FLIGHT DELAY PREDICTION SYSTEM

BUAN 6341.501(FALL 2022) – APPLIED MACHINE LEARNING

Presented to Professor Zixuan Meng

Group7

Sangram Tushar Mohite Patil (SXM210110)

Saurabh Dhananjay Jadhav (SDJ200000)

Ananya Reddy Aegumamidi (AXA210029)

Harshal Mahadik (HXM190046)

Ishika Bhargava (IXB200014)

TABLE OF CONTENTS

Table of Contents

Introduction

In the United States, the airlines reported 20.1% of the domestic flights were delayed by 15 minutes or more. The financial losses that the aviation industry is going through are primarily because of flight delays. They inconvenience airlines and passengers as the increased travel time increases the expenses associated with the passenger's accommodation and sometimes causes stress among passengers. The delays further affect the air traffic at the airport. These delays may arise due to air congestion, weather conditions, difficulties boarding passengers, or mechanical problems. So, what can be done as a passenger to avoid delayed flights? Is it possible to know if the flight gets delayed before it comes up on the departure boards? Or before the passenger is inside the plane? The answer to these questions is yes.

Objective & Approach

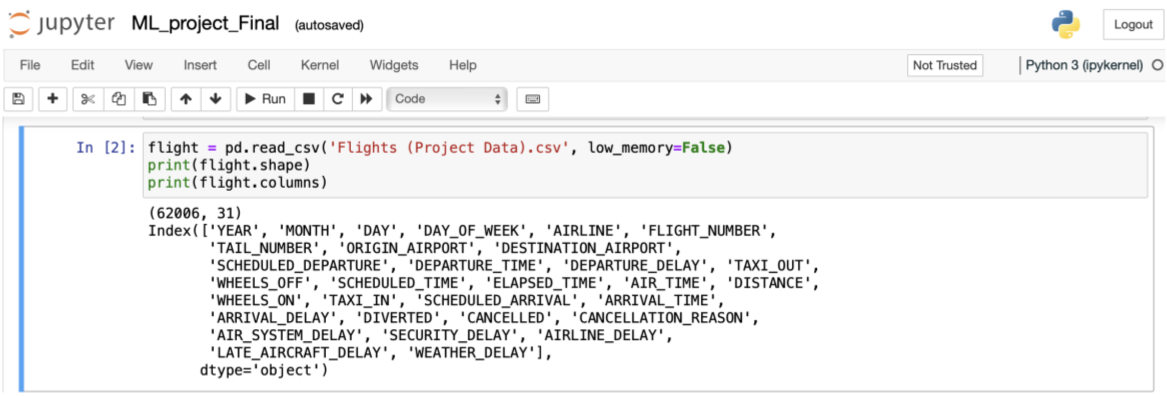
This project aims to predict the flight delay using the full spectrum of available data. This project will: Apply and compare different ML models to get the best accuracy for the predictions, deliver flight delay predictions, i.e., for a specific flight on a particular day, giving the probability of the flight delay and Alert travelers that the flight might get delayed even before it arrives.

This project follows a standard approach which helps to predict the flight delay using efficient techniques. Methods that have been performed in a sequential manner to get the best possible results are Reading the data, Data cleaning & Pre-processing, Exploratory Data Analysis, Visualization, Model Building, Finalize and Validate the Data model, and Making Prediction.

