# Flight Delay Prediction for aviation Industry using Machine Learning

# ABSTRACT:

**Growth in aviation industries has resulted in air-traffic jamming causing flight delays. Flight delays not only have economic impact but also injurious environmental properties. Air-traffic supervision is becoming increasingly challenging. Airlines delays make immense loss for business field as well as in budget loss for a country, there are so many reasons for impede in flights some of them are, some of them are due to security issues, mechanical problems, due to weather conditions, Airport congestion etc. we are proposing machine learning algorithms like Random Forest, Decision Tree , Classifier, ANN . The aim of this research work is to predict Flight Delay, Which is highest economy producing field for many countries and among many transportation this one is fastest and comfort, so to identify and reduce flight delays, can dramatically reduce the flight delays to saves huge amount of turnovers, using machine learning algorithms.**

# INTRODUCTION:

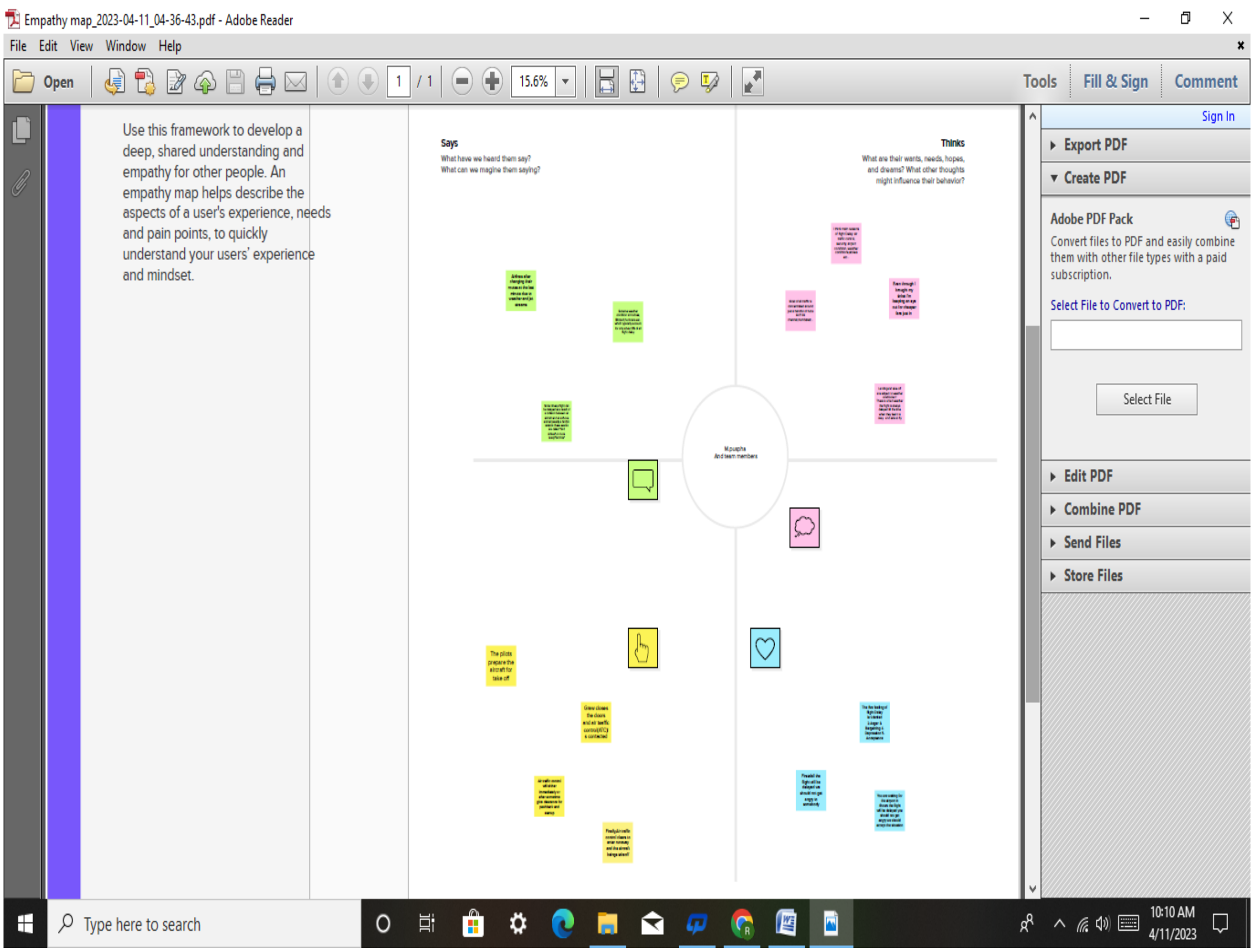
OVER the last twenty years, air travel has been increasingly preferred among travelers, mainly because of its speed and in some cases comfort. This has led to phenomenal growth in air traffic and on the ground. An increase in air traffic growth has also resulted in massive levels of aircraft delays on the ground and in the air. These delays are responsible for large economic and environmental losses. According to, taxi-out operations are responsible for 4,000 tons of hydrocarbons, 8,000 tons of nitrogen oxides and 45,000 tons of carbon monoxide emissions in the United States in 2007. Moreover, the economic impact of flight delays for domestic flights in the US is estimated to be more than $19 Billion per year to the airlines and over $41 Billion per year to the national economy In response to growing concerns of fuel emissions and their negative impact on health, there is active research in the aviation industry for finding techniques to predict flight delays accurately in order to optimize flight operations and minimize delays. Using a machine learning model, we can predict flight arrival delays. The input to our algorithm is rows of feature vector like departure date, departure delay, distance between the two airports, scheduled arrival time etc. We then use decision tree classifier to predict if the flight arrival will be delayed or not. A flight is delayed when difference between scheduled and actual arrival times is greater than 15 minutes. Furthermore, we compare decision tree classifier with logistic regression and a simple neural network for various figures of merit. Finally, it will be integrated to web based application.

