Phase #1

Title: "Data-Driven Flight Delay Forecasting to Improve Airlines Operations "

Class : CSE 587B Data Intensive Computing

Team Members:
1.) Deekshith Sagar Rangapuram
2.) Indushree Byrareddy
3.) Akhil Chaitanya Ghanta

➢ Problem Statement:

Flight delays are a major operational challenge for airlines all over the world, which affects customer satisfaction, raises operating costs, and disrupts the aviation sector as a whole. In addition to interfering with passengers' travel plans and causing significant financial losses for airlines, these delays also have wider economic and environmental repercussions. The goal of this project is to create predictive models that can overcome these difficulties by precisely predicting crucial aspects of airline operations, like flight delays.

➢ Objective:

- ❖ To develop a machine learning model that can accurately predict whether the flight is delayed or not & by how much time.
- ❖ To create a user interface application that allows users to input journey details and receive a prediction about flight status.
- ❖ To evaluate the performance of the machine learning model using a variety of metrics, including accuracy and score.
- ❖ To assess the usability and user experience of the user interface application, including its ease of use and the clarity of the prediction results.
- ❖ To optimize the machine learning model for robustness and generalizability, using techniques such as Regularization, Feature Engineering and hyperparameter tuning.

➢ Background:

The airline industry is a vital part of global transportation because it links people and organizations all over the world. Flight cancellations and delays, which mess with schedules and bother passengers, are just two of the ongoing issues it faces. These

problems frequently are the result of unanticipated occurrences like bad weather, defective equipment, or staffing issues.

The problem of predicting airline flight delays using machine learning models is significant due to several reasons:

Operational Efficacy: Flight delays raise the airlines' operating expenses. Staff, aircraft, and fuel allocation can all be negatively impacted by delays.

Passengers Satisfaction: Delays in flight can cause annoyance, anxiety, and missed connections for passengers. By providing customers with advance notice, airlines might potentially alleviate passenger annoyance and empower them to make well-informed travel decisions.

Competitive Advantage: Airline companies that possess the ability to precisely forecast and alleviate delays stand to benefit from a competitive edge. Airlines that offer dependable and punctual services have a higher chance of being chosen by passengers, potentially resulting in a larger market share.

Our project aim is to develop an accurate predictive model for airlines delay prediction using machine learning models, holding significant potential to make a substantial contribution.

- ❖ Increased cost savings and operational effectiveness for airlines.
- ❖ Improved experience and satisfaction for passengers.
- ❖ Compliance with regulations and penalties avoidance.
- ❖ Gaining a competitive edge by being dependable and on time.

➢ Data Source:

The Bureau of Transportation Statistics has a reliable source of information about the aviation sector in the United States. By utilizing the vast amount of data provided by the Bureau of Transportation Statistics, we can gain insightful knowledge for anticipating flight delays. The dataset is updated periodically and is up-to-date. We have selected the data time frame from January 2023.

Dataset:
"https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr"

➢ Data Cleaning/Processing

Step: 1 Data Inspection

Helps us to know the structure of the data and data types of the variables. Also we get some idea about the central tendency and variance values that the Dataset columns have.

```
df.describe()
```

	DAY_OF_MONTH	DAY_OF_WEEK	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	DE
count	59610.000000	59610.000000	59610.000000	59610.000000	59610.000000	58787.
mean	16.719393	3.822798	2742.099094	12613.149656	12613.958497	1335.
std	8.680311	2.064362	1928.899587	1239.224912	1239.327762	495.
min	1.000000	1.000000	1.000000	10257.000000	10257.000000	1.
25%	10.000000	2.000000	1013.000000	11697.000000	11697.000000	926.
50%	17.000000	4.000000	2264.000000	12478.000000	12478.000000	1336.
75%	24.000000	6.000000	4717.000000	13198.000000	13198.000000	1732.
max	31.000000	7.000000	9887.000000	15919.000000	15919.000000	2400.

8 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59610 entries, 0 to 59609
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   DAY_OF_MONTH          59610 non-null  int64
 1   DAY_OF_WEEK           59610 non-null  int64
 2   OP_UNIQUE_CARRIER     59610 non-null  object
 3   OP_CARRIER_FL_NUM     59610 non-null  int64
 4   ORIGIN_AIRPORT_ID     59610 non-null  int64
 5   ORIGIN                59610 non-null  object
 6   DEST_AIRPORT_ID       59610 non-null  int64
 7   DEST                  59610 non-null  object
 8   DEP_TIME              58787 non-null  float64
 9   DEP_DELAY             58786 non-null  float64
 10  WHEELS_OFF            58746 non-null  float64
 11  WHEELS_ON             58730 non-null  float64
 12  ARR_TIME              58730 non-null  float64
 13  ARR_DELAY             58606 non-null  float64
 14  CANCELLED             59610 non-null  float64
 15  DIVERTED              59610 non-null  float64
 16  ACTUAL_ELAPSED_TIME   58606 non-null  float64
 17  AIR_TIME              58606 non-null  float64
 18  FLIGHTS               59610 non-null  float64
 19  DISTANCE              59610 non-null  float64
 20  CARRIER_DELAY         13615 non-null  float64
 21  WEATHER_DELAY         13615 non-null  float64
 22  SECURITY_DELAY        13615 non-null  float64
 23  LATE_AIRCRAFT_DELAY   13615 non-null  float64
dtypes: float64(16), int64(5), object(3)
memory usage: 10.9+ MB
```

Step: 2 Dropping duplicate values

Removing the duplicate rows in the dataframe

```
#Dropping the duplicates From Dataset
df.drop_duplicates(inplace=True)
```

Step: 3 Renaming column names

Renaming column names for clarity, readability & to be in compliance with coding standards

```
#To Rename the columns of the dataset
df.rename(columns={'OP_UNIQUE_CARRIER': 'AIRLINE', 'OP_CARRIER_FL_NUM': 'FLIGHT_NUMBER','CRS_DEP_TIME':'SCHD_DEP_TIME','CRS_ARR_TIME':'SCHD_ARR_TIME','CRS_ELAPSED_TIME':'SCHD_ELAPSED_TIME'}, inplace=True)
```

Step: 4 Handling missing values

Identifying the missing values and filling the data with appropriate values. As our dataset has all columns with time data we have only few columns with null values. Those columns include Delay Reasons with number of minutes so, If the values are not available we can assume there is no reason for the delay and we can replace the empty value with the zero minutes.

```
#To fill the Not available values of minutes with zero
df[['CARRIER_DELAY','WEATHER_DELAY','SECURITY_DELAY','LATE_AIRCRAFT_DELAY']]=df[['CARRIER_DELAY','WEATHER_DELAY','SECURITY_DELAY','LATE_AIRCRAFT_DELAY']].fillna(0)
```

Step: 5 Dropping rows with null values

After filling the above empty rows now we are again going to check for the null values. This time we will only have the null values in the places where the flight is canceled or there is no Arrival time or Departure time.Dropping the rows which do not have the Departure and Arrival Time , which are useless for our data.For other rows we can ignore the rows which are not having the data because we have sufficient data to train the model.

```
#Dropping the rows which does not have the Departure and Arrival Time
df.dropna(axis=0,inplace=True)
```

Step: 6 Dropping columns

Now the two columns in the dataset are redundant and we have origin airport name and destination airport name so we can ignore the ID's. Dropping the Two ID columns as they are not useful for our model. "ORIGIN_AIRPORT_ID" & "DEST_AIRPORT_ID" are just the other form of airport name

```
#Dropping the Two ID columns as they are not useful for our model
df.drop(['ORIGIN_AIRPORT_ID', 'DEST_AIRPORT_ID'], axis=1, inplace=True)
```

Step: 7 Converting all values in a column to uppercase

When we are skimming through the dataset we observe the 'Origin' Column has format discrepancies.Checking the Unique values of 'ORIGIN' column and changing the string to uppercase format.

```
#Checking the Unique values and changing the names to one format
print(df['ORIGIN'].unique())
print('-------------------------------------------------After Changing----------------------------------')
df['ORIGIN']=df['ORIGIN'].str.upper()
print(df['ORIGIN'].unique())
```

```
['LGA' 'GSP' 'JFK' 'AVL' 'ALB' 'DTW' 'RDU' 'Jfk' 'BNA' 'ROC' 'CVG' 'Bna'
 'MEM' 'ORF' 'ATL' 'HPN' 'BUF' 'PVD' 'Ith' 'MSP' 'CLE' 'SYR' 'ind' 'ILM'
 'PWM' 'CHS' 'MKE' 'OMA' 'SAV' 'BTV' 'ORD' 'CLT' 'BWI' 'STL' 'BGR' 'LAX'
 'DFW' 'AUS' 'MIA' 'DCA' 'PHX' '1GA' 'SFO' 'clt' 'EGE' 'jfk' 'sna' 'STT'
 'SEA' 'PDX' 'FLL' 'BOS' 'SJU' 'PBI' 'MCO' 'ORH' 'RSW' 'SMF' 'SRQ' 'JAX'
 'SAT' 'IAH' 'MSY' 'PSP' 'DEN' 'MCI' 'TPA' 'LAS' 'ONT' 'BUR' 'BQN' 'SLC'
 'PSE' 'RNO' 'BZN' 'HNL' 'SAN' 'ISP' 'SWF' 'SFB' 'PBG' 'PIE' 'PGD' 'MYR'
 'PHL' 'ELM' 'EWR' 'DAL' 'MDW' 'HOU' 'TUL' 'PIT' 'GSO' 'CMH' 'SDF' 'LIT'
 'OKC' 'EYW' 'CHO' 'XNA' 'IND' 'RIC' 'MSN' 'IAD' 'BDL' 'ITH' 'BGM' 'SNA'
 'IAG' 'DAY' 'ROA' 'MTJ' 'JAC' 'TYS' 'GRR' 'BHM' 'CAE' 'DSM' 'SCE' 'LEX']
----------------------------------------------------After Changing-----------------------------------
['LGA' 'GSP' 'JFK' 'AVL' 'ALB' 'DTW' 'RDU' 'BNA' 'ROC' 'CVG' 'MEM' 'ORF'
 'ATL' 'HPN' 'BUF' 'PVD' 'ITH' 'MSP' 'CLE' 'SYR' 'IND' 'ILM' 'PWM' 'CHS'
 'MKE' 'OMA' 'SAV' 'BTV' 'ORD' 'CLT' 'BWI' 'STL' 'BGR' 'LAX' 'DFW' 'AUS'
 'MIA' 'DCA' 'PHX' 'SFO' 'EGE' 'SNA' 'STT' 'SEA' 'PDX' 'FLL' 'BOS' 'SJU'
 'PBI' 'MCO' 'ORH' 'RSW' 'SMF' 'SRQ' 'JAX' 'SAT' 'IAH' 'MSY' 'PSP' 'DEN'
 'MCI' 'TPA' 'LAS' 'ONT' 'BUR' 'BQN' 'SLC' 'PSE' 'RNO' 'BZN' 'HNL' 'SAN'
 'ISP' 'SWF' 'SFB' 'PBG' 'PIE' 'PGD' 'MYR' 'PHL' 'ELM' 'EWR' 'DAL' 'MDW'
 'HOU' 'TUL' 'PIT' 'GSO' 'CMH' 'SDF' 'LIT' 'OKC' 'EYW' 'CHO' 'XNA' 'RIC'
 'MSN' 'IAD' 'BDL' 'BGM' 'IAG' 'DAY' 'ROA' 'MTJ' 'JAC' 'TYS' 'GRR' 'BHM'
 'CAE' 'DSM' 'SCE' 'LEX']
```

Step: 8 Changing the datatype of Column

When we are evaluating the datatypes of the features we observed that the number of flights and diverted or not is in the Float data type so We are Changing the Datatype of the flights & Diverted to integer value

```
#Changing the Datatype of the flights to integer value
df['FLIGHTS'] = df['FLIGHTS'].astype('int64')
df['DIVERTED'] = df['DIVERTED'].astype('int64')
```

Step: 9 Converting few columns into categorical data

To obtain the relation between the columns and to build the model efficiently we need to find the columns which have categorical data and convert them into categorical types of columns. Converting columns “DAY_OF_WEEK” & “AIRLINE” to categorical data as they have only less number of values and which will be repeating.