Data cleaning & Preprocessing

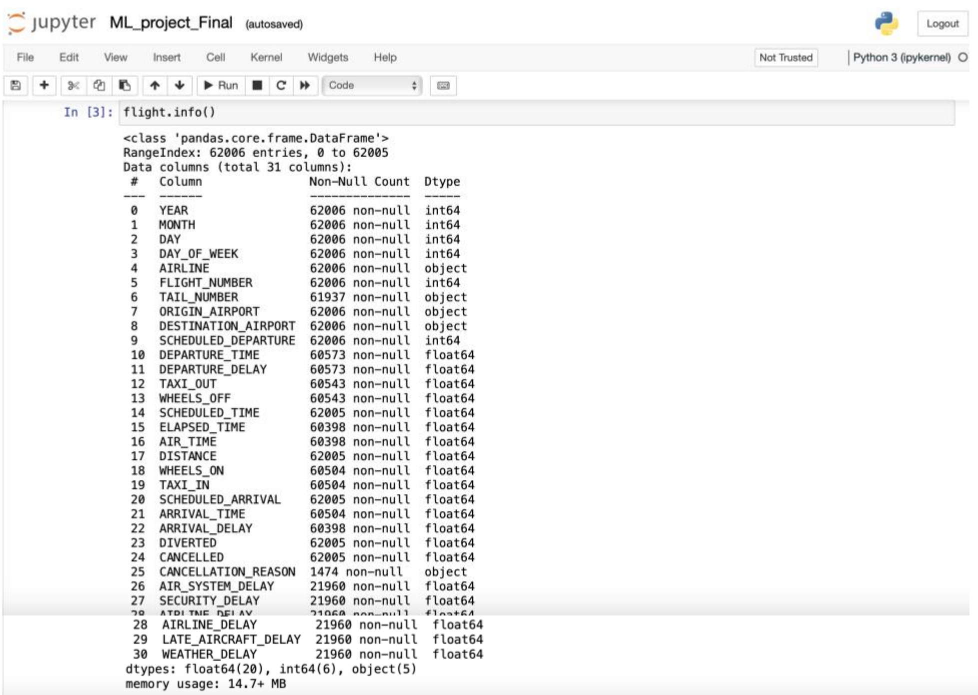
The first step included finding a good data set with all the initial requirements which could help us to get to the achievement of project objective. Our data is sourced from Kaggle. This dataset is of Monthly Air Travel Consumer Report from the Department of Transportation (DOT) from 2015 which consists of a summary of the number of on-time, delayed, cancelled, and diverted flights Following are three files (Airlines.csv, Airports.csv, Flights.csv) consume all the data and there are approximately 5819079 number of observations and 40 columns.

We must do some data cleaning before we can get to the actual analysis for our project. This dataset has a few issues that would make analysis difficult, so this section aims to fix those problems. While this section is mainly data cleaning, there is some exploratory data analysis for the purpose of examining how to best clean the data. The first thing we do is to read data set Flights.csv for predicting.



```
In [2]: flight = pd.read_csv('Flights (Project Data).csv', low_memory=False)
        print(flight.shape)
        print(flight.columns)

(62006, 31)
Index(['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'AIRLINE', 'FLIGHT_NUMBER',
       'TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT',
       'SCHEDULED_DEPARTURE', 'DEPARTURE_TIME', 'DEPARTURE_DELAY', 'TAXI_OUT',
       'WHEELS_OFF', 'SCHEDULED_TIME', 'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE',
       'WHEELS_ON', 'TAXI_IN', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME',
       'ARRIVAL_DELAY', 'DIVERTED', 'CANCELLED', 'CANCELLATION_REASON',
       'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY',
       'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY'],
      dtype='object')
```

The next thing we do is we have taken an initial look at the dataset to get an understanding of it. Initially there are 62006 observations and 31 columns.



```
In [3]: flight.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62006 entries, 0 to 62005
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   YEAR                 62006 non-null  int64
 1   MONTH                62006 non-null  int64
 2   DAY                  62006 non-null  int64
 3   DAY_OF_WEEK          62006 non-null  int64
 4   AIRLINE              62006 non-null  object
 5   FLIGHT_NUMBER        62006 non-null  int64
 6   TAIL_NUMBER          61937 non-null  object
 7   ORIGIN_AIRPORT       62006 non-null  object
 8   DESTINATION_AIRPORT  62006 non-null  object
 9   SCHEDULED_DEPARTURE  62006 non-null  int64
 10  DEPARTURE_TIME       60573 non-null  float64
 11  DEPARTURE_DELAY      60573 non-null  float64
 12  TAXI_OUT             60543 non-null  float64
 13  WHEELS_OFF           60543 non-null  float64
 14  SCHEDULED_TIME       62005 non-null  float64
 15  ELAPSED_TIME         60398 non-null  float64
 16  AIR_TIME             60398 non-null  float64
 17  DISTANCE             62005 non-null  float64
 18  WHEELS_ON            60504 non-null  float64
 19  TAXI_IN              60504 non-null  float64
 20  SCHEDULED_ARRIVAL    62005 non-null  float64
 21  ARRIVAL_TIME         60504 non-null  float64
 22  ARRIVAL_DELAY        60398 non-null  float64
 23  DIVERTED             62005 non-null  float64
 24  CANCELLED            62005 non-null  float64
 25  CANCELLATION_REASON  1474 non-null   object
 26  AIR_SYSTEM_DELAY     21960 non-null  float64
 27  SECURITY_DELAY       21960 non-null  float64
 28  AIRLINE_DELAY        21960 non-null  float64
 29  LATE_AIRCRAFT_DELAY  21960 non-null  float64
 30  WEATHER_DELAY        21960 non-null  float64
dtypes: float64(20), int64(6), object(5)
memory usage: 14.7+ MB
```