# Purpose statement:

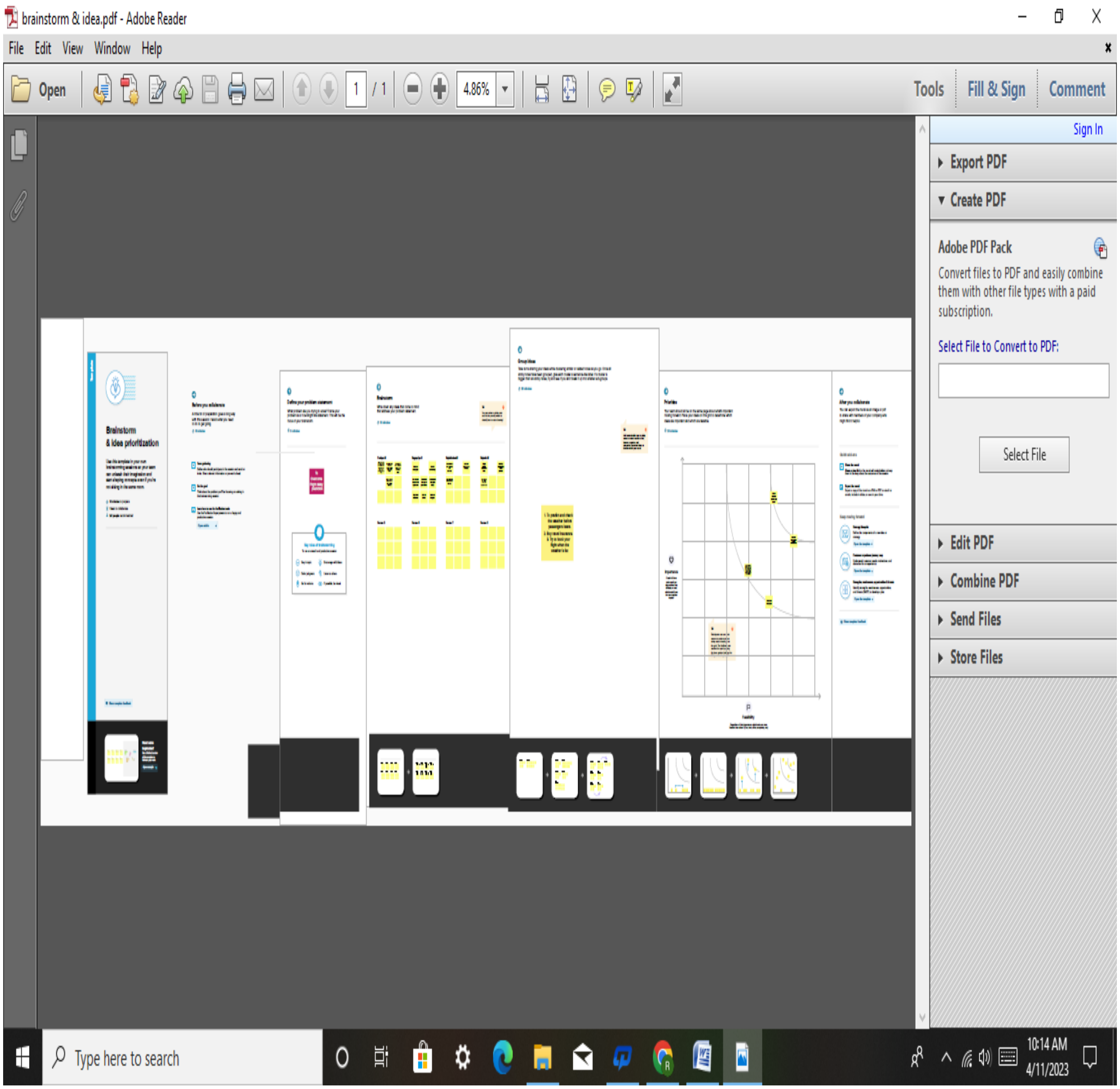
- Throughout the year 2015, there has been over 5,4 million domestic flights within the US. All of their metadata are recorded and saved in the Department of Transportation's (DOT) Bureau of Transportation Statistics.
- Flight delays cause significant financial and other losses to airlines, airports, and passengers. Their prediction is crucial during the decision-making process for all players of American aviation industry.
- Therefore, predicting the likelihood of delay based on flights' features bridges an important information asymmetry between airlines and passengers.
- The primary use case of the algorithm will be: predicting a potential delay, on a given day, for a given airport and airline.

# Problem Defining & Design Thinking:

## *Empathy Map:*



# *Ideation & Brainstorming Map:*



# RESULTS:

- At first, it seemed like our models were not going to perform very well, however after introducing 400k data points, we managed to get all three of them to get above chance.

That in and off itself feels like an accomplishment.

- Features were sparse, and many were not very efficient at predicting delay. Still this sparse data managed to find three methods that could predict delay vs non-delay with a 65% accuracy.
- Since all three were able to classify with a similar f1-score, comparing the models' performance seems out of place.
- One thing to note, is that our test set was built of 56% non-delayed and 44% delayed, which may be the reason our 0 classifier's precision is higher.
- In order to continue this research and get a better model, one would need to increase the number of features, and most likely improve the quality of the features.
- If this is done well, we hope to see a marketable app, that predicts the delay for us or even calculates whether a selected layover is manageable or not.

# Advantages of flight delay:

- ❖ Reimbursement of your ticket and a return flight to your departure airport if you have a connecting flight.
- ❖ Rerouting to your final destination.
- ❖ Rerouting at a later date under comparable transportation conditions.
- ❖ When we traveling by air, can sit comfortable in an arm-chair, reading magazines, listen to music, read books, play games or watching  a free film on television.

# Disadvantages of flight delay:

- ❖ Flight delays not only irritate air passengers and disrupt their schedules but also cause a decrease in efficiency, an increase in capital costs, reallocation of flight crews and aircraft, and additional crew expenses.
- ❖ There  are  plane crashes in which the crew and passengers have died.
- ❖ Airports can often be several miles from city center.

# Applications for Flight Delay:

- ✓ It is widely used by aircraft operators throughout the world to inform and facilitate corrective actions in a range of operational areas by offering the ability to track and evaluate flight operations trends, identify risk precursors, and take the appropriate remedial action.

- ✓ Therefore, predicting flight delays can improve airline operations and passenger satisfaction, which will result in a positive impact on the economy. In this study, the main goal is to compare the performance of machine learning classification algorithms when predicting flight delays.

- ✓ With predictive analytics, sensory equipment gathers information from each aircraft's systems, and sends that information to a cloud. That data is then analyzed and used to determine everything from fleet maintenance schedules to marketing strategies.

- ✓ In case of a delay of over 24 hours, the passenger should be offered free hotel accommodation.

- ✓ Customers should also be offered a free stay if a flight departs between 8 pm and 3 am and is delayed for over six hours.