```
#Converting the columns into Categorical Data
columns = ['DAY_OF_MONTH', 'DAY_OF_WEEK', 'AIRLINE','ORIGIN','DEST']
unique = {col: df[col].unique() for col in columns}
print(unique)
df = pd.get_dummies(df, columns=['DAY_OF_WEEK', 'AIRLINE'])
df.shape
```

```
{'DAY_OF_MONTH': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]), 'DAY_OF_WEEK': array([7, 1,
       'OO', 'UA', 'WN', 'YX'], dtype=object), 'ORIGIN': array(['LGA', 'GSP', 'JFK', 'AVL',
       'CVG', 'MEM', 'ORF', 'ATL', 'HPN', 'BUF', 'PVD', 'ITH', 'MSP',
       'CLE', 'SYR', 'IND', 'ILM', 'PWM', 'CHS', 'MKE', 'OMA', 'SAV',
       'BTV', 'ORD', 'CLT', 'BWI', 'STL', 'BGR', 'LAX', 'DFW', 'AUS',
       'MIA', 'DCA', 'PHX', 'SFO', 'EGE', 'SNA', 'STT', 'SEA', 'PDX',
       'FLL', 'BOS', 'SJU', 'PBI', 'MCO', 'ORH', 'RSW', 'SMF', 'SRQ',
       'JAX', 'SAT', 'IAH', 'MSY', 'PSP', 'DEN', 'MCI', 'TPA', 'LAS',
       'ONT', 'BUR', 'BQN', 'SLC', 'PSE', 'RNO', 'BZN', 'HNL', 'SAN',
       'ISP', 'SWF', 'SFB', 'PBG', 'PIE', 'PGD', 'MYR', 'PHL', 'ELM',
       'EWR', 'DAL', 'MDW', 'HOU', 'TUL', 'PIT', 'GSO', 'CMH', 'SDF',
       'LIT', 'OKC', 'EYW', 'CHO', 'XNA', 'RIC', 'MSN', 'IAD', 'BDL',
       'BGM', 'IAG', 'DAY', 'ROA', 'MTJ', 'JAC', 'TYS', 'GRR', 'BHM',
       'CAE', 'DSM', 'SCE', 'LEX'], dtype=object), 'DEST': array(['RDU', 'LGA', 'GSP', 'DTW', 'MSP', 'AVL', 'ALB', 'BUF', 'BNA',
       'MKE', 'JFK', 'CLT', 'ROC', 'MEM', 'PWM', 'ORF', 'HPN', 'ATL',
       'ITH', 'CHS', 'SAV', 'CLE', 'SYR', 'ILM', 'IND', 'CVG', 'BTV',
       'MCI', 'ORD', 'BGR', 'PVD', 'STL', 'BWI', 'OMA', 'LAX', 'DFW',
       'MIA', 'DCA', 'PHX', 'EGE', 'SFO', 'AUS', 'STT', 'SNA', 'SEA',
       'SAN', 'PDX', 'PBI', 'MCO', 'BOS', 'LAS', 'TPA', 'JAX', 'RSW',
       'FLL', 'SJU', 'SMF', 'SRQ', 'MSY', 'PSP', 'IAH', 'DEN', 'BQN',
       'ONT', 'BUR', 'SLC', 'SAT', 'PSE', 'ORH', 'RNO', 'BZN', 'ISP',
       'SWF', 'PBG', 'SFB', 'PIE', 'PGD', 'HNL', 'MYR', 'PHL', 'ELM',
       'EWR', 'DAL', 'MDW', 'HOU', 'CMH', 'RIC', 'TUL', 'LIT', 'SDF',
       'OKC', 'XNA', 'EYW', 'CHO', 'PIT', 'GSO', 'MSN', 'IAD', 'BDL',
       'BGM', 'IAG', 'DAY', 'ROA', 'JAC', 'MTJ', 'SCE', 'TYS', 'DSM',
       'BHM', 'GRR', 'CAE', 'LEX'], dtype=object)}
 (58606, 45)
```

Step: 10 Creating the reference date for better understanding

As Date is an important element in this problem domain in this dataset we need to have a proper column with date and it should be easily accessible for visualization and model purposes. So we are combining the columns which have the date related information and forming the date column. Adding date Column to the dataset using existing columns.

```
#Adding date Column to the dataset using existing columns
df['DATE'] = pd.to_datetime(df['DAY_OF_MONTH'].astype(str) + '-' + str(1) + '-' + str(2023), format='%d-%m-%Y')
```

Step: 11 Changing the timestamp into readable python format

In the dataset we have features with HHMM format of time which is hard to use in our model directly and it is also complex to derive the visualizations around it if the time format is not matching.Converting time data in HHMM format to datetime.datetime

```
import datetime
#Function to change the time format to Datetime Format
def to_time(value):
    if pd.isnull(value):
        return None
    value = int(value)
    if value == 2400:
```

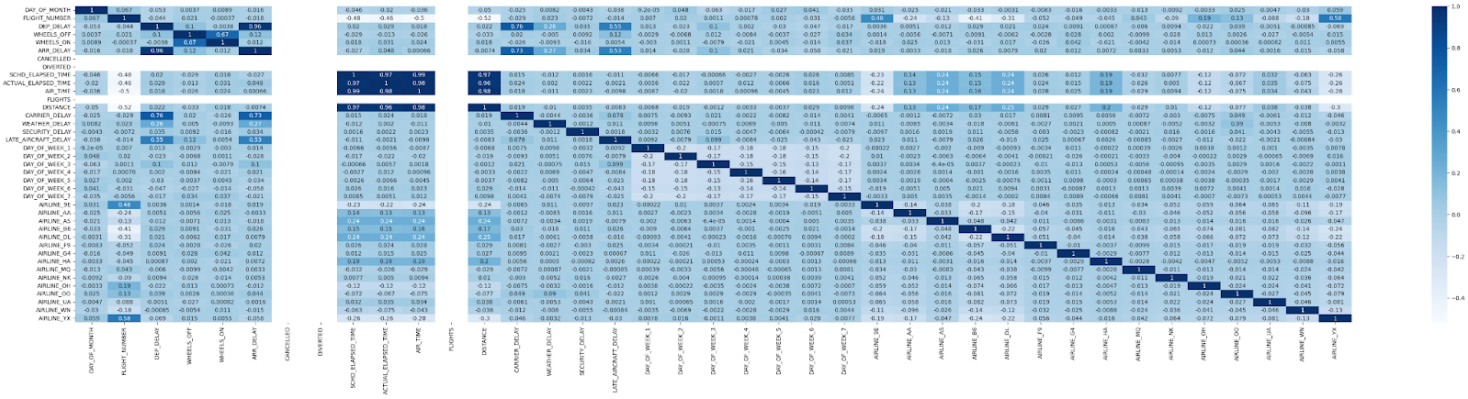
```
        return datetime.time(0,0)
    # Convert the HHMM value to a time object
    hours = value // 100
    minutes = value % 100
    return datetime.time(hours,minutes)
def merge_date_time(row, time_column):
    time=to_time(row[time_column])
    if time is None:
        return None
    if time== datetime.time(0, 0) and row[time_column] == 2400:
        return datetime.datetime.combine(row['DATE'] + datetime.timedelta(days=1),time)
    return datetime.datetime.combine(row['DATE'], time)
# Applying format to columns related to time.
df['SCHD_DEP_TIME'] = df.apply(merge_date_time, time_column='SCHD_DEP_TIME', axis=1)
df['DEP_TIME'] = df.apply(merge_date_time, time_column='DEP_TIME', axis=1)
df['SCHD_ARR_TIME'] = df.apply(merge_date_time, time_column='SCHD_ARR_TIME', axis=1)
df['ARR_TIME'] = df.apply(merge_date_time, time_column='ARR_TIME', axis=1)
```

➢ Exploratory Data Analysis

Identifying correlation between the columns

Correlation matrix gives the analysis of how close the columns are related to each other. Prediction and decision-making can both benefit from an understanding of these relationships. For instance, a large positive correlation between two variables might imply a causal connection, but a negative correlation might emphasize the necessity of taking trade-offs into account when making decisions. High value in the correlation matrix implies the columns are very closely related. In our dataset the columns ‘Flights’, ‘canceled’, ‘Diverted’ have no impact on the departure delay. This means we can safely remove those columns from the dataset as there is no significant impact.

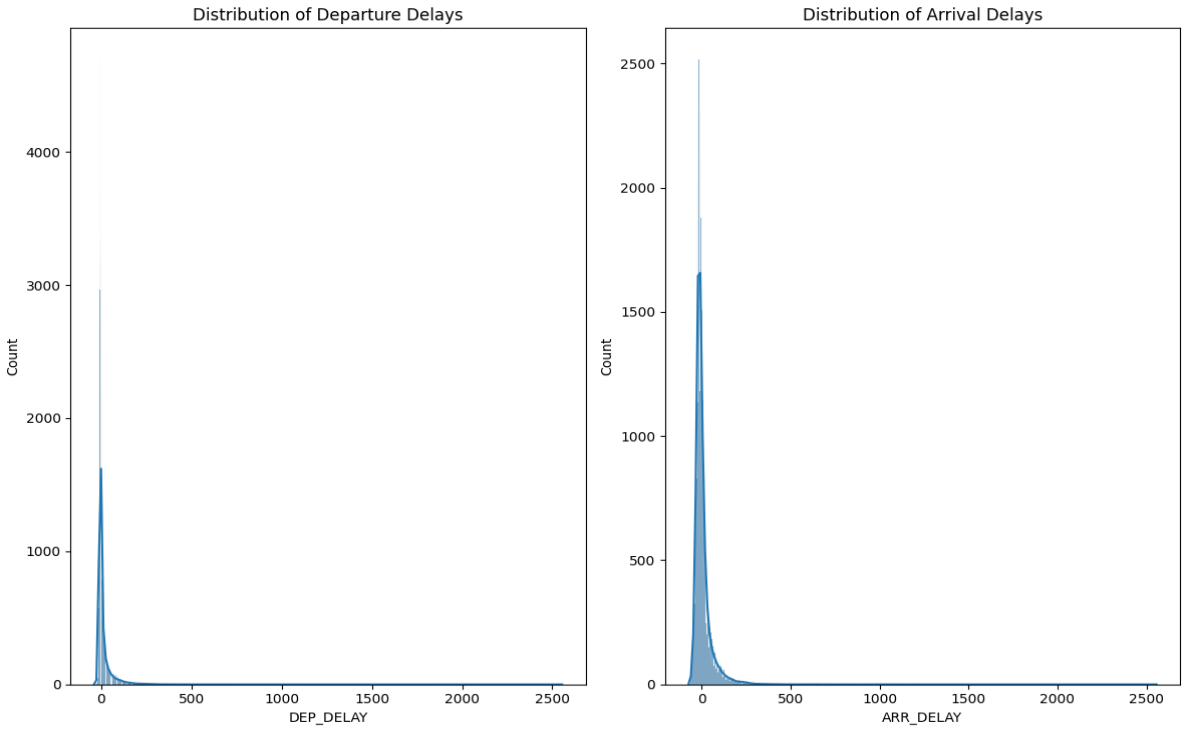
```
#Analyzing Correlation Matrix and dropping the unrelated columns from the Dataset
# Identify how different numeric columns correlate, especially how different delay reasons correlate with the actual delay.
plt.figure(figsize=(50,10))
sns.heatmap(df.corr(),cbar=True,annot=True,cmap='Blues')
df.drop(['CANCELLED', 'DIVERTED','FLIGHTS'], axis=1, inplace=True)
```



Distribution of delays

Histogram plots of delays give the distribution of delays. Which helps in analyzing the central tendency, skewness, Extreme values and how spread are the values of delay. Both Departure and Arrival delays follow the same distribution pattern, both are very narrowly distributed and skewed towards right. There are no such outliers that are very extreme, hence there is no need to find any exceptional circumstances that are impacting delays only at certain times like weather impacts.

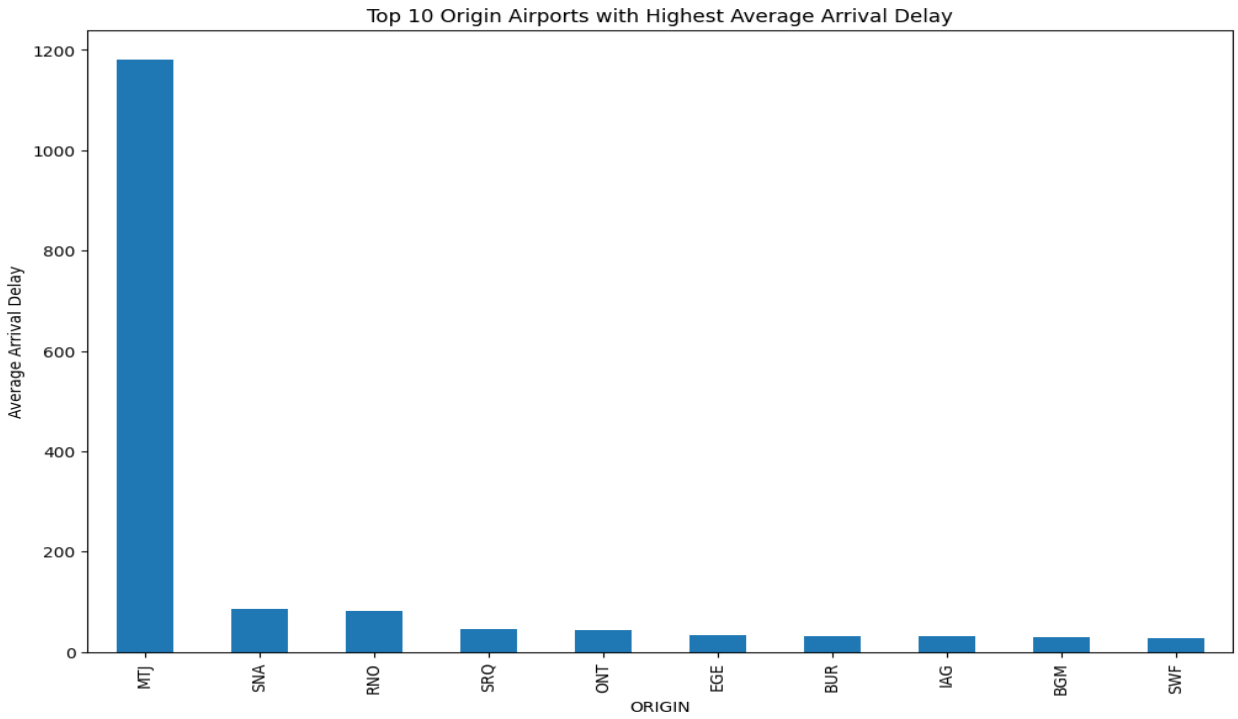
```
# Distribution of Delays
# Understanding the distribution of delays can give insight about the extent of delays.
plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
sns.histplot(df['DEP_DELAY'], kde=True)
plt.title('Distribution of Departure Delays')
plt.subplot(1, 2, 2)
sns.histplot(df['ARR_DELAY'], kde=True)
plt.title('Distribution of Arrival Delays')
plt.tight_layout()
plt.show()
```



Top 10 airports with high avg arrival delays

The histogram, just to check how is the average delay with respect to different airports from where flights land. The delay in arrival is around the same for all the airports, except for MTJ Airport, which has a very high average Arrival delay. This is not resulting in any trend thus the dataset has some extreme values that might need to be addressed later.

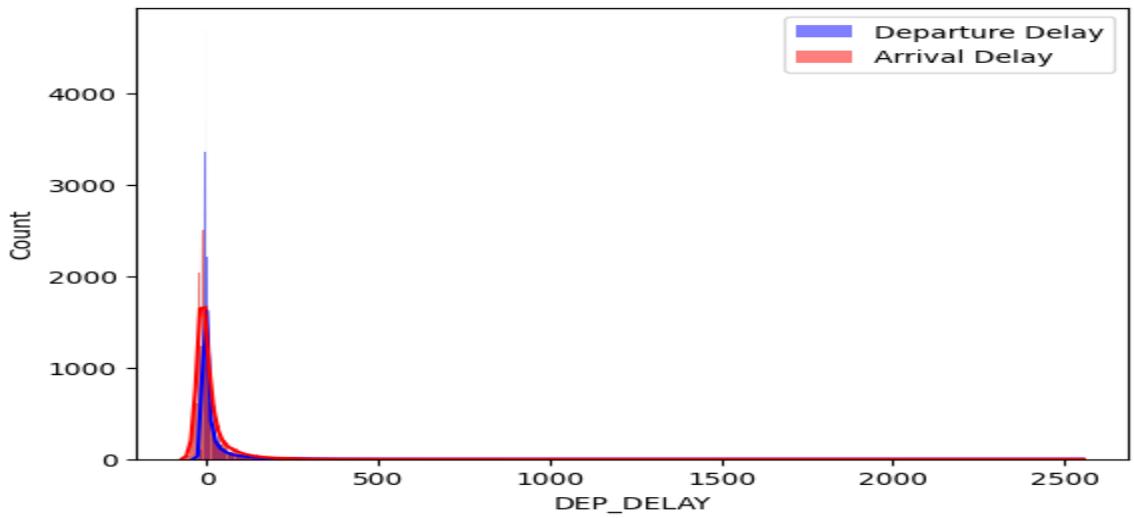
```
#Top 10 airports with High Avg Arrival Delays

