Project Title:

Prediction of departure and arrival delay for the flights using Machine Learning models

Description:

Growth in the aviation industry has resulted in air-traffic congestion causing flight delays. Flight delays not only have economic impact but also harmful environmental effects. Air-traffic management is becoming increasingly challenging. In this project, we apply machine learning algorithms like Random Forest Regressor, XGBoost Regressor, logistic regression, and Linear Regression to predict if a given flight's arrival will be delayed or not. [1]

Air transportation plays a vital role in the transportation infrastructure as well as contributes significantly to the economy. [2]

An aircraft is said to be delayed when it departs and/or arrives later than its actual planned time. There are several causes of an aircraft being delayed such as weather changes, problems in maintenance, previous delays being propagated down the line, traffic congestion, and many more. These delays are a huge challenge for the aviation industry as well as their customers and passengers. In the USA alone, these delays result in a loss of about 22 billion US dollars every year. [2]

Notable reasons for commercially scheduled flights to delay are adverse weather conditions, air traffic congestion, late reaching aircraft to be used for the flight from previous flight, maintenance and security issues. [3]

According to a survey in the USA, in 2015, 18% of the domestic flights were delayed and 1.5% were canceled and out of which 40% were delayed due to late arrival of the aircraft. Delays have a huge impact on the economy and if we are able to predict the delay we can take actions in order to save the loss or improve the delay [4]

This project has experimented on the confidential data provided by the Data team at AirAsia and it consists of data related to flights from date of departure 16th Feb 2020 to date of departure 15th July 2020.
The data consists of fields such as Airline Code, flight number, Tail number, Aircraft type, Scheduled departure date and time, Scheduled Arrival date and time, Departure Airport, Arrival Airport, Actual departure time and day, Actual arrival time and day, Flight Door close time, Flight take-off time, Flight touchdown time, Total passengers, Baggage pieces, longitude and latitude of the departure and arrival airport, the distance between departure and arrival airport, separation with previous flight, tail number of previous flights, departure delay, arrival delay, and net arrival delay.

The project involved a lot of reading for the understanding of the concepts of the machine learning models and how does it work. We will go through it in-depth, one by one.
Following are the models used in this project:

1. RandomForest Regressor:
    - Random forest regressor is used to improve performance obtained by a single decision tree
    - Decision trees are easy to implement, easy to understand and easy to use as well
    - The only reason why decision tree cannot be termed as the ideal tool for prediction analysis is that it has less accuracy
    - Random forest uses the ease with which the decision tree can be implemented and implements a large number of decision trees in order to get the result
    - We use a large number of decision trees that form a forest of such trees and that is why we call this model as a random forest
    - In this model, we take a subsample of the data and create a bootstrapped dataset A bootstrapped dataset is a subset of the original dataset which has fewer rows than the original data and these rows are selected from the original dataset at random. There is a possibility that there can be duplicate entries for the row.
    - When the bootstrap dataset is ready, we take a random subset of the variables or columns at each step, instead of considering all the variables, we randomly select two to decide the split at the root
    - Then at each depth level of the tree, we consider two more variables as candidates and build the tree by considering a random subset of variables at each step
    - We repeat the same process, create a new bootstrap dataset and build a tree by considering a new set of variables at each depth level of the tree
    - Ideally, we create 100 of such trees and more the trees, there is a high chance of getting a different variety of cases included in the random forest.
    - The variety is what makes the random forest more effective than the individual decision tree

- In the random forest model which I have trained, I have used the following features:
    - "AirlineCode", "FlightNumber", "TailNumber", "FT", "SchedDepApt", "SchedArrApt", "SchedDepUtc", "SchedArrUtc", "FlightDoorClose_Utc",

"FlightTakeOff_Utc", "Total_PAX", "Baggage_pieces", "DistanceGC", "arrival_delay"