For Modelling purposes, we have taken first 100k column records in the flight data and printing out to see all the columns available. The next thing we do is see if there are any NAs in the dataset, and if there are, r emove them.

Next, we will drop the useless columns that will not be helpful in the analysis. We have dropped the group of columns because they don't add value to machine learning values and also by doing this it improves the speed and efficiency of the model.

```
In [9]: # filtering out unnecessary columns for ML
        df=df.drop(['YEAR','FLIGHT_NUMBER','AIRLINE','DISTANCE','TAIL_NUMBER','TAXI_OUT',
                                                            'SCHEDULED_TIME','DEPARTURE_TIME','WHEELS_OFF','ELAPSED_TIME',
                                                            'AIR_TIME','WHEELS_ON','DAY_OF_WEEK','TAXI_IN','CANCELLATION_REASON'],
                                                           axis=1)
```

Now, we are filling Nan and 0 values with mean

```
In [10]: df=df.fillna(df.mean()) # filling 0 or nan values with mean
```

Now removing some more columns that will not be used in analysis and printing out the columns to reverify again.

```
In [15]: # removing some more columns
         df=df.drop(['ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', 'ARRIVAL_TIME', 'ARRIVAL_DELAY'],axis=1)
         df.head

Out[15]: <bound method NDFrame.head of        MONTH  DAY  SCHEDULED_DEPARTURE  DEPARTURE_DELAY  SCHEDULED_ARRIVAL  \
         0          1    1                    5       -11.000000         430.000000
         1          1    1                   10        -8.000000         750.000000
         2          1    1                   20        -2.000000         806.000000
         3          1    1                   20        -5.000000         805.000000
         4          1    1                   25        -1.000000         320.000000
         ...      ...  ...                  ...              ...                ...
         62001      1    4                 2155         5.000000        2353.000000
         62002      1    4                 2155        19.902646        2247.000000
         62003      1    4                 2155        47.000000        2355.000000
         62004      1    4                 2155        -2.000000        2206.000000
         62005      1    4                 2156        27.000000        1514.336666

                DIVERTED  CANCELLED  AIR_SYSTEM_DELAY  SECURITY_DELAY  AIRLINE_DELAY  \
         0      0.000000   0.000000         12.900729        0.084973      17.918169
         1      0.000000   0.000000         12.900729        0.084973      17.918169
         2      0.000000   0.000000         12.900729        0.084973      17.918169
         3      0.000000   0.000000         12.900729        0.084973      17.918169
         4      0.000000   0.000000         12.900729        0.084973      17.918169
         ...         ...        ...               ...             ...            ...
         62001  0.000000   0.000000         12.900729        0.084973      17.918169
         62002  0.000000   1.000000         12.900729        0.084973      17.918169
         62003  0.000000   0.000000          0.000000        0.000000      37.000000
         62004  0.000000   0.000000         12.900729        0.084973      17.918169
         62005  0.002145   0.023772         12.900729        0.084973      17.918169

                LATE_AIRCRAFT_DELAY  WEATHER_DELAY  result
         0                26.137477       1.932741       0
         1                26.137477       1.932741       0
         2                26.137477       1.932741       0
         3                26.137477       1.932741       0
         4                26.137477       1.932741       0
         ...                    ...            ...     ...
         62001            26.137477       1.932741       0
         62002            26.137477       1.932741       1
         62003             7.000000       0.000000       1
         62004            26.137477       1.932741       0
         62005            26.137477       1.932741       1

         [62006 rows x 13 columns]>
```

This concludes the data cleaning section. From here, we will move on to exploratory data analysis.