airports                                                              =
df.groupby('ORIGIN')['ARR_DELAY'].mean().sort_values(ascending=False).head
(10)
airports.plot(kind='bar', figsize=(12,8))
plt.title('Top 10 Origin Airports with Highest Average Arrival Delay')
plt.ylabel('Average Arrival Delay')
plt.show()
```



Departure and arrival delay comparison

A visual comparison of the two types of delays in the dataset is provided by the arrival and departure delays' histograms, which are overlaid in blue and red, respectively. Sometimes airlines have buffer time into their schedule to account for potential delays

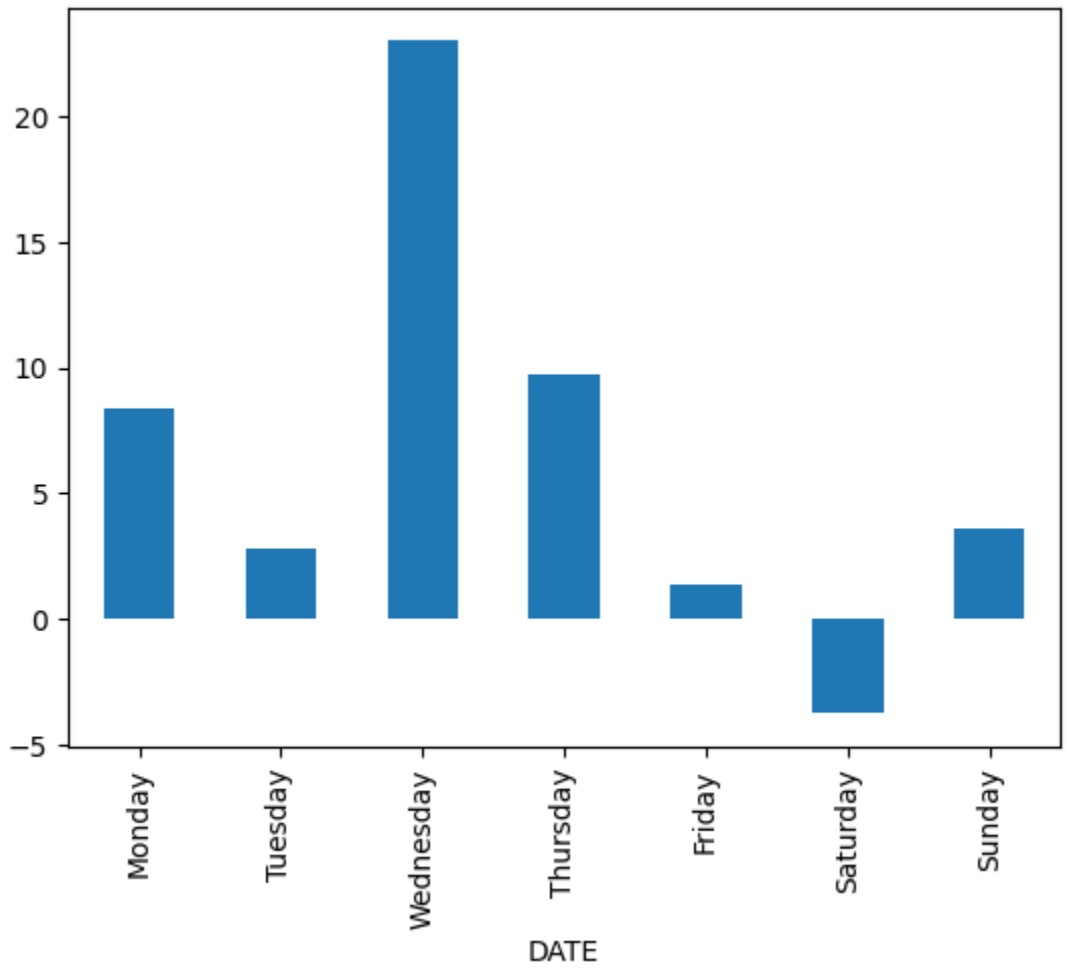
```
#Departure and Arrival Delay Comparison
sns.histplot(df['DEP_DELAY'], kde=True, color='blue', label='Departure Delay')
sns.histplot(df['ARR_DELAY'], kde=True, color='red', label='Arrival Delay')
plt.legend()
plt.show()
```



Average arrival delays for each day of the week

Now to check how the arrival delay is with respect to the day of the week. It is expected that day of the week might have an impact on flight delays and is evident from the below graph. We can say from the graph that more delays are seen on wednesday. This varying probability of average delay based on the day of the week seems like this could be the potential column in further analyzing or arriving at any predictions. Variation along the days of the week is not consistent since there are slight extremes in average delay.

```
# Day Of the week vs Arrival Delay
# See if certain days of the week are more prone to delays.
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
avg_daily_delay =
df.groupby(by=df['DATE'].dt.day_name())['ARR_DELAY'].mean().reindex(days)
avg_daily_delay.plot(kind='bar')
plt.show()
```

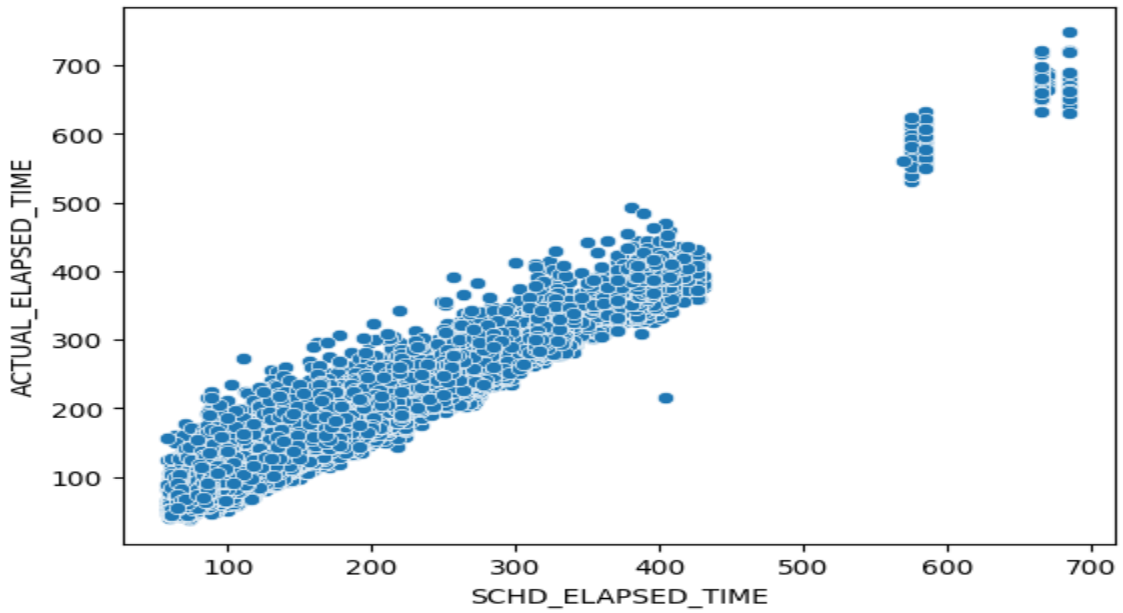


Scatter Plot of scheduled elapsed time and actual elapsed time

How well flight schedules match actual performance can be seen visually in the scatter plot. Weather, air traffic, or operational inefficiencies are just a few of the things that can cause deviations from the optimum line. It is essential to analyze these variations in order to schedule flights optimally, cut down on delays, and enhance the overall passenger experience.

In our case it seems to follow the linear trend, which means scheduled and actual time match almost perfectly.

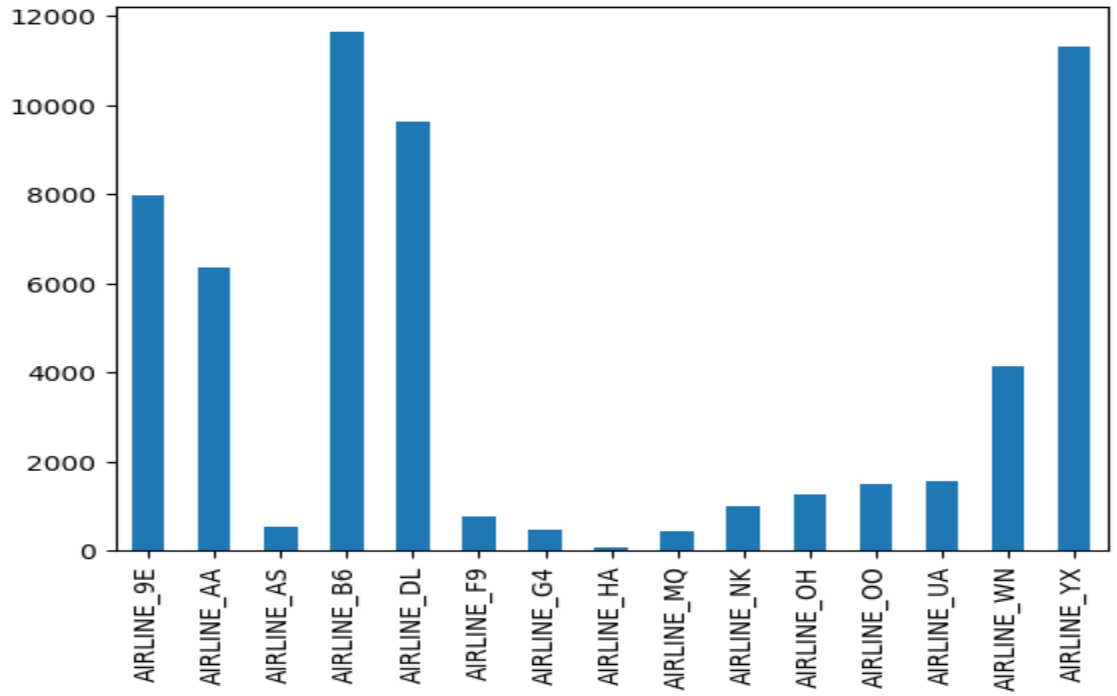
```
#Scatter Plot the Scheduled Elapsed Time and Actual Elapsed Time
sns.scatterplot(x='SCHD_ELAPSED_TIME', y='ACTUAL_ELAPSED_TIME', data=df)
plt.show()
```



Flight volumes for different airlines using a bar graph

This bar graph gives the count or frequency of flights managed by different airlines in a month. This frequency or number might have an impact on delay as well. The results show that only 5 airlines have multiple flights operated. For a variety of stakeholders, including airlines, airports, and aviation regulators, understanding flight volumes per airline is crucial. It can be useful in determining market share, competition, and market domination. Additionally, it offers information about the size of activities and capacity planning.

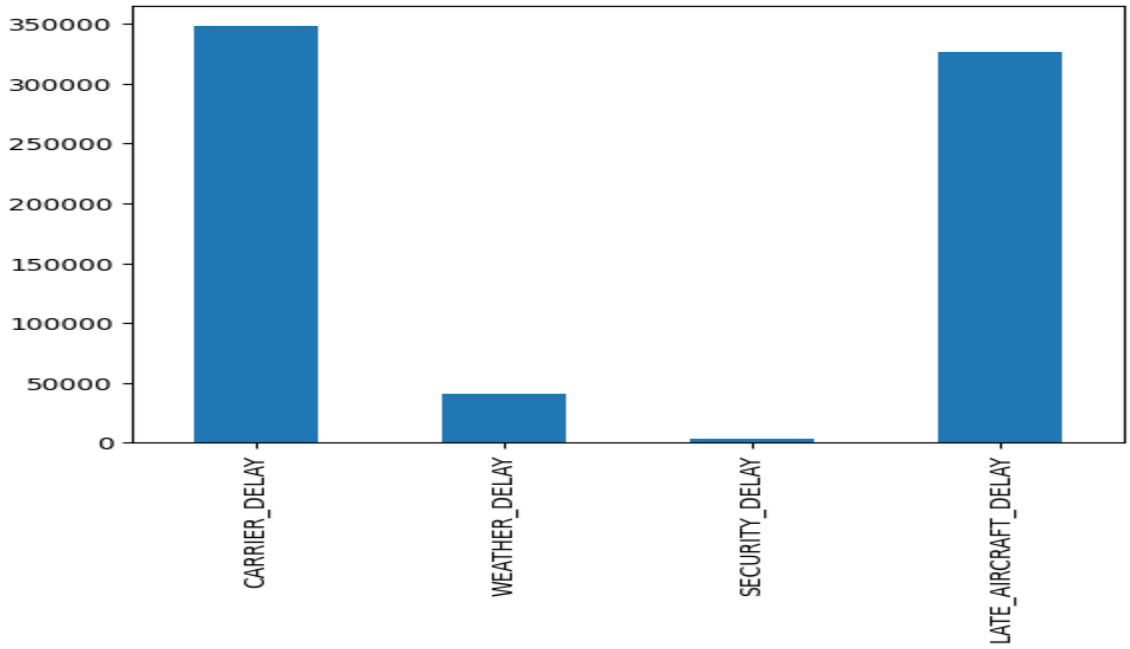
```
#Airline Analysis using Bar Graph
airlines_cols = [col for col in df.columns if 'AIRLINE_' in col]
flight_volume = df[airlines_cols].sum()
flight_volume.plot(kind='bar')
plt.show()
```



The total delay minutes for different delay reasons using bar graph

To compare and visualize the effects of different delay factors on flight operations, it can be helpful to analyze the total delay minutes for various delay causes using a bar graph. By examining the contribution of different delays the impact of security delay and weather delay is not that significant. It is the Carrier delay and late aircraft delay that has more impact on the total delay. It is true because of the fact that flights are scheduled considering all the weather conditions.

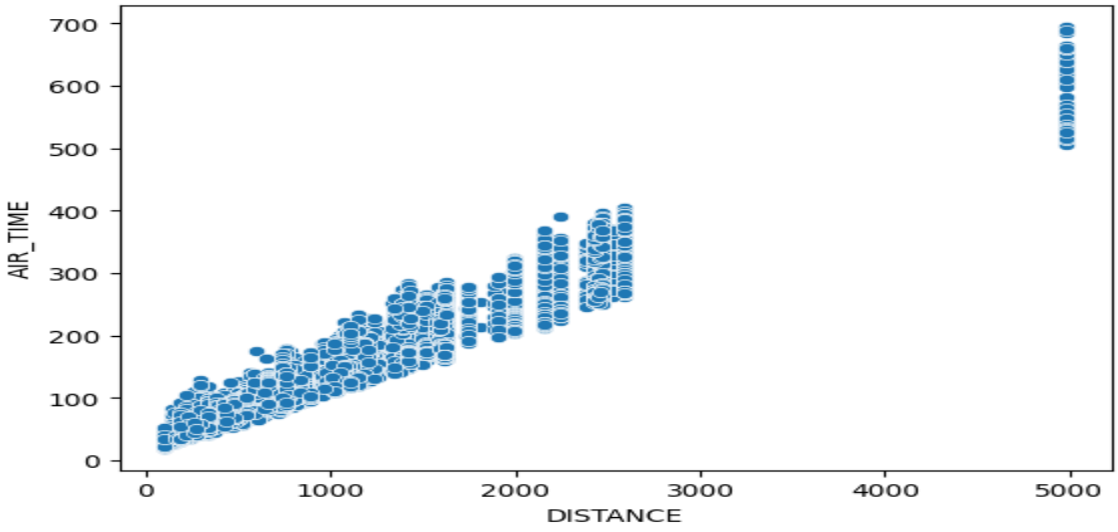
```
#Bar Graph of the Carrier and delay in minutes for each Delay.
# This can help identify the most common reasons for flight delays.
delay_reasons = ['CARRIER_DELAY', 'WEATHER_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY']
df[delay_reasons].sum().plot(kind='bar')
plt.show()
```



Scatter Plot visualization between flight distance and airtime

For the flights in our dataset, we look at the correlation between flight distance and airtime in this section. Airtime is the actual amount of time a flight spent in the air, whereas flight distance is the distance each flight has traveled. This association is graphically depicted in the scatter plot below. Here also linearity can be seen which implies that airtime and distance are in match. Though the slight variations might be the impact of weather delays. And also we can see from the plot that there are only a few flights that are covering very large distances of around 4800. This might act as an outlier or extremes compared to other data.