- Preprocessing the data:
    - As the fields, SchedDepUtc, SchedArrUtc, FlightDoorClose_Utc, FlightTakeOff_Utc are in the form “dd/mm/yyyy hh:mm:ss +00:00”, I had to extract the values such as day, month, week, year, hour, minutes etc
    - There were missing values for the fields TailNumber, FT, FlightTakeOff_Utc, Baggage_pieces, Total_PAX. I have put values such as the most occurring TailNumber, Most occurring FT, FlightTakeOff_UTC = FlightDoorClose_Utc of the respective flight and Avg values for Baggage_pieces and Total_PAX
    - For the fields, which have categorical values such as Airline Code, Tail Number, Flight type, Scheduled Arrival Airport, Scheduled Departure Airport, I have performed one-hot encoding in order to convert them into binary values for each of the values in all the fields

- Training of the model:
    - When we are going to do the training and testing, it is always great to take chunk of a data for training and remaining for the validation/testing. Here, I am taking 80% of the data for training and 20% of the data for testing. train_x represents 80% data without the departure_delay column train_y represents 20% data with only departure_delay column test_x represents 80% data without the departure_delay column test_y represents 20% data with only departure_delay column

  Following is the code which displays the training data and testing data split:

  {

  ```
  from sklearn.model_selection import train_test_split
  train_x, test_x, train_y, test_y = train_test_split(df.drop('departure_delay', axis=1), df['departure_delay'], test_size=0.2, random_state=42)
  ```

  }

Here, df is the data frame and we can replace departure_delay with the field that we want to predict.

- Finding the best parameters for the regressor:

- For the application of RandomForestRegressor, I used the library function from sklearn.ensemble and the parameters which I tried and gave me the best results are n_estimators=100 and random_State= 0

- Create a RandomForestRegressor object to be trained on the training data

- According to the documentation of the Scikit-learn, the most important parameters in the Random Forest are a number of trees i.e n_estimators and number of features selected for splitting at each node

- n_estimators = number of trees in the forest

- max_features = max number of features considered for splitting a node

- max_depth = max number of levels in each decision tree

- min_samples_split = min number of data points placed in a node before the node is split

- min_samples_leaf = min number of data points allowed in a leaf node

- bootstrap = method for sampling data points (with or without replacement)

- Creating the regressor object with the following code:

    - from sklearn.ensemble import RandomForestRegressor
    - regressor = RandomForestRegressor(n_estimators=100, random_state=0)

- Now, I have trained the model over the training data using the following inbuilt function provided by sklearn library:
    - model = regressor.fit(train_x, train_y)

- Now, Once, the model is trained, we can evaluate the model over the testing data using the following code snippet:
    - score = model.score(test_x, test_y)

- The score function calculates the prediction over the independent variables from test_x internally and compares it with the actual values present in test_y and gives the output as the accuracy of the model in terms of what percentage of the predicted values match with the actual values.
-
- I have used the K_Fold cross-validation technique to see the performance of the model over different folds of the data. I have divided the whole dataset into 10 folds and trained the model on 9 folds and tested it on the 10th unseen fold. After each, training of the model I have saved the model in a data structure and checked the score for testing data on each model.

- Then I have saved the predicted values obtained by model.predict(test_x) and while iterating through each value from test_y, I compared it with the corresponding value in the prediction and kept a count of how many times the difference between the actual and predicted value is less than 10 mins. At the end of the computation, I calculated the percentage of the time the difference between the actual and predicted values is less than 10 min, 15 min, and 20 mins.
- In the end, I have taken the mean of all the 10 folds accuracies and the values of the accuracy are:

- Departure Delay:
  - 10 min threshold: 91.57%
  - 15min threshold: 95.40%
  - 20min threshold: 97.03%
- Arrival Delay:
  - 10 min threshold: 87.14%
  - 15min threshold: 92.82%
  - 20min threshold: 95.57%
- NetArrival Delay:
  - 10 min threshold: 96.05%
  - 15min threshold: 98.59%
  - 20min threshold: 99.37%

