Flight Delay Prediction

Name: Rhythm Paudel

Email: rhythmpaudel12@gmail.com

Github: github.com/rhythm-paudel

LinkedIn: linkedin.com/in/rhythmpaudel

Table of Contents

Table of Figures

1. Introduction

John McCarthy is known to be the father of Artificial Intelligence or AI. However, the first use of AI was in before John McCarthy was given this title. The first use of AI was in the checkers game developed by a computer scientist named Arthur Samuel (Salesforce, Inc., 2024). Artificial Intelligence is an advanced technology which imitates human learning pattern, helping in problem solving and other intensive tasks as well (Stryker, 2024).The growth of AI has been growing exponentially. Many AI tools like ChatGPT, ClaudeAI, or Gemini has come in regular use for a normal email writing user to intense coder.

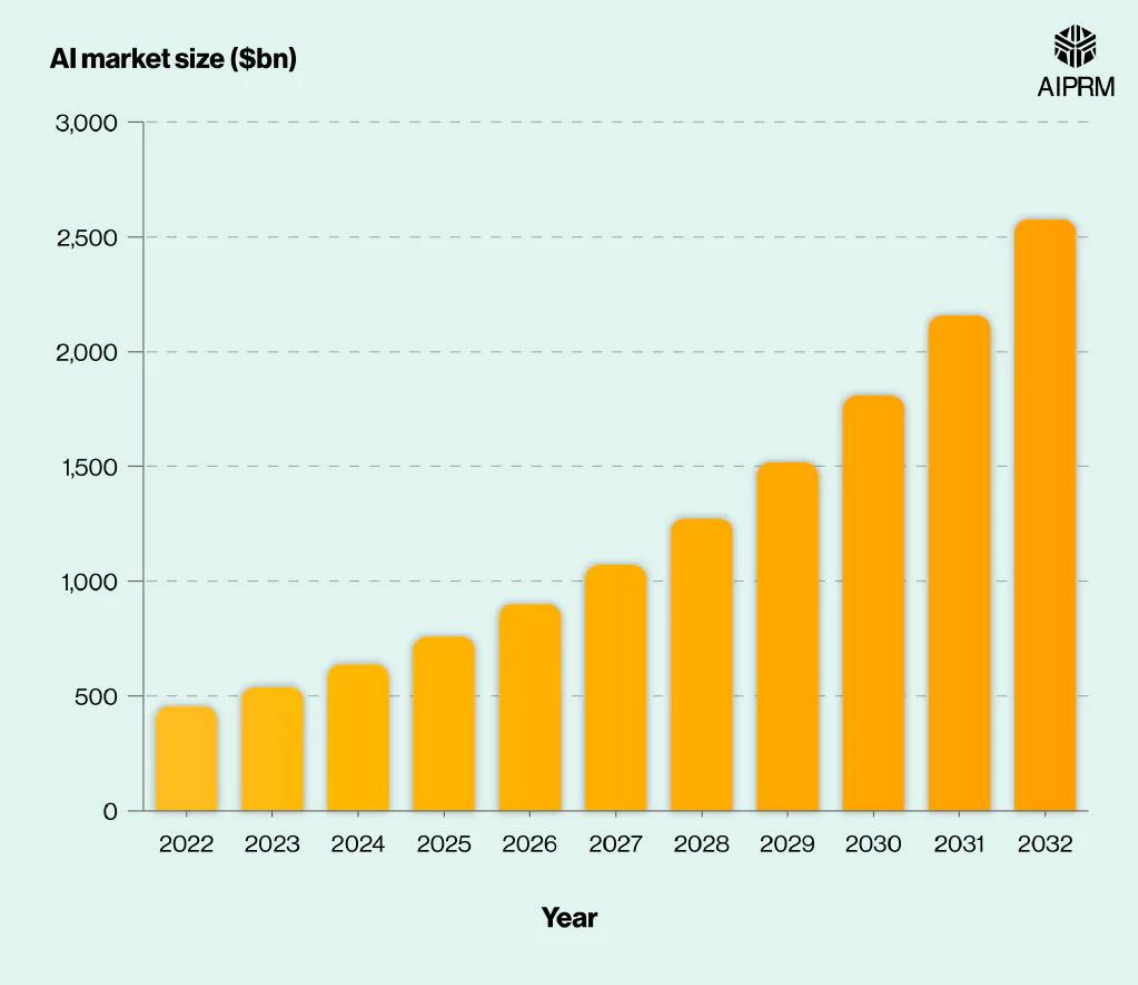


Figure 1: Growth of AI in Market

(AIPRM, 2024)

Several hundred of new models are released every day that is designed to handled certain tasks.Artificial Intelligence has many branches and subsets that deals with variety of domain that are categorized. Similarly, machine learning model is one subset that falls under AI.

Machine Learning is a self-learning tool that uses neural-network and deep-learning. Machine learning or ML tools are trained by providing a large dataset and letting ML algorithms work on it (Google , 2024).

1.1. Problem domain

This model will help aviation system to dive deep into the cause and coming up with efficient solutions and providing beforehand prediction with might enhance the customer experience. Throughout this development of this model several algorithms like Linear Regressor, Decision Tree, Random Forest, K—Nearest Neighbors, and XGBoost will be used.

Being a researcher, this project brings out a lot of potential from the machine learning field. This model could be contributing the aviation industry a lot. With the help of many pre-defined algorithms, a model for predicting flight delay, would be prepared, getting the answers with accuracy and precision.

This topic was chosen to build a model that could have the potential to solve a larger problem being faced all around the world, especially in context of Nepal.

1.2. Aim

To develop a proper working model to estimate the delay time of flight to contribute to aviation industry.

1.3. Objectives

- Select a dataset and clean the data to obtain a proper expected result.
- Analyse various different airplane delay factor such as weather, arrival time, delay time.
- Perform training on various five different algorithms to check the efficiency of each algorithm.
- Tune the model to improve accuracy further to prevent overfitting the data if necessary.
- Evaluate the model and work accordingly with further refinement.

2. Background

2.1. Research on problem scenario

Flight delays have been a major problem in the aviation industry. Even though the major flight is seldom delayed, the consequences faced by the customer due to the delayed flight are troublesome. Since flights are known to be fastest and quickest mean of transportation, one would use it for emergency cases, knowing its reliable. The results may not always turn out to be affirmative.

According to research done on British Airways, the satisfaction of customer is highly dependent on plane arrival record. This research also shows that dealing with flight delays plays a vital role on maintaining a loyalty of customers, also at the time of active plane delay (Efthymiou, et al., 2024).

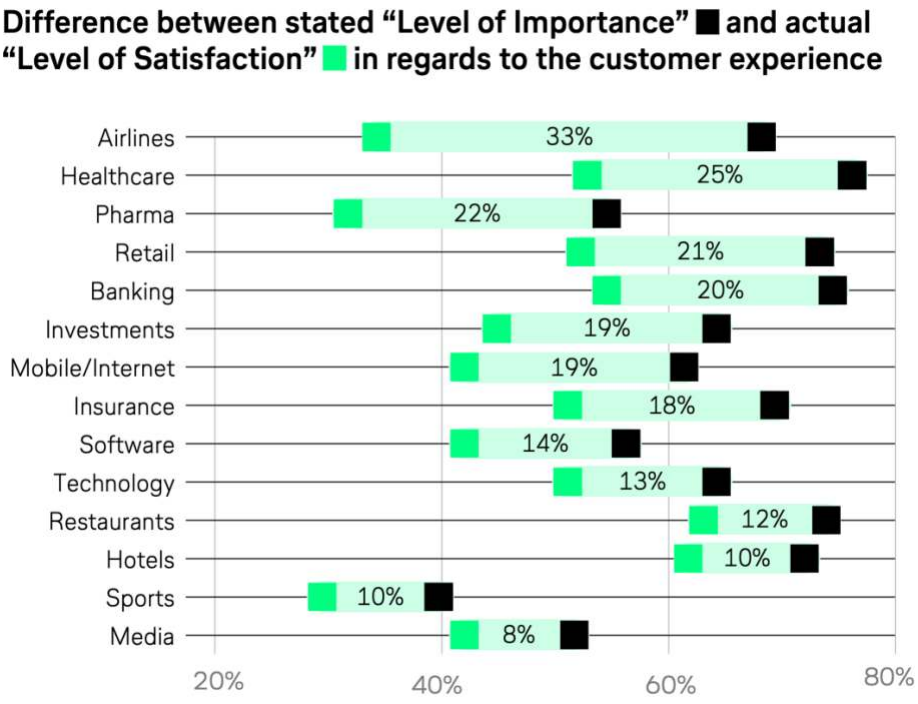


Figure 2: Satisfaction Level of Customers (Industry-Wise) *(Mehra, 2023)*

Above presented diagram shows that the highest dissatisfaction level of customers is in Airlines sector. According to TNMT, the major reason for this dissatisfaction is because of uninformed flight irregularities, most of the being

delayed or cancelled. This means that the uninformed flight delay is the major problem of aviation rather than flight being delayed.

2.2. Existing Work in a domain

There has been a couple of models that were made for predicting flight delays. The model worked fine in some scenario but were very inaccurate in some scenarios. For this model to be developed a lot research were performed, which included pre-existing models. The research was carried out along with learning what algorithms were used in each model and how well the algorithms performed.

With deep analysis and studying the effectiveness of the model, there were some things that were needed to redo or fix, like reducing the complexity of the model, for preparing as lightweight flight prediction ML model.

Spatiotemporal Propagation Learning was developed for predicting the delays. The datasets used for this model was very specific making it limited to adjust in only some scenarios. This model was heavily dependent on spatiotemporal data structures, making it hard to predict the delays only based on historical data.

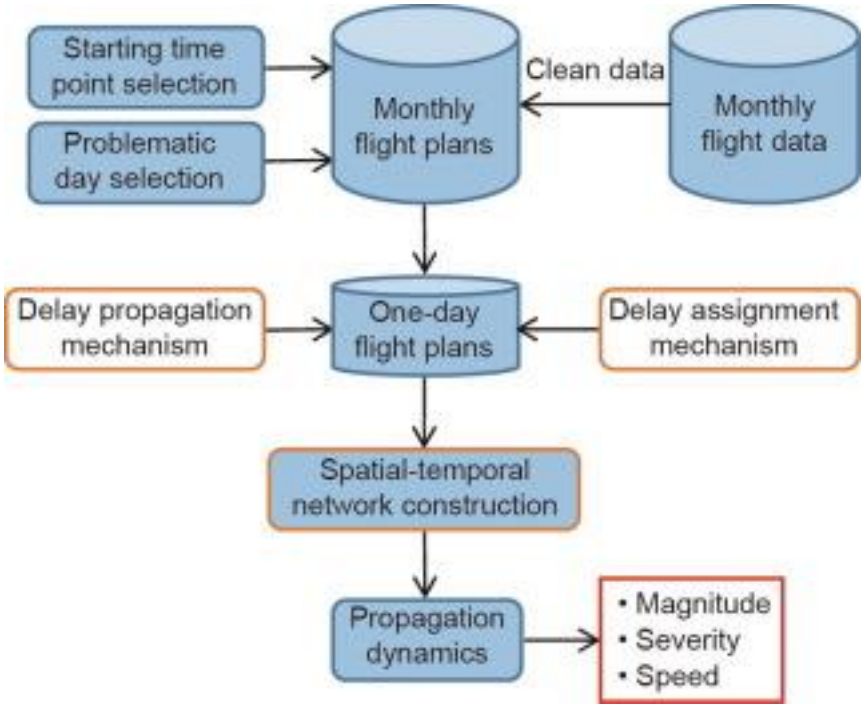


Figure 3: Spatiotemporal Propagation Flight Delay

(Duong, et al., 2024)

Similarly, Flight Delay Regression Prediction Model Based on Att-Conv-LSTM was also developed for a similar purpose of detecting plane delays.

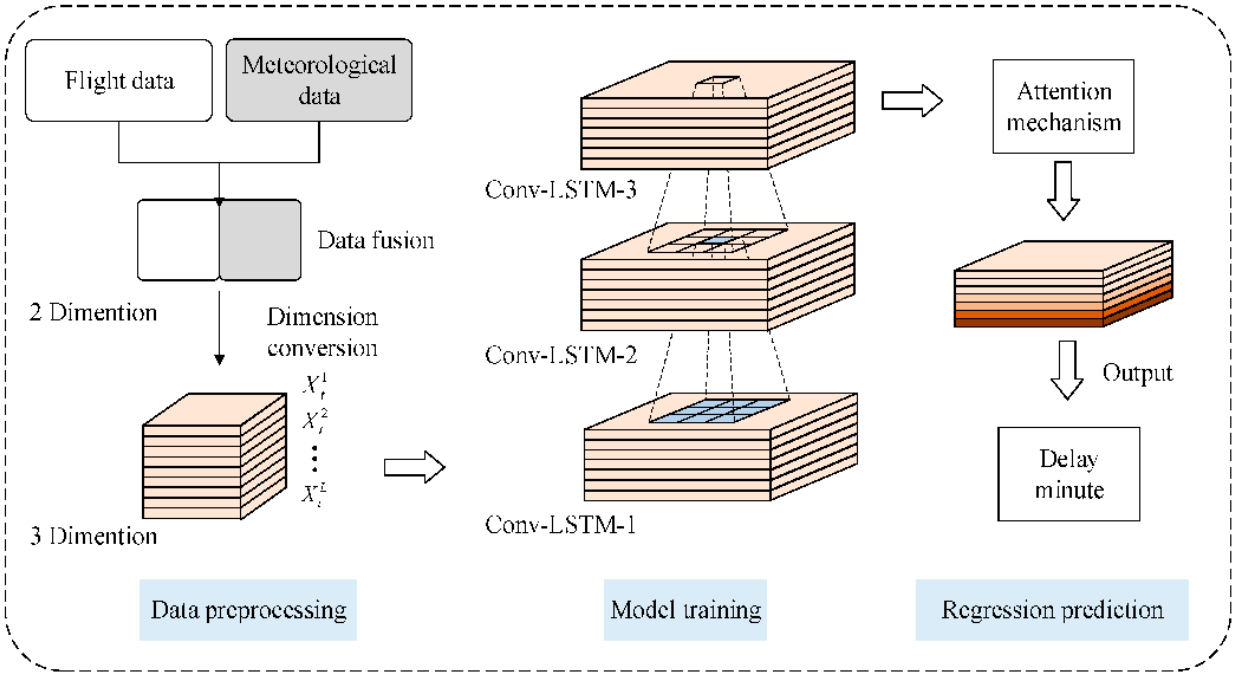


Figure 4: Flight Delay Regression Prediction Model Based on Att-Conv-LSTM

Since this model was developed using a deep learning architecture based on LSTMs, the computational power required was very high and the overall complexity of the process was very high. This could even result in overfitting most of the time and making the prediction less accurate for newer data.

(Qu , et al., 2023)

There are couple of other models prepared but most of them require a very high computation power, and sometimes very restrictive data. Like Flight Delay Prediction Using Deep Convolutional Neural Network Based on Fusion of Meteorological Data, requires a high-end hardware resource that cannot be compromised for obtaining the desired output.

Other models like Graph Convolutional Networks (GCNs), Deep Learning Models with Temporal Features and many more requires a very complex dataset, with a very high computational power for predicting delays.

2.3. Block diagram

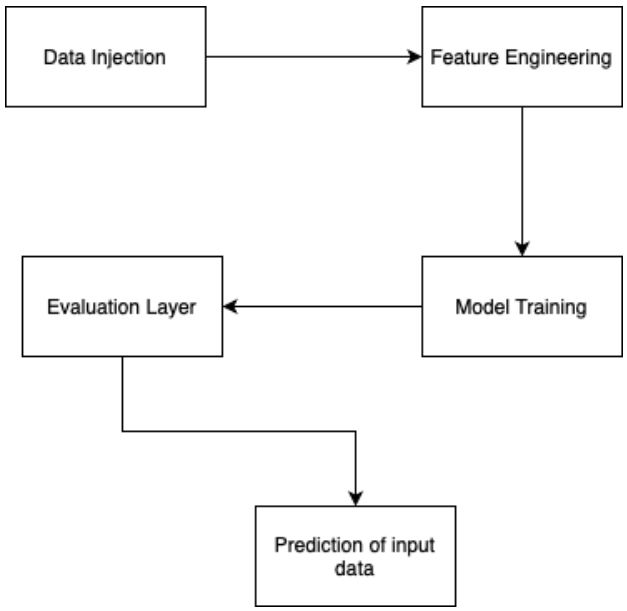


Figure 5: Block Diagram.

