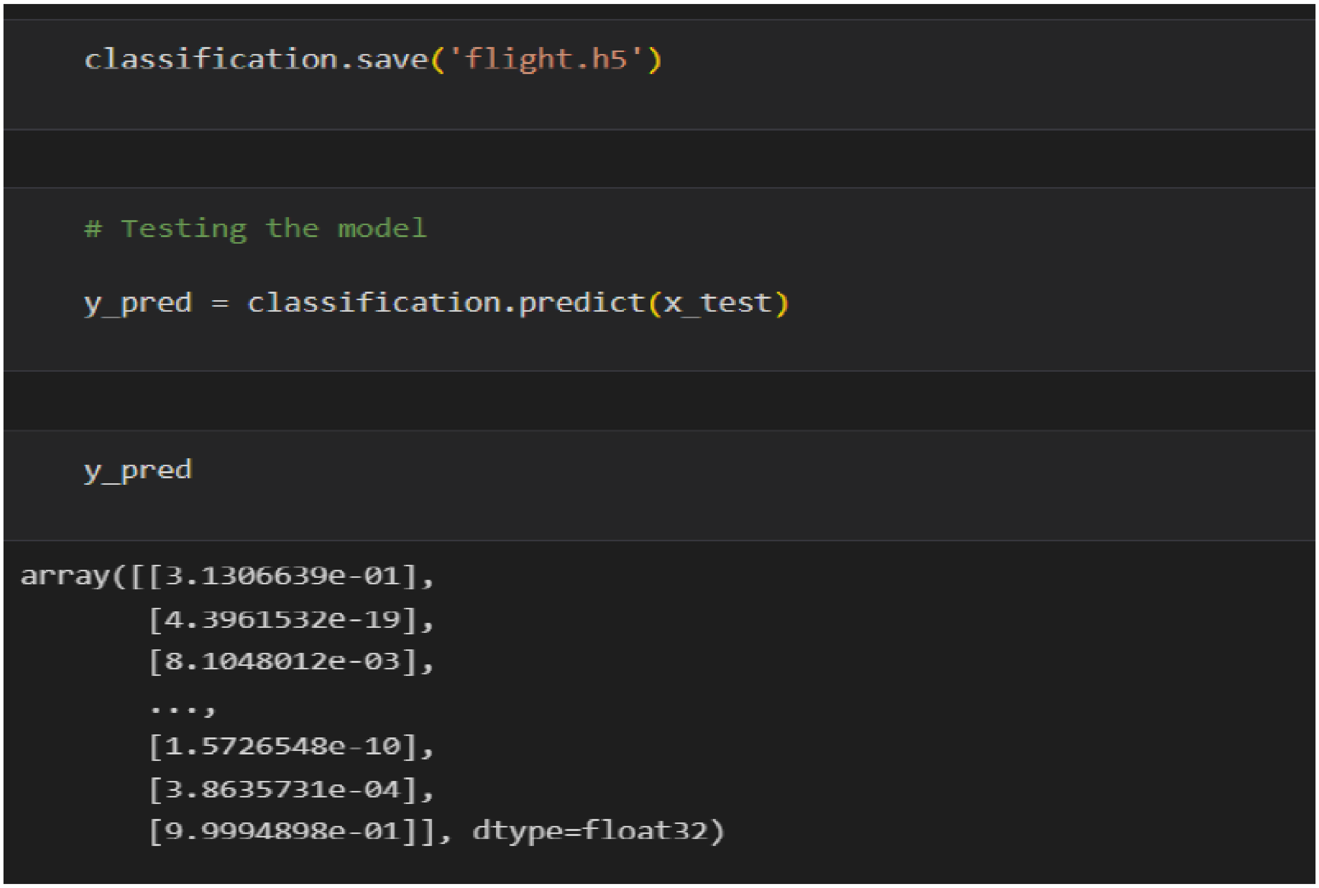


```
## Decision tree

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

print(y_pred)
(y_pred)
```

```
[0.]

array([0.])
```

```
## RandomForest

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

print(y_pred)
(y_pred)
```

```
[0.]

array([0.])
```

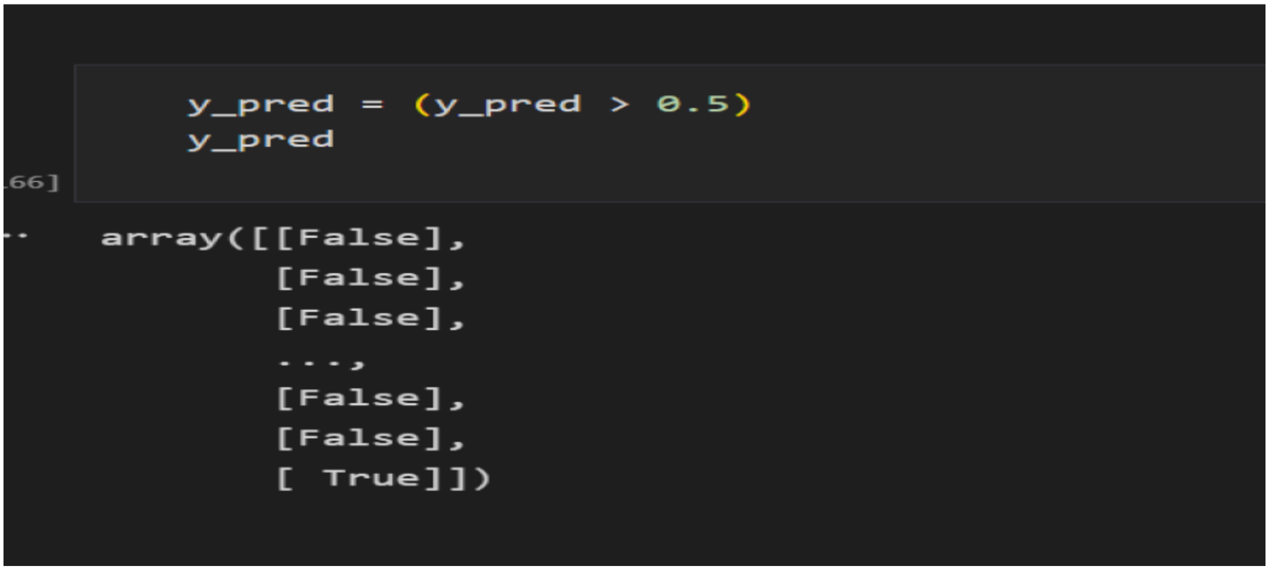


```
classification.save('flight.h5')
```

```
# Testing the model

y_pred = classification.predict(x_test)
```