Exploratory Data Analysis

First and foremost, we counted the number of flights that got diverted and the below is the output for the same.

```
In [6]: df.value_counts('DIVERTED')

Out[6]: DIVERTED
        0.0    61872
        1.0      133
        dtype: int64
```

Next, finding number of delayed flights and the count of number of delayed flights are taken from the first 100k records.

Finding the flights delayed we have created a list that is result, where we find: if the value is greater than 15, then true is appended to the list and if it is false 0 is appended to the list.

```
In [11]: #0 flight not delayed 1 flight is delayed
         result=[]
         for row in df['ARRIVAL_DELAY']:
             if row > 15:
                 result.append(1)
             else:
                 result.append(0)
```

```
In [12]: df['result'] = result
```

Now counting number of times, the values of 0 and 1 occurred in the result. Here 1 shows number of flights delayed. Therefore, the number of flights delayed are 22901 i.e., 22.9% taken from first 100k records.

```
In [14]: df.value_counts('result') #1 is delayed we getting count how many delayed
Out[14]: result
         0    39105
         1    22901
         dtype: int64
```

Data Modelling

In this section, we will go over the models we tested, their performance, and which model we ultimately chose for the flight delay prediction system. After validating all the classification models, the models that were chosen based on their performance are Random Forest Classifier, KNN Classifier, Decision Tree Classifier, and Naive Bayes (Gaussian).

This is the data which we are working with after data cleaning. Before using the models we must split the data into two parts they are as training set and test set in the ratio of 70:30. Here the training data set is used for fitting the model, that is, to estimate the unknown parameters in the model and the test data set is used for evaluating its accuracy.The training set should be larger than the test set so that there is more data with which to train the models. The training set consists of 70% of the data.

```
In [16]: ##ML starts here
         from sklearn.model_selection import train_test_split

         data = df.values
         X, y = data[:,:-1], data[:,-1]
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)  # splitting in the ratio
```

Models

A. Random Forest Classifier

Random forest algorithm builds decision trees on different samples and takes their majority vote for classification.

Firstly, we are importing the package of random forest classifier for building the random forest model. Then importing standard scaler package because the variables need to be scaled for the future analyses to be more accurate. We use the standard scaler to normalize the data.

In the below code, we are finding the testing and training accuracy of the random forest model.

Testing Score: 0.99586, Training Score: 1.0

Next, obtaining the confusion matrix and then the Model accuracy for RF: 0.995860

Random Forest Classifier

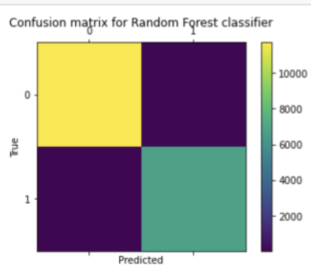
```
In [17]: #---------------------------------------RANDOMFOREST--------------------------------
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusio
         sc = StandardScaler()
         X_train_sc = sc.fit_transform(X_train)
         X_test_sc = sc.transform(X_test)
```

```
In [18]: clf = RandomForestClassifier(n_estimators=30, random_state=0)
         clf.fit(X_train, y_train)
         print(clf.score(X_test, y_test))
         print(clf.score(X_train, y_train))

         0.9958606601440705
         1.0
```

```
In [19]: y_pred = clf.predict(X_test)
         print(confusion_matrix(y_test, y_pred))
         print('Model accuracy for RF', accuracy_score(y_test,y_pred))

         [[11711    62]
          [   15  6814]]
         Model accuracy for RF 0.9958606601440705
```