2.4. Advantages on working in this problem domain

There are several advantages on why working in this domain is beneficial. Already the delays are major problem occurring in aviation history.

- Since the delays are not a solvable problem, the closest solution for it could be to aware the customers about it before hand.
- Similarly, the aviation is highly dependent of data for several reasons like to keep track of passengers, security reasons and more. This means the dataset for working in this domain is always adequate, mostly providing already accurate dataset in most scenarios.
- This addresses a spiking problem in aviation sector, which is the decrease in level of satisfaction to the customers due to flight delays. Furthermore, passengers could schedule accordingly, reducing other activity disruption.
- Due to various possibilities for delays in flights, by working in this domain a complex pattern that could contribute the field of AI would be useful.
- This model could also help in minimizing the overall financial loss for airlines.
- Furthermore, using various ML models from a simple linear regress to XGBoost, there will be a clear understanding of the models are working, tuning them and understanding which models works best for what situation.

2.5. Drawbacks on working in this problem domain

- The flight delays may not be accurate even with training data, as something new could come up every time making it hard to predict.
- If there are any data inconsistencies or if any steps are missed during the data cleaning process, then the result might not be as expected.
- If the dataset or conditions are changed, the model might struggle with the prediction.
- Some algorithms like XGBoost or Random forest, required computational power could be high.
- These algorithms sometime may result in overfitting due to noises capture.
- Since the data is highly dependent on historical data, sometimes biasness may be seen.

2.6. Understanding dataset

The dataset for this model preparation was collected from official United States government. The dataset can be extracted from this website https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp?20=E. From this dataset target variable and dependent variable are separated. The model will predict how many minutes the flight will be delayed on the basis of their past history on which it was used to train from the dataset. Currently the data are taken from the US government site, but studying the column, it can be understood that the data can easy be collected as the dataset only consist of frequently stored records of airplanes.

Various methods are followed before processing the dataset. Such as the dataset are first checked, the nature of the dataset is understood. After the dataset is understood, data should be pre-processed like filtering null values, encoding labels. After completion of these steps, the visualization should be performed in order to have a clear idea of the data on how it is correlated. For visualization various chart is used like histogram for comparison and, scatter plot for checking actual vs predicted data.

Data Loading

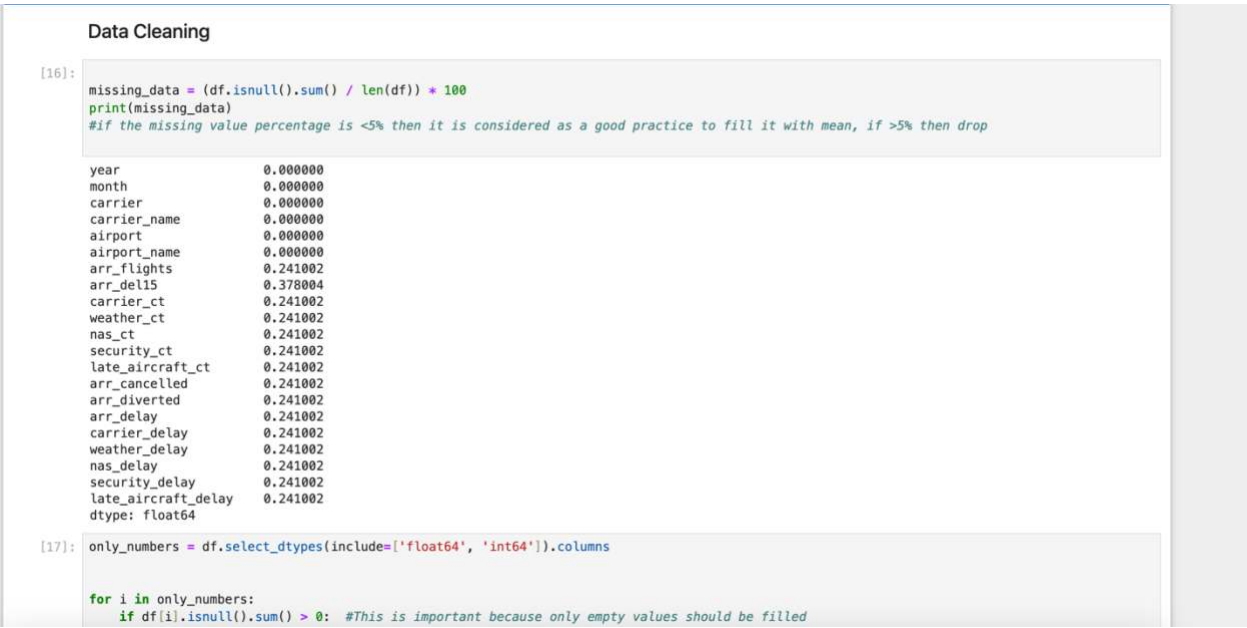
```
[3]: #Loading the dataset

pd.set_option('display.max_columns', None) #setting this as none so all columns will be displayed
df = pd.read_csv('Airline_Delay_Cause.csv')
df.tail()
```

[3]:

	year	month	carrier	carrier_name	airport	airport_name	arr_flights	arr_del15	carrier_ct	weather_ct	nas_ct	security_ct	late_aircraft_ct	arr_cance
99994	2020	4	DL	Delta Air Lines Network	SEA	Seattle, WA: Seattle/Tacoma International	945.0	72.0	37.44	0.56	30.66	0.0	3.33	1
99995	2020	4	DL	Delta Air Lines Network	SFO	San Francisco, CA: San Francisco International	398.0	21.0	7.98	0.00	13.02	0.0	0.00	
99996	2020	4	DL	Delta Air Lines Network	SJC	San Jose, CA: Norman Y. Mineta San Jose Intern...	111.0	4.0	2.78	0.00	1.22	0.0	0.00	
99997	2020	4	DL	Delta Air Lines Network	SJU	San Juan, PR: Luis Munoz Marin International	50.0	0.0	0.00	0.00	0.00	0.0	0.00	
99998	2020	4	DL	Delta Air Lines Network	SLC	Salt Lake City, UT: Salt Lake City International	1992.0	163.0	93.73	0.00	60.35	0.0	8.92	3

Figure 6: Data Loading.

Before proceeding with model training, the data should be filtered properly or else the inconsistencies will result in unwanted errors and prediction. The data are also scaled properly as it will be easier to compare the values.



Data Cleaning

[16]:
```
missing_data = (df.isnull().sum() / len(df)) * 100
print(missing_data)
#if the missing value percentage is <5% then it is considered as a good practice to fill it with mean, if >5% then drop
```

```
year                  0.000000
month                 0.000000
carrier               0.000000
carrier_name          0.000000
airport               0.000000
airport_name          0.000000
arr_flights           0.241002
arr_del15             0.378004
carrier_ct            0.241002
weather_ct            0.241002
nas_ct                0.241002
security_ct           0.241002
late_aircraft_ct      0.241002
arr_cancelled         0.241002
arr_diverted          0.241002
arr_delay             0.241002
carrier_delay         0.241002
weather_delay         0.241002
nas_delay             0.241002
security_delay        0.241002
late_aircraft_delay   0.241002
dtype: float64
```

[17]:
```
only_numbers = df.select_dtypes(include=['float64', 'int64']).columns


for i in only_numbers:
    if df[i].isnull().sum() > 0:  #This is important because only empty values should be filled
```

Figure 7: Example for data cleaning.

3. Proposed Solution to the chosen problem.

3.1. Approach to solving a problem

Even with already existing models the aviation system is yet implementing the system as it is not working properly for predicting the delays beforehand the flight which raises the dissatisfaction level of customers. This results in customers churn resulting in downfall of the business. If aviation system could just predict the approximate delay based on their past record, then it would be helpful for the customers as they would be mentally prepared instead of being informed right at the time of flight.

The reason of existing models not being widely used in aviation sector is because of the complexity, computational power required, and very specific data sets required for the model to work properly. This results in less affordability by small scale airports or aviation system.

Many airports in many countries are not ready to invest in such a huge investment mostly if the airport does not handle large airplane traffic. So, to solve this problem a simplistic model, that requires a normal processing power and is able to handle the prediction based on the past record or the flight is being developed.

With this airplane delay detection, the required dataset does not contain a very detailed requirement and can be used by any type of aviation system as the data required are easy to provide.

3.2. Algorithms Used

For any machine learning algorithm to be developed various algorithms are needed to be used. All ML algorithms are made for different purpose and chosen according to the dataset available and output required by the developer. Choosing an efficient algorithm is one of the crucial steps as the outcome may not be accurate and precise if any wrong one is chosen.

For this ML model which predicts the delay time of flight will be developed using several algorithms and a comparison will be performed to analyse the result. Various algorithms used in this development were chosen as they were suited best for the prediction model to get the accuracy along with the model being lightweight.

I) Linear Regression

This algorithm was developed by Sir Francis Galton. Linear Regression algorithm is used to predict the outcome variable which is dependent to another independent variable (IBM, 2024). Since linear regression is simple, the workload is also light. This algorithm is used as it would act as a good model for knowing the importance of features as well.

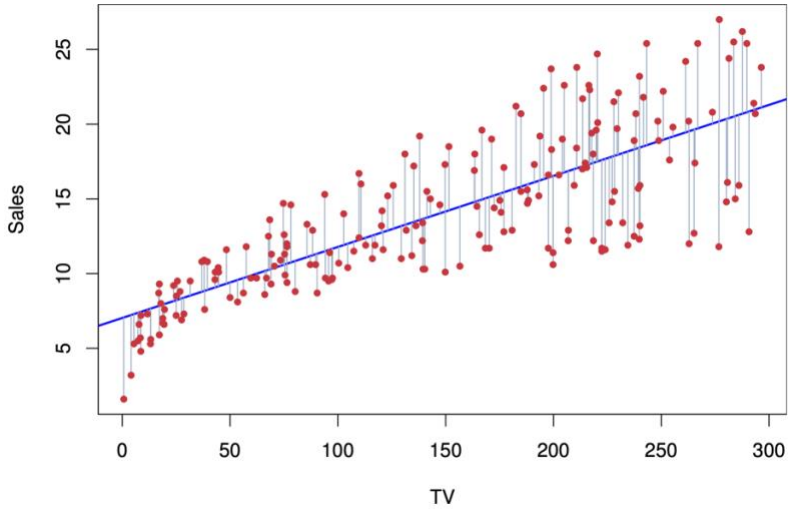


Figure 8: Linear Regression Model

(Bacallado & Taylor, 2022)

II) Decision Tree Regression

This algorithm was a part of contribution of Ross Quinlan. Decision Tree Regression learn by plotting a sine curve also by having noisy observation (Scikit Learn, 2024). Since decision tree regression can handle non-linear relationships as well, making prediction with this model could be more accurate.

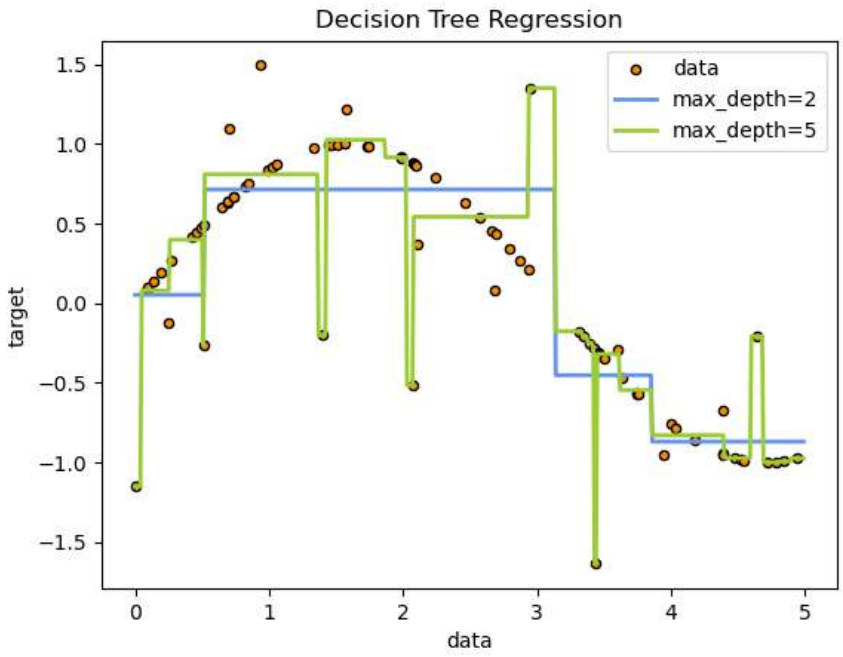


Figure 9: Decision Tree Regression

(Scikit Learn, 2024)

III) Random Forest Regression

Random Forest was developed by Leo Breiman. This algorithm can also be seen as group of decision tree. Each tree picks up the best method for splitting and each tree work together to give more accurate results and prevent overfitting (Scikit Learn, 2024).

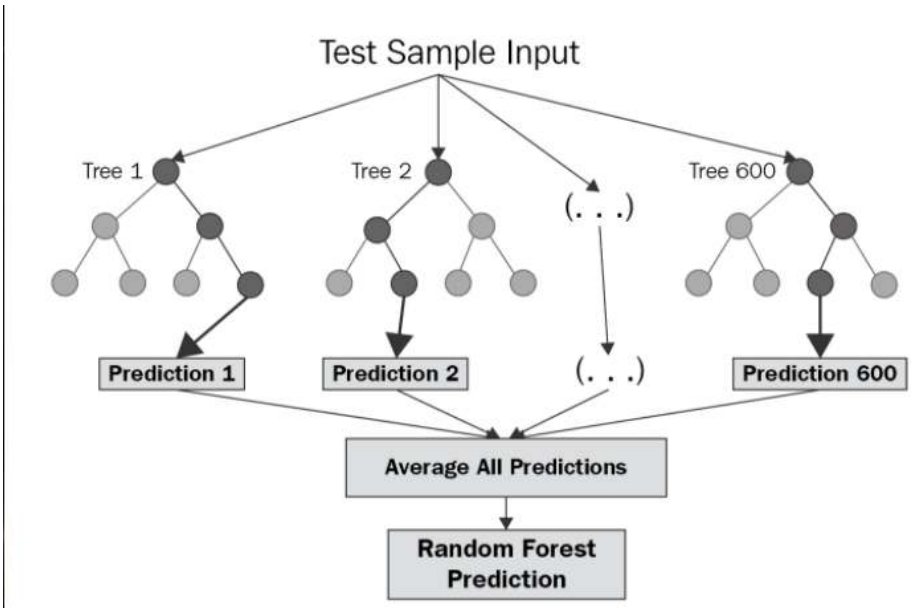


Figure 10: Random Forest Regression

(Chaya, 2020)

IV) K-Nearest Neighbours (KNN) Regression

KNN algorithm is introduced by Evelyn Fix and Joseph Hodges. This algorithm is used to predict the outcome by averaging the outcomes of nearby data points. The analyst of KNN algorithm is assigned to choose the value of K or the number of points to make the prediction (BIO Statistics Collaboration of Australia, 2024). Since this algorithm predicts the delay by making a comparison to the past data, it would be useful to recognize a pattern of clusters of flights.

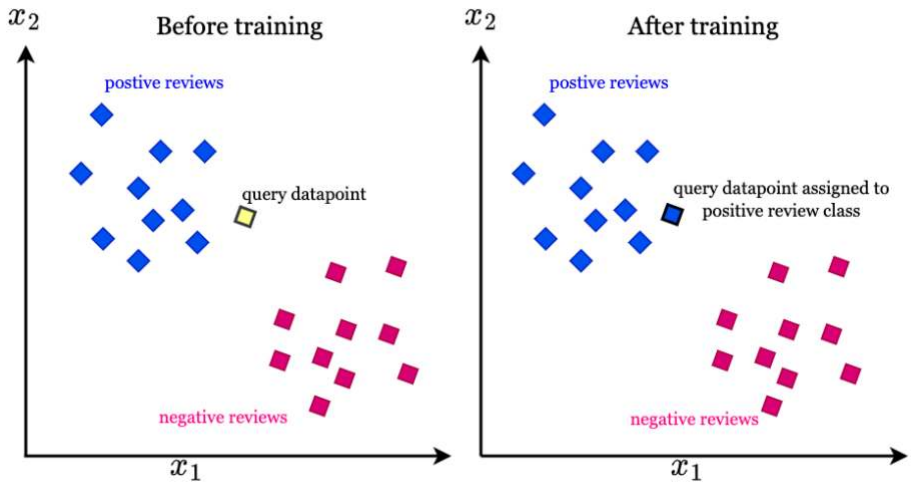


Figure 11: KNN Regression

(H, 2024)

V) XGBoost Regression

This algorithm was developed by Tianqi Chen. XGBoost Regression is similar to Random Forest Regression, where multiple decision trees are combined to make a better prediction (NVIDIA Corporation, 2024). However, unlike Random Forest XGBoost does not combines the trees result but build the trees sequentially.