```
#Scatter Plot the Distance and AirTime
sns.scatterplot(x='DISTANCE', y='AIR_TIME', data=df)
plt.show()
```

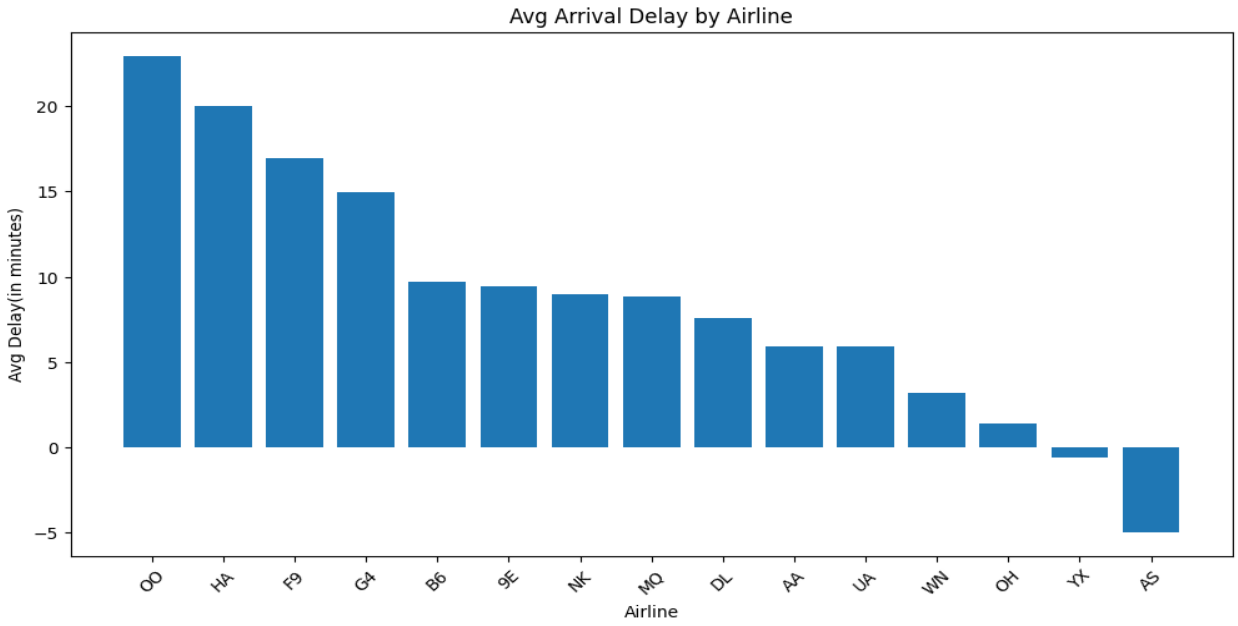


Average delay per airline

We investigated the typical delay times for several airlines, to analyze how different airlines operate in terms of arrival delays and how they affect the traveling experience.

A bar plot shows how average arrival delays vary across different airlines, making it easier to see which airlines, on average, have longer or shorter arrival delays. The plot is also sorted in descending order , Airlines 'OO' have the highest average delay time & airline 'AS' have the lowest average arrival delay time

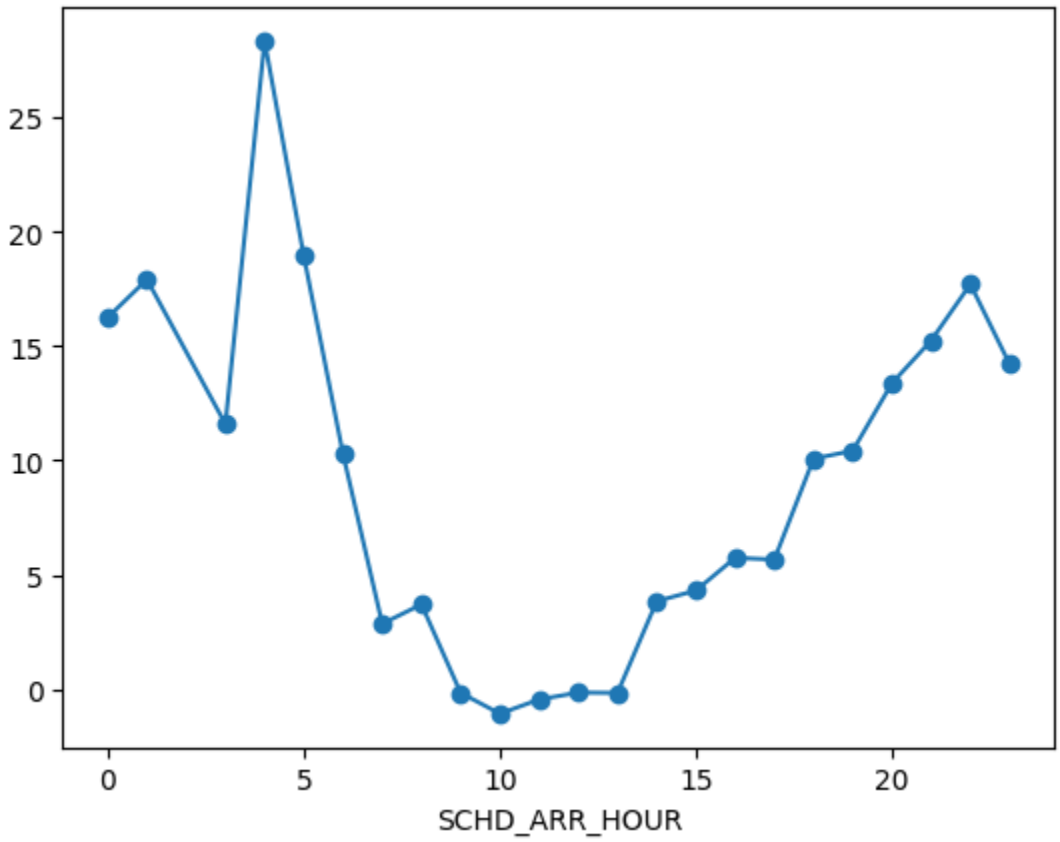
```
# Calculating average delay per airline
airlines_cols = [col for col in df.columns if 'AIRLINE_' in col]
avg_airline_delay = {}
for col in airlines_cols:
    airline_data = df[df[col] == 1]
    avg_airline_delay[col.split('_')[-1]] = airline_data['ARR_DELAY'].mean()
avg_airline_delay= dict(sorted(avg_airline_delay.items(), key=lambda item: item[1], reverse=True))
plt.figure(figsize=(12,6))
plt.bar(avg_airline_delay.keys(),avg_airline_delay.values())
plt.title('Avg Arrival Delay by Airline')
plt.xlabel('Airline')
plt.ylabel('Avg Delay(in minutes)')
plt.xticks(rotation=45)
plt.show()
```



Plotting the average arrival delay for each hour

The figure shows probable peak delays during rush hours and identifies hourly patterns in airline arrival delays.The correlation between scheduled arrival times and typical arrival delays is examined in this section. We have taken the scheduled arrival time for each flight from the dataset, which contains flight data. From the visualization we can say that most delay happens during 0000 to 0500 hrs.