Then Plotting the confusion matrix of Random Forest Classifier

```
In [20]: import matplotlib.pyplot as plt

         fig_1 = plt.figure()
         ax = fig_1.add_subplot(111)
         cax = ax.matshow(confusion_matrix(y_test,y_pred))
         plt.title('Confusion matrix for Random Forest classifier')
         fig_1.colorbar(cax)
         plt.xlabel('Predicted')
         plt.ylabel('True')
         plt.show()
```



After this we also found the F1, Precision and Recall score for the random forest classifier. Here, The macro average is the arithmetic mean of the individual class related to precision, memory, and f1 score. We use macro average scores when we need to treat all classes equally to evaluate the overall performance of the classifier against the most common class labels. The obtained F1, Precision, Recall score are 0.9955 , 0.99485, 0.99626

```
print("F1 score :",f1_score(y_test, y_pred, average="macro"))
print("Precision Score :" , precision_score(y_test, y_pred, average="macro"))
print("Recall Score :" , recall_score(y_test, y_pred, average="macro"))

F1 score : 0.9955524386849719
Precision Score : 0.9948519605651518
Recall Score : 0.9962685989347198
```

B. KNN(K-Nearest Neighbours) Classifier

KNN identifies the K nearest neighbours to a given observation point. It then uses K points to evaluate the proportions of each type of target variable and predicts the target variable that has the highest ratio.

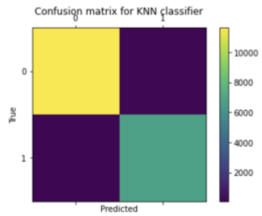
Here as random forest classifier , KNN classifier, and GridSearchCV packages are imported. Here we are taking the nearest neighbours as 3 and then finding the mean validation accuracy and test accuracy for the best K. The output obtained are: Mean Validation Accuracy for chosen K: 0.95212, Test Accuracy for the best K: 0.956079

KNN CLASSIFIER

```
In [22]: from sklearn.model_selection import GridSearchCV
         from sklearn.neighbors import KNeighborsClassifier

         Knn = KNeighborsClassifier()

         Knn_param = {'n_neighbors': range(3, 30, 2)}

         Knn_grid = GridSearchCV(Knn, Knn_param, cv = 10)
         Knn_grid.fit(X_train, y_train)

Out[22]: GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
                      param_grid={'n_neighbors': range(3, 30, 2)})

In [23]: print("Value of chosen K: ", Knn_grid.best_params_)
         print("Mean Validation accuracy for chosen K: ", Knn_grid.best_score_)
         print("Test Accuracy for the best K : ", Knn_grid.best_estimator_.score(X_test,y_test))

         Value of chosen K:  {'n_neighbors': 3}
         Mean Validation accuracy for chosen K:  0.9521240885055897
         Test Accuracy for the best K :  0.9560799913987743

In [24]: objClassifier = KNeighborsClassifier(n_neighbors = 3,  metric ='minkowski', p=2)
         objClassifier.fit(X_train_sc,y_train)
         objClassifier

Out[24]: KNeighborsClassifier(n_neighbors=3)
```

Next, made the confusion matrix and plotted. Also obtained the Model Accuracy for KNNC is 0.987

```
In [26]: import matplotlib.pyplot as plt

fig_2 = plt.figure()
ax = fig_2.add_subplot(111)
cax = ax.matshow(confusion_matrix(y_test,y_pred1))
plt.title('Confusion matrix for KNN classifier')
fig_2.colorbar(cax)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```




```
In [25]: y_pred1=objClassifier.predict(X_test_sc)
#Making the confussion matarix
print(confusion_matrix(y_test,y_pred1))
print('Model accuracy for KNNC', accuracy_score(y_test,y_pred1))

[[11619   154]
 [   74  6755]]
Model accuracy for KNNC 0.9877432534136115
```

Now, we obtained the F1, Precision and Recall score 0.98684, 0.98569, 0.98804 as follows

```
In [27]: print("F1 score :",f1_score(y_test, y_pred1, average="macro"))
print("Precision Score :" , precision_score(y_test, y_pred1, average="macro"))
print("Recall Score :" , recall_score(y_test, y_pred1, average="macro"))