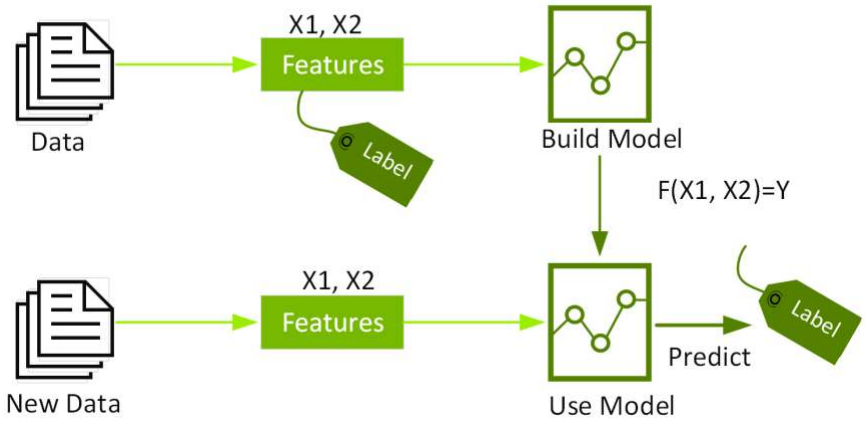


Figure 12: XGBoost Regression.

(NVIDIA Corporation, 2024)

3.3. Pseudocode

I) Overall flow of model development

START

LOAD dataset

    DISPLAY structure of dataset

    VISUALIZE correlation between data

CLEAN dataset

    HANDLE missing values

    DROP values or fill with mean

ENCODE categorical variables

    APPLY label encode to columns with more than two values

    APPLY one-hot encoding to categorical data

SELECT features and target variable

IDENTIFY columns with low correlation

REMOVE repeated type of columns and low correlated data

SCALE the feature for better comparison and to make sure all the features get equal weight.

SPLIT train test data

TRAIN models on various algorithms

EVALUATE models

    CALCULATE several error functions like R^2, MAE and MSE

GENERATE report of classification

    DEFINING a threshold for delay (for an example 15)

DIVIDING in on-time and delayed by comparison with threshold.

PRINTING the result with precision,recall, f1-score

CATEGORIZING feature importance //for knowing which feature afftects the output most

EMSEMBLING model

VISUALIZE the results

TEST new data

OUTPUT predictions

END

II) Linear Regression Model

START

INITIALIZE decision tree regressor model

SET max depth

TRAIN model (X_train,y_train)

EVALUATE model performance

IF newdata

THEN preprocess

ELSE

END

END

III) Decision Tree Regressor Model

START

INITIALIZE decision tree regressor model

SET max depth

TRAIN model (X_train,y_train)

EVALUATE model performance

IF newdata

THEN preprocess

ELSE

END

END

IV) Random Forest Regressor Model

START

INITIALIZE random forest model

SET max depth and number of trees

TRAIN model (X_train,y_train)

EVALUATE model performance

IF newdata

THEN preprocess

ELSE

END

END

V) KNN Regressor Model

START

INITIALIZE KNN model

SET value of K

TRAIN model (X_train,y_train)

EVALUATE model performance

IF newdata

THEN preprocess

ELSE

END

END

VI) XGBoost Regressor Model

START

INITIALIZE XGBoost Regressor Model

SET max_depth, n_estimators

TRAIN model (X_train,y_train)