```
# Hourly Analysis of Delays
# Grouping by hour can help identify peak delay times
df['SCHD_ARR_HOUR'] = df['SCHD_ARR_TIME'].dt.hour
avg_arr_hourly_delay = df.groupby('SCHD_ARR_HOUR')['ARR_DELAY'].mean()
avg_arr_hourly_delay.plot(kind='line', marker='o')
plt.show()
```



Phase #2

Preprocessing:

Apart from the preprocessing done in phase 1, there were few features found in the dataset that had to be removed because of very less correlation with the target. And 2 of the columns Origin and destination airports of type objects are encoded to convert into integer type.

After all these modifications, now the data looks like below

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11722 entries, 24558 to 47178
Data columns (total 38 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   DAY_OF_MONTH          11722 non-null  int64
 1   SCHD_DEP_TIME         11722 non-null  int64
 2   DEP_TIME              11722 non-null  int64
 3   DEP_DELAY             11722 non-null  float64
 4   WHEELS_OFF            11722 non-null  float64
 5   SCHD_ARR_TIME         11722 non-null  int64
 6   SCHD_ELAPSED_TIME     11722 non-null  int64
 7   ACTUAL_ELAPSED_TIME   11722 non-null  float64
 8   AIR_TIME              11722 non-null  float64
 9   DISTANCE              11722 non-null  int64
 10  CARRIER_DELAY         11722 non-null  float64
 11  WEATHER_DELAY         11722 non-null  float64
 12  SECURITY_DELAY        11722 non-null  float64
 13  LATE_AIRCRAFT_DELAY   11722 non-null  float64
 14  DAY_OF_WEEK_1         11722 non-null  int64
 15  DAY_OF_WEEK_2         11722 non-null  int64
 16  DAY_OF_WEEK_3         11722 non-null  int64
 17  DAY_OF_WEEK_4         11722 non-null  int64
 18  DAY_OF_WEEK_5         11722 non-null  int64
 19  DAY_OF_WEEK_6         11722 non-null  int64
 20  DAY_OF_WEEK_7         11722 non-null  int64
 21  AIRLINE_9E            11722 non-null  int64
 22  AIRLINE_AA            11722 non-null  int64
 23  AIRLINE_AS            11722 non-null  int64
 24  AIRLINE_B6            11722 non-null  int64
 25  AIRLINE_DL            11722 non-null  int64
 26  AIRLINE_F9            11722 non-null  int64
 27  AIRLINE_G4            11722 non-null  int64
 28  AIRLINE_HA            11722 non-null  int64
 29  AIRLINE_MQ            11722 non-null  int64
 30  AIRLINE_NK            11722 non-null  int64
 31  AIRLINE_OH            11722 non-null  int64
 32  AIRLINE_OO            11722 non-null  int64
 33  AIRLINE_UA            11722 non-null  int64
 34  AIRLINE_WN            11722 non-null  int64
 35  AIRLINE_YX            11722 non-null  int64
 36  ORIGIN                11722 non-null  float64
 37  DEST                  11722 non-null  float64
dtypes: float64(10), int64(28)
memory usage: 3.5 MB
```

Models:

We have applied a total 6 regression models, in order to predict the Arrival delay that a flight may face for reaching the destination airport. The models implemented are:

1. Random Forest Regression
2. Linear Regression
3. Support Vector Regression
4. Decision Tree Regression
5. Gradient Boosting
6. XG Boost

1. Random Forest Regression:

Random forest is a tree-based model. The variance in the dataset is very high. And also there are many features, whose values hugely fall into some interval. Since there might be a nonlinearity relation in the data and the target, Random forest is being used so that any non linearity can be captured. And also in many features more values fall into some category as compared to remaining features, since Random forest is robust to overfitting this model might address the issue of overfitting the model.

Implementation:

For implementation of 'RandomForestRegressor' we import from the Scikit learn library. 'sklearn.linear_model' is used in the implementation and the default parameters are kept as such without any modifications.

n_estimators: is the no of trees in the forest. We have taken 100

max_depth: depth of each tree. We have taken 10

min_samples_split: The minimum number of samples required to split a node in a tree. We have taken 4

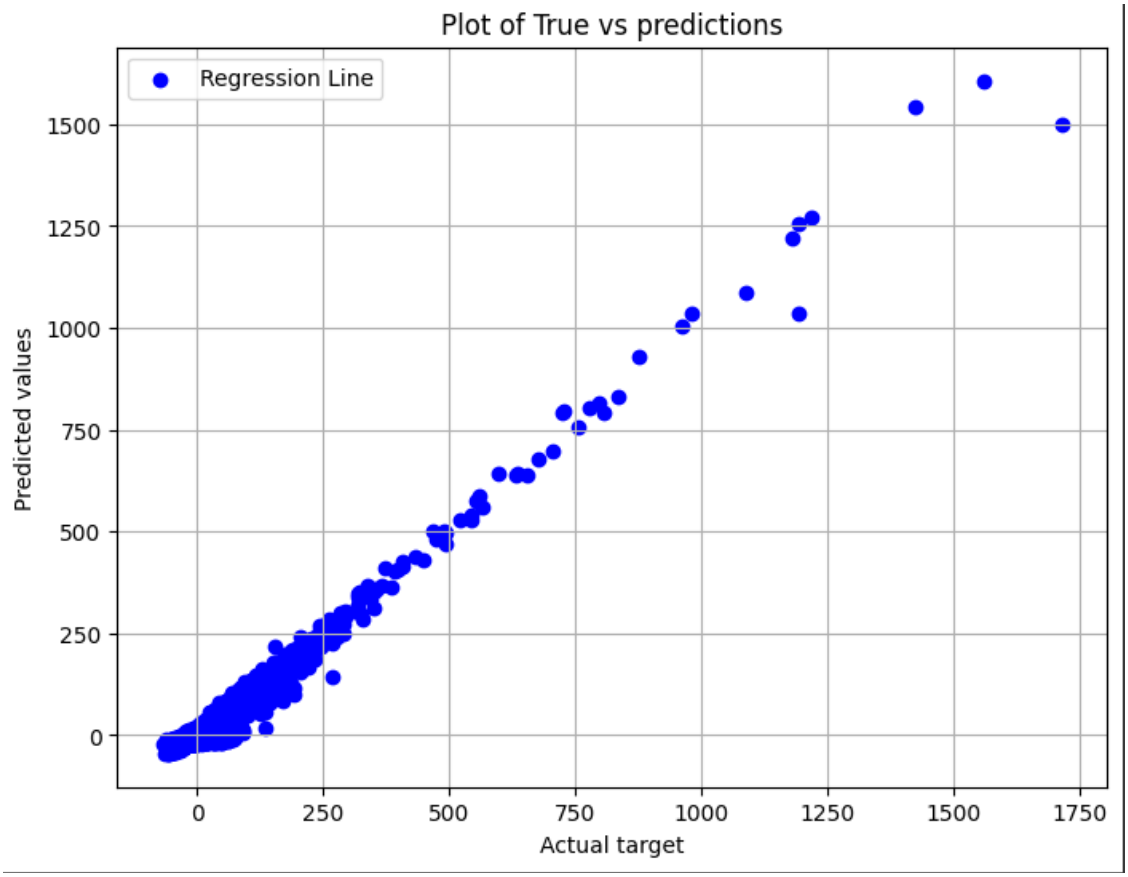
```
#1.Random Forest Regressor
model1= RandomForestRegressor(n_estimators=100, max_depth=10, min_samples_split=4, random_state=42)
model1.fit(X_train, y_train)
y_pred=model1.predict(X_test)
model1_mse= mean_squared_error(y_test,y_pred)
model1_rmse=np.sqrt(model1_mse)
model1_mae=mean_absolute_error(y_test,y_pred)
model1_r2= r2_score(y_test,y_pred)
print(f"Random Forest Metrics:  Mean Square Error: {model1_mse}, Root Mean Square Error: {model1_rmse}")
print(f"Mean Absolute Error: {model1_mae}, r2: {model1_r2}")
```

Results:

Root mean square error of 12 is achieved. R2 score 0.96 implies 96% accuracy on the test data which is a very good fit. The R-squared score of 0.964832779587538 indicates that the model explains 96.48% of the variance in the target variable (flight delays). This is a very high R-squared score, which suggests that the model is a good fit for the data.

```
Random Forest Metrics:  Mean Square Error: 155.32770157971459, Root Mean Square Error: 12.463053461319765
Mean Absolute Error: 8.3309018286301, r2: 0.964832779587538
```

Plot:



The plot of true vs. predicted values shows that the model is able to accurately predict flight delays across a wide range of values. There is a strong correlation between the true and predicted values, and the model is not biased in either direction. However, there are data points where the model's prediction is significantly different from the actual value. This might be due to several reasons: like aircraft doesnt have a clear runway i.e having traffic , technical glitch, heavy aircraft traffic at destination airports etc.

2. Linear Regression:

Linear regression is a simple and transparent model that assumes linear relation. It is interpretable. This means that it is possible to understand how the input features contribute to the predicted arrival delay. This can be useful for identifying factors that airlines can control to reduce arrival delays. This model can be used as a benchmark to find if the target has linear relation with the data or not. Since the dataset is not that complex it might provide reasonable prediction without overfitting.

Implementation:

For implementation of "LinearRegression" we imported it from the Scikit learn library. 'sklearn.linear_model' is used in the implementation

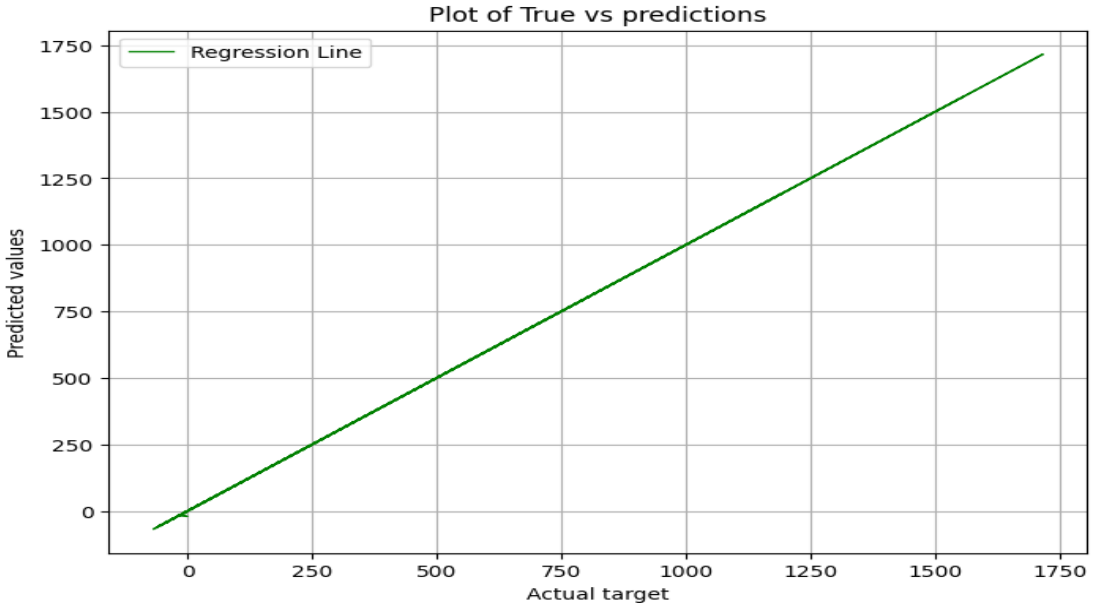
```
#2 Linear Regression
lin_model=LinearRegression()
lin_model.fit(X_train,y_train)
lin_y_pred=lin_model.predict(X_test)
lin_mse= mean_squared_error(y_test,lin_y_pred)
lin_rmse=np.sqrt(lin_mse)
lin_mae=mean_absolute_error(y_test,lin_y_pred)
lin_r2= r2_score(y_test,lin_y_pred)
print(f"Linear Regression Metrics:  Mean Square Error: {lin_mse}, Root Mean Square Error: {lin_rmse}")
print(f"Mean Absolute Error: {lin_mae}, r2: {lin_r2}")
```

Results:

The model achieves a mean squared error (MSE) of 0.03756930058176234, which is very good. The root mean squared error (RMSE) is 0.1938280180514735, and the mean absolute error (MAE) is 0.04868980833407491. These metrics indicate that the model is able to predict flight delays with a high degree of accuracy. The R-squared score is 0.9999914940615172 indicating the model has very high accuracy.

```
Linear Regression Metrics:  Mean Square Error: 0.03756930058176234, Root Mean Square Error: 0.1938280180514735
Mean Absolute Error: 0.04868980833407491, r2: 0.9999914940615172
```

Plots of predicted vs True values:



Strong correlation between the two variables can be seen in the plot of what was predicted versus the actual values. The model can forecast flight delays over a wide range of values, as evidenced by the regression line passing near to the data points. Overall, the plot suggests that the Linear Regression model is a good fit for the data and is able to accurately predict flight delays in most cases.

3. Support vector Regression:

Few of the features in the data have some values that vary more compared to most of the values.Since this data is also important and cannot be considered redundant, our model should be robust to outliers. So we tried with SVR, rbf and the polynomial kernel did not turn out to be that good, hence linear kernel is used. Also SVR is capable of recognizing both local and global trends in the data, it can function effectively for regressive data.

Implementation:

We used a linear kernel in the SVR model, with an epsilon value of 0.2 and a C value of 1.0. For applications like flight delay prediction, where the connection between the input data and the output target is probably linear, the linear kernel is a reasonable option.