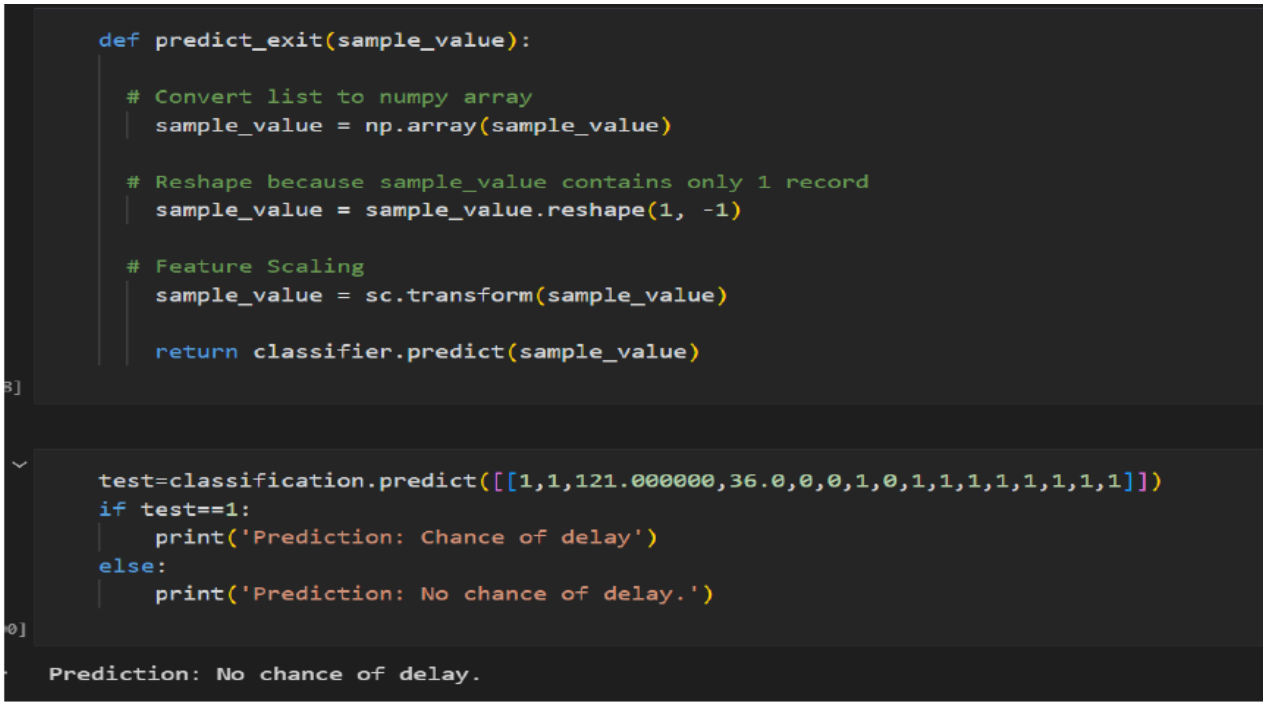
```
y_pred
```

```
array([[3.1306639e-01],
       [4.3961532e-19],
       [8.1048012e-03],
       ...,
       [1.5726548e-10],
       [3.8635731e-04],
       [9.9994898e-01]], dtype=float32)
```



```
y_pred = (y_pred > 0.5)
y_pred
```

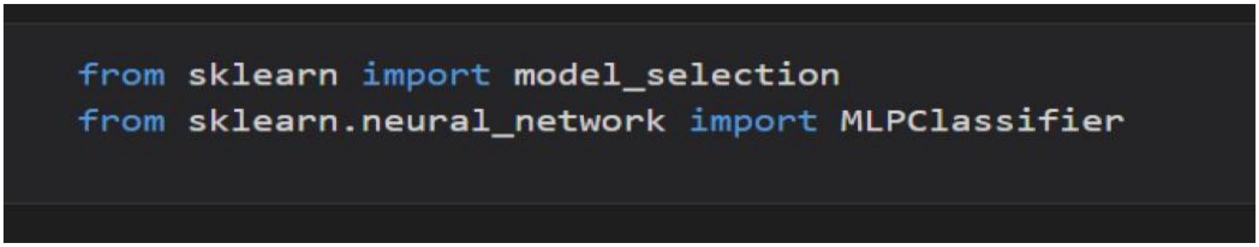
```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [ True]])
```



