2020 TAMIDS Data Science Competition

Report Submission

Team: Data Hungry

Roline Stapny Saldanha
Vipul Tiwari
Manseerat Batra
Devi Sandeep

Introduction:

Flight delays have been one of the important problems in airport management and flight scheduling. Although some airports and/or airlines have put efforts in airport/airline management to reduce the possible delays, flight delays become unavoidable in some airports. In reality, multiple factors that impact flight delay are in many cases independent. Delay is caused by some of the external triggers, like weather and airport capacity. A comprehensive overview on the potential factors that influence flight delay has been given by Xu et al. [1], where more than 50 potential factors were identified based on a detailed airport analysis.

Dataset:

The U.S. Bureau of Transportation Statistics tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights is published in their monthly Air Travel Consumer Report. In this dataset, we are provided with the data for the years 2018 and for the Quarters 1 and 2 for the year 2019.

Following is the data provided:

FlightDelays.csv: 10,915,495 flight logs for 2018 and the first half of 2019

Airports.csv: Airport locations and codes for 362 passenger airports in the United States

Routes.csv: Distance for 6,684 airport routes with origins and destinations in the U.S.

Airfares.csv: The average airfares between cities for 2018 and the first half of 2019.

The airline data provided reports the causes of delay in broad categories such as Air Carrier, National Aviation System, Weather, Late-Arriving Aircraft and Security. Below are the categories and their detailed descriptions are per Bureau's report [2]

- **Air Carrier:** The cause of the cancellation or delay was due to circumstances within the airline's control (e.g. maintenance or crew problems, aircraft cleaning, baggage loading, fueling, etc.).
- **Extreme Weather:** Significant meteorological conditions (actual or forecasted) that, in the judgment of the carrier, delays or prevents the operation of a flight such as tornado, blizzard or hurricane.
- **National Aviation System (NAS):** Delays and cancellations attributable to the national aviation system that refer to a broad set of conditions, such as non-extreme weather conditions, airport operations, heavy traffic volume, and air traffic control.
- **Late-arriving aircraft:** A previous flight with the same aircraft arrived late, causing the present flight to depart late.
- **Security:** Delays or cancellations caused by evacuation of a terminal or concourse, re-boarding of aircraft because of security breach, inoperative screening equipment and/or long lines in excess of 29 minutes at screening areas.

Focusing on these delays, we see the following from the dataset given. From this, we can understand that the delays to air carriers and late aircraft are more significant since the mean and variance of the delays are higher. Also, to note is that the security delay is the least significant one. We will further show various analyses on these in the Data Visualization section.

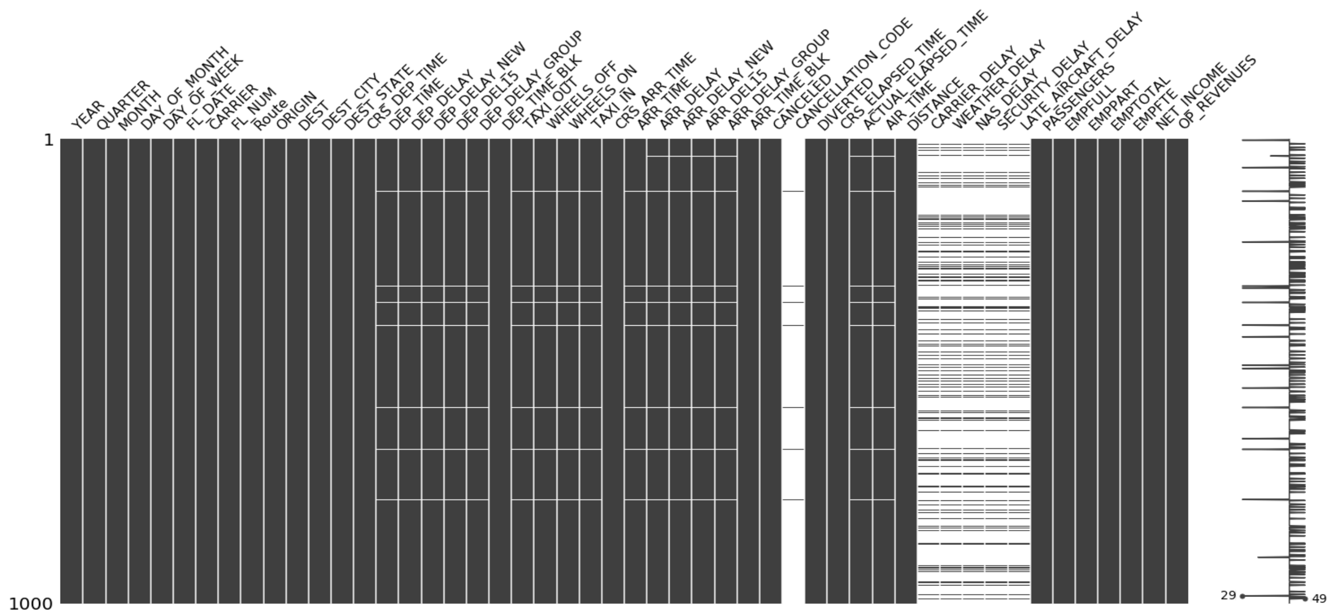
	CARRIER_DELAY	WEATHER_DELAY	NAS_DELAY	SECURITY_DELAY	LATE_AIRCRAFT_DELAY
count	2083263	2083263	2083263	2083263	2083263
mean	19.95377732	3.802843424	16.15089549	0.098475805	26.35087265
std	60.8905727	31.40153473	37.00454574	3.517429549	51.18834281
min	0	0	0	0	0
25%	0	0	0	0	0
50%	0	0	3	0	3
75%	17	0	20	0	32
max	2592	2692	1848	1078	2454