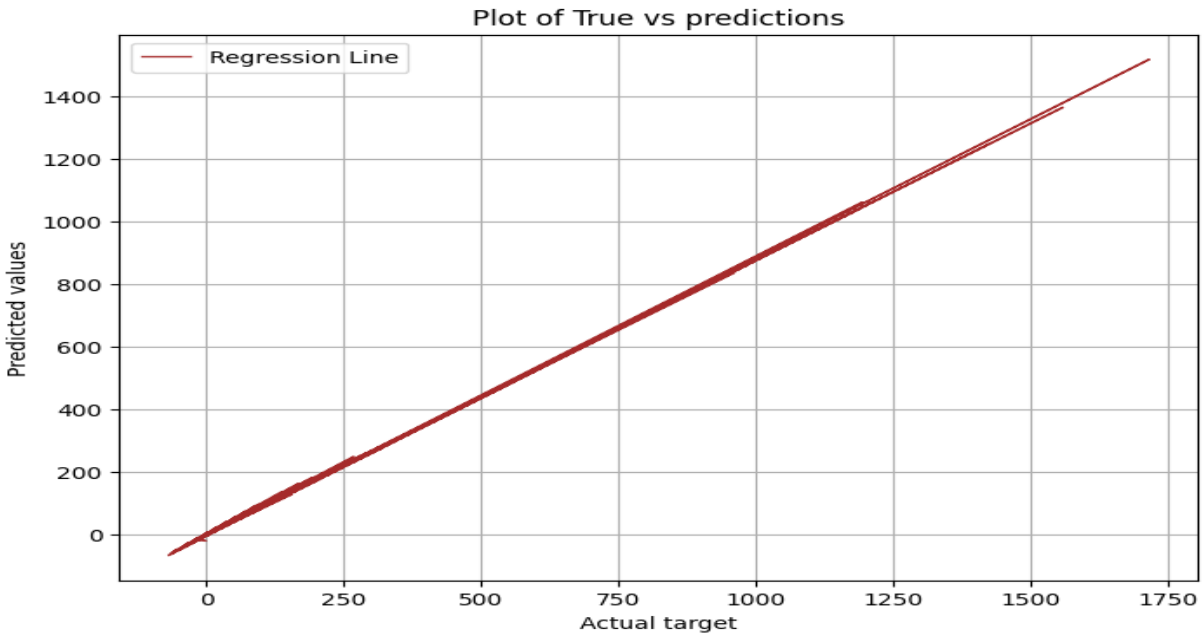
```
#3 Support vector Regression
scaler = StandardScaler()
X_train_scaled, X_test_scaled = scaler.fit_transform(X_train), scaler.fit_transform(X_test)
svr_model = SVR(kernel = 'linear', C=1.0, epsilon=0.2)
svr_model.fit(X_train_scaled, y_train)
y_pred_svr = svr_model.predict(X_test_scaled)
svr_mse = mean_squared_error(y_test, y_pred_svr)
svr_rmse=np.sqrt(svr_mse)
svr_mae=mean_absolute_error(y_test,y_pred_svr)
svr_model_r2= r2_score(y_test,y_pred_svr)
print(f"SVR Metrics:  Mean Square Error: {svr_mse}, Root Mean Square Error: {svr_rmse}")
print(f"Mean Absolute Error: {svr_mae}, r2: {svr_model_r2}")
```

Results:

With an R-squared score of 0.985, Mean Squared Error (MSE) of 64.42, Root Mean Squared Error (RMSE) of 8.03, and Mean Absolute Error (MAE) of 2.83, the SVR model in this instance performs exceptionally well on all evaluation criteria. This suggests that the model can be used to enhance airline operations in several ways, including identifying flights at high risk of delay, optimizing flight schedules

```
SVR Metrics:  Mean Square Error: 64.41996626913537, Root Mean Square Error: 8.026204973032733
Mean Absolute Error: 2.8262707038818653, r2: 0.985414892966871
```

Plot:



It can be observed that the model is able to predict delays accurately because the data points are closely packed around the regression line. There are very very few outliers compared to the other model.

4. Decision tree Regressor:

Decision trees can learn complex relationships between the input features and the target variable. This is important for arrival delay prediction, as the relationship is likely to be complex, with many different factors like weather, air traffic, departure delay, and arrival time etc. The relationship among several variables such as weather, air traffic, departure delay, and arrival time can be complex and nonlinear. Such nonlinear interactions can be captured by decision trees. The model is also good for interpretability of the features that are of different types and also robust to outliers.

Implementation:

For implementation of "DecisionTreeRegressor" we imported it from the Scikit learn library.
'sklearn.linear_model' is used in the implementation

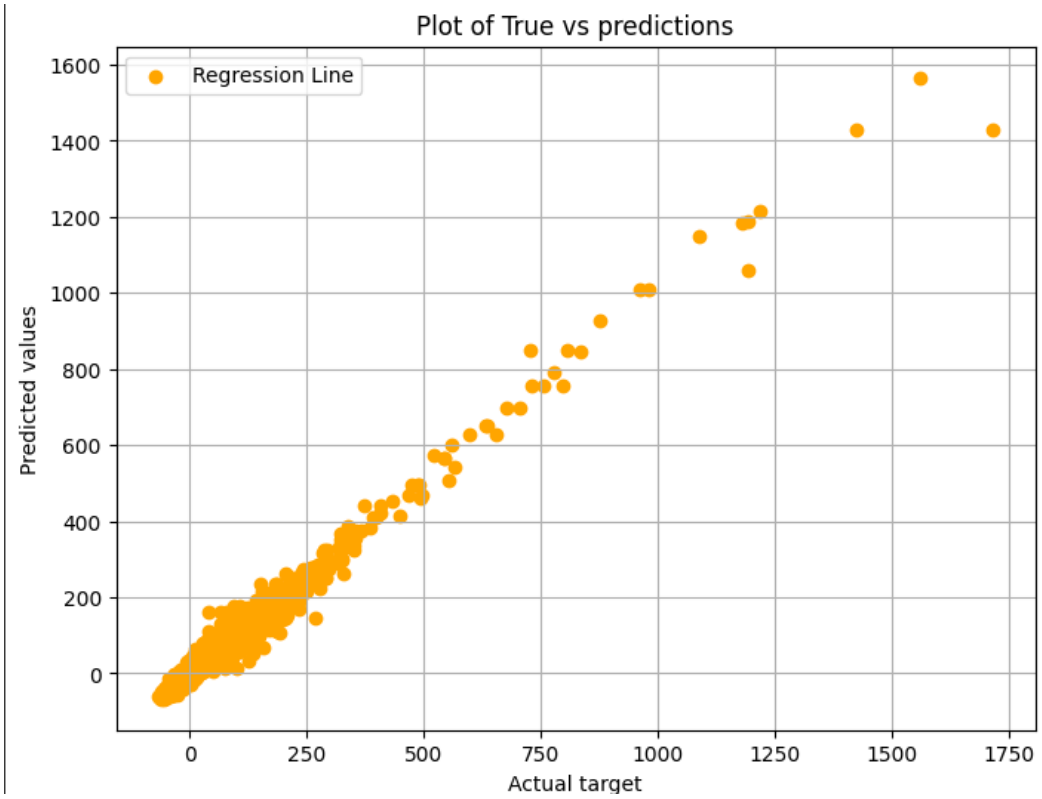
```
#4 Decision tree Regessor
from sklearn.tree import DecisionTreeRegressor
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
dt_mae=mean_absolute_error(y_test,y_pred_dt)
dt_mse = mean_squared_error(y_test, y_pred_dt)
dt_rmse=np.sqrt(dt_mse)
dt_r2= r2_score(y_test,y_pred_dt)
print(f"Decision tree Metrics:  Mean Square Error: {dt_mse}, Root Mean Square Error: {dt_rmse}")
print(f"Mean Absolute Error: {dt_mae}, r2: {dt_r2}")
```

Results:

The decision tree regressor model we provided achieved a mean squared error (MSE) of 79.48686231018597 and a root mean squared error (RMSE) of 8.915540494562624 on the test dataset. This indicates that the model is able to predict flight delays with a high degree of accuracy.

```
Decision tree Metrics:  Mean Square Error: 79.48686231018597, Root Mean Square Error: 8.915540494562624
Mean Absolute Error: 4.369476198600921, r2: 0.9820036479112983
```

Plots:



The scatter plot provided above shows the true vs. predicted delays for the test set. The points are tightly clustered around the regression line, which indicates that the model is able to make accurate predictions for a wide range of flight delays. There are a few outliers which indicate the model's prediction is significantly different from the actual value.

5. Gradient Boosting Regressor:

For aerial delay detection, the features in the data exhibit different patterns and complexity. Gradient Boosting Regressor is efficient at capturing interdependency of the features and patterns. Flight data also has some temporal variations like on certain days of the week or month the feature values exhibit variability as compared to usual data. Gradient Boosting is good at capturing temporal variations which is very important in delay prediction.

Implementation:

The GradientBoostingRegressor class in scikit-learn can be used to implement a gradient boosting regressor. For the parameters

n_estimators: we have chosen the value 100

Learning rate we have taken 0.1

Max_depth:The maximum depth of the individual decision trees : we have taken 3

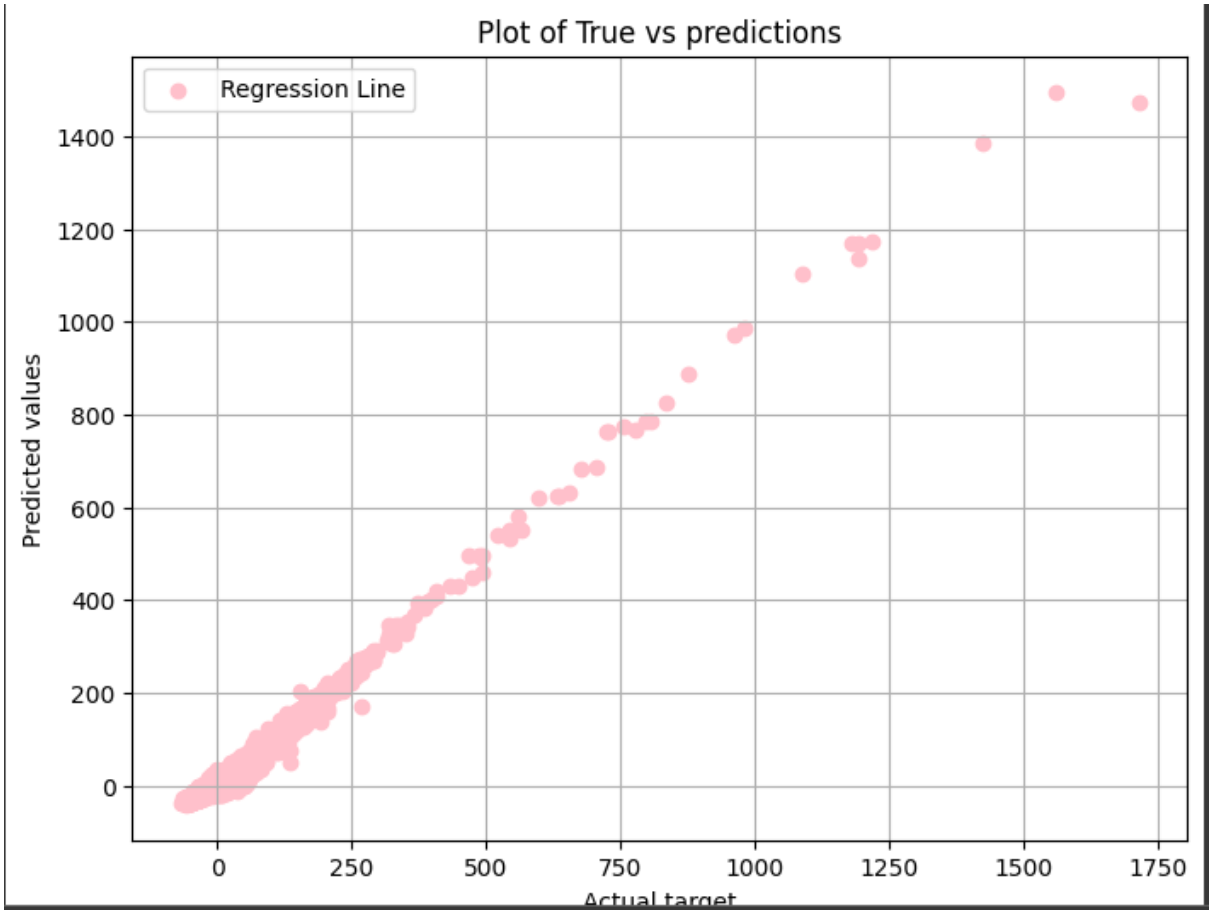
```
#5 Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)
gb_mse = mean_squared_error(y_test, y_pred_gb)
gb_rmse=np.sqrt(gb_mse)
gb_mae=mean_absolute_error(y_test,y_pred_gb)
gb_r2= r2_score(y_test,y_pred_gb)
print(f"Gradient Boosting Regressor Metrics:  Mean Square Error: {gb_mse}, Root Mean Square Error: {gb_rmse}")
print(f"Mean Absolute Error: {gb_mae}, r2: {gb_r2}")
```

Results:

It has a mean squared error (MSE) of 68.14, a root mean squared error (RMSE) of 8.25, a mean absolute error (MAE) of 5.39, and an r-squared score of 0.9845. This means that the model is able to predict flight delays with a good degree of accuracy.

```
Gradient Boosting Regressor Metrics:  Mean Square Error: 68.14221719592477, Root Mean Square Error: 8.254829931375012
Mean Absolute Error: 5.3982494345716265, r2: 0.9845721507036327
```

Plot:



The plot of the true versus predicted values shows that the model is able to fit the data very well. The majority of the points fall along the regression line. Even though the prediction is not a perfect fit as in the above plot the difference is not very high.

6. XG Boost

XGBoost is a gradient boosting algorithm that uses decision trees as base learners and builds an ensemble of these trees to make predictions. It can train large models on large datasets quickly. This is important for arrival delay prediction, as the dataset is likely to be large and complex. XGBoost is robust to overfitting. This means that it is less likely to learn the training data too well, which can lead to poor performance on new data.