# Conclusion:

- In this project, we use flight data, weather, and demand data to predict flight departure delay. Our result shows that the Random Forest method yields the best performance compared to the SVM model.
- Somehow the SVM model is very time consuming and does not necessarily produce better results. In the end, our model correctly predicts 91% of the non-delayed flights.
- However, the delayed flights are only correctly predicted 41% of time. As a result, there can be additional features related to the causes of flight delay that are not yet discovered using our existing data sources.
- In the second part of the project, we can see that it is possible to predict flight delay patterns from just the volume of concurrently published tweets, and their sentiment and objectivity.
- This is not unreasonable; people tend to post about airport delays on Twitter; it stands to reason that these posts would become more frequent, and more profoundly emotional, as the delays get worse. Without more data, we cannot make a robust model and find out the role of related factors and chance on these results.
- However, as a proof of concept, there is potential for these results. It may be possible to routinely use tweets to ascertain an understanding of concurrent airline delays and traffic patterns, which could be useful in a variety of circumstances.

# Future Scope:

- This project is based on data analysis from year 2008. A large dataset is available from 1987-2008 but handling a bigger dataset requires a great amount of preprocessing and cleaning of the data.
- Therefore, the future work of this project includes incorporating a larger dataset. There are many different ways to preprocess a larger dataset like running a Spark cluster over a server or using a cloud-based services like AWS and Azure to process the data.
- With the new advancement in the field of deep learning, we can use Neural Networks algorithm on the flight and weather data. Neural Network works on the pattern matching methodology. It is divided into three basic parts for data modelling that includes feed forward networks, feedback networks, and self organization network.
- Feed-forward and feedback networks are generally used in the areas of prediction, pattern recognition, associative memory, and optimization calculation, whereas self-organization networks are generally used in cluster analysis. Neural Network offers distributed computer architecture with important learning abilities to represent nonlinear relationships.

# APPENDIX:

# SOURCE CODE:

## Milestone 1:

```
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
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classificat
ion_report, confusion_matrix, f1_score
from sklearn.utils import shuffle
import numpy as np
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
```

## Read the data set

```
df= pd.read_csv('/content/flightdata.csv')
df.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 | DIVE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2016 | 1 | 1 | 1 | 5 | DL | N836DN | 1399 | 10397 | ATL | ... | 2143 | 2102.0 | -41.0 | 0.0 | 0.0 | |
| 1 | 2016 | 1 | 1 | 1 | 5 | DL | N964DN | 1476 | 11433 | DTW | ... | 1435 | 1439.0 | 4.0 | 0.0 | 0.0 | |
| 2 | 2016 | 1 | 1 | 1 | 5 | DL | N813DN | 1597 | 10397 | ATL | ... | 1215 | 1142.0 | -33.0 | 0.0 | 0.0 | |
| 3 | 2016 | 1 | 1 | 1 | 5 | DL | N587NW | 1768 | 14747 | SEA | ... | 1335 | 1345.0 | 10.0 | 0.0 | 0.0 | |
| 4 | 2016 | 1 | 1 | 1 | 5 | DL | N836DN | 1823 | 14747 | SEA | ... | 607 | 615.0 | 8.0 | 0.0 | 0.0 | |

5 rows × 26 columns

```
df.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
 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
```

```
df= df.drop('Unnamed: 25',axis=1)
df.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
```

```
df=df[["FL_NUM","MONTH","DAY_OF_MONTH","DAY_OF_WEEK","ORIGIN","DEST","CRS_ARR_TIME","DEP_DEL15","ARR_DEL15"]]
df.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
```

```
df[df.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 |

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

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

```
df=df.fillna({'ARR_DEL15': 1})
df=df.fillna({'DEP_DEL15': 0})
df.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 |

```
import math

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

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

```
df['ORIGIN'].unique()
```

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

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

```
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()

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.]])
```

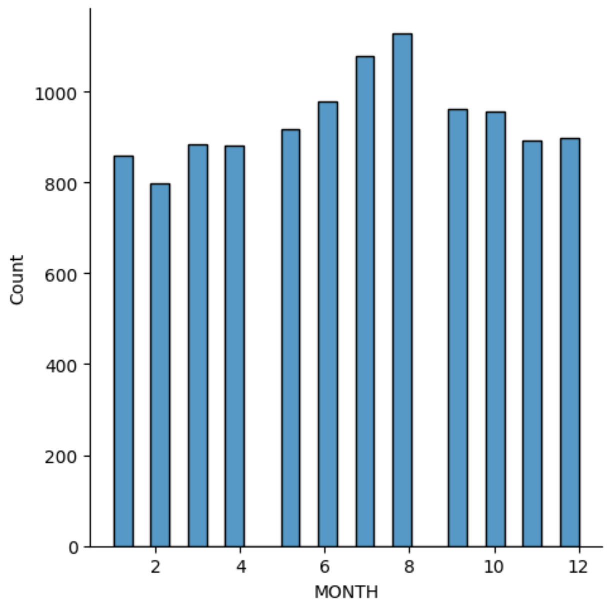
# Milestone 3

```
df.describe()
```

| | FL_NUM | MONTH | DAY_OF_MONTH | DAY_OF_WEEK | ORIGIN | DEST | CRS_ARR_TIME | DEP_DEL15 | ARR_DEL15 |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 11231.000000 | 11231.000000 | 11231.000000 | 11231.000000 | 11231.000000 | 11231.000000 | 11231.000000 | 11231.000000 | 11231.000000 |
| **mean** | 1334.325617 | 6.628973 | 15.790758 | 3.960199 | 1.837325 | 1.806607 | 15.067314 | 0.141483 | 0.139168 |
| **std** | 811.875227 | 3.354678 | 8.782056 | 1.995257 | 1.489464 | 1.496328 | 5.023534 | 0.348535 | 0.346138 |
| **min** | 7.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 624.000000 | 4.000000 | 8.000000 | 2.000000 | 0.000000 | 0.000000 | 11.000000 | 0.000000 | 0.000000 |
| **50%** | 1267.000000 | 7.000000 | 16.000000 | 4.000000 | 2.000000 | 2.000000 | 15.000000 | 0.000000 | 0.000000 |
| **75%** | 2032.000000 | 9.000000 | 23.000000 | 6.000000 | 3.000000 | 3.000000 | 19.000000 | 0.000000 | 0.000000 |
| **max** | 2853.000000 | 12.000000 | 31.000000 | 7.000000 | 4.000000 | 4.000000 | 23.000000 | 1.000000 | 1.000000 |

```
sns.displot(df.MONTH)
```