EVALUATE model performance

IF newdata

THEN preprocess

ELSE

END

END

3.4. Flowchart

I) Overall flowchart of model development

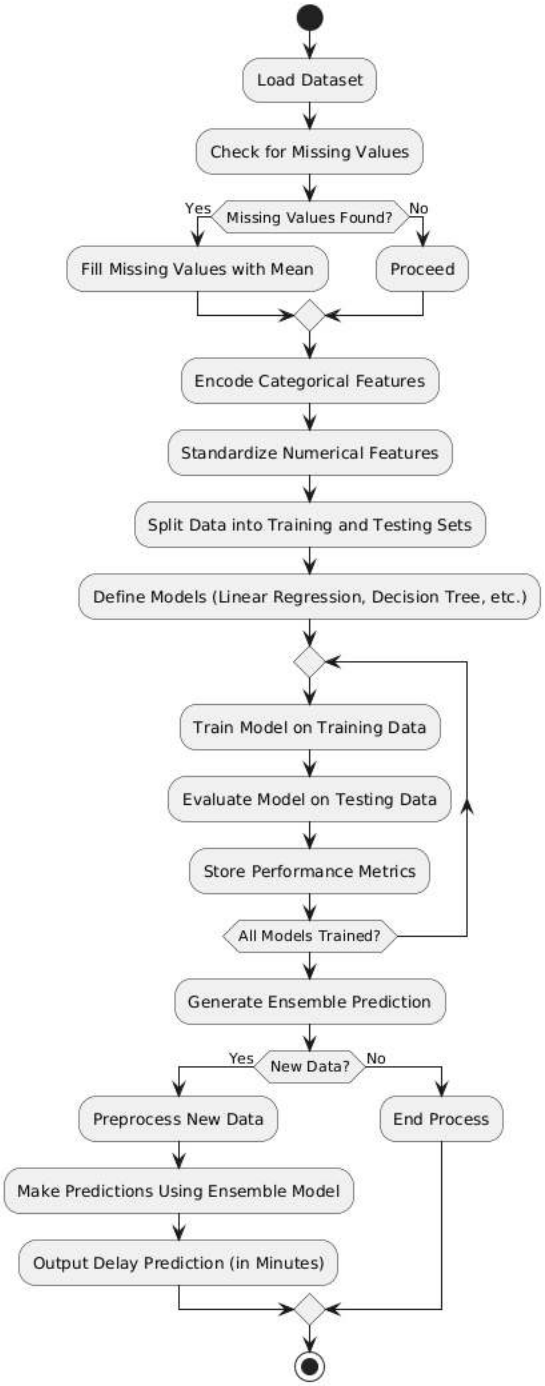


Figure 13: Flowchart of ML Model

II) Linear Regression Model

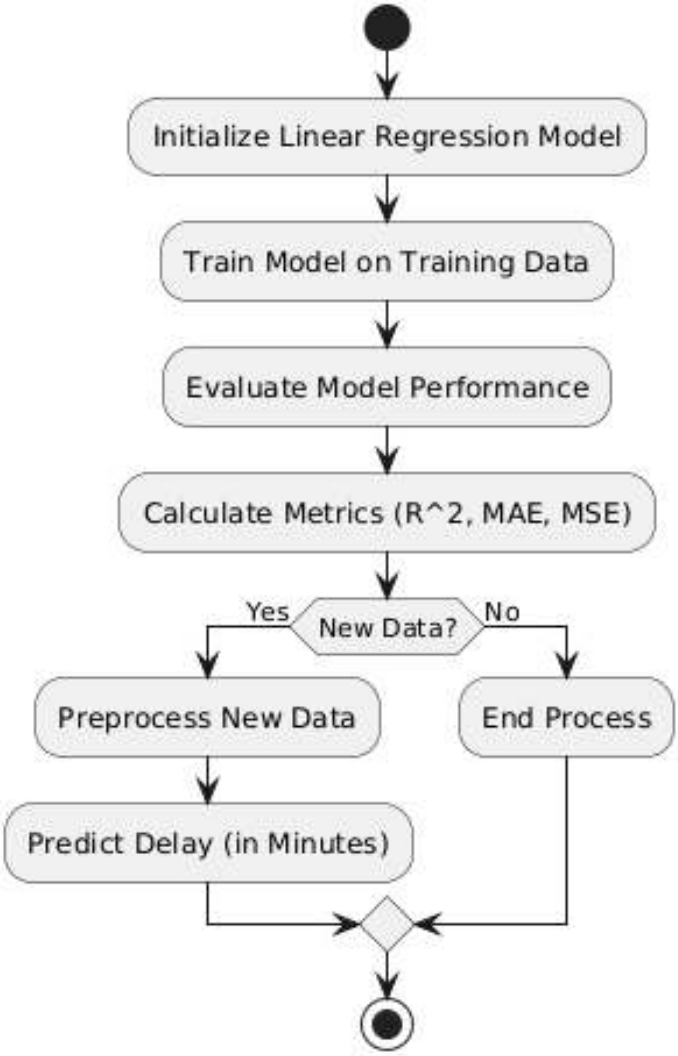


Figure 14: Flowchart of Linear Regression Model

III) Decision Tree Regressor Model

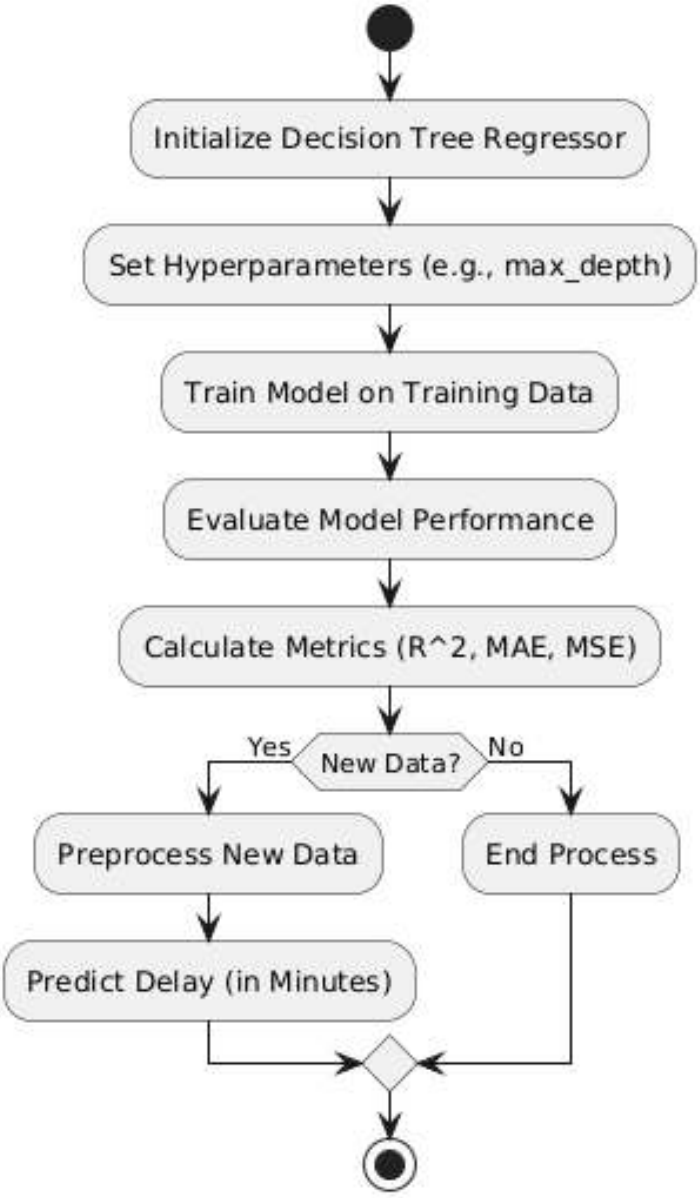


Figure 15: Flowchart of Decision Tree Regressor Model

IV) Random Forest Regressor Model

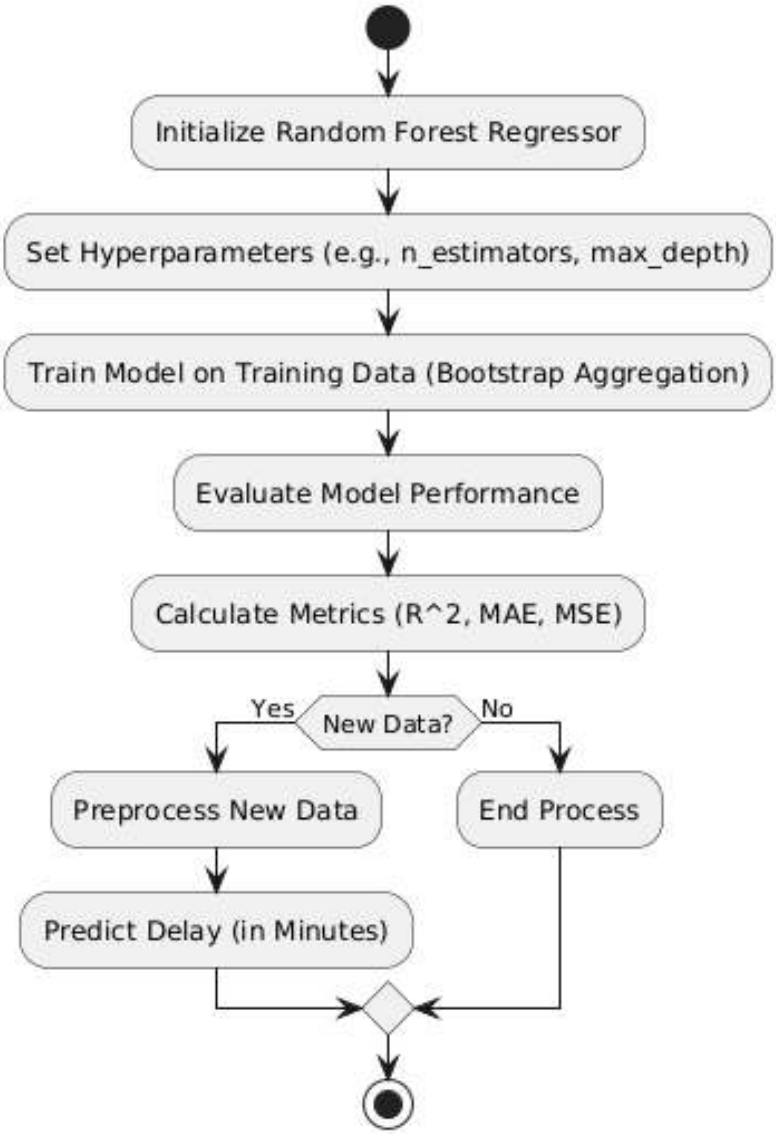


Figure 16: Flowchart of Random Forest Regressor Model

V) KNN Regressor Model

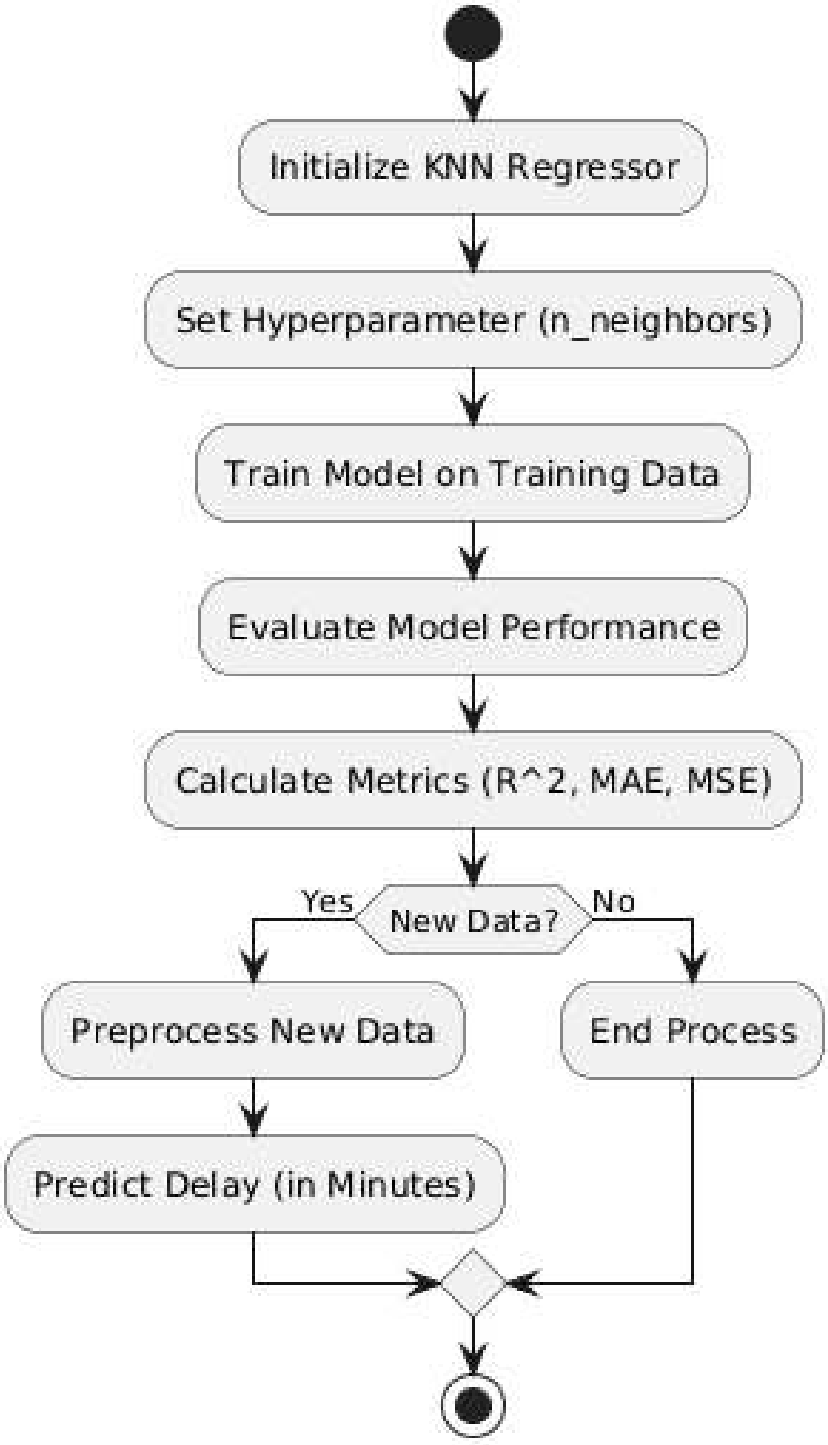


Figure 17: Flowchart of KNN Regressor Model

VI) XGBoost Regressor Model

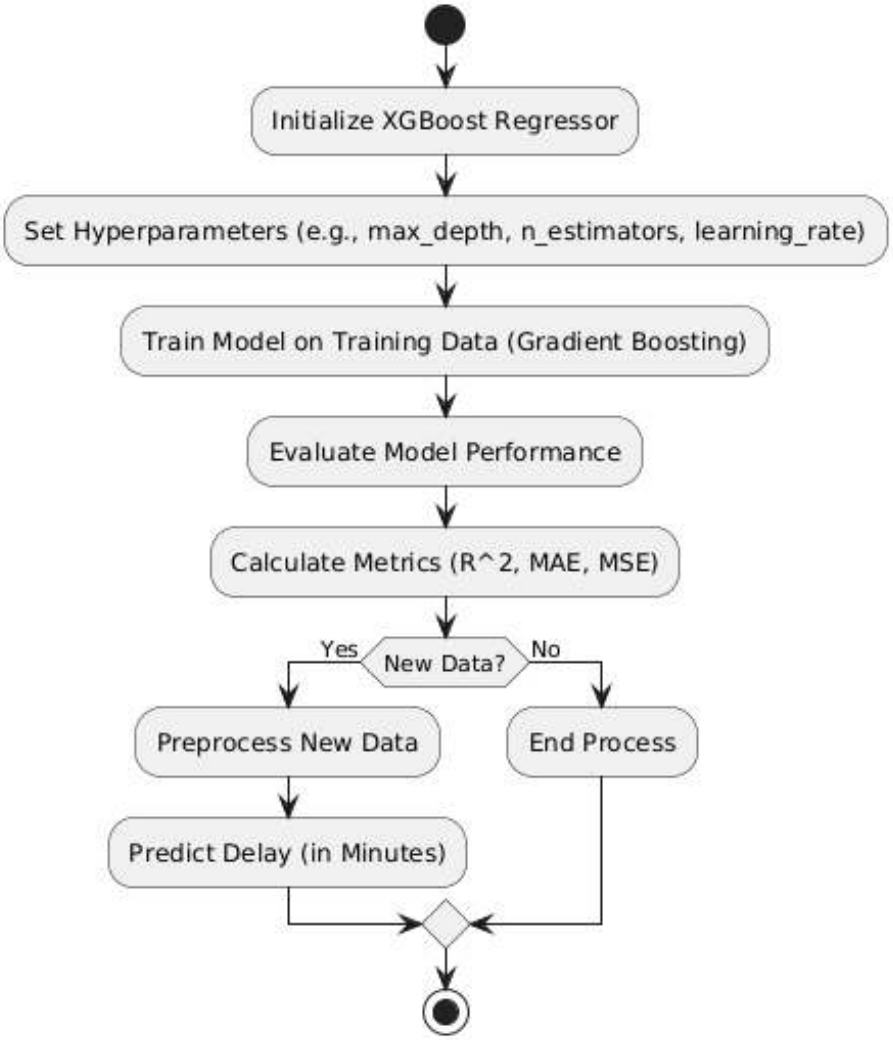


Figure 18: Flowchart of XGBoost Regressor Model

3.5. Development

For this model development there are several dependencies, which are required for completing the model. This model will be developed in the language python, which also is among the highest recommended language for AI/ML model development. Additionally, using python, jupyter notebook (a web-based application) was used to create this model for proper segregation of tasks and for a proper documentation.

3.5.1. Libraries used

Several libraries were also involved in this project. Various libraries used were in-use for various purpose.

```
[1]: #IMPORTING NECESSARY LIBRARIES

     import matplotlib.pyplot as plt
     import numpy as np
     from sklearn.metrics import mean_absolute_error, mean_squared_error
     from sklearn.linear_model import LinearRegression
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.ensemble import RandomForestRegressor
     from sklearn.neighbors import KNeighborsRegressor
     from xgboost import XGBRegressor
     import pandas as pd
     from sklearn.preprocessing import LabelEncoder
     from sklearn.preprocessing import StandardScaler
```

Figure 19: Importing libraries.

Numpy was used for performing complex mathematical computions for this project. Similarly, for visualization matplotlib was used. To calculate the function errors of the model like mean absolute error, mean squared error, a method from sklearn.metrics library was used. For training the model several models like linear regression, decision tree regressor, random forest regressor, and KNeighborsRegressor were imported from sklearn library. However, for XGBRegressor model xgboost library was used. For converting the data and scaling pandas library was used in the project.

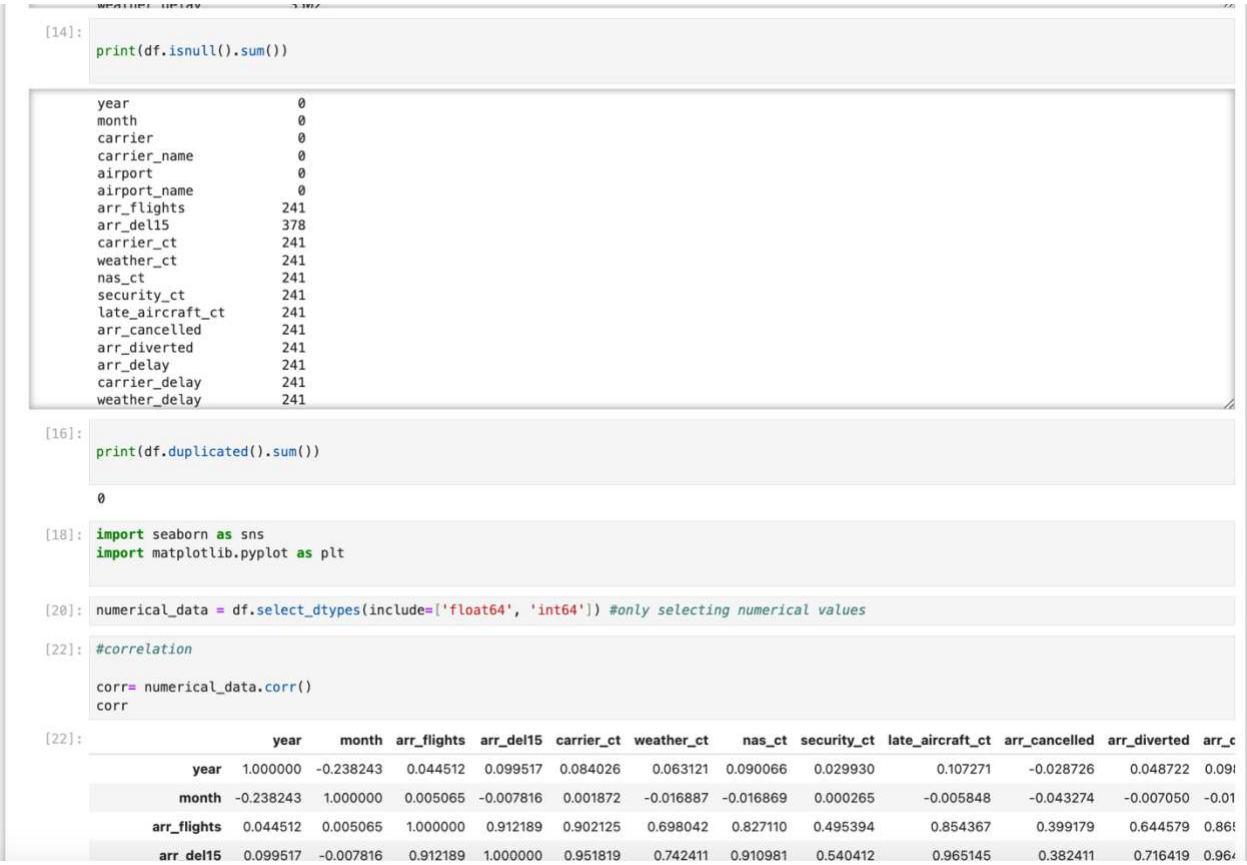
3.5.2. Preprocessing data

Before processing the data, required data should be loaded properly. Once, the data is loaded, there should be a clear understanding of the model.

Data Understanding

```
[6]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   year                99999 non-null  int64
 1   month               99999 non-null  int64
 2   carrier             99999 non-null  object
 3   carrier_name        99999 non-null  object
 4   airport             99999 non-null  object
 5   airport_name        99999 non-null  object
 6   arr_flights         99758 non-null  float64
 7   arr_del15           99621 non-null  float64
 8   carrier_ct          99758 non-null  float64
 9   weather_ct          99758 non-null  float64
 10  nas_ct              99758 non-null  float64
 11  security_ct         99758 non-null  float64
 12  late_aircraft_ct    99758 non-null  float64

[8]: len(df)

[8]: 99999

[10]: df.memory_usage()

[10]: Index                 132
      year                799992
      month               799992
      carrier             799992
      carrier_name        799992
      airport             799992
      airport_name        799992
      arr_flights         799992
      arr_del15           799992
      carrier_ct          799992
      weather_ct          799992
      nas_ct              799992
      security_ct         799992
      late_aircraft_ct    799992
      arr_cancelled       799992
      arr_diverted        799992
      arr_delay           799992
      carrier_delay       799992

[12]: df.nunique()
```

Figure 20: Data understanding 1.



[14]:
```
print(df.isnull().sum())
```

```
year                  0
month                 0
carrier               0
carrier_name          0
airport               0
airport_name          0
arr_flights         241
arr_del15           378
carrier_ct          241
weather_ct          241
nas_ct              241
security_ct         241
late_aircraft_ct    241
arr_cancelled       241
arr_diverted        241
arr_delay           241
carrier_delay       241
weather_delay       241
```

[16]:
```
print(df.duplicated().sum())
```

```
0
```

[18]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

[20]:
```
numerical_data = df.select_dtypes(include=['float64', 'int64']) #only selecting numerical values
```

[22]:
```
#correlation

corr= numerical_data.corr()
corr
```

[22]:

	year	month	arr_flights	arr_del15	carrier_ct	weather_ct	nas_ct	security_ct	late_aircraft_ct	arr_cancelled	arr_diverted	arr_c
year	1.000000	-0.238243	0.044512	0.099517	0.084026	0.063121	0.090066	0.029930	0.107271	-0.028726	0.048722	0.09
month	-0.238243	1.000000	0.005065	-0.007816	0.001872	-0.016887	-0.016869	0.000265	-0.005848	-0.043274	-0.007050	-0.01
arr_flights	0.044512	0.005065	1.000000	0.912189	0.902125	0.698042	0.827110	0.495394	0.854367	0.399179	0.644579	0.86
arr_del15	0.099517	-0.007816	0.912189	1.000000	0.951819	0.742411	0.910981	0.540412	0.965145	0.382411	0.716419	0.96

Figure 21: Data understanding 2.

[24]:
```
#correlation heatmap

plt.figure(figsize=(12, 10))
sns.heatmap(numerical_data.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

Figure 22: Code for heatmap of correlation.

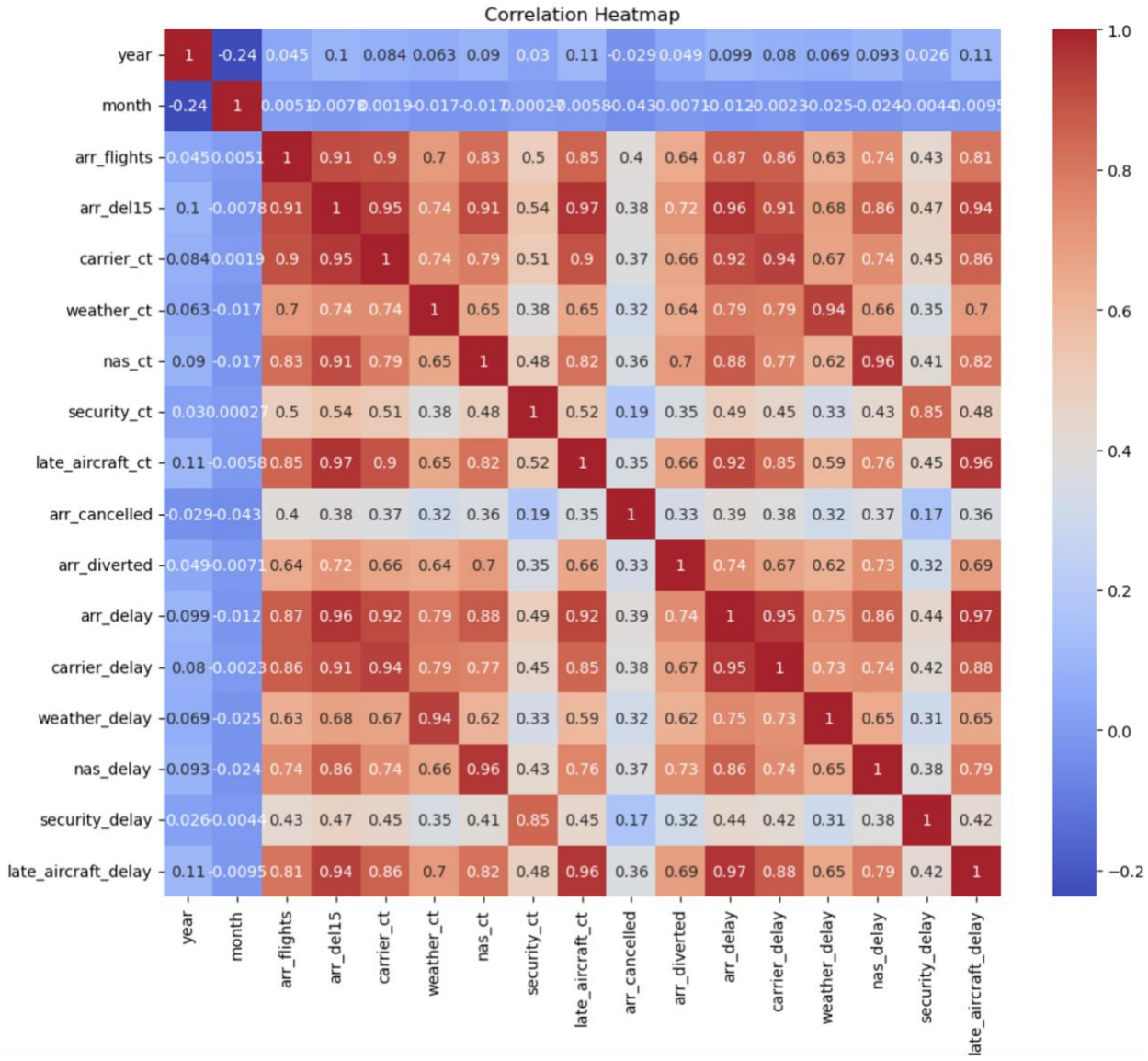


Figure 23: Heatmap for showing correlation.

The correlation heatmap shows, how much the data are dependent with each other or correlated with each other.

If the missing percentage of the null values are below 5 percent, then it is considered good fill the empty values with mean of the column. However, if the missing rate of percentage is greater than 5 then it is good practice to drop the rows.