Implementation:

For implementation of "XGBRegressor " we imported it from the Scikit learn library.
'sklearn.linear_model' is used in the implementation. The following parameters are also passed :
Objective : This parameter specifies the learning objective of the model. Our model objective is 'squared error'

colsample_bytree: This parameter controls the fraction of columns that are randomly sampled when constructing each tree. We have given value as 0.3
Learning_rate: we have given 0.1
Max_depth: we have given 5
alpha: This parameter controls the L1 regularization term. We have given value of 10
N_estimators: we have given the value 100

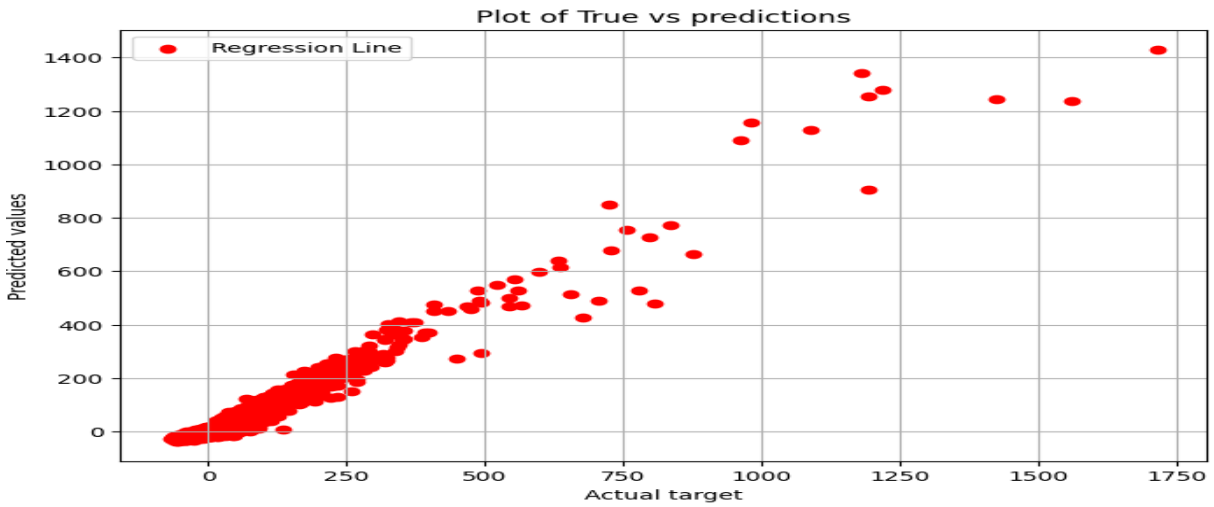
```
#6 XGBoost model
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, learning_rate = 0.1,max_depth = 5, alpha = 10, n_estimators = 100)
xgb_model.fit(X_train,y_train)
y_pred_xgb = xgb_model.predict(X_test)
xgb_mse = mean_squared_error(y_test, y_pred_xgb)
xgb_mse = mean_squared_error(y_test, y_pred_xgb)
xgb_rmse=np.sqrt(xgb_mse)
xgb_mae=mean_absolute_error(y_test,y_pred_xgb)
xgb_r2= r2_score(y_test,y_pred_xgb)
print(f"XGBoost Metrics:  Mean Square Error: {xgb_mse}, Root Mean Square Error: {xgb_rmse}")
print(f"Mean Absolute Error: {xgb_mae}, r2: {xgb_r2}")
```

Result:

The XGBoost model we have trained shows good performance on the flight delay forecasting task. The mean squared error (MSE) is 217.1498, and the root mean squared error (RMSE) is 14.7360, which means that the model is able to predict flight delays with a high degree of accuracy. The mean absolute error (MAE) is 8.9006, which means that the model's predictions are on average very close to the actual values. Finally, the R-squared score is 0.9508, which means that the model explains 95% of the variation in the flight delay data.

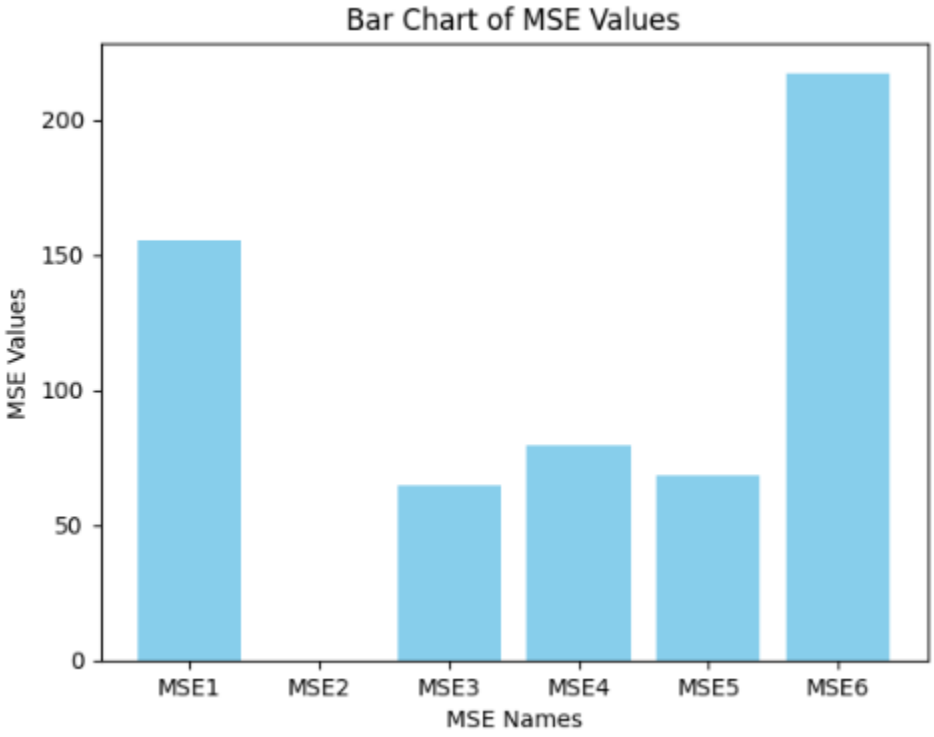
```
XGBoost Metrics:  Mean Square Error: 217.14981600307792, Root Mean Square Error: 14.73600407176511
Mean Absolute Error: 8.900649594335622, r2: 0.9508358434185243
```

Plot:



The plot of the predicted values versus the actual values shows that the model is able to capture the overall trend of the data very well. However, there are data points where the model's prediction is significantly different from the actual value. This might be due to several reasons: like aircraft doesnt have a clear runway i.e having traffic , technical glitch, heavy aircraft traffic at destination airports etc.

Comparison of MSE values of different models



The lower the MSE value, the better the model is at predicting the target variable. Therefore, MSE2 i.e Linear Regression model has the best performance followed by support vector machine

Phase #3

Objective:

To build the web application integrated with the designed ML model, so that users can do estimations on the Arrival delay based on their own data.

Model Selection:

In phase 2 we explored 6 models in total and tuned a few of them to check the variations. The details of tuned models are as discussed below.

In the **RandomForestRegressor** implementation, we have tuned a few hyperparameters: n_estimators: This parameter determines the number of trees in the forest. A higher number can lead to better performance but may increase computational cost. It has been set to 100, which provides a balance between performance and computational efficiency.

max_depth: Controls the maximum depth of each tree in the forest. Setting this can help prevent overfitting. With max_depth=10, the trees are limited to this depth, which balances model complexity and generalization. min_samples_split: Specifies the minimum number of samples required to split an internal node. Setting min_samples_split=4 ensures that a node won't split further if it contains fewer samples than this value, helping to control for overfitting.
These settings aim to strike a balance between model complexity and performance by limiting tree depth and the number of splits while ensuring a substantial number of trees for robustness. Adjusting these parameters can impact the model's predictive power and generalizability.

For **SVM Regressor**, a linear kernel is used with a regularization parameter that is kept low (1.0) to widen the margin and improve generalization. In SVM, the choice of kernel and tuning C and epsilon significantly impacts the model's ability to fit the data and its generalizability.

In **Gradient Boosting Regressor** n_estimators is set to 100, learning_rate is set to 0.1 to balance between convergence and precision, max_depth is set to 3, so that the individual trees in the ensemble are constrained to a moderate depth, aiding in controlling model complexity.

In **XG Boost model** colsample_bytree=0.3 and alpha=10 for feature and weight regularization, max_depth=5 to control tree depth, learning_rate=0.1 balances model convergence and generalization. These settings aim to optimize the XGBoost model for regression, controlling overfitting through regularization, tree depth, and balancing learning rate for accurate predictions.

All those 6 models and the results are analyzed as in the below table.

Evaluation metrics considered are: Mean Square Error(MSE), Root Mean Squared Error(RMSE), R2, Mean Absolute Error(MAE)

Sl.No	**ML Algorithm**	**MSE**	**RMSE**	**R2**	**MAE**
1	Random Forest Regression	155.327	12.463	0.96	8.330
2	Linear Regression	0.037569	0.1938	0.999	0.0486
3	Support vector Regression	64.41996	8.0262	0.9854	2.8262
4	Decision tree Regressor	79.4868	8.9155	0.9820	4.369
5	Gradient Boosting Regressor	68.14	8.25	0.9845	5.39
6	XG Boost	217.1498	14.7360	0.9508	8.9006

Based on the provided evaluation metrics, Linear Regression is the optimal choice for our flight delay prediction model. Its high R2 score of 0.999 showcases a good level of variance explanation, signifying an efficient fit to the data compared to other models. Moreover, low Mean Absolute Error (MAE) of 0.0486 signifies its precision in predicting the actual arrival delays, outperforming all other models significantly. Both R2 and MAE metrics, demonstrates its capability to predict flight delays with good accuracy and minimal error. Despite the complexity of other models like Random Forest, Gradient Boosting, or XGBoost, Linear Regression performs better by providing comparatively good accuracy with good R2 and exceptionally low MAE. The negligible MSE and RMSE values reinforce the model's precision in minimizing errors when predicting flight delay values. Given the massive datasets in airline operations, Linear Regression's computational efficiency allows us to handle this volume well, enabling faster training and prediction.

Integrating model with the application:

We are reusing the Linear Regression model developed in Phase 2. Since we performed ordinal encoding on certain fields during that phase, we have now mapped the data back to its original names to enhance the user interface. The final pickle file is saved.

```
import pickle
with open('linear_regression_model_new.pkl', 'wb') as file:
    pickle.dump(lin_model, file)
```

We are loading a Linear Regression model from a pickled file and exposing a /predict endpoint to receive POST requests with input data. Using the input data we are performing a prediction using the loaded model, and return the prediction as JSON. Then the model will respond to the interface with the predicted delay value. The code used for integrating model with the application is as below:

```
from flask import Flask, request, jsonify
import pickle
import numpy as np

app = Flask(__name__)

with open('linear_regression_model_new.pkl', 'rb') as file:
    model = pickle.load(file)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    input_data = np.array(data['input']).reshape(1, -1)
    prediction = model.predict(input_data)
    return jsonify({'prediction': prediction.tolist()})


if __name__ == '__main__':
    app.run(debug=True)
```

Data Product:

We have designed a web application that supports user interactions.

Instructions on How to get the product run in the local machine:

1. Install flask: In the project folder install flask using the command ***pip install flask*** as shown in the picture below

```
PS C:\Users\indus\Desktop\indushre_drangapu_aghanta_phase3\src\phase3> pip install flask
Requirement already satisfied: flask in c:\users\indus\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: Werkzeug>=2.2.2 in c:\users\indus\anaconda3\lib\site-packages (from flask) (2.2.3)
Requirement already satisfied: Jinja2>=3.0 in c:\users\indus\anaconda3\lib\site-packages (from flask) (3.1.2)
Requirement already satisfied: itsdangerous>=2.0 in c:\users\indus\anaconda3\lib\site-packages (from flask) (2.0.1)
Requirement already satisfied: click>=8.0 in c:\users\indus\anaconda3\lib\site-packages (from flask) (8.0.4)
Requirement already satisfied: colorama in c:\users\indus\anaconda3\lib\site-packages (from click>=8.0->flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\indus\anaconda3\lib\site-packages (from Jinja2>=3.0->flask) (
2.1.1)
PS C:\Users\indus\Desktop\indushre_drangapu_aghanta_phase3\src\phase3>
```

2. Install streamlit: In the project folder install streamlit using the command ***pip install streamlit*** as shown in the picture below

```
PS C:\Users\indus\Desktop\indushre_drangapu_aghanta_phase3\src\phase3> pip install streamlit
Requirement already satisfied: streamlit in c:\users\indus\anaconda3\lib\site-packages (1.29.0)
Requirement already satisfied: altair<6,>=4.0 in c:\users\indus\anaconda3\lib\site-packages (from streamlit) (5.2.0)
Requirement already satisfied: blinker<2,>=1.0.0 in c:\users\indus\anaconda3\lib\site-packages (from streamlit) (1.7.0)
Requirement already satisfied: cachetools<6,>=4.0 in c:\users\indus\anaconda3\lib\site-packages (from streamlit) (5.3.2)
Requirement already satisfied: click<9,>=7.0 in c:\users\indus\anaconda3\lib\site-packages (from streamlit) (8.0.4)
Requirement already satisfied: importlib-metadata<7,>=1.4 in c:\users\indus\anaconda3\lib\site-packages (from streamlit)
 (6.0.0)
```

3. Run the flask using the command: ***flask run***

```
PS C:\Users\indus\Desktop\indushre_drangapu_aghanta_phase3\src\phase3> flask run
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

4. Run the file predict.py using the command: ***streamlit run predict.py***