<seaborn.axisgrid.FacetGrid at 0x7f965c2cef10>

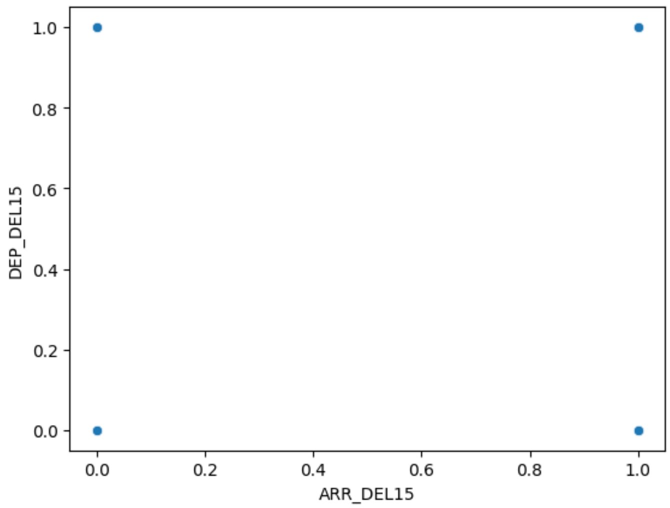



```
import seaborn as sns
sns.__version__
```

'0.12.2'

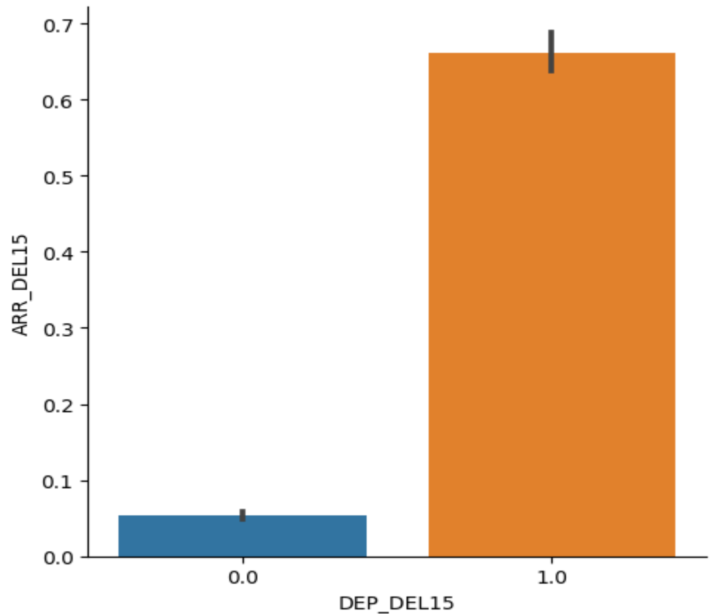
```
import seaborn as sns
sns.scatterplot(x='ARR_DEL15',y='DEP_DEL15', data=df)
```

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



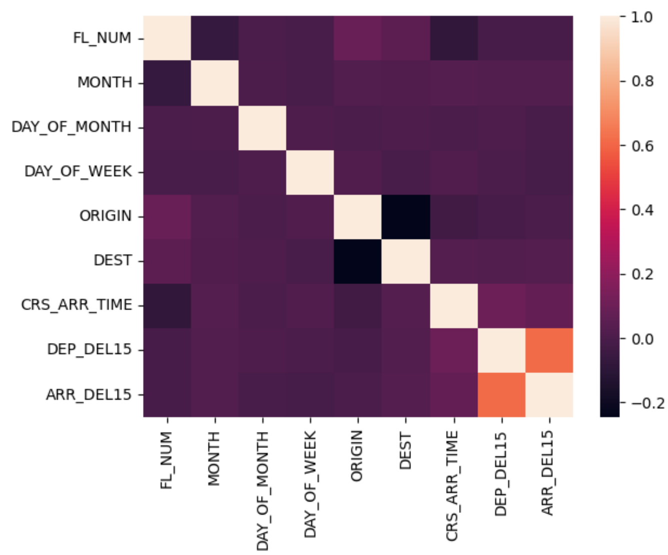
```
sns.catplot(x="DEP_DEL15",y="ARR_DEL15",kind='bar',data=df)
```

<seaborn.axisgrid.FacetGrid at 0x7f965968e070>




```
sns.heatmap(df.corr())
```

<Axes: >



```
x=df.iloc[:,0:8].values

y=df.iloc[:,8:9].values

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)

x_test.shape
```

```
(2247, 8)
```

```
x_train.shape
```

```
(8984, 8)
```

```
y_test.shape
```

```
(2247, 1)
```

```
y_train = np.arange(8984).reshape((8984))
y_train
```

array([ 0, 1, 2, ..., 8981, 8982, 8983]

```
y_train.shape
```

```
(8984,)
```

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

# Milestone4

# DECISIONTREE MODEL

[ ]

```
from sklearn.tree import DecisionTreeClassifier

Classifier = DecisionTreeClassifier(random_state = 0)

Classifier.fit(x_train,y_train)
```

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

```
decisionTree = Classifier.predict(x_test)
```

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

```
decisionTree
```

```
array([5161, 4854, 4601, ..., 2682, 3532, 6819])
```

# RANDOM FOREST **MODEL**

```
from sklearn.ensemble import RandomForestClassifier

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

rfc.fit(x_train,y_train)
```

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

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

# ANN MODEL

```
import tensorflow

from keras.models import Sequential

from tensorflow.keras.layers import Dense
```

```
classification = Sequential()
classification.add(Dense(80, 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'))

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

classification.fit(x_train,y_train,batch_size=4, validation_split=0
.2,epochs=100)
```

Epoch 1/100
1797/1797 [==============================] - 6s 3ms/step - loss: -2957390118912.0000 - accuracy: 1.3914e-04 - val_loss: -37840700309504.0000 - val_accuracy: 0.0000e+00
Epoch 2/100
1797/1797 [==============================] - 6s 3ms/step - loss: -122504261664768.0000 - accuracy: 1.3914e-04 - val_loss: -731546008944640.0000 - val_accuracy: 0.0000e+00
Epoch 3/100
1797/1797 [==============================] - 4s 2ms/step - loss: -852444774924288.0000 - accuracy: 1.3914e-04 - val_loss: -3646220002131968.0000 - val_accuracy: 0.0000e+00
Epoch 4/100
1797/1797 [==============================] - 4s 2ms/step - loss: -3067490875736064.0000 - accuracy: 1.3914e-04 - val_loss: -11172752903897088.0000 - val_accuracy: 0.0000e+00
Epoch 5/100
1797/1797 [==============================] - 5s 3ms/step - loss: -8045248987004928.0000 - accuracy: 1.3914e-04 - val_loss: -26641158151077888.0000 - val_accuracy: 0.0000e+00
Epoch 6/100
1797/1797 [==============================] - 4s 2ms/step - loss: -17430914317418496.0000 - accuracy: 1.3914e-04 - val_loss: -54104282848296960.0000 - val_accuracy: 0.0000e+00