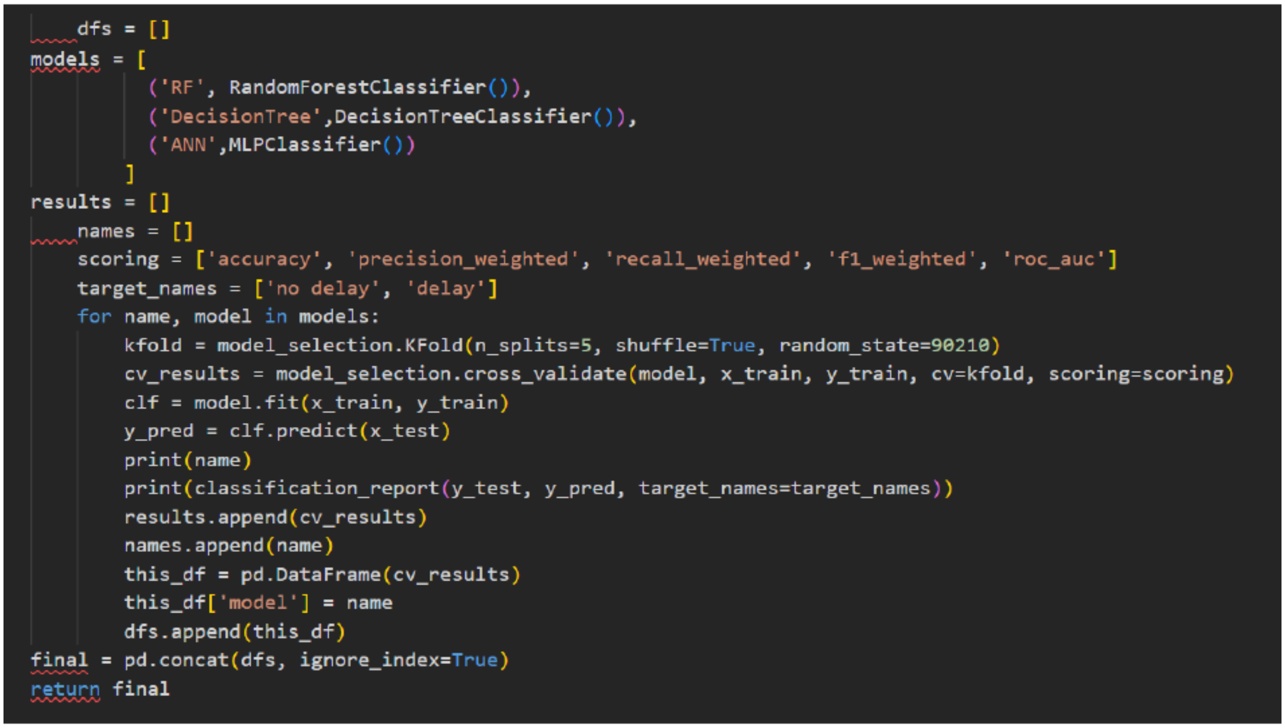
```
def predict_exit(sample_value):

  # Convert list to numpy array
  sample_value = np.array(sample_value)

  # Reshape because sample_value contains only 1 record
  sample_value = sample_value.reshape(1, -1)

  # Feature Scaling
  sample_value = sc.transform(sample_value)

  return classifier.predict(sample_value)
```