Libraries needed:

- Missingno [3] (For missing data visualizations)
- Basemap (For visualizations involving map)
- Pyproj
- Leaflet, shiny for R (for development of visualization platform)
- Plotly (visualization graphs)
- Scikit-learn (model building and tuning)

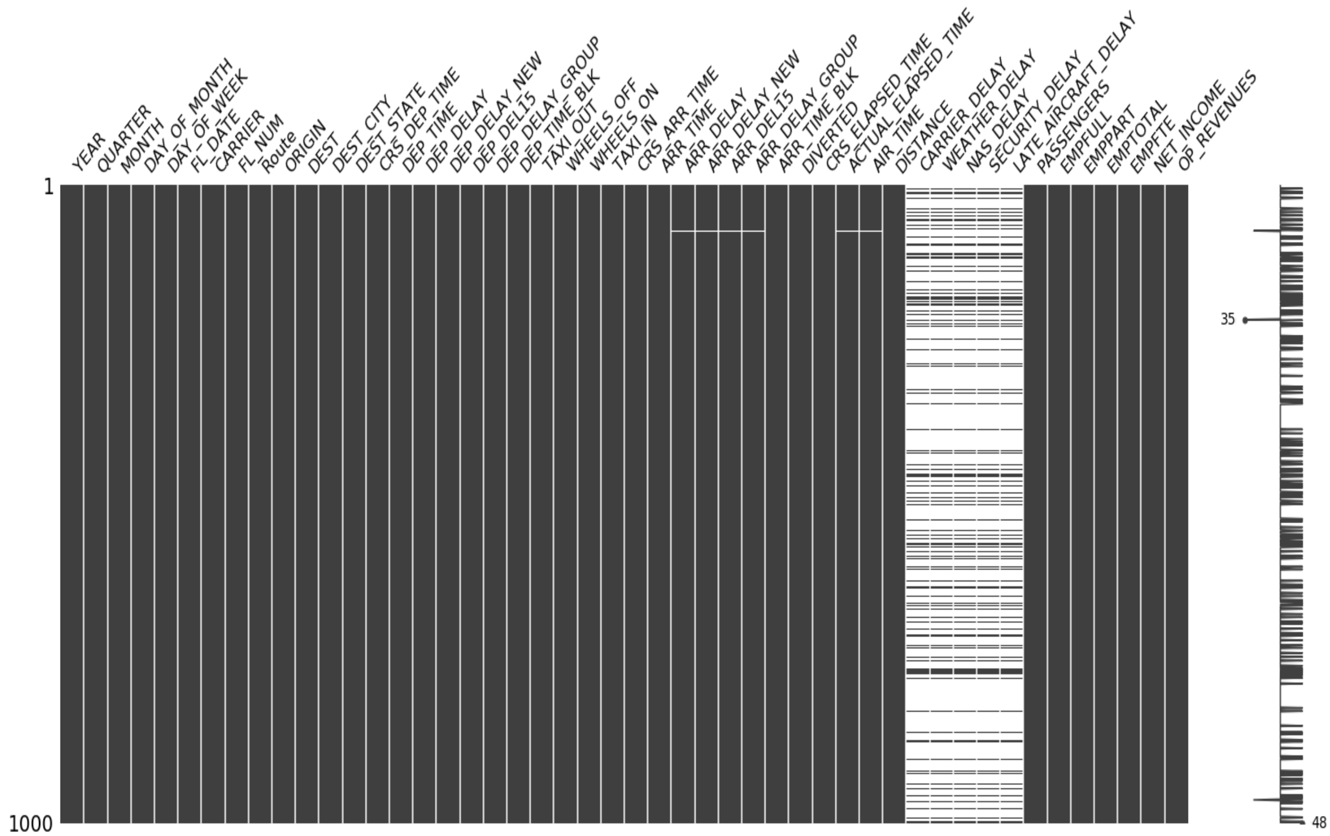
Data cleaning:

We used missingno python package to analyze the missing data. Below is the visualization for missing data in the dataset provided.