Epoch 7/100
1797/1797 [==============================] - 5s 3ms/step - loss: -3330482568062566 4.0000 - accuracy:
1.3914e-04 - val_loss: -9889958485976678 4.0000 - val_accuracy: 0.0000e+00
Epoch 8/100
1797/1797 [==============================] - 4s 2ms/step - loss: -58181409403043840.0000 - accuracy:
1.3914e-04 - val_loss: -167120957058580480.0000 - val_accuracy: 0.0000e+00
Epoch 9/100
1797/1797 [==============================] - 4s 2ms/step - loss: -95225566755553280.0000 - accuracy:
1.3914e-04 - val_loss: -266594169160466432.0000 - val_accuracy: 0.0000e+00
Epoch 10/100
1797/1797 [==============================] - 6s 4ms/step - loss: -148073704813756416.0000 -
accuracy: 1.3914e-04 - val_loss: -406234740048265216.0000 - val_accuracy: 0.0000e+00
Epoch 11/100
1797/1797 [==============================] - 4s 2ms/step - loss: -220848025814171648.0000 -
accuracy: 1.3914e-04 - val_loss: -595615069493002240.0000 - val_accuracy: 0.0000e+00
Epoch 12/100
1797/1797 [==============================] - 4s 2ms/step - loss: -318661937430790144.0000 -
accuracy: 1.3914e-04 - val_loss: -847241123008086016.0000 - val_accuracy: 0.0000e+00
Epoch 13/100
1797/1797 [==============================] - 5s 3ms/step - loss: -447027411712737280.0000 -
accuracy: 1.3914e-04 - val_loss: -1174974271886196736.0000 - val_accuracy: 0.0000e+00
Epoch 14/100
1797/1797 [==============================] - 4s 2ms/step - loss: -612550503498252288.0000 -
accuracy: 1.3914e-04 - val_loss: -1593252409470091264.0000 - val_accuracy: 0.0000e+00
Epoch 15/100
1797/1797 [==============================] - 5s 3ms/step - loss: -821773616210247680.0000 -
accuracy: 1.3914e-04 - val_loss: -2117740209201217536.0000 - val_accuracy: 0.0000e+00
Epoch 16/100
1797/1797 [==============================] - 5s 3ms/step - loss: -1083024450917498880.0000 -
accuracy: 1.3914e-04 - val_loss: -2770174191326986240.0000 - val_accuracy: 0.0000e+00
Epoch 17/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1405101093507039232.0000 -
accuracy: 1.3914e-04 - val_loss: -3567472403825033216.0000 - val_accuracy: 0.0000e+00
Epoch 18/100
1797/1797 [==============================] - 6s 3ms/step - loss: -1796332894315085824.0000 -
accuracy: 1.3914e-04 - val_loss: -4531824640611319808.0000 - val_accuracy: 0.0000e+00
Epoch 19/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2269566547200573440.0000 -
accuracy: 1.3914e-04 - val_loss: -5696079361321467904.0000 - val_accuracy: 0.0000e+00
Epoch 20/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2834431849877471232.0000 -
accuracy: 1.3914e-04 - val_loss: -7074537089064239104.0000 - val_accuracy: 0.0000e+00
Epoch 21/100
1797/1797 [==============================] - 5s 3ms/step - loss: -3503502817321025536.0000 -
accuracy: 1.3914e-04 - val_loss: -8705394857788571648.0000 - val_accuracy: 0.0000e+00
Epoch 22/100
1797/1797 [==============================] - 4s 2ms/step - loss: -4288141802692673536.0000 -
accuracy: 1.3914e-04 - val_loss: -10604813839155331072.0000 - val_accuracy: 0.0000e+00
Epoch 23/100
1797/1797 [==============================] - 4s 2ms/step - loss: -5203917066008002560.0000 -
accuracy: 1.3914e-04 - val_loss: -12821298438868041728.0000 - val_accuracy: 0.0000e+00
Epoch 24/100
1797/1797 [==============================] - 5s 3ms/step - loss: -6266192632997740544.0000 -
accuracy: 1.3914e-04 - val_loss: -15381550846563057664.0000 - val_accuracy: 0.0000e+00
Epoch 25/100