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

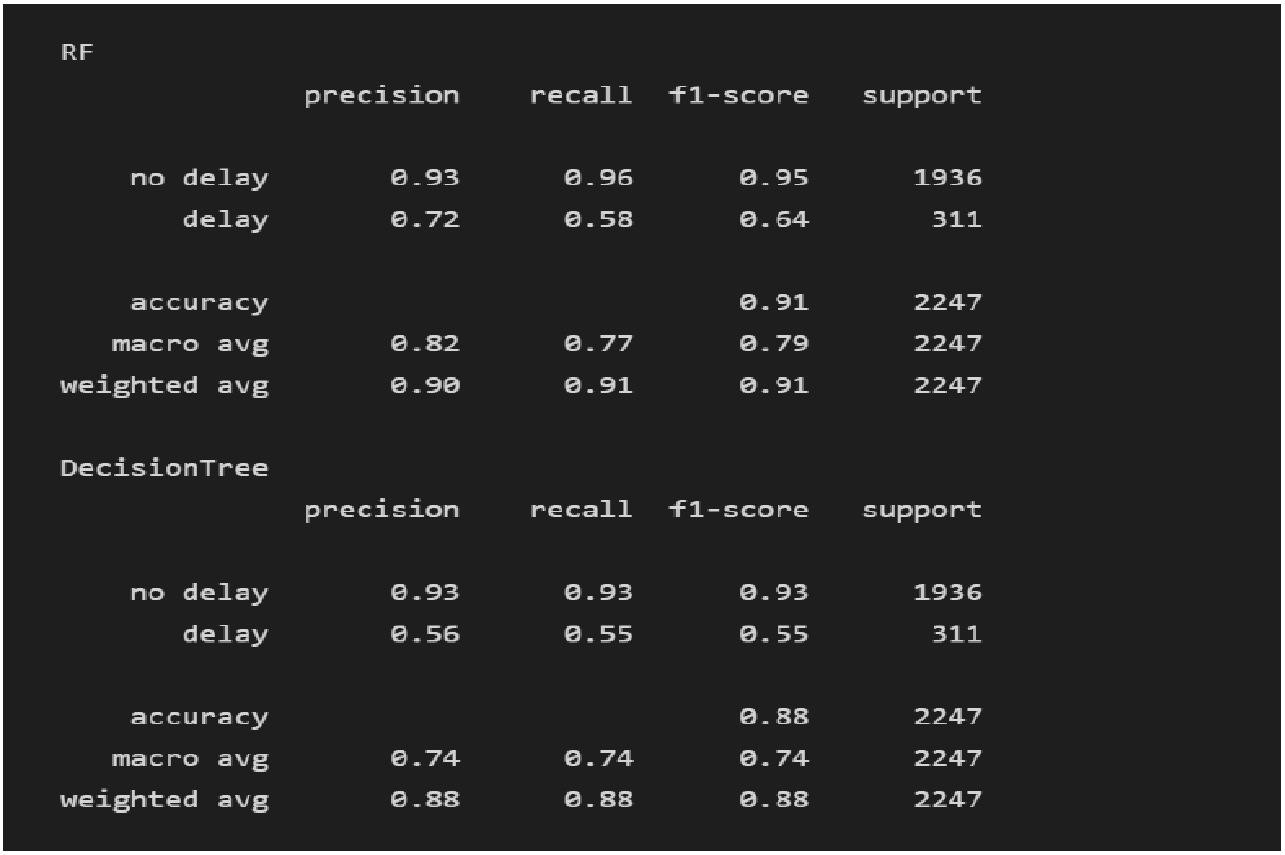
```
Prediction: No chance of delay.
```



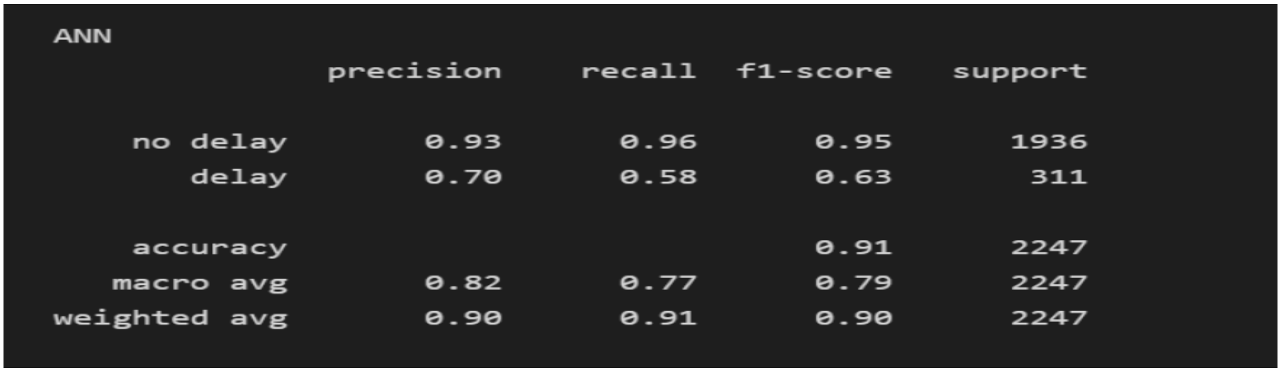
```
from sklearn import model_selection
from sklearn.neural_network import MLPClassifier
```



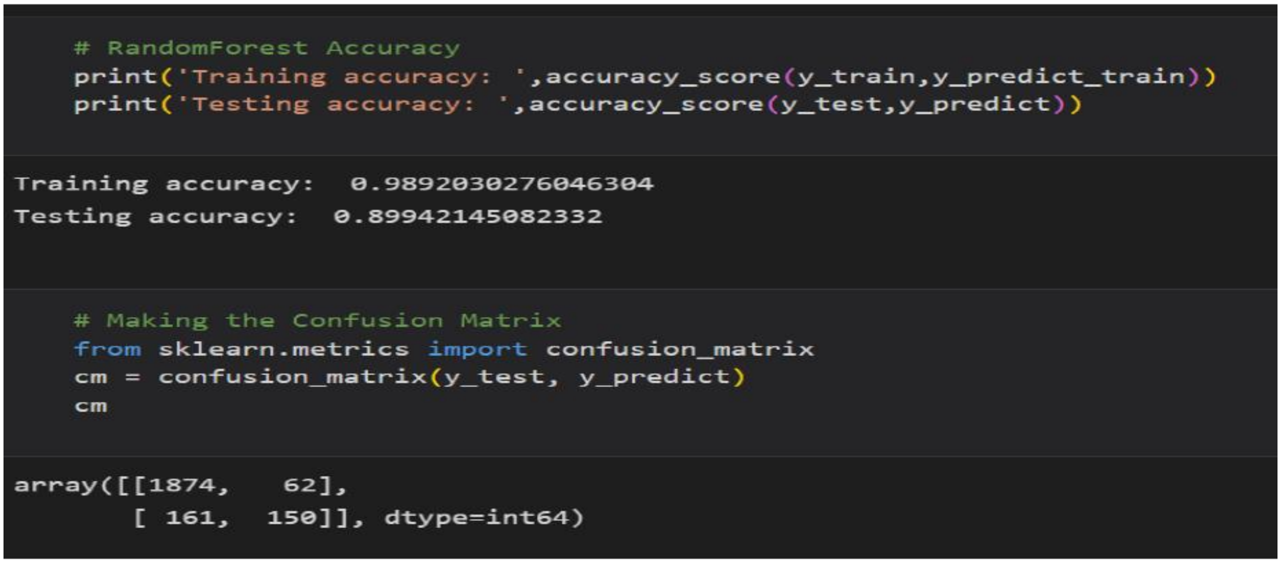
```
    dfs = []
models = [
          ('RF', RandomForestClassifier()),
          ('DecisionTree',DecisionTreeClassifier()),
          ('ANN',MLPClassifier())
        ]
results = []
    names = []
    scoring = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted', 'roc_auc']
    target_names = ['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       0.93      0.96      0.95      1936
       delay       0.72      0.58      0.64       311

    accuracy                           0.91      2247
   macro avg       0.82      0.77      0.79      2247
weighted avg       0.90      0.91      0.91      2247

DecisionTree
              precision    recall  f1-score   support

    no delay       0.93      0.93      0.93      1936
       delay       0.56      0.55      0.55       311

    accuracy                           0.88      2247
   macro avg       0.74      0.74      0.74      2247
weighted avg       0.88      0.88      0.88      2247
```



```
ANN
              precision    recall  f1-score   support

    no delay       0.93      0.96      0.95      1936
       delay       0.70      0.58      0.63       311

    accuracy                           0.91      2247
   macro avg       0.82      0.77      0.79      2247
weighted avg       0.90      0.91      0.90      2247
```