```
PS C:\Users\indus\Desktop\indushre_drangapu_aghanta_phase3\src\phase3> streamlit run predict.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.234:8501
```

5. In the local browser use the url: http://localhost:8501/ to access the application.

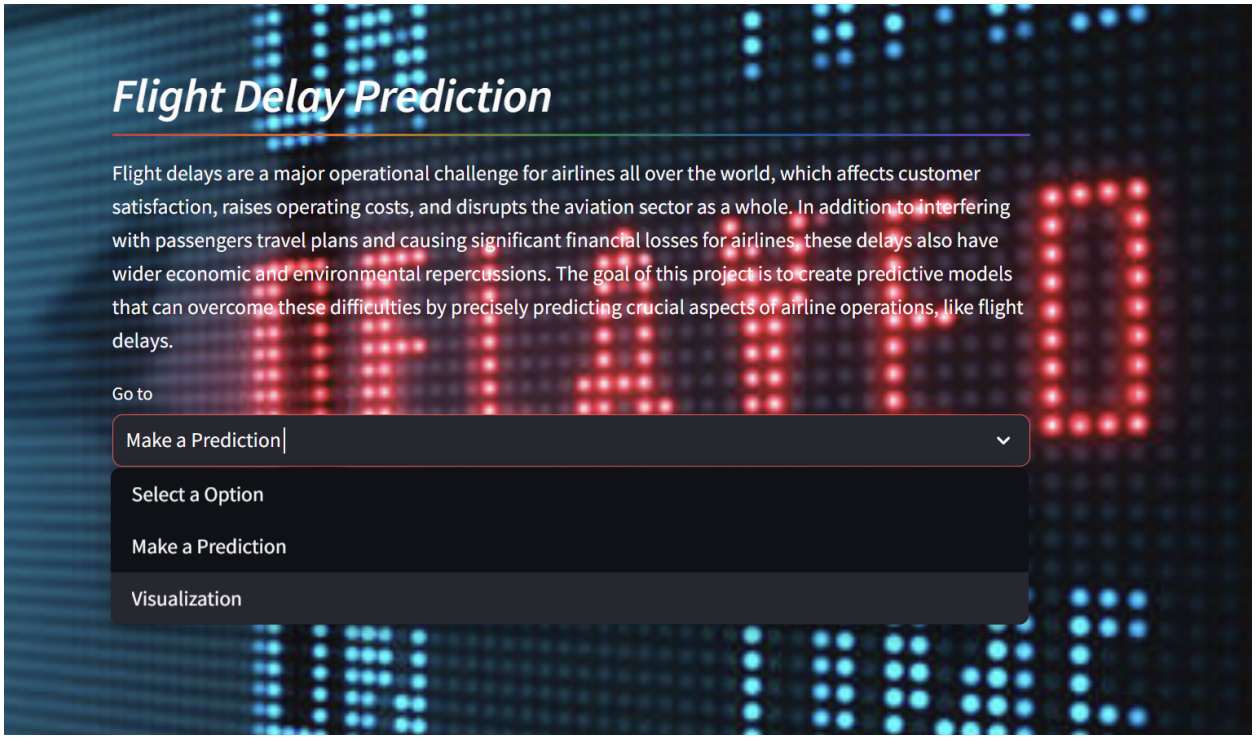
Web application provides 2 features:

1. Users can use their own data and input that into the respective fields provided in the web application to get the prediction of the arrival delay.
2. Can have a look at the dashboard that contains different plots and visualizations, which users can use to look at how the trends with respect to delays have been.

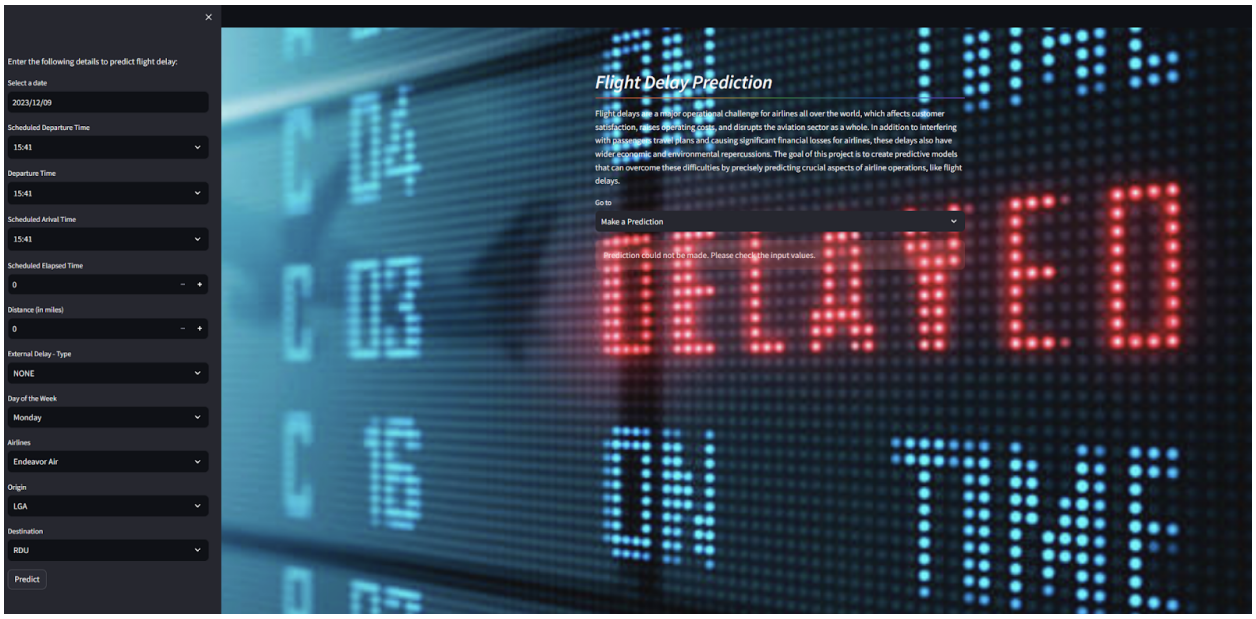
1. User interface: While the designed Linear regression model takes way more input parameters because of the one hot encoding employed on the data for feeding the model, the user interface has been made very simple.

The screenshot of the Home page of designed website is provided below:

In the Home page users can choose an option to either get predictions for their own data or have a look at the previous trends visualizations.



Once user selects an option ‘Make a Prediction’ the Interface looks like:



Here in the respective fields users can input their data, most of which can be found on the Airline ticket. Provided fields for user input are:

Day of the month: User can chose the date from the calendar

Scheduled Departure Time: User should input the scheduled departure time in the 24 hour format.

Departure Time: User should input the departure time in the 24 hour time format.

Scheduled Arrival Time: User should input the scheduled arrival time in the 24 hour format.

Journey time in minutes: Scheduled journey time has already been calculated in the back end based on scheduled departure and arrival time. But if the user wishes to get some estimation based on journey time this field can be used, otherwise the default value is considered based on the scheduled departure and arrival time.

DISTANCE: User must input the distance in kms

External Delay & External Delay - Time: - Type: External factors can be the reason for higher delays most of the time, hence users can select the delay from the options provided and if selected has to provide the delay time that the selected factor might cause. Otherwise the default values will be None and 0

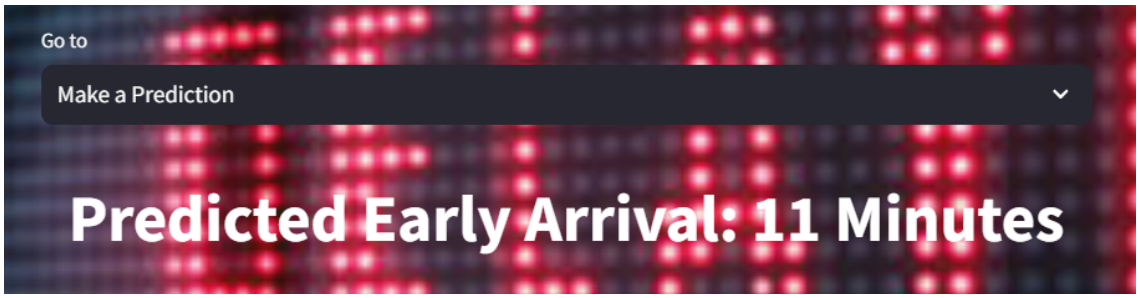
Day of the Week: User has to choose the day from the drop down.

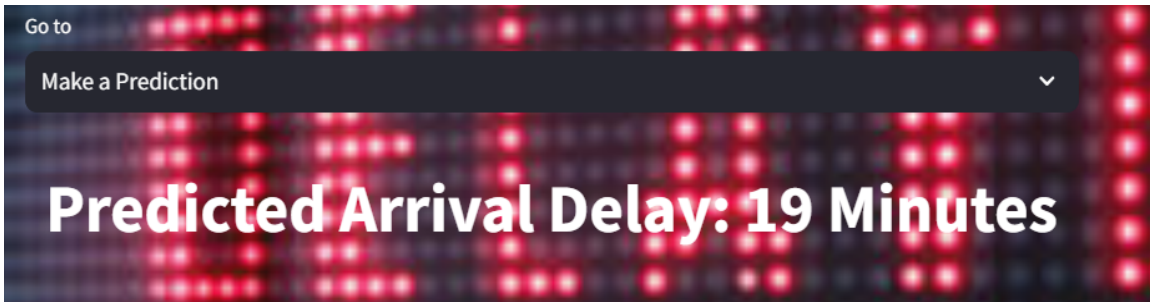
Airlines: Users are provided with a bunch of AIrline options to choose from.

Origin: User has to select the origin airport from the options provided in the drop down.

Destination: User has to select the Destination Airport from the options provided in the drop down.

Finally, after providing values when user clicks on predict, the prediction is displayed on the screen as below:





2. When user selects Visualization from the drop down:

When users opt for visualization a dashboard appears on the screen which shows the cumulative history where user can see the trends in delays as:

1. Which Airports record more departure delays.
2. Days on which most delays occur.
3. Average delays for each Aircrafts.
4. At what time of the day most delays occur.

Web pages reflecting all these details are as provided in the below screenshot.

Recommendations related to our problem statement based on your analysis:

For starters, for airlines, our predictive model can be a game changer in operational planning. Airlines can optimize resource allocation, such as crew scheduling and aircraft maintenance, by accurately forecasting delays. This proactive approach can result in significant cost savings and efficiency improvements. Furthermore, by understanding the patterns and causes of delays, airlines can implement strategic changes to mitigate these issues. This could include revising flight schedules, improving maintenance protocols, or improving crew management.

Passengers will also benefit significantly from our project. They can make informed travel decisions with access to reliable delay predictions, reducing the stress and inconvenience frequently associated with unexpected delays. This information is especially useful for connecting flights, arranging ground transportation, and managing time-sensitive commitments at the destination.

From a regulatory standpoint, our model helps to ensure compliance with aviation standards and avoid penalties associated with delays. Airlines can use the information to improve punctuality and reliability, which are frequently scrutinized by regulatory bodies.

Our web application, a key project output, serves as a useful tool for both airlines and passengers. It is a resource for airlines to monitor and analyze delay trends, enabling data-driven decision-making. The app provides a user-friendly platform for passengers to check potential delays, enhancing their travel planning experience.

In summary, our project not only addresses the immediate issue of flight delay prediction, but also offers broader solutions for improving airline operations and passenger experiences.

How does this product helps users:

Users can learn more about the causes of delays in aircraft arrival times. This involves being aware of the airlines, routes, times of day, and even days of the week that frequently encounter delays. With this information, consumers may prepare for any flight delays.
With the expected delay information, users can make more efficient travel plans. To alleviate the anxiety brought on by erratic arrival times, they can choose to change their itinerary, take a different aircraft, or take a different route.

Potential Extensions:

Training can be extended over multiple years to capture long-term trends and seasonal shifts, similar to how we understand changing patterns over time. Integrating weather data based on user location could improve predictions, similar to how we consider the impact of weather on our plans. Regular updates or real-time data integration would ensure that predictions remained current, much like keeping up with the latest news. A redesigned user interface with interactive elements and live updates could provide users with a more engaging experience, such as map exploration or real-time flight status updates.

References

1. *https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr*
2. *https://www.revfine.com/airline-industry/#:~:text=The%20airline%20industry%20forms%20a,flight%20attendants%2C%20and%20ground%20crew*.
3. *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*
4. *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html*
5. *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html*
6. *https://www.geeksforgeeks.org/major-kernel-functions-in-support-vector-machine-svm/*
7. *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html*
8. *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html*
9. *https://xgboost.readthedocs.io/en/stable/parameter.html*
10. *https://machinelearningmastery.com/regression-metrics-for-machine-learning/*
11. *https://discuss.streamlit.io/t/how-to-create-text-output-box/8227*
12. *https://docs.streamlit.io/library/api-reference/write-magic/st.write*
13. *https://discuss.streamlit.io/t/how-to-make-colored-box-in-streamlit/11092*
14. *https://docs.streamlit.io/library/api-reference/widgets/st.date_input*
15. *https://docs.streamlit.io/library/api-reference/widgets/st.time_input*
16. *https://discuss.streamlit.io/t/drop-down-menu/3180*
17. *https://discuss.streamlit.io/t/how-do-i-use-a-background-image-on-streamlit/5067*
18. *https://flask.palletsprojects.com/en/3.0.x/quickstart/#sessions*
19. *https://flask.palletsprojects.com/en/3.0.x/quickstart/#html-escaping*

Contribution:

Name:	**Contribution**
Deekshith Sagar Rangapuram - drangapu	33.3%
Indushree Byrareddy -indushre	33.3%
Akhil Chaitanya Ghanta - aghanta	33.3%