We see that most of the features are filled. There are very few missing data instances for some of the features. Also from the above visualization, we see that “CANCELLATION_CODE” is mostly empty, which seems appropriate and also suggests that only few of the flights are cancelled. For the ones which have the CANCELLATION_CODE, DEP_DELAY (Departure Delays) and ARR_DELAY (Arrival Delays) are not applicable. Therefore, these cancelled flights are not really helpful in flight delay analysis, and can be removed.

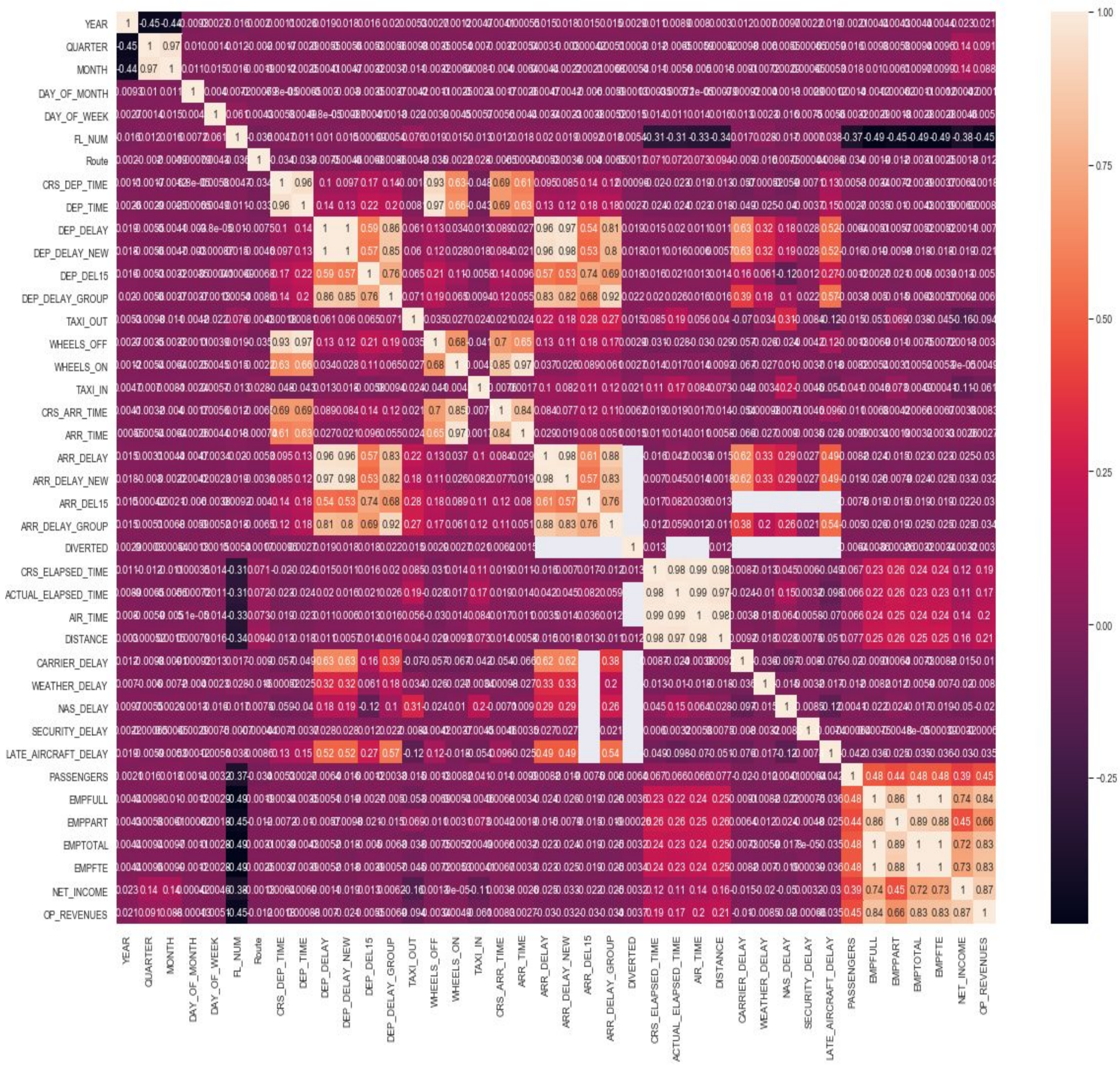
After dropping CANCELED and CANCELLATION_CODE, we get the below visualization. Now we see that mostly the following columns are sparse (CARRIER_DELAY, WEATHER_DELAY, NAS_DELAY, SECURITY_DELAY, LATE_AIRCRAFT_DELAY).