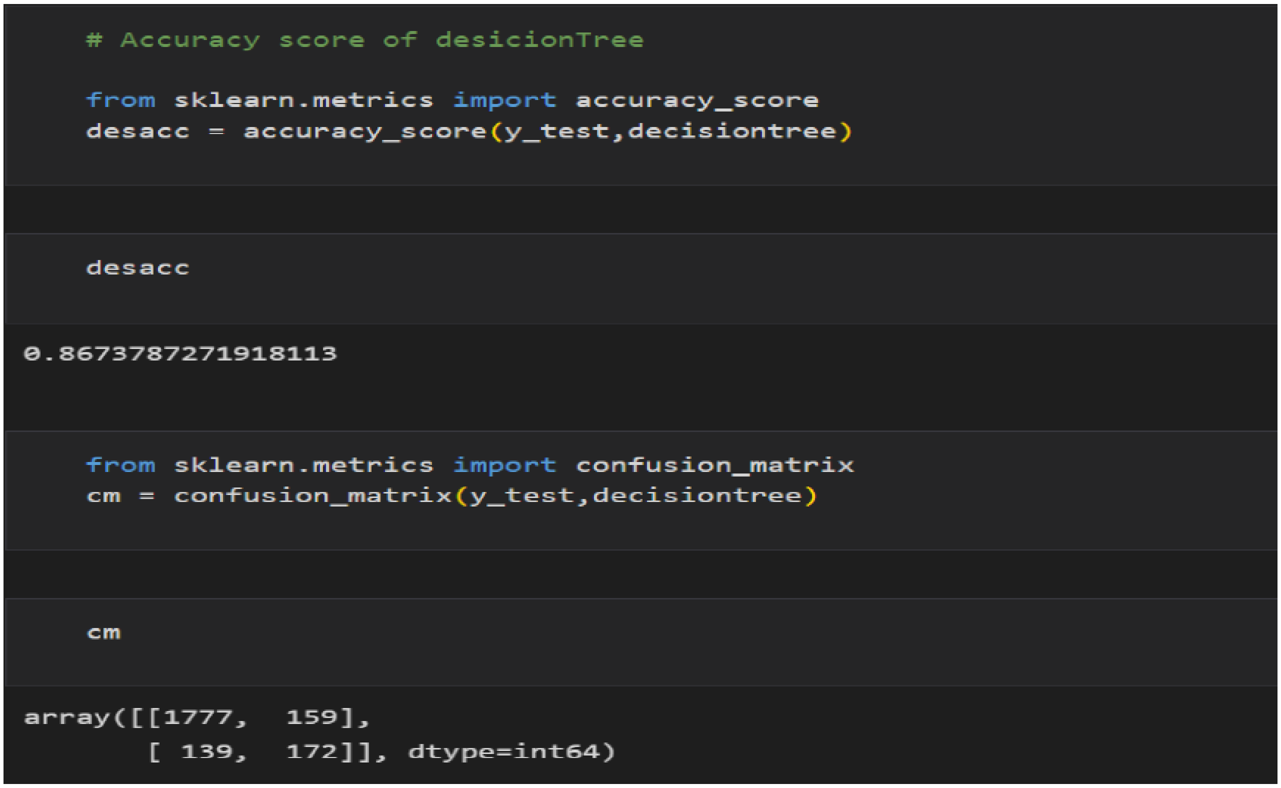


```
# RandomForest Accuracy
print('Training accuracy: ',accuracy_score(y_train,y_predict_train))
print('Testing accuracy: ',accuracy_score(y_test,y_predict))
```

```
Training accuracy:  0.9892030276046304
Testing accuracy:  0.89942145082332
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[1874,   62],
       [ 161,  150]], dtype=int64)
```



```
# Accuracy score of desicionTree

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

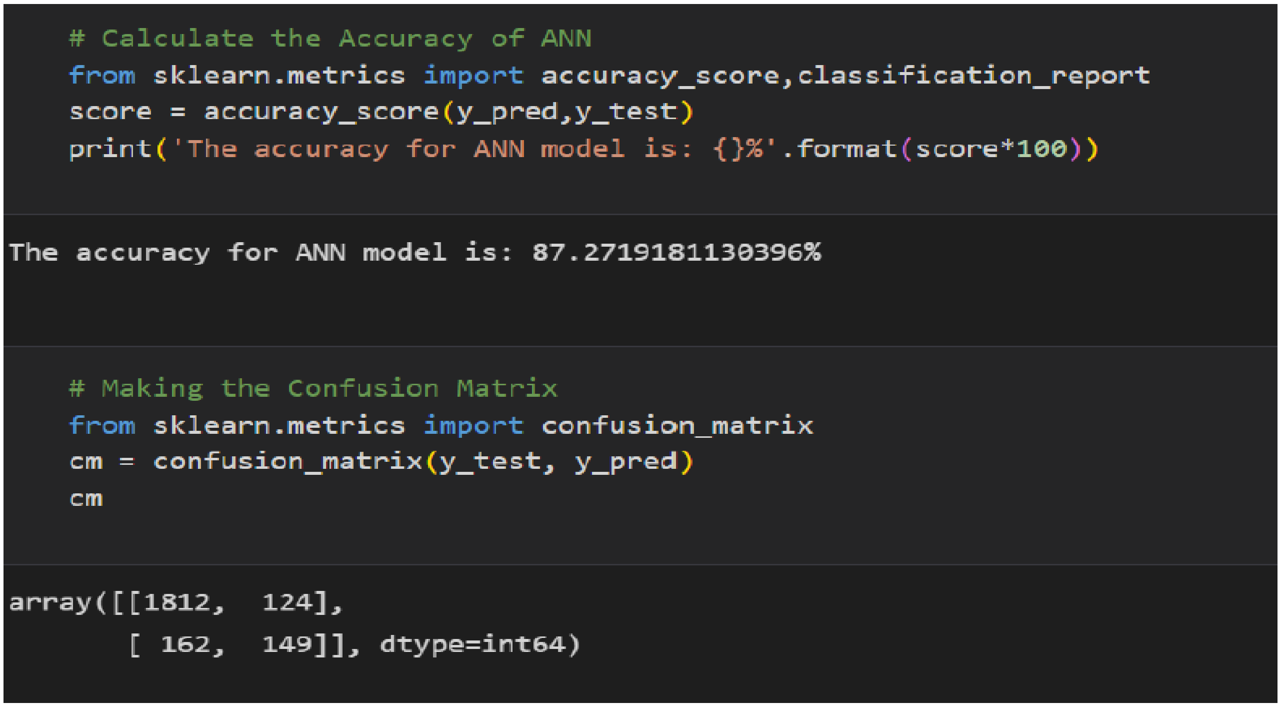
```
desacc
```