```
[16]:
missing_data = (df.isnull().sum() / len(df)) * 100
print(missing_data)
#if the missing value percentage is <5% then it is considered as a good practice to fill it with mean, if >5% then drop
```

```
year                  0.000000
month                 0.000000
carrier               0.000000
carrier_name          0.000000
airport               0.000000
airport_name          0.000000
arr_flights           0.241002
arr_del15             0.378004
carrier_ct            0.241002
weather_ct            0.241002
nas_ct                0.241002
security_ct           0.241002
late_aircraft_ct      0.241002
arr_cancelled         0.241002
arr_diverted          0.241002
arr_delay             0.241002
carrier_delay         0.241002
weather_delay         0.241002
```

Figure 24: Null value checking.

```
[17]:
only_numbers = df.select_dtypes(include=['float64', 'int64']).columns


for i in only_numbers:
    if df[i].isnull().sum() > 0:  #This is important because only empty values should be filled
        df[i] = df[i].fillna(df[i].mean())

df.isnull().sum()
```

```
[17]:
year                  0
month                 0
carrier               0
carrier_name          0
airport               0
airport_name          0
arr_flights           0
arr_del15             0
carrier_ct            0
weather_ct            0
nas_ct                0
security_ct           0
late_aircraft_ct      0
arr_cancelled         0
arr_diverted          0
arr_delay             0
carrier_delay         0
weather_delay         0
```

Figure 25: Filling empty data with mean.

Label encoding is one of the most necessary steps if the columns are not numerical. The computer is only able to understand the language of numbers, hence for faster processing the conversion to numerical data. For development of this model, two different encoding technique was used. Label encoding was used as because there were many data to be converted into machine understandable data. However, for certain columns like “carrier” and “carrier_name” which are limited and could be easily converted into binary values like true or false are done using one hot encoding. Since the airport and airport name have larger unique values, using one hot encoding would create hundreds of new binary data, which might also require high computational power.

[18]:
```
# One-hot encoding.
#This is performed because field like carrier and airport are not numberical which is why machine wont understand it.


 #since this has many unique values label encoding is performed also because there is no direct comparison between the data
label_encoding_col= ['airport', 'airport_name']
onehot_encoding_col = ['carrier', 'carrier_name']  #one hot is used to convert columns into binary value like true false


label_encoding = LabelEncoder()
for i in label_encoding_col:
    df[i] = label_encoding.fit_transform(df[i])


df = pd.get_dummies(df, columns=onehot_encoding_col, drop_first=True)


df.head()
```

[18]:

	year	month	airport	airport_name	arr_flights	arr_del15	carrier_ct	weather_ct	nas_ct	security_ct	late_aircraft_ct	arr_cancelled	arr_diverted	arr_delay
0	2024	9	0	10	81.0	7.0	5.61	0.56	0.84	0.0	0.00	0.0	0.0	243.0
1	2024	9	11	9	81.0	6.0	4.43	0.00	1.13	0.0	0.44	0.0	0.0	647.0
2	2024	9	12	21	133.0	12.0	3.49	2.04	3.84	0.0	2.63	10.0	0.0	2382.0
3	2024	9	14	7	73.0	4.0	0.82	1.00	0.23	0.0	1.95	0.0	0.0	160.0
4	2024	9	23	19	2204.0	280.0	61.70	17.36	88.36	0.0	112.57	40.0	4.0	24383.0

Figure 26: Label Encoding.

Separating features for model training are necessary to identify the X and y for prediction. Here features are the determining factor of the result or “arr_delay” in this case which also is y. Correlation is also checked with arr_delay to remove the redundant features and to remove columns not having correlation between the data. Here the features are initially separated to make sure only these columns are standardized but when choosing actual features, simple not including arr_delay and including all

the columns would work. Since every data are not meant to be standardized features are separated here.

```
[19]: # separating features
features = ['arr_flights', 'arr_del15', 'carrier_ct', 'weather_ct',
                       'nas_ct', 'security_ct', 'late_aircraft_ct',
                       'arr_cancelled', 'arr_diverted',
                       'carrier_delay', 'weather_delay', 'nas_delay',
                       'security_delay', 'late_aircraft_delay']

[20]: correlations = df.corrwith(df['arr_delay'])
print(correlations)

year                                  0.098775
month                                -0.011562
airport                              -0.027527
airport_name                         -0.027282
arr_flights                           0.865804
                                        ...
carrier_name_SkyWest Airlines Inc.   -0.019198
carrier_name_Southwest Airlines       0.122552
carrier_name_Spirit Airlines          0.020865
carrier_name_Trans States Airlines   -0.000363
carrier_name_United Air Lines Network 0.042931
Length: 68, dtype: float64
```

Figure 27: Separating features.

After checking the correlation with the target, the columns holding low value should be removed. Some redundant columns also should be dropped in order for obtain efficient result from the model.

```
[21]: #checking correlation after data encoding
correlations = df.corrwith(df['arr_delay'])

#checking if the correlation between data is too low if it is too low then removing it
threshold = 0.1
low_features = correlations[correlations.abs() < threshold].index.tolist()

#since this data is almost correlated with arr_delay it is being removed
redundant_features = ['late_aircraft_delay']

#now dropping all those unncessary column
dropping_features = list(set(low_features + [f for f in redundant_features if f not in low_features]))


df = df.drop(columns=dropping_features)

print("Dropped Features:", dropping_features)

Dropped Features: ['carrier_name_Delta Air Lines Network', 'carrier_name_SkyWest Airlines Inc.', 'carrier_name_Horizon Air', 'carrier_name_PS
A Airlines Inc.', 'carrier_B6', 'carrier_name_Hawaiian Airlines Network', 'late_aircraft_delay', 'carrier_name_Trans States Airlines', 'carri
er_name_United Air Lines Network', 'carrier_UA', 'airport_name', 'carrier_F9', 'carrier_G7', 'carrier_DL', 'carrier_CP', 'carrier_name_Envoy
Air', 'carrier_OH', 'carrier_name_ExpressJet Airlines LLC', 'carrier_name_CommuteAir LLC dba CommuteAir', 'carrier_name_GoJet Airlines LLC d/
b/a United Express', 'carrier_name_Piedmont Airlines', 'carrier_name_JetBlue Airways', 'carrier_AS', 'carrier_name_Compass Airlines', 'carrie
r_name_Frontier Airlines', 'carrier_C5', 'carrier_PT', 'carrier_name_Allegiant Air', 'carrier_HA', 'carrier_NK', 'carrier_QX', 'carrier_name_
Commutair Aka Champlain Enterprises, Inc.', 'carrier_name_Republic Airline', 'carrier_name_Alaska Airlines Network', 'carrier_G4', 'airport',
'carrier_EM', 'carrier_MQ', 'carrier_OO', 'carrier_YX', 'carrier_name_Empire Airlines Inc.', 'carrier_name_Spirit Airlines', 'carrier_name_Me
sa Airlines Inc.', 'carrier_name_Endeavor Air Inc.', 'carrier_EV', 'month', 'carrier_YV', 'carrier_ZW', 'carrier_AX', 'year']

[22]: #removing some more redundant features that has low impact

redundant_features = [
    'late_aircraft_ct',
    'nas_ct',
    'carrier_ct',
    'weather_ct'
]

df = df.drop(columns=redundant_features)

print("Remaining Columns:", df.columns.tolist())

Remaining Columns: ['arr_flights', 'arr_del15', 'security_ct', 'arr_cancelled', 'arr_diverted', 'arr_delay', 'carrier_delay', 'weather_dela
y', 'nas_delay', 'security_delay', 'carrier_AA', 'carrier_WN', 'carrier_name_American Airlines Network', 'carrier_name_Southwest Airlines']
```

Finally, the data in the dataset are standardized in order to provide equal weights, so no biasness is seen during the model training and testing.

```
[24]: #since there are variation in range of numbers
      #for an example te range of late_aircraft_ct maybe between 0-500
      #but arr_delay maybe from 0-120000
      #for this model training the range should be scaled (like assigning equal weights to all columns)


      #this is done so that encoded data also dont get standarized
      features = [i for i in features if i in df.columns]




      scaler = StandardScaler()


      df[features] = scaler.fit_transform(df[features])
```

3.5.3. Preparing the model

Now that the data is ready to be trained, the first step involved is to split the data into train, and test split. Before splitting the data, features (X) and target(y) is also separated.

Test Train Split

```
[26]: from sklearn.model_selection import train_test_split


      X = df.drop(columns=['arr_delay'])
      y = df['arr_delay']


      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


      print("Training Features Shape:", X_train.shape)
      print("Test Features Shape:", X_test.shape)
      print("Training Target Shape:", y_train.shape)
      print("Test Target Shape:", y_test.shape)
      df
```

```
Training Features Shape: (69999, 13)
Test Features Shape: (30000, 13)
Training Target Shape: (69999,)
Test Target Shape: (30000,)
```

[26]:

	arr_flights	arr_del15	security_ct	arr_cancelled	arr_diverted	arr_delay	carrier_delay	weather_delay	nas_delay	security_delay	carrier_AA	carrier_WN
0	-0.248430	-0.286580	-0.211414	-0.153206	-0.215469	243.0	-0.250497	-0.226528	-0.244467	-0.180516	False	Fals
1	-0.248430	-0.292643	-0.211414	-0.153206	-0.215469	647.0	-0.200729	-0.242624	-0.182192	-0.180516	False	Fals
2	-0.187494	-0.256265	-0.211414	0.090271	-0.215469	2382.0	-0.028105	0.394774	-0.190649	-0.180516	False	Fals
3	-0.257805	-0.304769	-0.211414	-0.153206	-0.215469	160.0	-0.286521	-0.226528	-0.251387	-0.180516	False	Fals
4	2.239413	1.368612	-0.211414	0.820702	1.016797	24383.0	1.340191	2.222196	1.163648	-0.180516	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 28: Train test split.

Here the data is split into two different parts, one being test which is used to test the final data after the model is being trained, and the other set of data is used for training the data. In this case the test size is given as 0.3

which means 70 percent of the data is used for training purpose and 30 percent of the data is used for testing purpose.

After the data are split, the model is ready to be trained. In the screenshot below, all five models are stored into a dictionary and are trained using for each loop. Along with the training of data, the performance score of the data are also simultaneously stored.

```
[30]: from sklearn.metrics import mean_absolute_error, mean_squared_error
      from xgboost import XGBRegressor

      models = {
          "Linear Regression": LinearRegression(),
          "Decision Tree": DecisionTreeRegressor(max_depth=3),
          "Random Forest": RandomForestRegressor(n_estimators=5, max_depth=3, random_state=42),
          "K-Nearest Neighbors (KNN)": KNeighborsRegressor(n_neighbors=10),
          "XGBoost": XGBRegressor(max_depth=3, n_estimators=5, learning_rate=0.2, random_state=42)
      }

      model_scores = {}
      performance = {}


      for name, model in models.items():
          print(f"\nTraining {name}...")
          model.fit(X_train, y_train)

          y_pred = model.predict(X_test)  # MakING prediction
          r2 = model.score(X_test, y_test)  # R2 score
          mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error
          mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error


          model_scores[name] = r2
          performance[name] = {"R^2": r2, "MAE": mae, "MSE": mse}

          print(f"{name} R^2 Score: {r2:.4f}")
          print(f"{name} Mean Absolute Error (MAE): {mae:.4f}")
          print(f"{name} Mean Squared Error (MSE): {mse:.4f}")
```

Figure 29: Training models.

```
Training Linear Regression...
Linear Regression R^2 Score: 0.9825
Linear Regression Mean Absolute Error (MAE): 592.8160
Linear Regression Mean Squared Error (MSE): 2720227.7995

Training Decision Tree...
Decision Tree R^2 Score: 0.8756
Decision Tree Mean Absolute Error (MAE): 1406.0859
Decision Tree Mean Squared Error (MSE): 19301405.5925

Training Random Forest...
Random Forest R^2 Score: 0.9295
Random Forest Mean Absolute Error (MAE): 1220.9498
Random Forest Mean Squared Error (MSE): 10935903.6992

Training K-Nearest Neighbors (KNN)...
K-Nearest Neighbors (KNN) R^2 Score: 0.9732
K-Nearest Neighbors (KNN) Mean Absolute Error (MAE): 538.4541
K-Nearest Neighbors (KNN) Mean Squared Error (MSE): 4151608.0381

Training XGBoost...
XGBoost R^2 Score: 0.8270
XGBoost Mean Absolute Error (MAE): 1769.9313
XGBoost Mean Squared Error (MSE): 26837488.1067
```

Figure 30: Performance score.

After the model is trained, the split data for testing purpose is used. Following the testing process, a report is generated accordingly.

```
[31]: from sklearn.metrics import classification_report, accuracy_score

#defining delay threshold
delay_threshold = 15

classification_reports = {}

for name, model in models.items():
    print(f"\nGenerating report for {name}...")

    # Predicting
    y_pred = model.predict(X_test)


    y_testing = ["Delayed" if y > delay_threshold else "On-Time" for y in y_test]
    y_prediction = ["Delayed" if y > delay_threshold else "On-Time" for y in y_pred]

    # report generation
    report = classification_report(
        y_testing,
        y_prediction,
        target_names=["On-Time", "Delayed"],
        zero_division=0  # removing warnings
    )
    accuracy = accuracy_score(y_testing, y_prediction)


    print(f"Classification Report {name}:")
    print(report)
    print(f"Accuracy: {accuracy:.4f}")

    # Storing for further use
    classification_reports[name] = {
        "report": report,
        "accuracy": accuracy
    }
```

Figure 31: Predicting data.

```
Generating report for Linear Regression...
Classification Report Linear Regression:
              precision    recall  f1-score   support

     On-Time       1.00      0.83      0.90     28238
     Delayed       0.26      0.96      0.40      1762

    accuracy                           0.83     30000
   macro avg       0.63      0.89      0.65     30000
weighted avg       0.95      0.83      0.87     30000

Accuracy: 0.8335

Generating report for Decision Tree...
Classification Report Decision Tree:
              precision    recall  f1-score   support

     On-Time       0.94      1.00      0.97     28238
     Delayed       0.00      0.00      0.00      1762

    accuracy                           0.94     30000
   macro avg       0.47      0.50      0.48     30000
weighted avg       0.89      0.94      0.91     30000

Accuracy: 0.9413

Generating report for Random Forest...
Classification Report Random Forest:
              precision    recall  f1-score   support

     On-Time       0.94      1.00      0.97     28238
     Delayed       0.00      0.00      0.00      1762

    accuracy                           0.94     30000
   macro avg       0.47      0.50      0.48     30000
weighted avg       0.89      0.94      0.91     30000

Accuracy: 0.9413

Generating report for K-Nearest Neighbors (KNN)...
Classification Report K-Nearest Neighbors (KNN):
              precision    recall  f1-score   support

     On-Time       0.99      1.00      1.00     28238
     Delayed       1.00      0.92      0.96      1762

    accuracy                           0.99     30000
   macro avg       1.00      0.96      0.98     30000
weighted avg       0.99      0.99      0.99     30000

Accuracy: 0.9950

Generating report for XGBoost...
Classification Report XGBoost:
              precision    recall  f1-score   support

     On-Time       0.94      1.00      0.97     28238
     Delayed       0.00      0.00      0.00      1762

    accuracy                           0.94     30000
   macro avg       0.47      0.50      0.48     30000
weighted avg       0.89      0.94      0.91     30000

Accuracy: 0.9413
```

Figure 32: Prediction report.

The best model is also selected to know which model performed the best based on this current scenario.