Next, we check missing values for DEP_DELAY. This measures the Departure Delay of the flight. DEP_DELAY = DEP_TIME - CRS_DEP_TIME. We can see that in the data, whenever DEP_DELAY is empty (NaN), the DEP_TIME and CRS_DEP_TIME is the same i.e., there is no delay. Also note that none of DEP_TIME and CRS_DEP_TIME is empty, so we can safely change DEP_DELAY to 0 whenever it is empty.

Similar is the case for DEP_DELAY_NEW, DEP_DEL15 and DEP_DELAY_GROUP. All these features depend on DEP_DELAY. Since DEP_DELAY is being set to 0 now, all these features also will be 0.

Next, we draw the correlation heatmap to check the dependent (correlated) data which won’t be useful in building the model and that can be removed. Below is the initial correlation heatmap for the dataset. Following the plot, we have shared the data cleaning steps we have taken based on the correlation heatmap.



Correlation heatmap analysis:

- 'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY' are mostly empty, but we think these are the important features and play a major role in the prediction of the arrival/departure delays, hence we choose not to remove these features, but instead we intend to fill the missing values for these features.

- We can see that AIR_TIME, DISTANCE, CRS_ELAPSED_TIME and ACTUAL_ELAPSED_TIME are highly correlated. Therefore, we can have one of these features and drop the other correlated fields from the dataframe for the flight delay info. We have chosen to keep DISTANCE since the number of non-null values for this feature are less in the dataset.

- Similarly, Intuitively as well, EMPTOTAL (Total number of Employees) is correlated to EMPFULL, EMPPART and EMPFTE. Hence, we can drop these other features and use only EMPTOTAL.
- ARR_DELAY is correlated to ARR_DELAY_GROUP and ARR_DELAY_LABEL. This is also easily understandable since Delay groups and delay labels are calculated based on the Arrival Delay (ARR_DELAY), these features are heavily dependent on ARR_DELAY and can be removed. Similar is the case for departure delays i.e., DEP_DELAY is correlated to DEP_DELAY_GROUP and DEP_DELAY_LABEL.

- ARR_TIME is correlated to CRS_ARR_TIME and DEP_TIME is correlated to CRS_DEP_TIME. This implies that the actual Arrival time is dependent on scheduled arrival time, which is true.

- WHEELS_ON is correlated to ARR_TIME and WHEELS_OFF is correlated to DEP_TIME. Since, there exists dependent equations between WHEELS_ON and ARR_TIME, it is quite obvious that these two are dependent features and one of them can be removed.

- From the heatmap, we see that EMP_TOTAL is correlated to NET_INCOME and OP_REVENUES. Hence, NET_INCOME and OP_REVENUES are removed.

- ARR_DEL15 is correlated to DELAY_LEVEL. Hence, only DELAY_LEVEL is kept.

- We saved the best observation for the last! We observe that the Arrival delays are dependent on departure delays i.e., if the flights are delayed in their departure by 30 min, these flights try to maintain the journey time. Therefore, there would be arrival delays of the same magnitude or less for these flights. We can see that from the given dataset as well, and is inferred from the heatmap that the arrival delays are heavily correlated to departure delays. Therefore, we can remove all the departure delay related features from the dataset.

Other data cleaning steps:

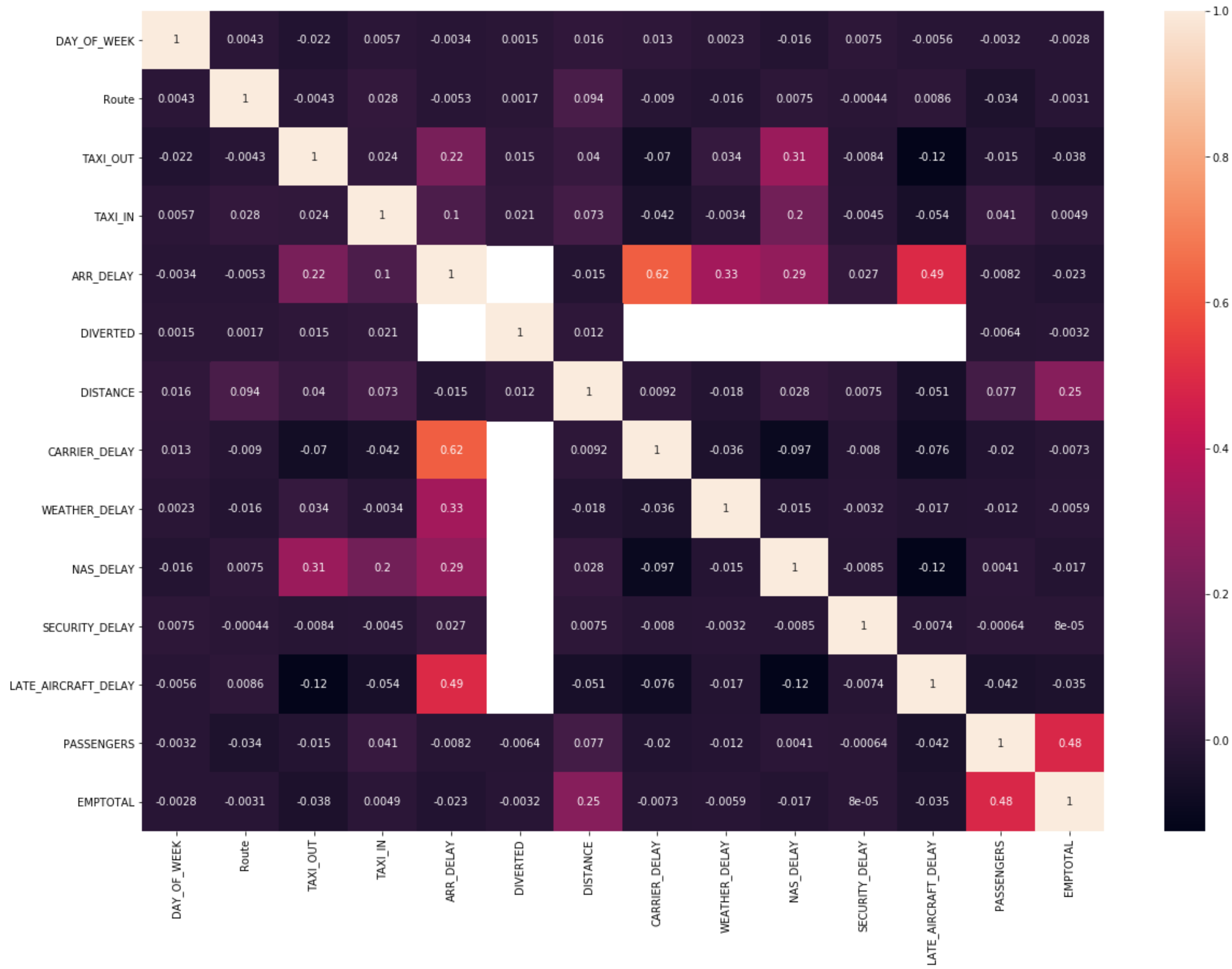
- We removed other features that seemed unnecessary for building the data models. These are the ones that are removed: 'ARR_TIME_BLK', 'DEP_TIME_BLK', 'YEAR', 'QUARTER', 'MONTH', 'DAY_OF_MONTH', 'FL_NUM', 'DEST_CITY', 'DEST_STATE'

- For the sake of easy understanding, visualizations and to fill the empty values in the arrival and departure times, we have changed the format of the ARR_TIME and other related features to date time format. I.e., if the value in the ARR_TIME is 700, then the corresponding datetime is 7:00, likewise if the value in the ARR_TIME is 1553, then the corresponding datetime is 15:53.

- Finally, we removed the entries which have null inputs from any of the features.

- As a final step, we have imputed 0 value to the 5 different categories of delays whenever these are empty.

Here is the correlation matrix after cleaning the data.



We can see that none of the features have high correlation values (greater than 0.7-0.8), therefore these all are kept and this cleaned data is converted to a csv file to be further used for building data models.

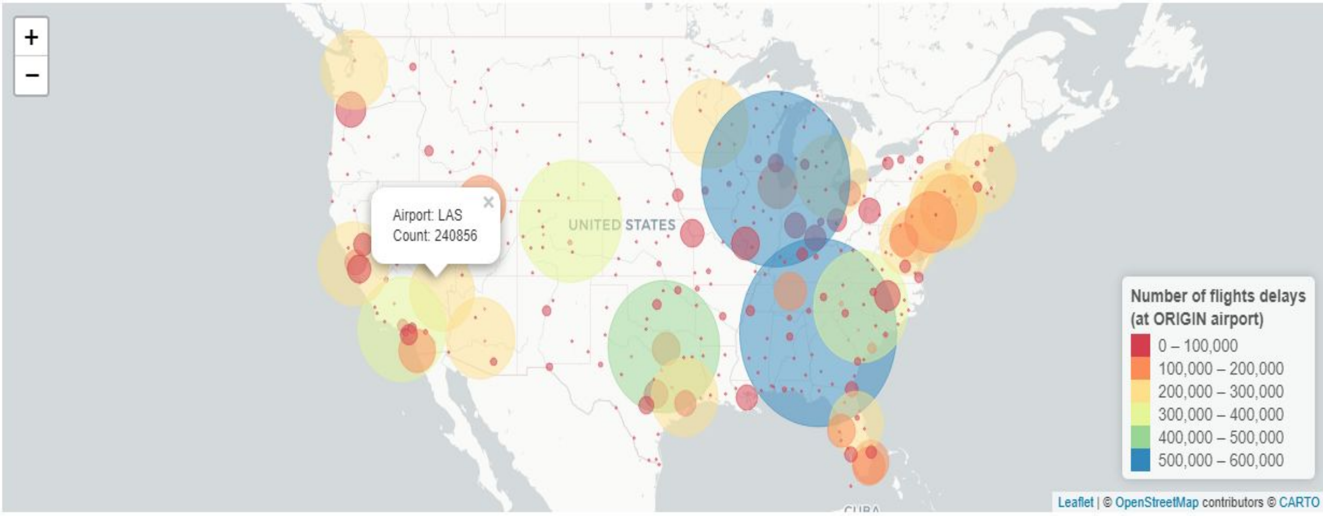
Exploratory data analysis:

Additional Data:

Additional data related to airports is downloaded from https://openflights.org/data.html to get the latitude and longitude information. The given dataset doesn't have latitude and longitude information, so we used this additional data to get this info to use in our visualizations.

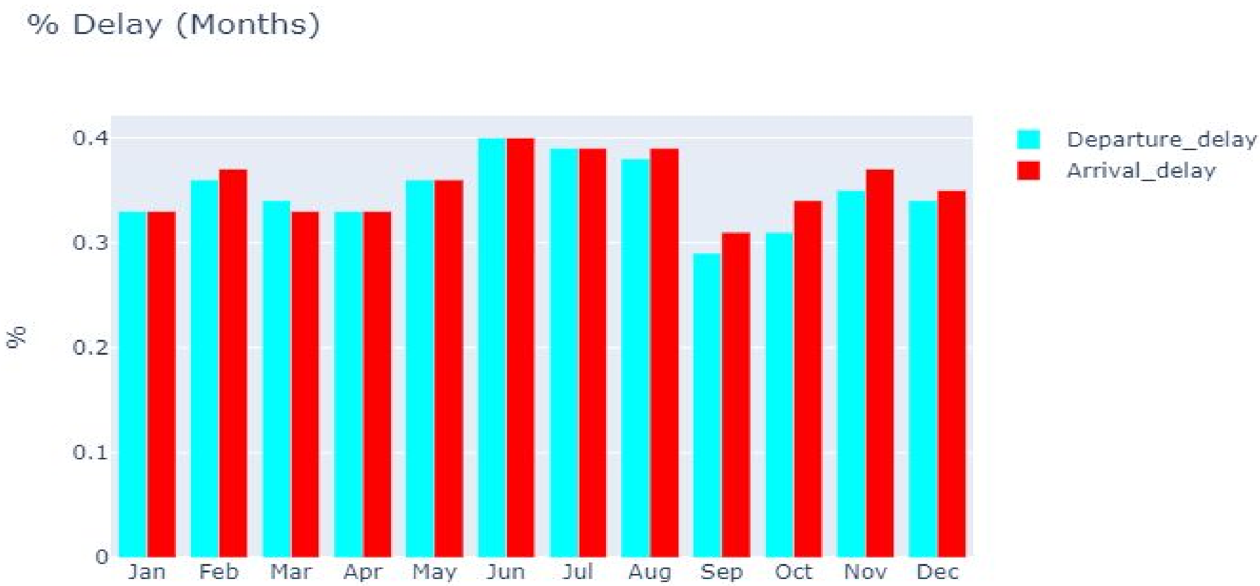
Visualizations and Analysis:

Below here is the demographic showing the total number of flight delays at the ORIGIN airport on the whole dataset and their magnitude.