- How does Random Forest work?
  - Whenever the new data arrives and we have to predict the delay for the new data, we start putting values from the new data in the trees one by one and depending upon the conditions in the branches of the trees, we arrive at a certain value from each tree
  - At the end of running through all the trees, we take the aggregated data output from all the trees and save it as the prediction for that particular entry
  - As the different trees cover the different scenarios, the aggregated value gives us the aptest prediction for the values of the features in the unseen data

2. XGBoost Regressor:
   - Extreme Gradient Boost technique is most widely used and very easy to implement a machine learning model
   - We need to understand how Adaboost works to understand the working of the XGBoost
   - In Adaboost, it starts with a very small tree called a stump from the training data and the amount of impact that the stump has on the final output is based on how it compensated on the previous error which is also called as ensemble learning
   - Adaboost builds the next stump based on the errors in the previous stump
   - It continues to make new stumps until it reached the number of stumps that we have configured it for
   - In contrast, gradient boost starts off with a single leaf instead of a stump which represents an initial guess for the weights of all of the samples

- When we try to predict the continuous value like delay, the first prediction is the average value and then the tree is formed, this tree is based on the errors made by the previous trees but it is quite larger than the stump
- We can set the maximum number of leaves between 8 to 32
- Like Adaboost, Gradient Boost can build fixed size trees where each tree is built based on the errors by the previous tree and each tree can be larger than the stump. Also, like AdaBoost, XGB can scale all the trees by the same amount. XGB continues to make such trees until it has created the number of trees we have asked for or the additional trees show no improvement
- In XGB, the first thing we do is calculate the average delay (departure/arrival/net arrival), now we build a tree based on the errors in the previous tree. Initially, we have only one leaf which has average value for the delay and now we get the error by (observed weight - predicted weight) and save the difference as a pseudo residual in a new column
- After calculating all the residual values, we go and build the new tree and if we get more than one values in the leaf of the tree, we take the average value
- After this, we can combine the original leaf with the new tree, for the data wee combine the original leaf with learning rate multiplied by the output from the tree and check the output with the original value
- If we keep the learning rate very low, then it is same as taking a lot of small steps toward the accurate prediction with less variance
- We repeat this step with each new tree and get a step closer to the accurate predictions and we keep adding the trees until it reaches the maximum specified count of the trees or there is no further improvement in the accuracy value

- In the XGBoost model, I have used the following features to train the model:
  - "AirlineCode", "FlightNumber", "TailNumber", "SchedDepApt", "SchedArrApt", "SchedDepUtc", "SchedArrUtc","FlightDoorClose_Utc", "FlightTakeOff_Utc", "Total_PAX", "Baggage_pieces", "DistanceGC", "arrival_delay/departure_delay/netarrival_delay"
- Preprocessing the data:
  - As the fields, SchedDepUtc, SchedArrUtc, FlightDoorClose_Utc, FlightTakeOff_Utc are in the form “dd/mm/yyyy hh:mm:ss +00:00”, I had to extract the values such as day, month, week, year, hour, minutes etc
  - There were missing values for the fields TailNumber, FT, FlightTakeOff_Utc, Baggage_pieces, Total_PAX. I have put values such as the most occurring TailNumber, Most occurring FT, FlightTakeOff_UTC = FlightDoorClose_Utc of the respective flight and Avg values for Baggage_pieces and Total_PAX
  - For the fields, which have categorical values such as Airline Code, Tail Number, Flight type, Scheduled Arrival Airport, Scheduled Departure Airport, I have performed one-hot encoding in order to convert them into binary values for each of the values in all the fields

- Training of the model:

- When we are going to do the training and testing, it is always great to take chunk of a data for training and remaining for the validation/testing. Here, I am taking 80% of the data for training and 20% of the data for testing. train_x represents 80% data without the departure_delay column train_y represents 20% data with only departure_delay column test_x represents 80% data without the departure_delay column test_y represents 20% data with only departure_delay column

Following is the code which displays the training data and testing data split:

{

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('departure_delay', axis=1),
df['departure_delay'], test_size=0.2, random_state=42)
```

}

Here, df is the data frame and we can replace departure_delay with the field that we want to predict.

- Importing the XGBoost regressor from sklearn library as follows:

```
import xgboost as xgb

xg_reg = xgb.XGBRegressor(objective='reg:linear', colsample_bytree = 0.8,
learning_rate = 0.3, max_depth = 8, alpha = 10, n_estimators = 100,
subsample=0.8)
```

- How to select the best parameters and its significance:
  - Here, I have imported the xgb from the xgboost library and have decided several initial parameters for which the model gives good results.

  - The parameters and their importance are as follows:
  - 1- Objective: It has a default value of "reg: linear" and other values are
    1. "bin:logistic": it gives binary classification based on logistic regression, returns prediction probability
    2. "Multi: softmax": it is used for multiclass classification, it just gives classification and not the probabilities, "multi:softprob": it is same as above but also provides probabilities

- 2- colsample_bytree: Denotes the fraction of columns to be randomly sampled for each tree and its values are in the range 0.5-1
  Reference for colsample parameters:
  https://medium.com/analytics-vidhya/xgboost-colsample-by-hyperparameters-explained-6c0bac1bdc1

- 3- learning_rate: (default = 0.3)
  step size shrinkage used in updates to prevent overfitting. After each boosting step, we can get the weight of the feature and eta shrinks the feature weights in order to make the boosting process more conservative

- 4- max_depth: The maximum depth of a tree, same as GBM.
  Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.
  Should be tuned using CV.
  Typical values: 3-10

- 5- alpha: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger alpha is, the more conservative the algorithm will be

- 6- n_estimators: No. of decision trees in the model

- 7- subsample: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration.

- Now, I have trained the model over the training data using the following inbuilt function provided by sklearn library:
  - model = xg_reg.fit(train_x, train_y)

- Now, Once, the model is trained, we can evaluate the model over the testing data using the following code snippet:
  - score = model.score(test_x, test_y)

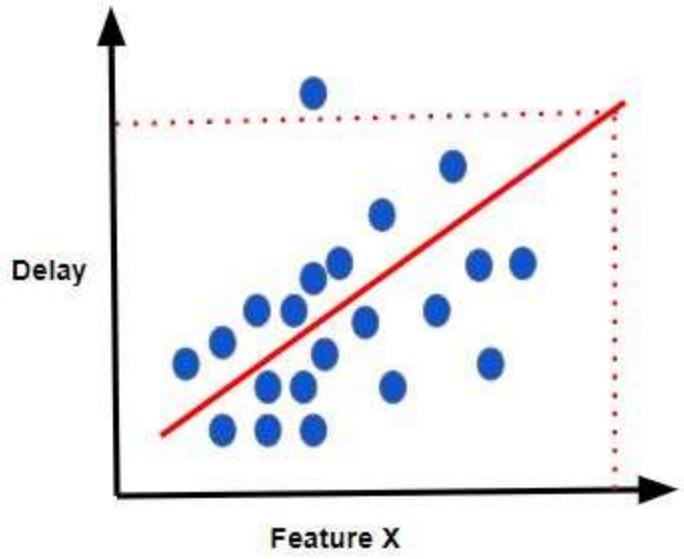
- The score function calculates the prediction over the independent variables from test_x internally and compares it with the actual values present in test_y and gives the output as the accuracy of the model in terms of what percentage of the predicted values match with the actual values.
- I have used the K_Fold cross-validation technique to see the performance of the model over different folds of the data. I have divided the whole dataset into 10 folds and trained

the model on 9 folds and tested it on the 10th unseen fold. After each, training of the model I have saved the model in a data structure and checked the score for testing data on each model.

- Then I have saved the predicted values obtained by model.predict(test_x) and while iterating through each value from test_y, I compared it with the corresponding value in the prediction and kept a count of how many times the difference between the actual and predicted value is less than 10 mins. At the end of the computation, I calculated the percentage of the time the difference between the actual and predicted values is less than 10 min, 15 min, and 20 mins.
- In the end, I have taken the mean of all the 10 folds accuracies and the values of the accuracy are:

- Departure Delay:
  - 10 min threshold: 86.25%
  - 15min threshold: 93.97%
  - 20min threshold: 96.80%
- Arrival Delay:
  - 10 min threshold: 73.73%
  - 15min threshold: 87.75%
  - 20min threshold: 93.79%
- NetArrival Delay:
  - 10 min threshold: 88.36%
  - 15min threshold: 96.20%
  - 20min threshold: 98.48%

3. Linear Regressor:



- In Linear regression, we try to fit a line which best suits the given data i.e. if we plot all the data points and try to fit a line which suits or satisfies all the data points in the best way, then we can use that line to predict the future values
- Let us say we are given the data across various features and we plot the data points across all these features in a multidimensional plane, then our goal in the linear regression is to find a line or equation of a line which suits the arrangement of the given points in such a way that the square of the perpendicular distance between the points and the line on aggregate is minimized