1797/1797 [==============================] - 4s 2ms/step - loss: -7495243772955983872.0000 - accuracy: 1.3914e-04 - val_loss: -18340919378070994944.0000 - val_accuracy: 0.0000e+00
Epoch 26/100
1797/1797 [==============================] - 4s 2ms/step - loss: -8902706067683672064.0000 - accuracy: 1.3914e-04 - val_loss: -21713061067320459264.0000 - val_accuracy: 0.0000e+00
Epoch 27/100
1797/1797 [==============================] - 5s 3ms/step - loss: -10507685180980854784.0000 - accuracy: 1.3914e-04 - val_loss: -25555605716769701888.0000 - val_accuracy: 0.0000e+00
Epoch 28/100
1797/1797 [==============================] - 4s 2ms/step - loss: -12333902526460985344.0000 - accuracy: 1.3914e-04 - val_loss: -29923248532342439936.0000 - val_accuracy: 0.0000e+00
Epoch 29/100
1797/1797 [==============================] - 5s 3ms/step - loss: -14407509988190715904.0000 - accuracy: 1.3914e-04 - val_loss: -34873920582682935296.0000 - val_accuracy: 0.0000e+00
Epoch 30/100
1797/1797 [==============================] - 4s 2ms/step - loss: -16746536258330689536.0000 - accuracy: 1.3914e-04 - val_loss: -40434434558346133504.0000 - val_accuracy: 0.0000e+00
Epoch 31/100
1797/1797 [==============================] - 4s 2ms/step - loss: -19362030329228427264.0000 - accuracy: 1.3914e-04 - val_loss: -46647277787652554752.0000 - val_accuracy: 0.0000e+00
Epoch 32/100
1797/1797 [==============================] - 7s 4ms/step - loss: -22289653761019215872.0000 - accuracy: 1.3914e-04 - val_loss: -53588415528965242880.0000 - val_accuracy: 0.0000e+00
Epoch 33/100
1797/1797 [==============================] - 4s 2ms/step - loss: -25568775667047202816.0000 - accuracy: 1.3914e-04 - val_loss: -61363502053620449280.0000 - val_accuracy: 0.0000e+00
Epoch 34/100
1797/1797 [==============================] - 5s 3ms/step - loss: -29207539030427697152.0000 - accuracy: 1.3914e-04 - val_loss: -69958454816689618944.0000 - val_accuracy: 0.0000e+00
Epoch 35/100
1797/1797 [==============================] - 5s 3ms/step - loss: -33251450438411091968.0000 - accuracy: 1.3914e-04 - val_loss: -79516971191830052864.0000 - val_accuracy: 0.0000e+00
Epoch 36/100
1797/1797 [==============================] - 4s 2ms/step - loss: -37723175223692361728.0000 - accuracy: 1.3914e-04 - val_loss: -90052623550575017984.0000 - val_accuracy: 0.0000e+00
Epoch 37/100
1797/1797 [==============================] - 6s 3ms/step - loss: -42657532720499916800.0000 - accuracy: 1.3914e-04 - val_loss: -101684893622491676672.0000 - val_accuracy: 0.0000e+00
Epoch 38/100
1797/1797 [==============================] - 4s 2ms/step - loss: -48084541795295297536.0000 - accuracy: 1.3914e-04 - val_loss: -114441110868600029184.0000 - val_accuracy: 0.0000e+00
Epoch 39/100
1797/1797 [==============================] - 4s 2ms/step - loss: -54035951945842163712.0000 - accuracy: 1.3914e-04 - val_loss: -128428948263584923648.0000 - val_accuracy: 0.0000e+00
Epoch 40/100
1797/1797 [==============================] - 5s 3ms/step - loss: -60566171405529382912.0000 - accuracy: 1.3914e-04 - val_loss: -143767241024176390144.0000 - val_accuracy: 0.0000e+00
Epoch 41/100
1797/1797 [==============================] - 4s 2ms/step - loss: -67673674052217602048.0000 - accuracy: 1.3914e-04 - val_loss: -160409545779217170432.0000 - val_accuracy: 0.0000e+00
Epoch 42/100
1797/1797 [==============================] - 4s 2ms/step - loss: -75441732498548064256.0000 - accuracy: 1.3914e-04 - val_loss: -178628541412395712512.0000 - val_accuracy: 0.0000e+00
Epoch 43/100
1797/1797 [==============================] - 5s 3ms/step - loss: -83926760487118700544.0000 - accuracy: 1.3914e-04 - val_loss: -198514660556073336832.0000 - val_accuracy: 0.0000e+00

Epoch 44/100
1797/1797 [==============================] - 4s 2ms/step - loss: -93113857436349890560.0000 -
accuracy: 1.3914e-04 - val_loss: -219930772120033820672.0000 - val_accuracy: 0.0000e+00
Epoch 45/100
1797/1797 [==============================] - 5s 3ms/step - loss: -103087597780650164224.0000 -
accuracy: 1.3914e-04 - val_loss: -243308095768597889024.0000 - val_accuracy: 0.0000e+00
Epoch 46/100
1797/1797 [==============================] - 5s 3ms/step - loss: -113929380564847034368.0000 -
accuracy: 1.3914e-04 - val_loss: -268595913229399490560.0000 - val_accuracy: 0.0000e+00
Epoch 47/100
1797/1797 [==============================] - 4s 2ms/step - loss: -125621639991175151616.0000 -
accuracy: 1.3914e-04 - val_loss: -295838081822247354368.0000 - val_accuracy: 0.0000e+00
Epoch 48/100
1797/1797 [==============================] - 5s 3ms/step - loss: -138241764086043901952.0000 -
accuracy: 1.3914e-04 - val_loss: -325271955321252741120.0000 - val_accuracy: 0.0000e+00
Epoch 49/100
1797/1797 [==============================] - 4s 2ms/step - loss: -151831982892052905984.0000 -
accuracy: 1.3914e-04 - val_loss: -356919699880831614976.0000 - val_accuracy: 0.0000e+00
Epoch 50/100
1797/1797 [==============================] - 4s 2ms/step - loss: -166483547078214549504.0000 -
accuracy: 1.3914e-04 - val_loss: -391068947742810177536.0000 - val_accuracy: 0.0000e+00
Epoch 51/100
1797/1797 [==============================] - 5s 3ms/step - loss: -182271091493822267392.0000 -
accuracy: 1.3914e-04 - val_loss: -427811143090247303168.0000 - val_accuracy: 0.0000e+00
Epoch 52/100
1797/1797 [==============================] - 4s 2ms/step - loss: -199169951894041788416.0000 -
accuracy: 1.3914e-04 - val_loss: -467031620054505488384.0000 - val_accuracy: 0.0000e+00
Epoch 53/100
1797/1797 [==============================] - 4s 2ms/step - loss: -217276181614675623936.0000 -
accuracy: 1.3914e-04 - val_loss: -509122860306235654144.0000 - val_accuracy: 0.0000e+00
Epoch 54/100
1797/1797 [==============================] - 5s 3ms/step - loss: -236657616125111042048.0000 -
accuracy: 1.3914e-04 - val_loss: -554074730746276216832.0000 - val_accuracy: 0.0000e+00
Epoch 55/100
1797/1797 [==============================] - 4s 2ms/step - loss: -257339130776414846976.0000 -
accuracy: 1.3914e-04 - val_loss: -602043168511724879872.0000 - val_accuracy: 0.0000e+00
Epoch 56/100
1797/1797 [==============================] - 4s 2ms/step - loss: -279465720365965115392.0000 -
accuracy: 1.3914e-04 - val_loss: -653379419189144453120.0000 - val_accuracy: 0.0000e+00
Epoch 57/100
1797/1797 [==============================] - 5s 3ms/step - loss: -303164171778583953408.0000 -
accuracy: 1.3914e-04 - val_loss: -708439724545934360576.0000 - val_accuracy: 0.0000e+00
Epoch 58/100
1797/1797 [==============================] - 4s 2ms/step - loss: -328406531030646784000.0000 -
accuracy: 1.3914e-04 - val_loss: -766800183267168354304.0000 - val_accuracy: 0.0000e+00
Epoch 59/100
1797/1797 [==============================] - 5s 3ms/step - loss: -355209598659826024448.0000 -
accuracy: 1.3914e-04 - val_loss: -828878433849540870144.0000 - val_accuracy: 0.0000e+00
Epoch 60/100
1797/1797 [==============================] - 5s 3ms/step - loss: -383835146394462584832.0000 -
accuracy: 1.3914e-04 - val_loss: -895254666588796747776.0000 - val_accuracy: 0.0000e+00
Epoch 61/100
1797/1797 [==============================] - 4s 2ms/step - loss: -414212453646657912832.0000 -
accuracy: 1.3914e-04 - val_loss: -965352631838865096704.0000 - val_accuracy: 0.0000e+00
Epoch 62/100