```
0.8673787271918113
```

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

```
cm
```

```
array([[1777,  159],
       [ 139,  172]], dtype=int64)
```

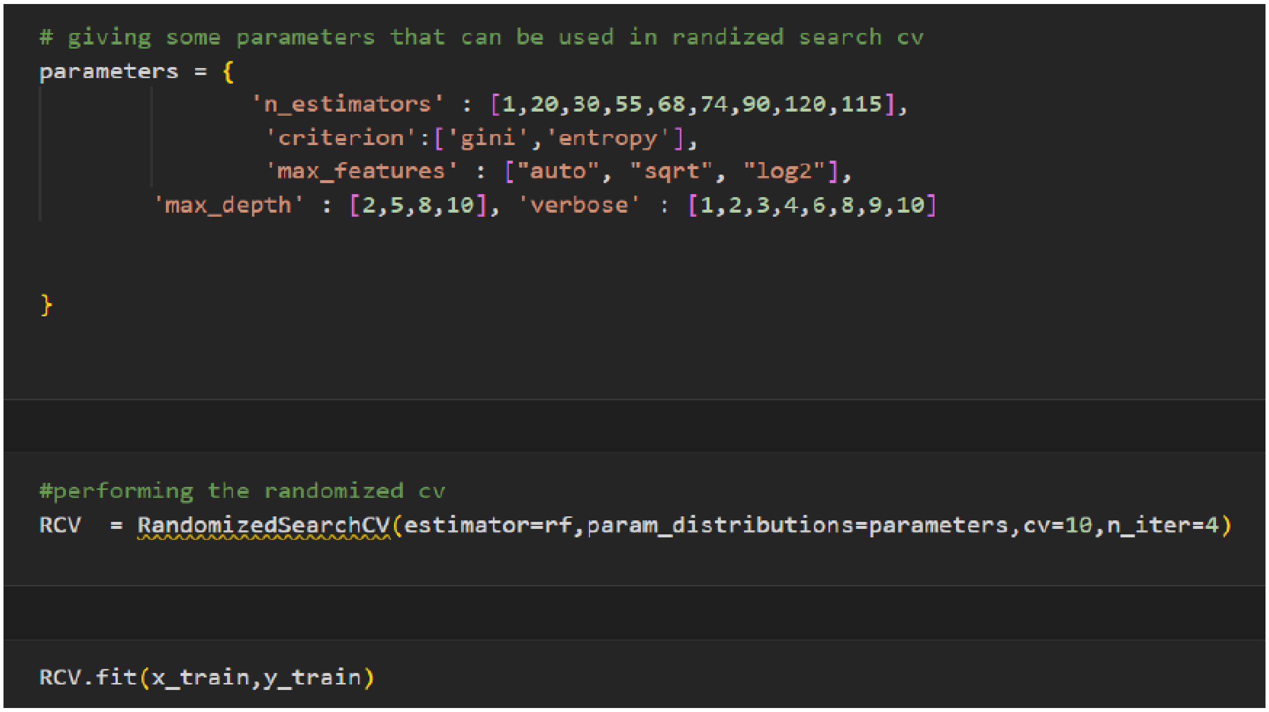


```
# Calculate the Accuracy of ANN
from sklearn.metrics import accuracy_score,classification_report
score = accuracy_score(y_pred,y_test)
print('The accuracy for ANN model is: {}%'.format(score*100))
```

```
The accuracy for ANN model is: 87.2719181130396%
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

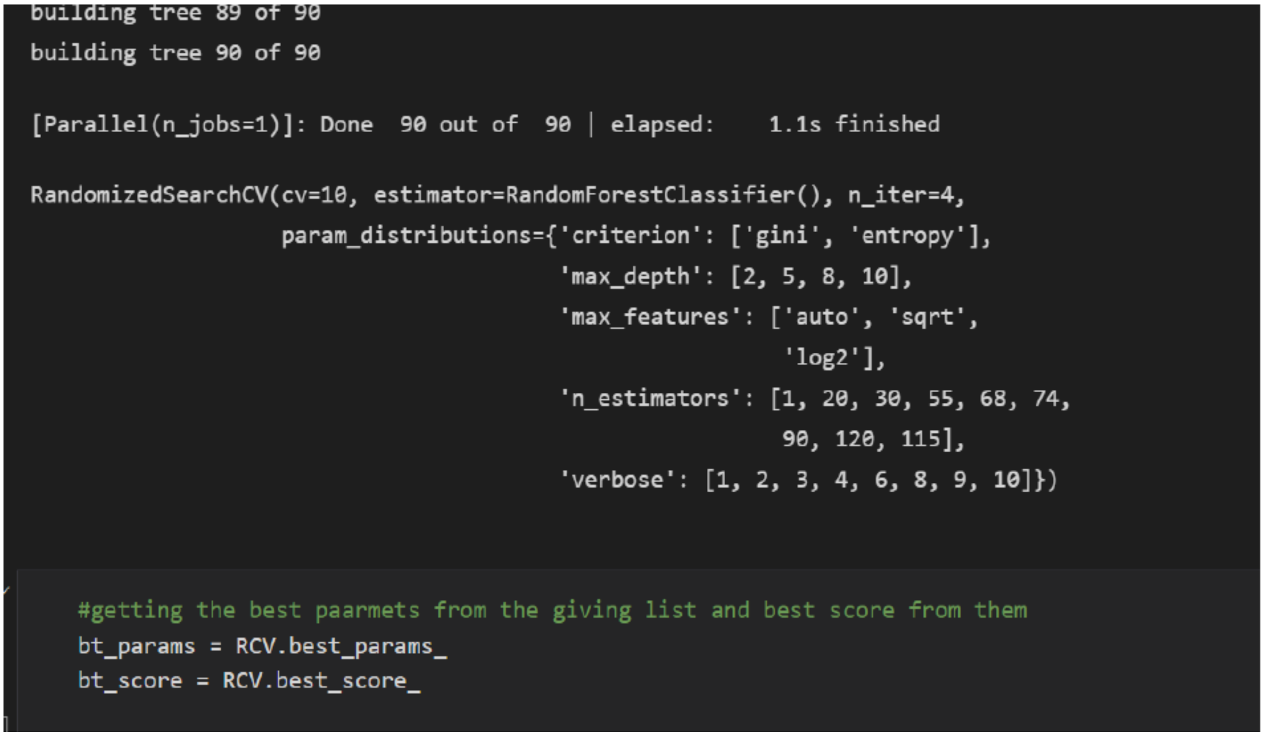
```
array([[1812,  124],
       [ 162,  149]], dtype=int64)
```



```
# giving some parameters that can be used in randized search cv
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' : [1,2,3,4,6,8,9,10]


}
```