- When we are able to fit a line, then with new given features and multiple dependent variables we will be able to predict the future values for the delay of an aircraft
- Linear does not refer to a straight line but it refers to falling in one line
- When we create a linear equation, the right-hand side of the equation is the dependent variable which we want to predict and the left-hand side is the sum of all the independent variable multiplied by their respective weights in the trained model
  For example, departure_delay = x0w0 + x1w1 + x2w2 + ……. + XnWn, where Xi is the independent variable and Wi is the weight associated with the independent variable.
- Wights are dependent on how much is the correlation between the independent variable and the dependent variable. If the independent variable is having a huge impact on deciding or predicting the dependent variable, then the weight associated with that independent variable will be high.

- In the Linear regression model, I have used the following features to train the model:
  - "AirlineCode", "FlightNumber", "TailNumber", "SchedDepApt", "SchedArrApt", "SchedDepUtc", "SchedArrUtc","FlightDoorClose_Utc", "FlightTakeOff_Utc", "Total_PAX", "Baggage_pieces", "DistanceGC", "arrival_delay/departure_delay/netarrival_delay"

- Preprocessing the data:
    - As the fields, SchedDepUtc, SchedArrUtc, FlightDoorClose_Utc, FlightTakeOff_Utc are in the form “dd/mm/yyyy hh:mm:ss +00:00”, I had to extract the values such as day, month, week, year, hour, minutes etc
    - There were missing values for the fields TailNumber, FT, FlightTakeOff_Utc, Baggage_pieces, Total_PAX. I have put values such as the most occurring TailNumber, Most occurring FT, FlightTakeOff_UTC = FlightDoorClose_Utc of the respective flight and Avg values for Baggage_pieces and Total_PAX
    - For the fields, which have categorical values such as Airline Code, Tail Number, Flight type, Scheduled Arrival Airport, Scheduled Departure Airport, I have performed one-hot encoding in order to convert them into binary values for each of the values in all the fields
- Training of the model:
    - When we are going to do the training and testing, it is always great to take chunk of a data for training and remaining for the validation/testing. Here, I am taking 80% of the data for training and 20% of the data for testing. train_x represents 80% data without the departure_delay column train_y represents 20% data with only departure_delay column test_x represents 80% data without the departure_delay column test_y represents 20% data with only departure_delay column

    Following is the code which displays the training data and testing data split:

    {

    from sklearn.model_selection import train_test_split
    train_x, test_x, train_y, test_y = train_test_split(df.drop('departure_delay', axis=1), df['departure_delay'], test_size=0.2, random_state=42)

    }

Here, df is the data frame and we can replace departure_delay with the field that we want to predict.

- Now importing the LinearRegression for sklearn by using the following code snippet, also we call the .fit() method to train the model over the training data where trains_x represents the 80% of the whole data without the predicting variable column and train_y represents the 80% of the whole data with only predicting variable column.

        from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(train_x, train_y)

- Now, Once, the model is trained, we can evaluate the model over the testing data using the following code snippet:
    - score = model.score(test_x, test_y)

- Here, test_x is 20% of the whole data without the column of the predicting or dependent variable and test_y is 20% of the whole data with only the predicting column or the dependent variable

- The score function calculates the prediction over the independent variables from test_x internally and compares it with the actual values present in test_y and gives the output as the accuracy of the model in terms of what percentage of the predicted values match with the actual values.
- I have used the K_Fold cross-validation technique to see the performance of the model over different folds of the data. I have divided the whole dataset into 10 folds and trained the model on 9 folds and tested it on the 10th unseen fold. After each, training of the model I have saved the model in a data structure and checked the score for testing data on each model.