```
[35]:
best_model = max(model_scores, key=model_scores.get)
best_score = model_scores[best_model]

print(f"The best model is {best_model} and its r2 score is {best_score:.4f}")
The best model is Linear Regression and its r2 score is 0.9825
```

Figure 33: Best model.

The models used here did not go under hyperparameter tuning process, as the parameters selected after hyper-parameter tuning lead to the overfitting of the data. The hyper-parameter tuning is not the cause for overfitting however, sometimes the parameters chosen might cause overfitting.

3.5.4. Visualization of data

Before proceeding further with the development of model, the output, and performance score should be clarified. The visualization is important as it is easy for anyone to understand data on the basis of image rather that text. It is also easy to spot the difference between the comparison in various scenarios.

After the model has been trained and tested, its performance score was visualized. The functions errors like R squared, MAE, and MSE were calculated.

[32]:
```
#COMPARISONS OF DATA

# After model_performance dictionary is populated
models_list = list(models.keys())
r2_scores = [metrics["R^2"] for metrics in performance.values()]
mae_scores = [metrics["MAE"] for metrics in performance.values()]
mse_scores = [metrics["MSE"] for metrics in performance.values()]

# Bar chart for R^2
plt.figure(figsize=(10, 6))
plt.bar(models_list, r2_scores, alpha=0.7, color='blue')
plt.title("Compare R2")
plt.ylabel("R2 Score")
plt.xlabel("Models")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Bar chart for MAE
plt.figure(figsize=(10, 6))
plt.bar(models_list, mae_scores, alpha=0.7, color='orange')
plt.title("Compare MAE")
plt.ylabel("MAE")
plt.xlabel("Models")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Bar chart for MSE
plt.figure(figsize=(10, 6))
plt.bar(models_list, mse_scores, alpha=0.7, color='green')
plt.title("Compare MSE")
plt.ylabel("MSE")
plt.xlabel("Models")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 34: Code for plotting functions error.

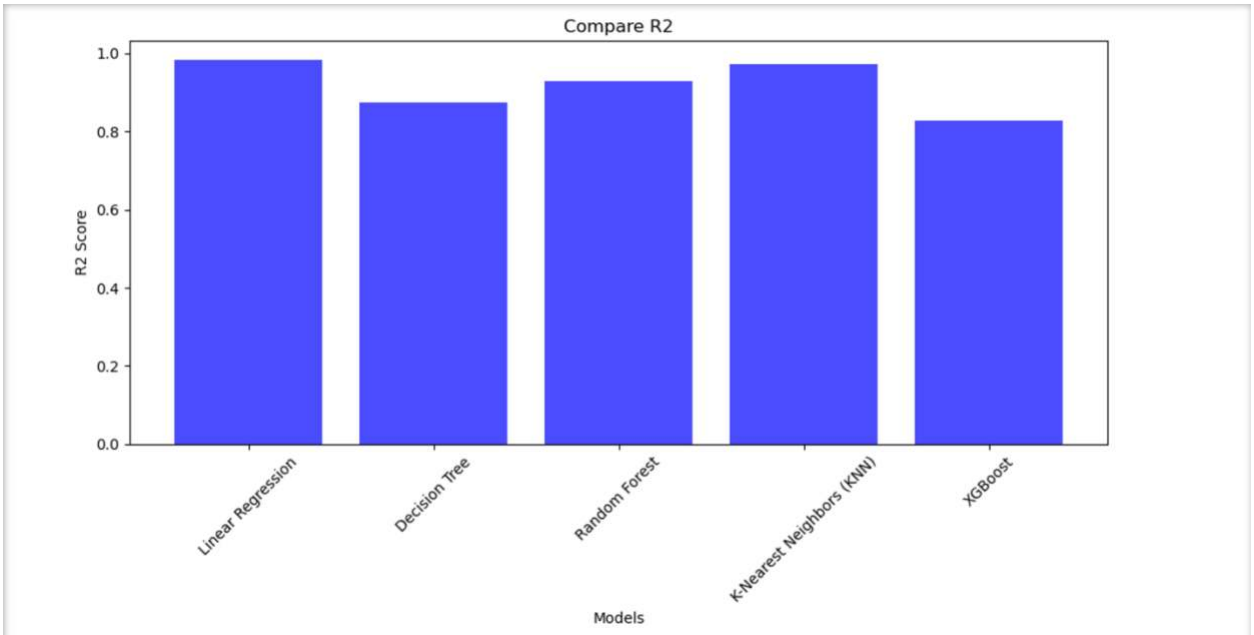


Figure 35: R2 comparison.

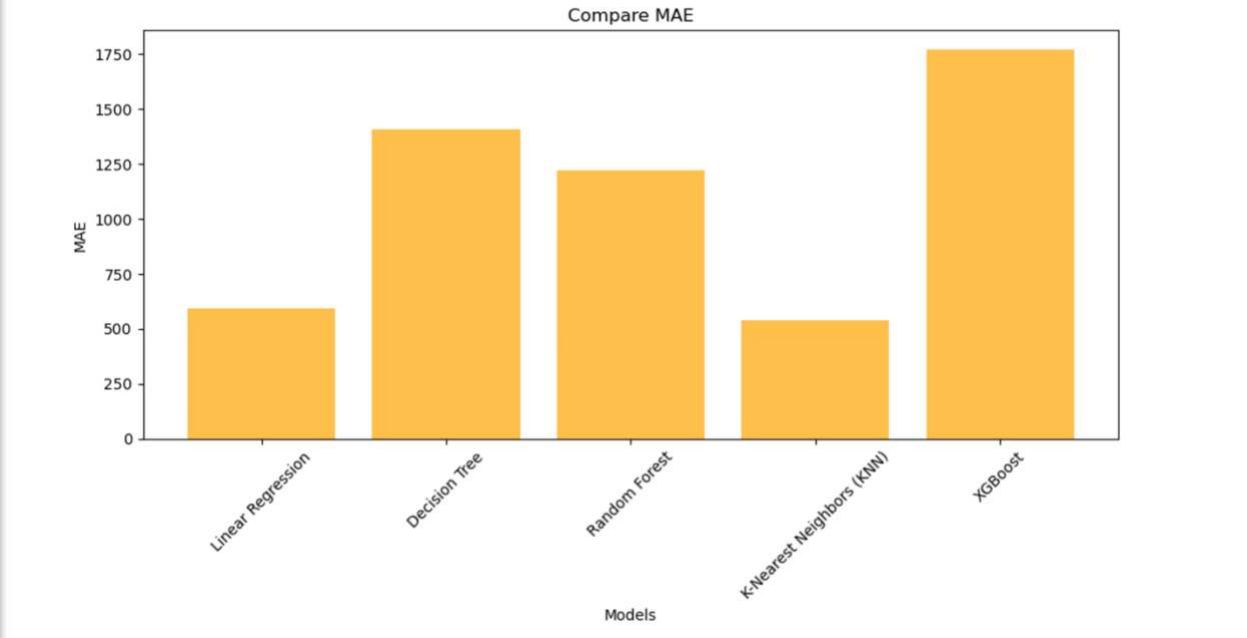


Figure 36: MAE Comparison

Here the KNN and linear regression has the lowest MAE, which means the average prediction error are the smallest in those models.

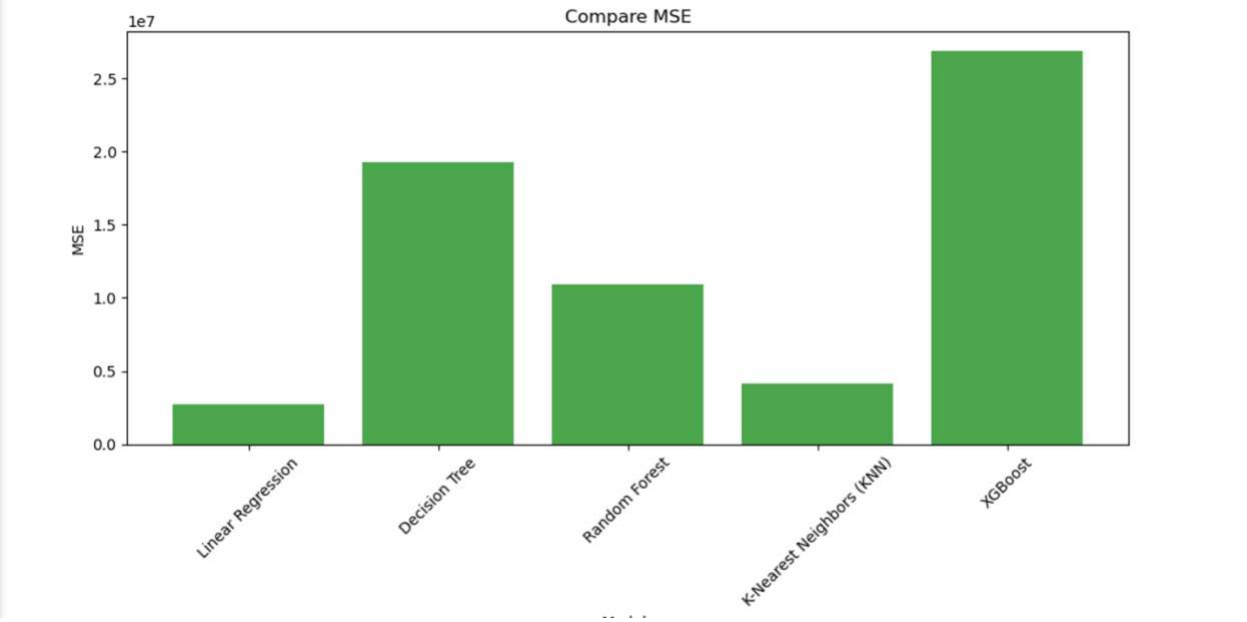


Figure 37: MSE Comparison

Similar to MAE, in MSE linear regression and KNN have the lowest MSE resulting in few larger errors in the output.

After visualizing the functions errors, the feature importance is tested to have a clear understanding what features were responsible to changing the outcome.

```
[34]: #visualizing the importance


for name, model in models.items():
    if hasattr(model, "feature_importances_"):
        important_feature = model.feature_importances_
        sortedd = importance.argsort()[::-1]

        plt.figure(figsize=(10, 6))
        plt.barh(
            [X_train.columns[i] for i in sortedd[:10]],
            important_feature[sortedd[:10]],
            color="blue"
        )
        plt.title(f"Top 10 Feature Importances ({name})")
        plt.xlabel("Importance")
        plt.ylabel("Features")
        plt.tight_layout()
        plt.show()
```

Figure 38: Code for visualizing feature importance.

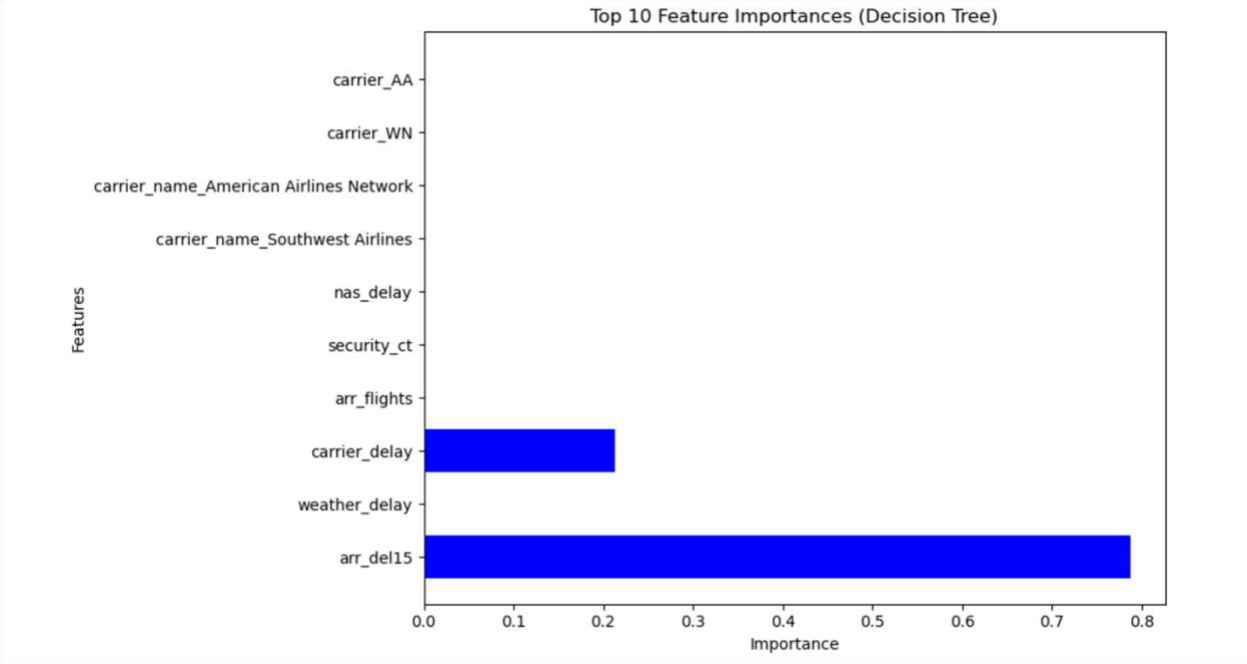


Figure 39: Feature importance for decision tree.

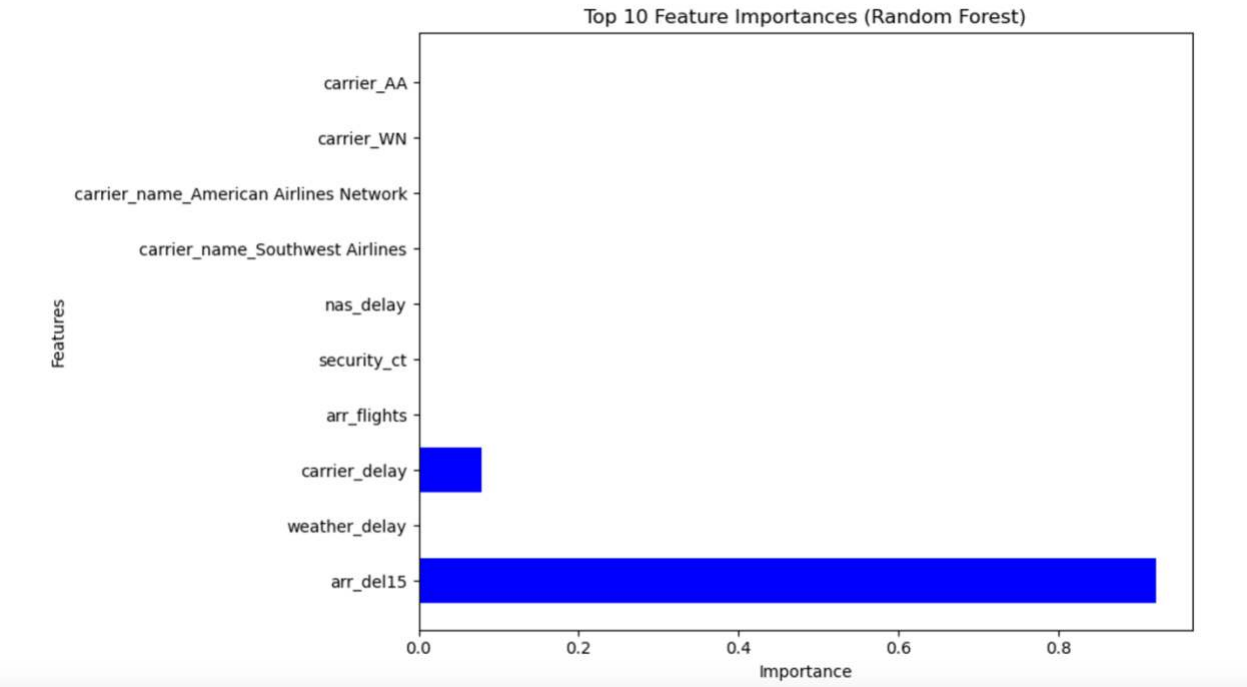


Figure 40: Feature importance for random forest.