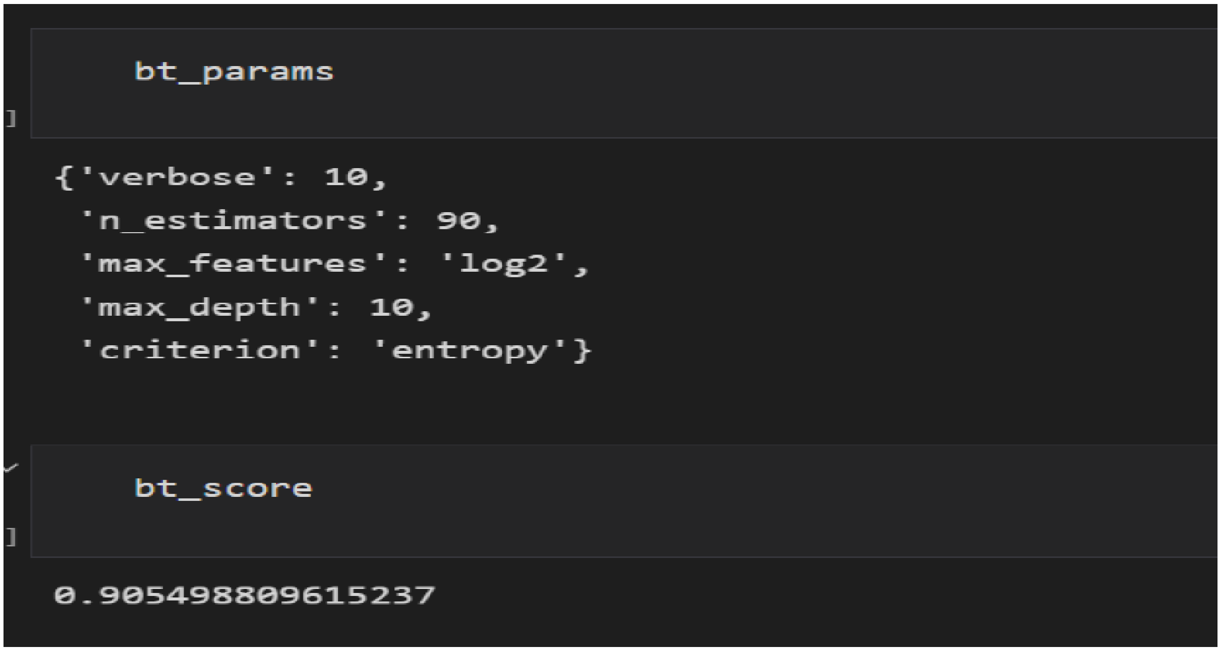
```
#performing the randomized cv
RCV  = RandomizedSearchCV(estimator=rf,param_distributions=parameters,cv=10,n_iter=4)
```

```
RCV.fit(x_train,y_train)
```



```
building tree 89 of 90
building tree 90 of 90

[Parallel(n_jobs=1)]: Done  90 out of  90 | elapsed:    1.1s finished

RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(), n_iter=4,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [2, 5, 8, 10],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'n_estimators': [1, 20, 30, 55, 68, 74,
                                                         90, 120, 115],
                                        'verbose': [1, 2, 3, 4, 6, 8, 9, 10]})
```

```
#getting the best paarmets from the giving list and best score from them
bt_params = RCV.best_params_
bt_score = RCV.best_score_
```