- Then I have saved the predicted values obtained by model.predict(test_x) and while iterating through each value from test_y, I compared it with the corresponding value in the prediction and kept a count of how many times the difference between the actual and predicted value is less than 10 mins. At the end of the computation, I calculated the percentage of the time the difference between the actual and predicted values is less than 10 min, 15 min, and 20 mins.
- In the end, I have taken the mean of all the 10 folds accuracies and the values of the accuracy are:

- Departure Delay:
    - 10 min threshold: 73.25%
    - 15min threshold: 85.68%
    - 20min threshold: 91.72%

- Arrival Delay:
    - 10 min threshold: 58.70%
    - 15min threshold: 75.05%
    - 20min threshold: 84.38%

- NetArrival Delay:
    - 10 min threshold: 79.75%
    - 15min threshold: 91.48%
    - 20min threshold: 96.17%

- Logistic Regression:

- Logistic regression is a technique that has been used in traditional statistics as well as machine learning
- Logistic regression predicts if something is true or false except instead of predicting something like an actual delay value
- Instead of fitting a line to the data like linear regression, logistic regression fits an "S" shaped logistic function to the data
- The curve goes from 0 to 1 and that tells us the probability whether there is a delay or no delay in the case current flight
- Now, the curve is spanned over different dimensions, and depending upon the different dimensions and the sigmoid function used for plotting the "S" curve, if the predicted value is over 0.5 on the curve, then there is a high probability that the given flight is going to be delayed. On the other hand, if the sigmoid function gives the output as a probability of less than 0.5 then there is a strong chance that the given flight will not be delayed.
- Logistic regression is usually used for classification. If the probability value of a certain record is greater than 0.5 then we classify it as "Delay" otherwise we classify it as "No Delay"
- We can have complicated models with many independent variables and we decide which variables to consider and make the model simpler based on the variable's effect on the model
- Logistic regression's ability to provide probabilities and classify new samples using continuous and discrete measurements makes it a popular machine learning method
- One big difference between linear regression and logistic regression is how we fit the line to the data, in linear regression we fit the line using least squares, in other words, we find the line that minimizes the sum of the squares of the residuals.
- In Logistic regression, we don't have the same concept of the least-squares and it can't be used to fit the "S" curve, instead, it uses something like maximum likelihood. We pick up the probability of observing a flight being delayed or not delayed on the curve and find the likelihood of the flight being delayed and we do the same for all the flights and lastly, we multiply all those likelihoods together and that's the likelihood of the data given the curve. Then we shift the curve and calculate the new likelihood of the data and we repeat this process and finally, the curve with the maximum likelihood is selected.

- In the logistic regression model, I have used the following features to train the model:
    - "AirlineCode", "FlightNumber", "TailNumber", "SchedDepApt", "SchedArrApt", "SchedDepUtc", "SchedArrUtc","FlightDoorClose_Utc", "FlightTakeOff_Utc",

"Total_PAX", "Baggage_pieces", "DistanceGC",
"arrival_delay/departure_delay/netarrival_delay"

- Preprocessing the data:
    - As the fields, SchedDepUtc, SchedArrUtc, FlightDoorClose_Utc, FlightTakeOff_Utc are in the form “dd/mm/yyyy hh:mm:ss +00:00”, I had to extract the values such as day, month, week, year, hour, minutes etc
    - There were missing values for the fields TailNumber, FT, FlightTakeOff_Utc, Baggage_pieces, Total_PAX. I have put values such as the most occurring TailNumber, Most occurring FT, FlightTakeOff_UTC = FlightDoorClose_Utc of the respective flight and Avg values for Baggage_pieces and Total_PAX
    - For the fields, which have categorical values such as Airline Code, Tail Number, Flight type, Scheduled Arrival Airport, Scheduled Departure Airport, I have performed one-hot encoding in order to convert them into binary values for each of the values in all the fields
    - For the dependent binary class in the data, for predicting the delay value, I have used the following code snippet:

```
for i in range(0, len(df)):
    if df.loc[i, 'departure_delay'] > 15:
        df.loc[i, 'delay_bool'] = 1
    else:
        df.loc[i, 'delay_bool'] = 0
```

- Training of the model:
    - When we are going to do the training and testing, it is always great to take chunk of a data for training and remaining for the validation/testing. Here, I am taking 80% of the data for training and 20% of the data for testing. train_x represents 80% data without the departure_delay column train_y represents 20% data with only departure_delay column test_x represents 80% data without the departure_delay column test_y represents 20% data with only departure_delay column

Following is the code which displays the training data and testing data split:

{

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(df.drop('delay_bool', axis=1),
df['delay_bool'], test_size=0.2, random_state=42)
```

```
}
```

Here, df is the data frame and we can replace departure_delay with the field that we want to predict.

This is how I imported the Logistic Regression from the sklearn library and use .fit() function to fit the model over the training data.

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(n_jobs=-1)
logreg.fit(train_x, train_y)
```

The error that I was continuously receiving was the n-iterations limit exceeded and after doing a lot of research on it and trying different ways of scaling the data, I came to the conclusion that if we put the parameter n_jobs = -1, it uses all the number of CPU cores which does not give us the error of max_iterations reached.

Logistic regression gives us the probability of a given record to be classified as class A or class B, here in this condition, delay_bool = 0 or delay_bool = 1

To get all the predicted probabilities over the test data, I have used following line of code:

```
myArray = logreg.predict_proba(test_x)
```

Where test_x is the unseen 20% of the data without the ‘delay_bool’ column and the output myArray looks like this:

```
[[0.75339511 0.24660489]
 [0.83949572 0.16050428]
 [0.76533832 0.23466168]
 ...
 [0.80749656 0.19250344]
 [0.79183364 0.20816636]
 [0.81313736 0.18686264]]
```

Here, each element of the list is itself a list of two elements out of which the first element is the probability of the record being classified as delay_bool = 0 and the second element is the probability of the record being classified as delay_bool = 1

To find the accuracy of the model, we use the score function and get the accuracy equal to 76.78%

Finally,
All the accuracies are as follows:

Model	Departure Delay	Arrival Delay	NetArrival Delay
XGBoost	86.25%	73.73%	88.36%
RandomForest	91.57%	87.14%	96.05%
Linear Regression	73.25%	58.70%	79.75%

Overall, Random Forest Regressor performs the best followed by XGBoost followed by Linear regression model.

REFERENCE:

[1] Gopalakrishnan, Karthik, and Hamsa Balakrishnan. "A comparative analysis of models for predicting delays in air traffic networks." (2017).

[2] R. Nigam and K. Govinda, "Cloud-based flight delay prediction using logistic regression," 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, 2017, pp. 662-667, DOI: 10.1109/ISS1.2017.8389254.

[3] Kuhn, Nathalie, and Navaneeth Jamadagni. "Application of Machine Learning Algorithms to Predict Flight Arrival Delays." (2017).

[4] Chakrabarty, Navoneel. (2019). A Data Mining Approach to Flight Arrival Delay Prediction for American Airlines.