F1 score : 0.9868437563268531
Precision Score : 0.9856908301895392
Recall Score : 0.9880415409786563
```

C. Decision Tree Classifier

Decision Tree is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

First we are importing the package to use the model and then using standard scaler to normalize the data. We are optimizing first before building the tree to avoid overfitting(pre-pruning). Then we obtained the training and test data set scores as 0.99182 and 0.99205

Decision Tree Classifier

```
In [28]: #-----------------------------------DECISIONTREE-----------------------------------------------------------------
from sklearn.tree import DecisionTreeClassifier
scaled_features = StandardScaler().fit_transform(X_train, X_test)
```

Pre-pruning

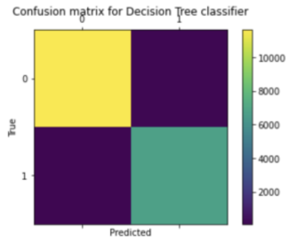
-Before building the tree we will optimize it to avoid overfitting.

```
In [50]: dct_clf = DecisionTreeClassifier(max_depth= 5,random_state=0)
dct_clf = dct_clf.fit(X_train,y_train)
print(dct_clf.score(X_test, y_test))
print(dct_clf.score(X_train, y_train))
#The difference is train and test accuracy indicates a chance of overfitting.
#Let us visualize the tree and calculate other performance measures.

0.9918288356090743
0.9920514238319049
```

As a next step, the confusion matrix is done and plotted and therefore printing Model Accuracy for DCT as 0.99182.

```
In [51]: # Predicting the Test set results
         y_pred2 = dct_clf.predict(X_test)# Making the Confusion Matrix
         print(confusion_matrix(y_test, y_pred2))
         print('Model accuracy for DCT', accuracy_score(y_test,y_pred2)) #DCT accuracy

         [[11629   144]
          [    8  6821]]
         Model accuracy for DCT 0.9918288356090743
```

```
In [52]: import matplotlib.pyplot as plt

         fig_3 = plt.figure()
         ax = fig_3.add_subplot(111)
         cax = ax.matshow(confusion_matrix(y_test,y_pred1))
         plt.title('Confusion matrix for Decision Tree classifier')
         fig_3.colorbar(cax)
         plt.xlabel('Predicted')
         plt.ylabel('True')
         plt.show()
```



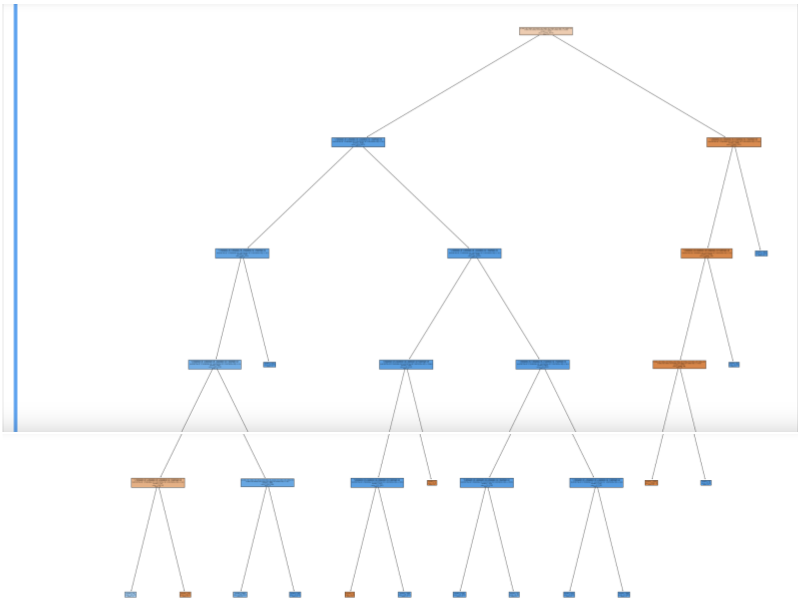
Then, obtained values.F1 score: 0.99124, Precision score: 0.98931, and Recall score: 0.99329