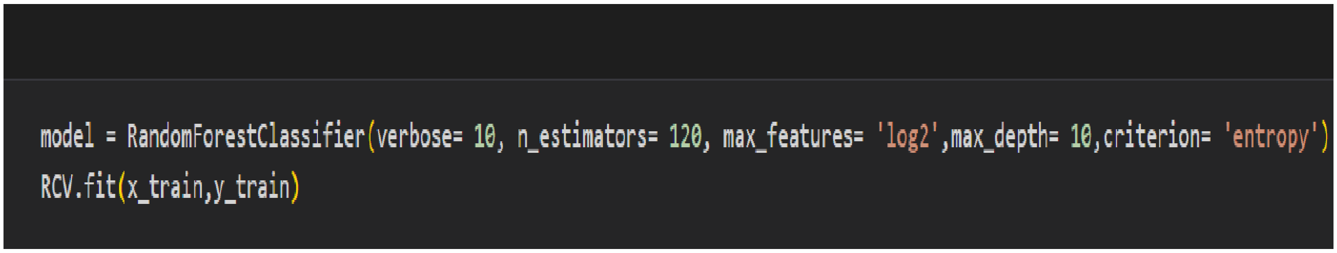


```
bt_params
```

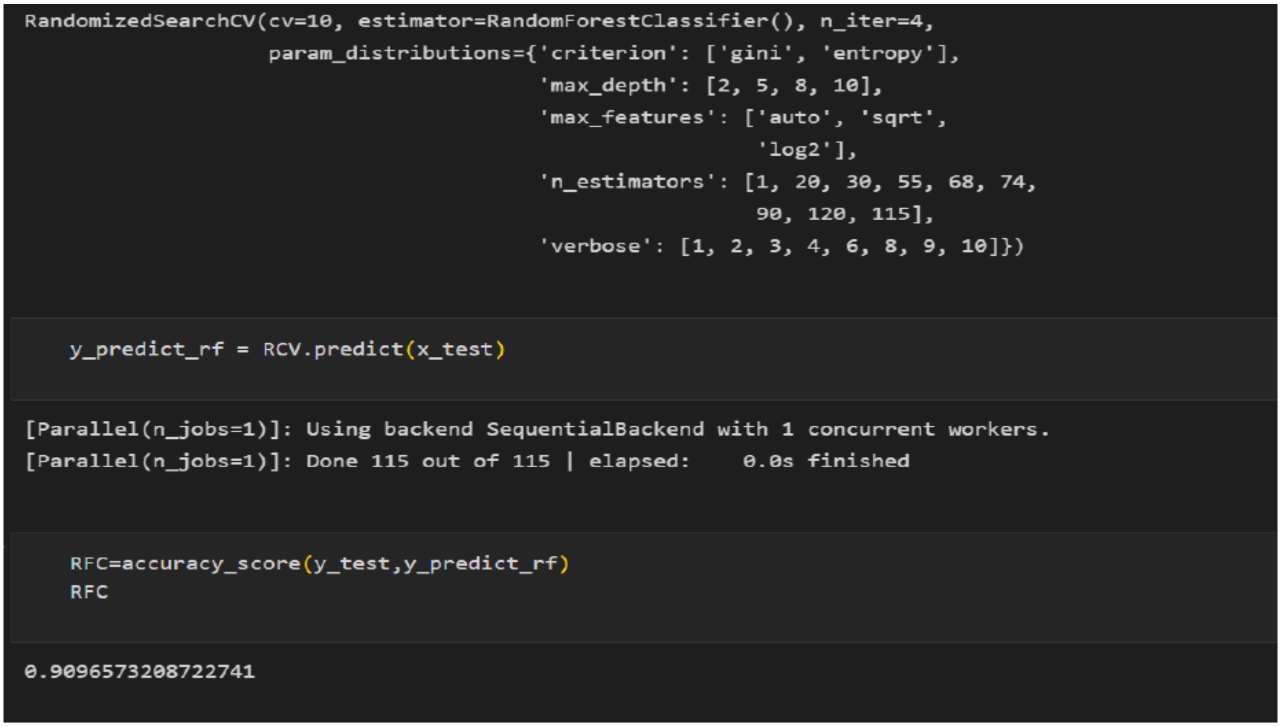
```
{'verbose': 10,
 'n_estimators': 90,
 'max_features': 'log2',
 'max_depth': 10,
 'criterion': 'entropy'}
```

```
bt_score
```

```
0.905498809615237
```



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



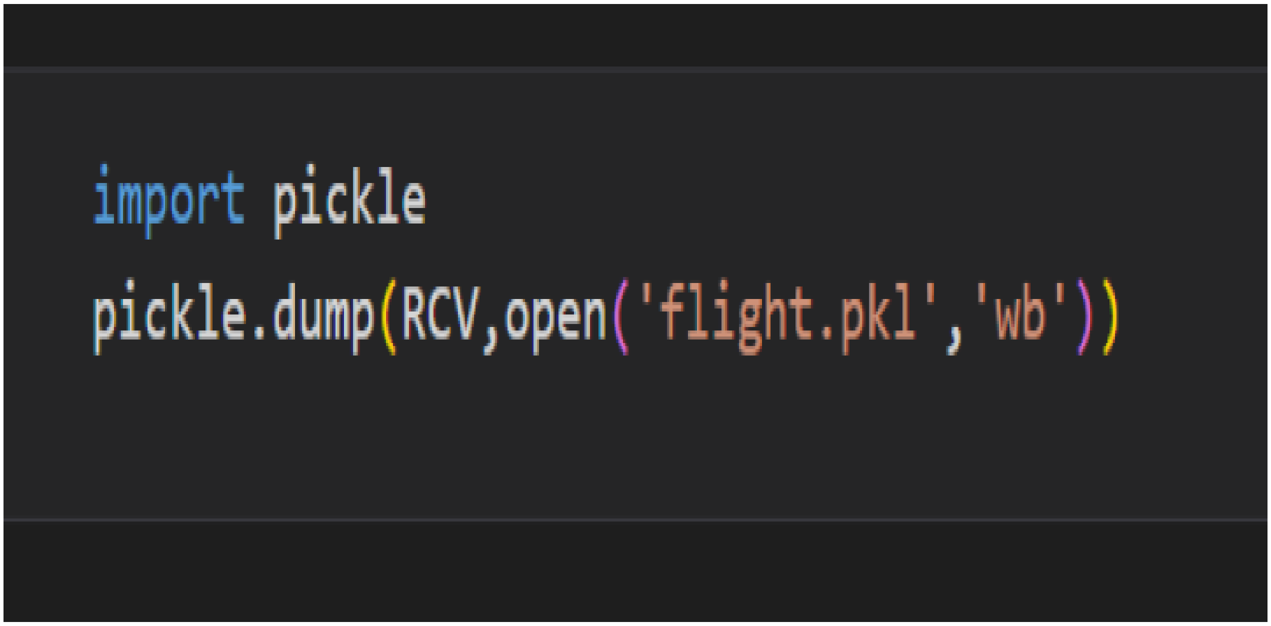
```
RandomizedSearchCV(cv=10, estimator=RandomForestClassifier(), n_iter=4,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [2, 5, 8, 10],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'n_estimators': [1, 20, 30, 55, 68, 74,
                                                         90, 120, 115],
                                        'verbose': [1, 2, 3, 4, 6, 8, 9, 10]})
```

```
y_predict_rf = RCV.predict(x_test)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 115 out of 115 | elapsed:    0.0s finished
```

```
RFC=accuracy_score(y_test,y_predict_rf)
RFC
```

```
0.9096573208722741
```



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