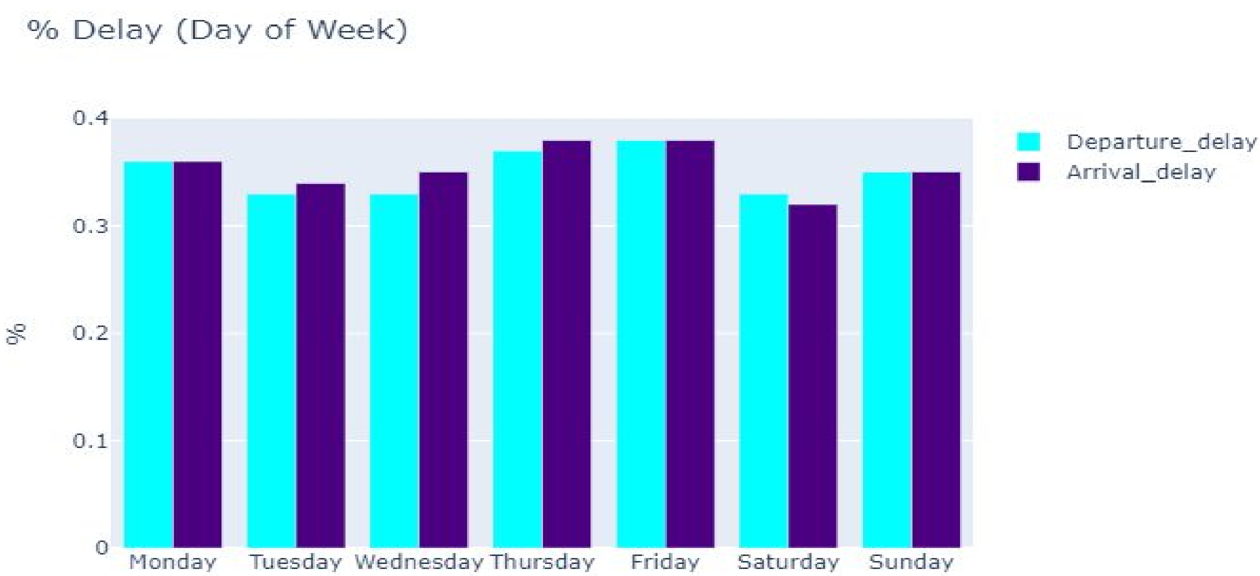


Then we went on to analyze the total number of flight delays per quarter. We see that there is consistency in the number of flight delays observed each quarter i.e., we see over 1.5 million flight delays every quarter. Also, we see that the number of delays tend to increase in Quarter 2 and Quarter 3. And they seem to be slightly less in Quarter 1 and 4.

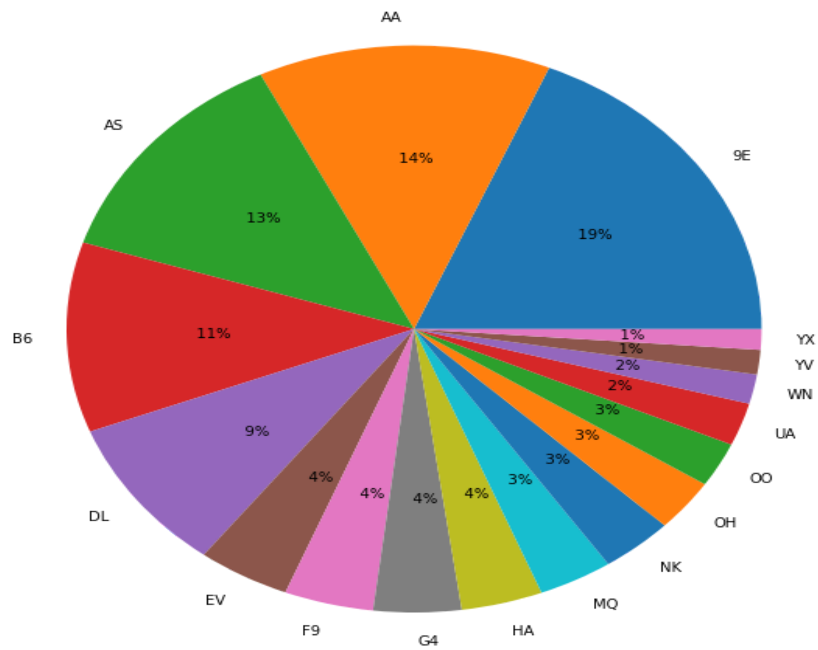
Next, we analyze the flight delays per month. Once again, we see the consistency that almost all months have approximate 0.3% of delays compared to the total flights scheduled. We also observe that the months of June, July have the highest number of flight delays compared to the other months. This may be due to the increase in the total number of flights scheduled leading to more delays.