```
In [53]: print("F1 score for DCT:",f1_score(y_test, y_pred2, average='macro'))
         print("Precision Score fro DCT:" , precision_score(y_test, y_pred2, average='macro'))
         print("Recall Score for DCT:" , recall_score(y_test, y_pred2, average="macro"))

         F1 score for DCT: 0.9912438822617068
         Precision Score fro DCT: 0.989318867505625
         Recall Score for DCT: 0.9932985742635276
```

Now, plotting the decision tree,

```
In [54]: # Plot the decision tree

         from sklearn import tree
         fig = plt.figure(figsize=(40,40)) # set a proper figure size (in case that the figure is too small to read or ratio

         tree.plot_tree(dct_clf,
                        feature_names = X_train, # specify variable names
                        class_names = ['0','1'], # specify class (Y) names
                        filled = True, impurity = False) # whether to color the boxes, whether to report gini index
                      #   fontsize = 12) # set fontsize to read
         plt.show()
```

Below is the decision tree plot.



D. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. In this present naïve bayes algorithm, we are using gaussian naïve bayes model.

As a part of first step, we are importing the package for building the gaussian naïve bayes model. and then found Testing Accuracy: 0.98940 and Training Accuracy: 0.98914

Then, as a part of second step made the confusion matrix and found the Model Accuracy for GNB as 0.98940

Finally, obtained the values as.F1 Score:0.98867, Precision Score: 0.98598 , and Recall Score: 0.99163

#Naive Bayes

```
In [55]: from sklearn.naive_bayes import GaussianNB

         Guassian_NB = GaussianNB()
         Guassian_NB = Guassian_NB.fit(X_train, y_train)

         print('Testing accuracy for Gaussain Niave Bayes:', Guassian_NB.score(X_test, y_test))
         print('Training accuracy for Gaussain Niave Bayes:', Guassian_NB.score(X_train, y_train))

         gnb_pred = Guassian_NB.predict(X_test)
         print(confusion_matrix(y_test, gnb_pred))
         print('Model accuracy for GNB', accuracy_score(y_test,gnb_pred)) #GNB accuracy

         print("F1 score for GNB:",f1_score(y_test, gnb_pred, average='macro'))
         print("Precision Score fro GNB:" , precision_score(y_test, gnb_pred, average='macro'))
         print("Recall Score for GNB:" , recall_score(y_test, gnb_pred, average="macro"))

         Testing accuracy for Gaussain Niave Bayes: 0.9894097408880765
         Training accuracy for Gaussain Niave Bayes: 0.9891484655792093
         [[11576   197]
          [    0  6829]]
         Model accuracy for GNB 0.9894097408880765
         F1 score for GNB: 0.9886720571083074
         Precision Score fro GNB: 0.9859806433247936
         Recall Score for GNB: 0.9916333984540899
```

Results

we obtained the output of performance scores of all the four models

Models	**Model Accuracy**	**F1 Score**	**Precision Score**	**Recall Score**
Random Forest	99.58%	0.99555	0.99485	0.99626
KNN	98.77%	0.98684	0.98569	0.98804
Decision Tree	99.18%	0.9893	0.98931	0.99329
Naive Bayes(Gaussian)	98.94%	0.98867	0.98598	0.99163

We can see from the above outputs the model accuracy that random forest performed the best, followed by decision tree, and then followed by Naïve Bayes (Gaussian) much lower is the KNN classifier. The training score should not be the one with which we decide the best model, however, since it is prone to overfitting. We use the test data for an unbiased score since it is data that has not been shown to the model during training. For the test scores, the random forest model performed better than the Decision tree model and better than the Naïve Bayes (Gaussian) and much better than the KNN classifier in F1, Precision and Recall score. Based on these scores, the best choice is the random forest model, and this is the one that we will select for predicting the flight delay.

Conclusion

The purpose of this project was to find which model is best at predicting the flight delay of flights using the data from Kaggle. We performed data cleaning to make the data usable and more robust for the models, we did exploratory data analysis to gain insights about the data, and we created four models and compared their test performances. In the end, the random forest model had the best test scores, so we selected this model to perform flight delay predictions in the future.

References