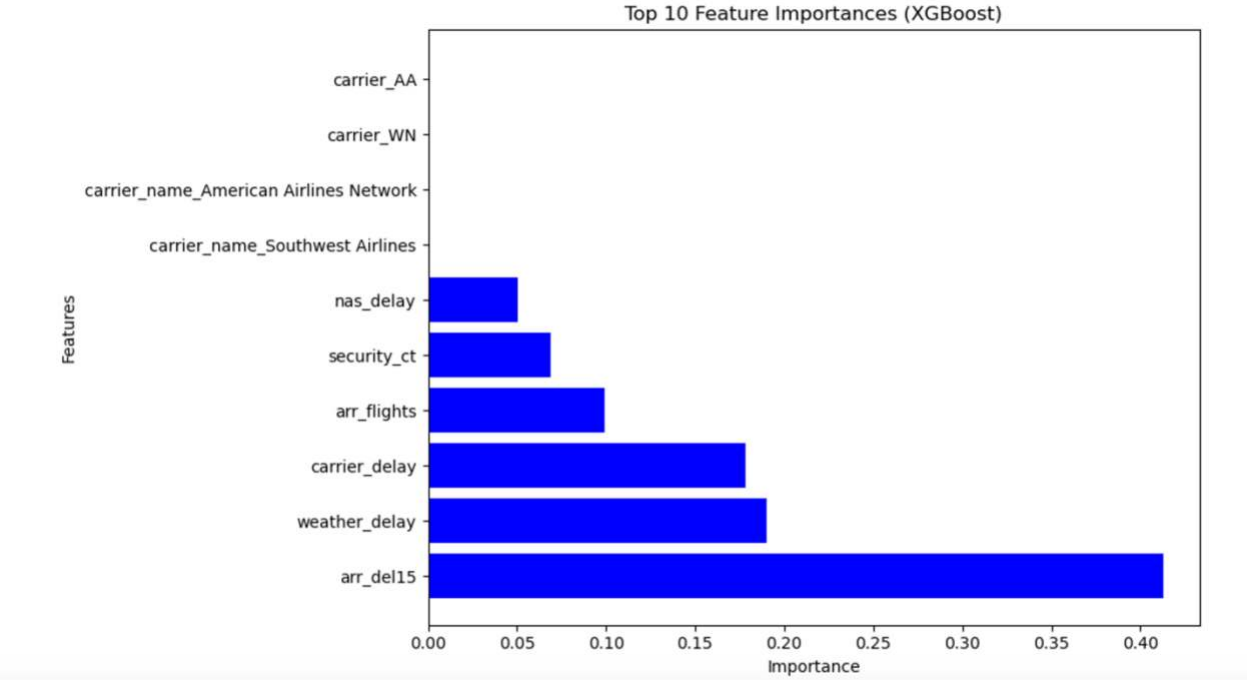


Figure 41: Feature importance for XGBoost.

By analysing the feature importance graph, the most common feature that determines the out is arr_del15, and other features are varying with the models.

3.5.5. Model Evaluation

This section will cover the evaluation of model after being trained and test. Here several additional visualization and other comparison will also be performed.

The comparison between the R squared score is visualised for different models.

```
[38]: import matplotlib.pyplot as plt

#for storing r2 scores
scores = {}

for name, model in models.items():

    score = model.score(X_test, y_test)
    scores[name] = score

# Print R-squared scores
print("\nModel R^2 Scores:")
for model, score in model_scores.items():
    print(f"{model}: {score:.4f}")

# Visualize the results
plt.figure(figsize=(8, 5))
plt.bar(scores.keys(), model_scores.values(), alpha=0.7, color='blue')
plt.ylabel("R^2 Score")
plt.title("Model Comparison")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 42: R square visualization code.

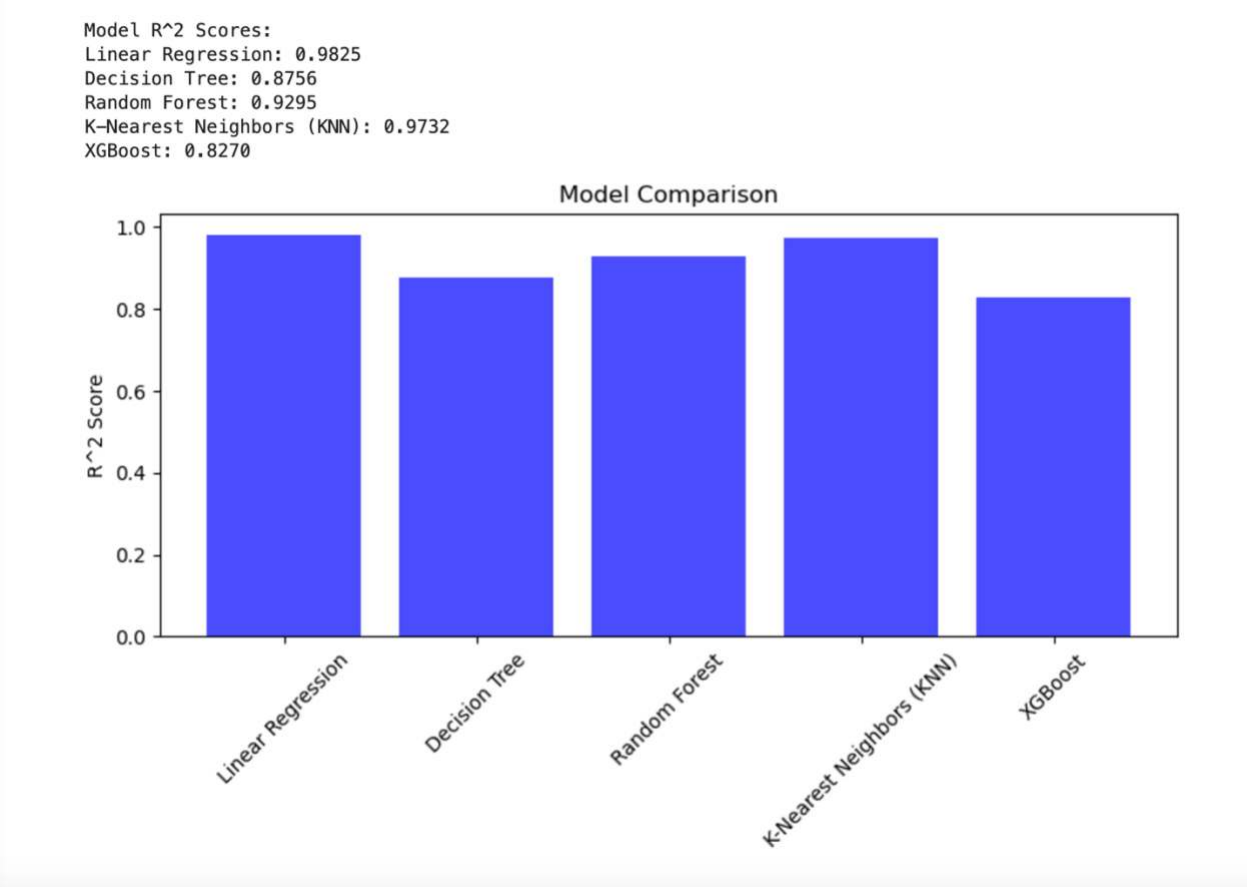


Figure 43: R Squared visualization graph

After the visualisation of R square score, actual vs predicted data will be visualized. For predicting the actual vs predicted data, scatter plot is being used. With the visualization of R squared score it can be seen that the linear regression has the highest score and XGBoost having the lowest score.

```
[40]: from sklearn.metrics import mean_absolute_error, mean_squared_error

# for storing performance
performance = {}

# Evaluating every models
for name, model in models.items():

    y_pred = model.predict(X_test)

    # Calculate scores
    r2 = model.score(X_test, y_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    # storing in previously stored dictionary
    performance[name] = {"R^2": r2, "MAE": mae, "MSE": mse}


print("\nModel Evaluation Results:")
for model, metrics in performance.items():
    print(f"\n{model} Performance:")
    print(f"R2: {metrics['R^2']:.4f}")
    print(f"MAE: {metrics['MAE']:.4f}")
    print(f"MSE: {metrics['MSE']:.4f}")

# plotting models
for name, model in models.items():
    y_pred = model.predict(X_test)

    plt.figure(figsize=(8, 6))
    plt.scatter(y_test, y_pred, alpha=0.6, label='Predicted vs Actual', color='blue')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2, label='Best fit')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(f'Actual vs Predicted for {name}')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Figure 44: Code for actual vs predicted data plotting.

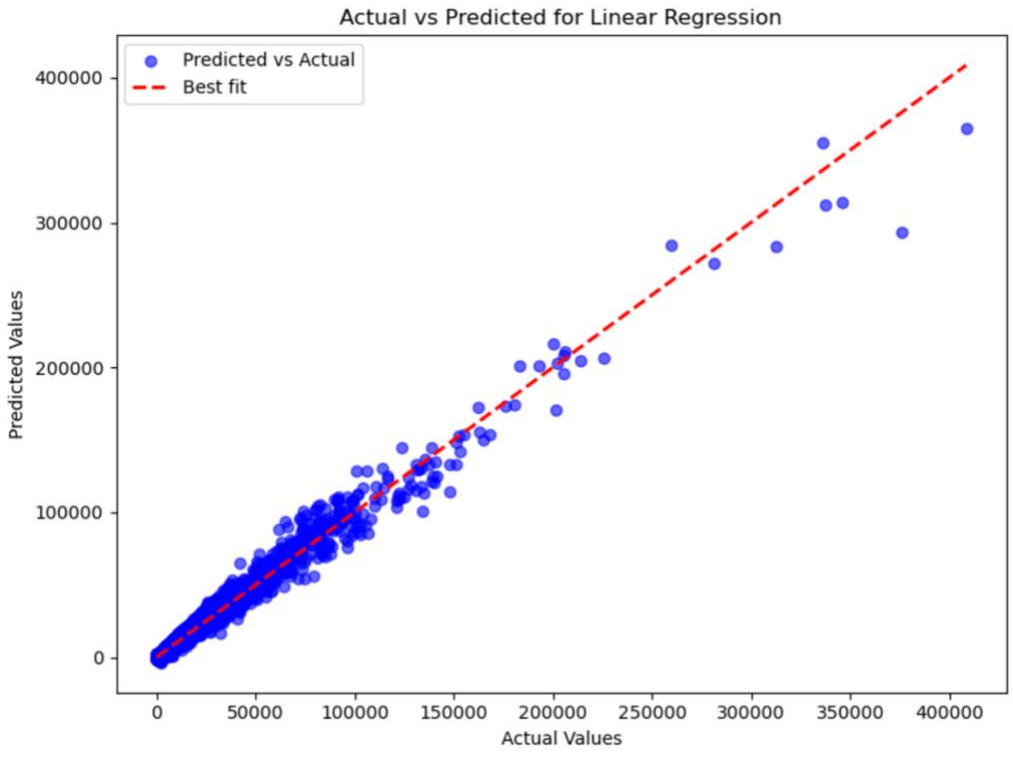


Figure 45: Actual vs Predicted data for linear regression.

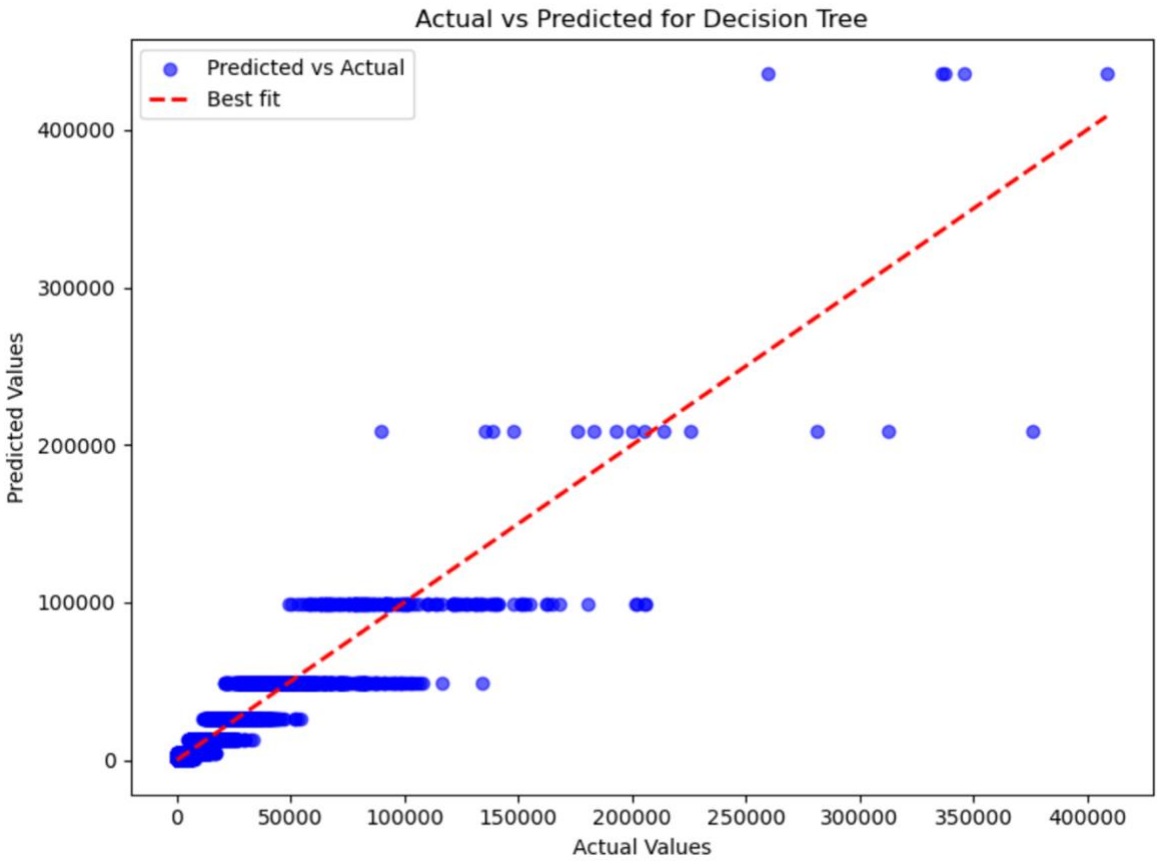


Figure 46: Actual vs Predicted data for decision tree.

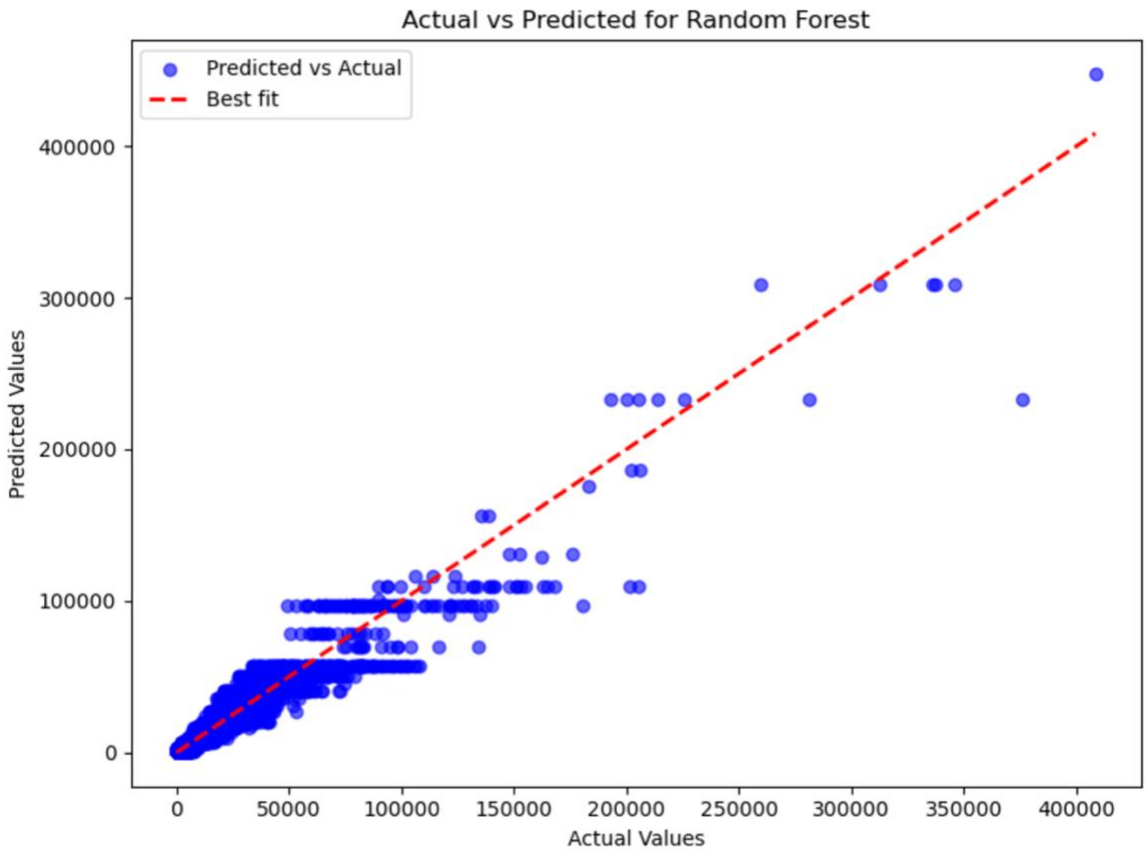


Figure 47: Actual vs Predicted data for random forest.