1797/1797 [==============================] - 5s 3ms/step - loss: -446345531434830135296.0000 - accuracy: 1.3914e-04 - val_loss: -1039676380914290524160.0000 - val_accuracy: 0.0000e+00
Epoch 63/100
1797/1797 [==============================] - 4s 2ms/step - loss: -480515397338852753408.0000 - accuracy: 1.3914e-04 - val_loss: -1118781193575378976768.0000 - val_accuracy: 0.0000e+00
Epoch 64/100
1797/1797 [==============================] - 4s 2ms/step - loss: -516848363254524674048.0000 - accuracy: 1.3914e-04 - val_loss: -1202747360559237169152.0000 - val_accuracy: 0.0000e+00
Epoch 65/100
1797/1797 [==============================] - 5s 3ms/step - loss: -555313502118779813888.0000 - accuracy: 1.3914e-04 - val_loss: -1291475310092853706752.0000 - val_accuracy: 0.0000e+00
Epoch 66/100
1797/1797 [==============================] - 4s 2ms/step - loss: -595784713142051799040.0000 - accuracy: 1.3914e-04 - val_loss: -1384657178920451309568.0000 - val_accuracy: 0.0000e+00
Epoch 67/100
1797/1797 [==============================] - 4s 2ms/step - loss: -638554413640770912256.0000 - accuracy: 1.3914e-04 - val_loss: -1483531175464580153344.0000 - val_accuracy: 0.0000e+00
Epoch 68/100
1797/1797 [==============================] - 5s 3ms/step - loss: -683860274048397213696.0000 - accuracy: 1.3914e-04 - val_loss: -1587994702096229203968.0000 - val_accuracy: 0.0000e+00
Epoch 69/100
1797/1797 [==============================] - 4s 2ms/step - loss: -731708979395627581440.0000 - accuracy: 1.3914e-04 - val_loss: -1698411002272843563008.0000 - val_accuracy: 0.0000e+00
Epoch 70/100
1797/1797 [==============================] - 5s 3ms/step - loss: -782252666907374125056.0000 - accuracy: 1.3914e-04 - val_loss: -1814725751323918073856.0000 - val_accuracy: 0.0000e+00
Epoch 71/100
1797/1797 [==============================] - 5s 3ms/step - loss: -835370372712395440128.0000 - accuracy: 1.3914e-04 - val_loss: -1937282207983551381504.0000 - val_accuracy: 0.0000e+00
Epoch 72/100
1797/1797 [==============================] - 4s 2ms/step - loss: -891608721215463620608.0000 - accuracy: 1.3914e-04 - val_loss: -2067005299025214701568.0000 - val_accuracy: 0.0000e+00
Epoch 73/100
1797/1797 [==============================] - 6s 3ms/step - loss: -950695385376611106816.0000 - accuracy: 1.3914e-04 - val_loss: -2202795864664852922368.0000 - val_accuracy: 0.0000e+00
Epoch 74/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1012668927267647258624.0000 - accuracy: 1.3914e-04 - val_loss: -2345445131061999697920.0000 - val_accuracy: 0.0000e+00
Epoch 75/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1077903778976341426176.0000 - accuracy: 1.3914e-04 - val_loss: -2495936290310305349632.0000 - val_accuracy: 0.0000e+00
Epoch 76/100
1797/1797 [==============================] - 5s 3ms/step - loss: -1146548488921652658176.0000 - accuracy: 1.3914e-04 - val_loss: -2653252936268867698688.0000 - val_accuracy: 0.0000e+00
Epoch 77/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1218431427736531632128.0000 - accuracy: 1.3914e-04 - val_loss: -2818855642591838339072.0000 - val_accuracy: 0.0000e+00
Epoch 78/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1293922312802887794688.0000 - accuracy: 1.3914e-04 - val_loss: -2992392565558328950784.0000 - val_accuracy: 0.0000e+00
Epoch 79/100
1797/1797 [==============================] - 5s 3ms/step - loss: -1373155970634565550080.0000 - accuracy: 1.3914e-04 - val_loss: -3174548815261653270528.0000 - val_accuracy: 0.0000e+00
Epoch 80/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1455953664621353631744.0000 - accuracy: 1.3914e-04 - val_loss: -3364468707772610904064.0000 - val_accuracy: 0.0000e+00

Epoch 81/100
1797/1797 [==============================] - 5s 3ms/step - loss: -1542987697745125441536.0000 - accuracy: 1.3914e-04 - val_loss: -3565037080127509364736.0000 - val_accuracy: 0.0000e+00
Epoch 82/100
1797/1797 [==============================] - 7s 4ms/step - loss: -1634108606793247621120.0000 - accuracy: 1.3914e-04 - val_loss: -3773805662978919366656.0000 - val_accuracy: 0.0000e+00
Epoch 83/100
1797/1797 [==============================] - 8s 4ms/step - loss: -1729139766217834233856.0000 - accuracy: 1.3914e-04 - val_loss: -3992083877918498881536.0000 - val_accuracy: 0.0000e+00
Epoch 84/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1828917860387157704704.0000 - accuracy: 1.3914e-04 - val_loss: -4221354816598536355840.0000 - val_accuracy: 0.0000e+00
Epoch 85/100
1797/1797 [==============================] - 4s 2ms/step - loss: -1933018847248803430400.0000 - accuracy: 1.3914e-04 - val_loss: -4459881215462773620736.0000 - val_accuracy: 0.0000e+00
Epoch 86/100
1797/1797 [==============================] - 5s 3ms/step - loss: -2041569672017267916800.0000 - accuracy: 1.3914e-04 - val_loss: -4708843299088558456832.0000 - val_accuracy: 0.0000e+00
Epoch 87/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2155028435217147756544.0000 - accuracy: 1.3914e-04 - val_loss: -4968964458166037250048.0000 - val_accuracy: 0.0000e+00
Epoch 88/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2273270584171248484352.0000 - accuracy: 1.3914e-04 - val_loss: -5239794614207449661440.0000 - val_accuracy: 0.0000e+00
Epoch 89/100
1797/1797 [==============================] - 5s 3ms/step - loss: -2396835143459971006464.0000 - accuracy: 1.3914e-04 - val_loss: -5523659876420332552192.0000 - val_accuracy: 0.0000e+00
Epoch 90/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2526100837664479510528.0000 - accuracy: 1.3914e-04 - val_loss: -5820586140502543302656.0000 - val_accuracy: 0.0000e+00
Epoch 91/100
1797/1797 [==============================] - 5s 3ms/step - loss: -2660856701289729359872.0000 - accuracy: 1.3914e-04 - val_loss: -6128991517084968026112.0000 - val_accuracy: 0.0000e+00
Epoch 92/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2800939478849228374016.0000 - accuracy: 1.3914e-04 - val_loss: -6449016180706008629248.0000 - val_accuracy: 0.0000e+00
Epoch 93/100
1797/1797 [==============================] - 4s 2ms/step - loss: -2946482870956914114560.0000 - accuracy: 1.3914e-04 - val_loss: -6783271093249633157120.0000 - val_accuracy: 0.0000e+00
Epoch 94/100
1797/1797 [==============================] - 6s 3ms/step - loss: -3098527209126709166080.0000 - accuracy: 1.3914e-04 - val_loss: -7130846527591112769536.0000 - val_accuracy: 0.0000e+00
Epoch 95/100
1797/1797 [==============================] - 4s 2ms/step - loss: -3256586949023787646976.0000 - accuracy: 1.3914e-04 - val_loss: -7493037831573269905408.0000 - val_accuracy: 0.0000e+00
Epoch 96/100
1797/1797 [==============================] - 4s 2ms/step - loss: -3420781436038274875392.0000 - accuracy: 1.3914e-04 - val_loss: -7868853650328129634304.0000 - val_accuracy: 0.0000e+00
Epoch 97/100
1797/1797 [==============================] - 5s 3ms/step - loss: -3591649413275595046912.0000 - accuracy: 1.3914e-04 - val_loss: -8260316100088381308928.0000 - val_accuracy: 0.0000e+00
Epoch 98/100
1797/1797 [==============================] - 4s 2ms/step - loss: -3769444771164741173248.0000 - accuracy: 1.3914e-04 - val_loss: -8666489558031438708736.0000 - val_accuracy: 0.0000e+00
Epoch 99/100

```
1797/1797 [==============================] - 4s 2ms/step - loss: -3953492532711561101312.0000 - accuracy: 1.3914e-04 - val_loss: -9088042808701966352384.0000 - val_accuracy: 0.0000e+00
Epoch 100/100
1797/1797 [==============================] - 5s 3ms/step - loss: -4144999662616190124032.0000 - accuracy: 1.3914e-04 - val_loss: -9525544431552919764992.0000 - val_accuracy: 0.0000e+00
<keras.callbacks.History at 0x7f960164d910>
```

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

#classification.predict([[129,99,1,0,0,1,0,1,1,1,0,1,1,1,1,1]])

print(y_pred)

(y_pred)
```

```
WARNING:tensorflow:Layers in a Sequential model should only have a single input tensor. Received: inputs=(<tf.Tensor 'IteratorGetNext:0' shape=(None, 8) dtype=float32>,). Consider rewriting this model with the Functional API.
71/71 [==============================] - 0s 3ms/step
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
array([[1.],
       [1.],
       [1.],
       ...,
       [1.],
       [1.],
       [1.]], dtype=float32)
```

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

```
[932]
array([932])
```

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

```
y_pred = classification.predict(x_test)
y_pred
```

```
71/71 [==============================] - 0s 2ms/step
array([[1.],
       [1.],
       [1.],
       ...,
       [1.],
       [1.],
       [1.]], dtype=float32)
```

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

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

```
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)

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

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

# Milestone5

```
from sklearn import model_selection
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import get_scorer_names
import sklearn
```

```
dfs = []

models = [
          ('RF',RandomForestClassifier()),
          ('DecisionTree',DecisionTreeClassifier()),
          ('ANN',MLPClassifier())
        ]
results = []
names = []
x_train=[]
y_train=[]
cv_kfold=[]
scoring_scoring=[]
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_s
tate=90210)
    cv_results = model_selection.cross_validate[model,x_train, y_tr
ain,cv_kfold, scoring_scoring(0)]
    clf = model.fit(x_train, y_train)
    y_pred= clf.predict([[129,99,1,0,0,1,0,1]])
    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

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

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

```
array([[0, 0, 2, ..., 0, 1, 1], [0, 0, 0, ..., 1, 0, 0], [0, 0, 0,
..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0,
0], [0, 0, 0, ..., 0, 0, 0]])
```

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

desacc
```

```
0.0
```

```
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 : 13.840676457498887%
```

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

```
array([[ 0, 1936], [ 0, 311]])
```

```
parameters = { 'n_estimators' :[1,20,30,55,68,74,90,120,115], 'crit
erion':['gini','entropy'],'max_features' : ["auto", "sqrt", "log2"]
, 'max_depth' : [2,5,8,10], 'verbose' : [1,2,3,4,6,8,9,10]
}
estimator =[]
RCV = RandomizedSearchCV[estimator:rf],[params_distributions_parame
ters],[cv_10],[n_iter_4]
RCV.fit(x_train,y_train)
```

```
bt_params = []
bt_params
[]
```

```
bt_params = RCV.best_params_

bt_score = RCV.best_score_

bt_score = []
bt_score

[]

entry.get= []
entry.get[fit]
fit =[]
model = RandomForestClassifier (verbose=10,n_estimators=120,max_fea
tures='log2',max_depth=10, criterion='entropy')
Classifier.fit(x_train,y_train)

y_predict_rf = Classifier.predict(x_test)
```

# Milestone 6

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

from flask import Flask, request,render_template
import numpy as np
import pandas as pd
import pickle
import os

import pickle
from flask import Flask, request,render_template
model = pickle.load(open('flight.pkl','rb'))
app = Flask(__name__)

@app.route('/')
def home():
  return render_template("index.html")
  @app.route('/prediction') methods =['post']
```

# Thank You