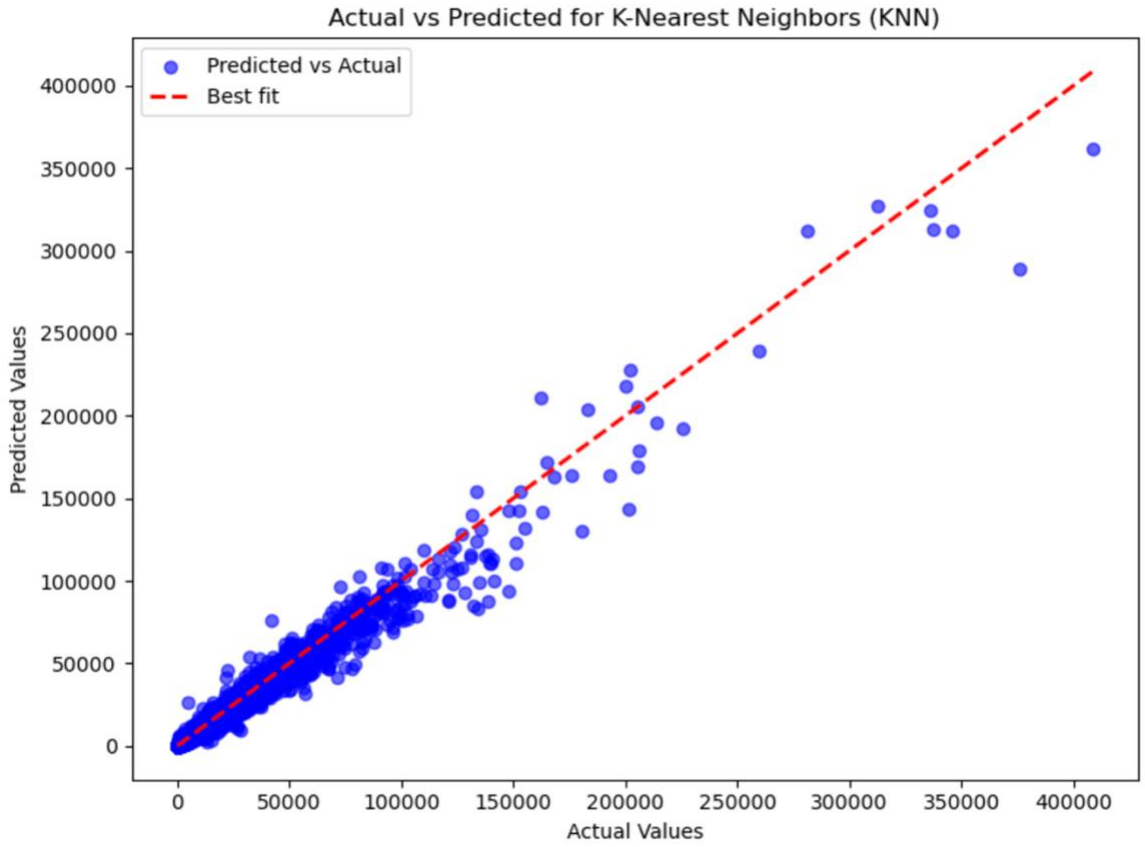


Figure 48: Actual vs Predicted data for KNN

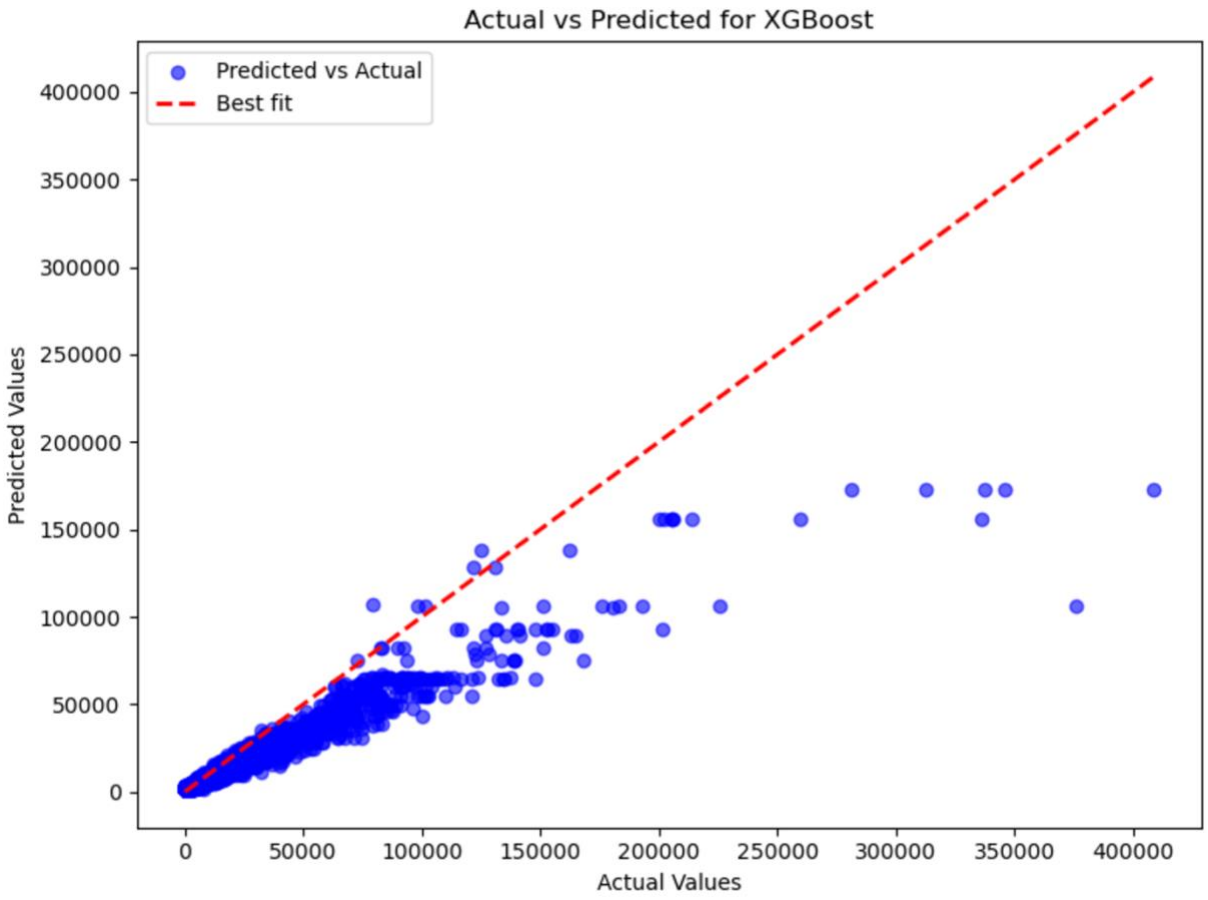


Figure 49: Actual vs Predicted data for XGBoost

This scatter plot represents the predicted delay time and the actual delay time. The graph also shows the best fit line. The actual values are on X-axis and the predicted values are on Y-axis. By analysing the graphs, it can be seen that the best performing models are linear regression or KNN as the points are closest to the best fit line. Here the worst performing model can be seen as decision tree model due to highly scattered data.

After the models are evaluated, the final step performed was to ensemble to model by assigning the weights according to their performance score, to act as a one model for getting more accurate and precise output.

Emsemble

```
[43]: import numpy as np
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# Get predictions for all models
predictions = []
for name, model in models.items():
    predictions.append(model.predict(X_test)) #adding for reference

# Assigning weights to the algorithms
weights = {
    "XGBoost": 0.2,
    "Linear Regression": 0.1,
    "Decision Tree": 0.1,
    "Random Forest": 0.5,
    "K-Nearest Neighbors (KNN)": 0.1
}


ensemble_predictions = np.zeros(len(X_test))
for name, model in models.items():
    ensemble_predictions += weights[name] * model.predict(X_test)

# Evaluate the ensemble
ensemble_r2 = r2_score(y_test, ensemble_predictions)
ensemble_mae = mean_absolute_error(y_test, ensemble_predictions)
ensemble_mse = mean_squared_error(y_test, ensemble_predictions)

print("Random Forest-Based Ensemble Performance:")
print(f"R^2: {ensemble_r2:.4f}")
print(f"MAE: {ensemble_mae:.4f}")
print(f"MSE: {ensemble_mse:.4f}")
```

```
Random Forest-Based Ensemble Performance:
R^2: 0.9475
MAE: 998.3896
MSE: 8138576.3756
```

Figure 50: Ensembling model.

After the model is ensembled, the model was exported for further use.

```
[44]: #saving model for future reference

import joblib


for name, model in models.items():
    joblib.dump(model, f"{name}_model.pkl")

# Saving the ensemble function
def combined_model(X):
    ensemble_predictions = np.zeros(len(X))
    for name, model in models.items():
        ensemble_predictions += weights[name] * joblib.load(f"{name}_model.pkl").predict(X)
    return ensemble_predictions

joblib.dump(combined_model, "combined_model.pkl")
```

```
[44]: ['combined_model.pkl']
```

Figure 51: Exporting model.

The ensembled model was put into test with a new unseen data. Since it is known that arr_del15 has the highest impact on the output the results obtain by tweaking the output is changed drastically.

[45]:
```
# testing data
data = pd.DataFrame({
    "arr_flights": [1000],
    "arr_del15": [0.5],
    "security_ct": [0],
    "arr_cancelled": [0],
    "arr_diverted": [0],
    "carrier_delay": [1],
    "weather_delay": [1],
    "nas_delay": [1],
    "security_delay": [0],
    "carrier_AA": [1],
    "carrier_WN": [0],
    "carrier_name_American Airlines Network": [0],
    "carrier_name_Southwest Airlines": [1]
})

# to make sure the data are consistent
#this is performed to scale and match with the data that were in preprocessing
data[features] = scaler.transform(data[features])
```

[46]:
```
import joblib

prediction = np.zeros(len(data))
for name, model in models.items():
    exported_combined_model = joblib.load(f"{name}_model.pkl")
    prediction += weights[name] * exported_combined_model.predict(data)

print(f"Delay Prediction: {prediction}")
```

```
Delay Prediction: [697.76011353]
```

Figure 52: Testing model with new data 1.

[89]:
```
# testing data
data = pd.DataFrame({
    "arr_flights": [1000],
    "arr_del15": [0.7],
    "security_ct": [0],
    "arr_cancelled": [0],
    "arr_diverted": [0],
    "carrier_delay": [1],
    "weather_delay": [1],
    "nas_delay": [1],
    "security_delay": [0],
    "carrier_AA": [1],
    "carrier_WN": [0],
    "carrier_name_American Airlines Network": [0],
    "carrier_name_Southwest Airlines": [1]
})

# to make sure the data are consistent
#this is performed to scale and match with the data that were in preprocessing
data[features] = scaler.transform(data[features])
```

[91]:
```
import joblib

prediction = np.zeros(len(data))
for name, model in models.items():
    exported_combined_model = joblib.load(f"{name}_model.pkl")
    prediction += weights[name] * exported_combined_model.predict(data)

print(f"Delay Prediction: {prediction}")
```

```
Delay Prediction: [705.15638544]
```

Figure 53: Testing model with new data 2.

Here values in arr_del15 means the flight is delayed 15 or more minutes 70 percent of the total time. The output 705 means the flight is going to be delayed by 705 minutes.

4. Conclusion

This project aims to deliver a proper working model, which can be used by aviation system for predicting flight delays. With a proper analysis of past record of flights, and proper implementation of various algorithms making sure the model stays lightweight but accurate this model will be prepared. In this proposal a model along with the flow of development process has been briefly designed.

For future work, the model can be redefined with further hyper parameters tuning, and addition of more easily available features. The model can be made more advance, with more efficient algorithms over time. The model can be scaled more to handle huge data keeping the model simple that requires low computational power.

5. Works Cited

Stryker, C., 2024. *What is Artificial Intelligence (AI) | IBM.* [Online] Available at: https://www.ibm.com/think/topics/artificial-intelligence [Accessed 20 December 2024].

Google , 2024. *What is Machine Learning? Types & Uses | Google Cloud.* [Online] Available at: https://cloud.google.com/learn/what-is-machine-learning [Accessed 20 December 2024].

Efthymiou, M. et al., 2024. The Impact of Delays on Customers' Satisfaction: An Empirical Analysis of the British Airways On-Time Performance at Heathrow Airport - University of Huddersfield Research Portal. *Journal of Aerospace Technology and Management,* Volume XI, pp. 1-13.

Mehra, S., 2023. *A new deep dive into passenger frustration with airlines - TNMT.* [Online] Available at: https://tnmt.com/passenger-frustration-with-airlines/?utm_source=chatgpt.com [Accessed 20 December 2024].

Qu , J., Xiao , M., Yang , L. & Xie , W., 2023. Flight Delay Regression Prediction Model Based on Att-Conv-LSTM.

Duong, V. N., Alam , S. & Cai , Q., 2024. A Spatial–Temporal Network Perspective for the Propagation Dynamics of Air Traffic Delays. *Research Novel Methodologies in Air Transportation,* 7(4), pp. 452-464.

IBM, 2024. *What is linear regression? | IBM.* [Online] Available at: https://www.ibm.com/think/topics/linear-regression [Accessed 21 December 2024].

Bacallado, S. & Taylor, J., 2022. *Simple Linear Regression -- Stats 202.* [Online] Available at: https://web.stanford.edu/class/stats202/notes/Linear-regression/Simple-linear-regression.html [Accessed 21 December 2024].

Scikit Learn, 2024. *Decision Tree Regression --sckit-learn 1.5.2 documentation.* [Online] Available at: https://scikit-learn.org/1.5/auto_examples/tree/plot_tree_regression.html [Accessed 21 December 2024].

Scikit Learn, 2024. *Random Forest Regressor --skicit-learn 1.5.2 documentation.* [Online] Available at: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestRegressor.html [Accessed 21 December 2024].

BIO Statistics Collaboration of Australia, 2024. *2 K-nearest Neighbours Regression | Machine Learning for BioStatistics.* [Online] Available at: https://bookdown.org/tpinto_home/Regression-and-Classification/k-nearest-neighbours-regression.html#knn.intro [Accessed 21 December 2024].

NVIDIA Corporation, 2024. *XGBoost - What is it and why does it matter?.* [Online] Available at: https://www.nvidia.com/en-us/glossary/xgboost/ [Accessed 21 December 2024].

H, R. S., 2024. *K-Nearest Neighbors Algorithm - Intuitive Tutorials.* [Online] Available at: https://intuitivetutorial.com/2023/04/07/k-nearest-neighbors-algorithm/ [Accessed 21 December 2024].

Chaya, 2020. *Random Forest Regression. Random Forest Regression is a… | by Chaya | Level Up Coding.* [Online] Available at: https://levelup.gitconnected.com/random-forest-regression-209c0f354c84?gi=049745d9db1b [Accessed 21 December 2024].

Salesforce, Inc., 2024. *What is history of Artificial Intelligence (AI)? | Tableu.* [Online] Available at: https://www.tableau.com/data-insights/ai/history#:~:text=1952%3A%20A%20computer%20scientist%20named,it%20came%20into%20popular%20usage. [Accessed 22 December 2024].

AIPRM, 2024. *AI Statistics 2024 · AIPRM.* [Online] Available at: https://www.aiprm.com/ai-statistics/ [Accessed 22 December 2024].