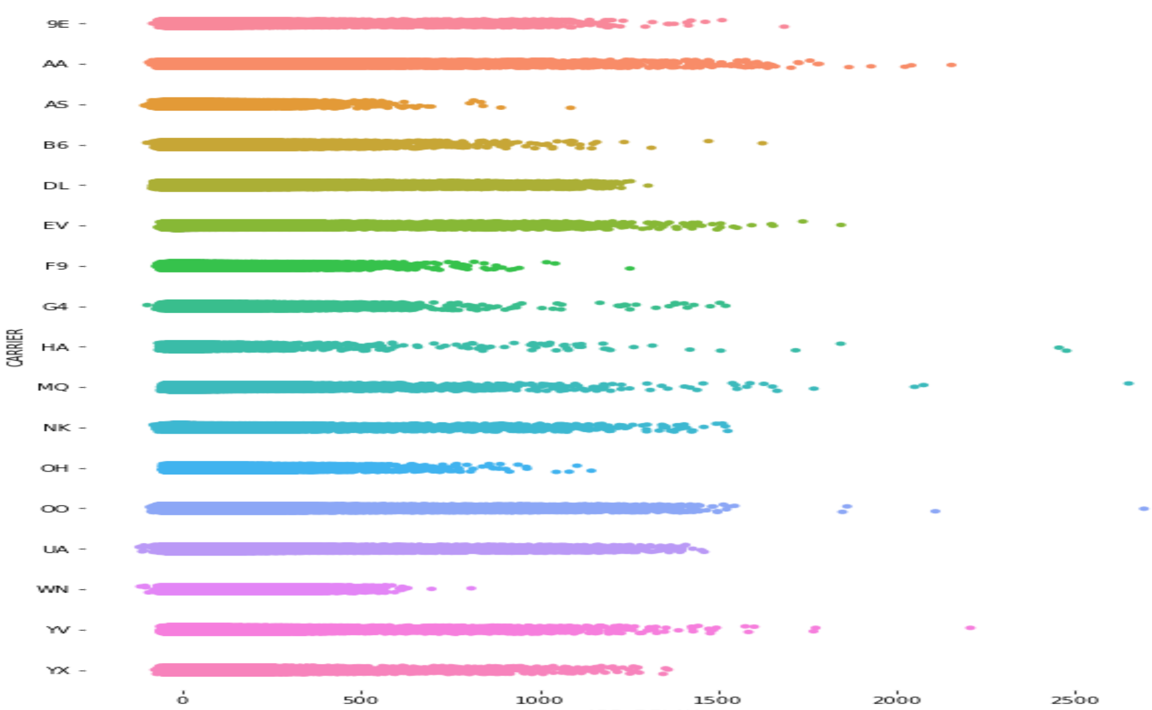
Next, we tried to analyze the flight delay data based on the day of the week. We surprisingly see that the delays on the weekdays are higher (on Thursday and Friday) compared to the delays on the weekend (Saturday and Sunday).



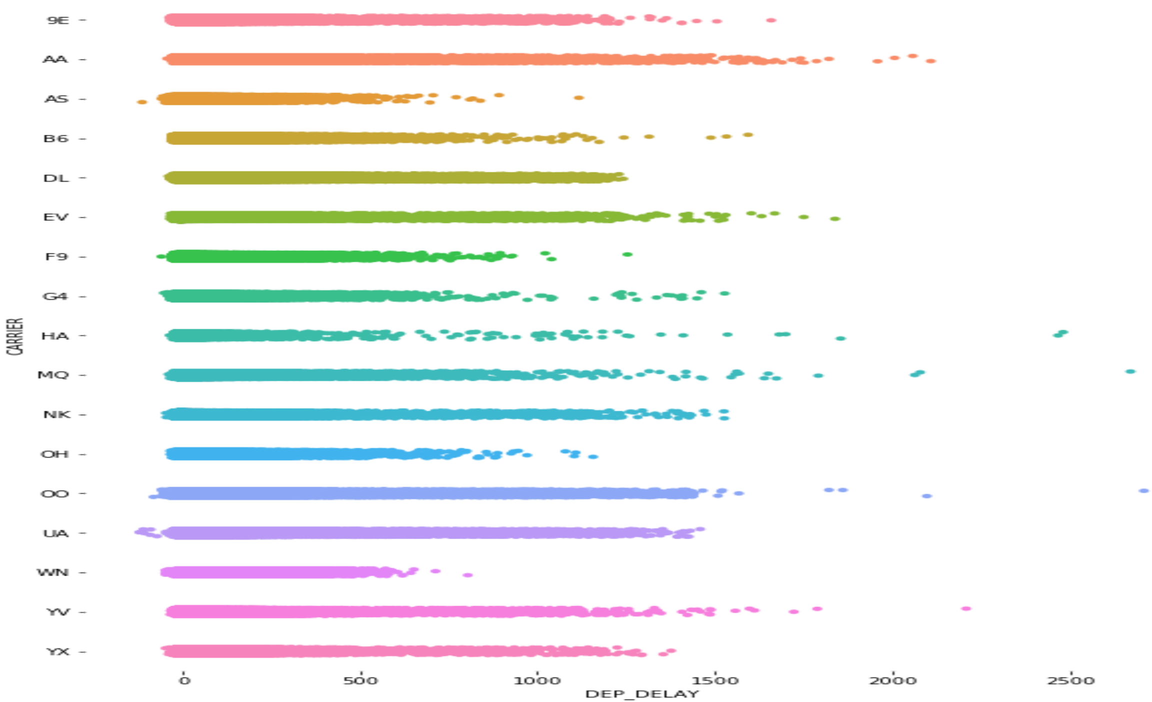
Next, we shifted our focus to analyze the arrival and departure delays with respect to the airlines. We calculated the total flights per each airline and shown them in a pie chart as below:



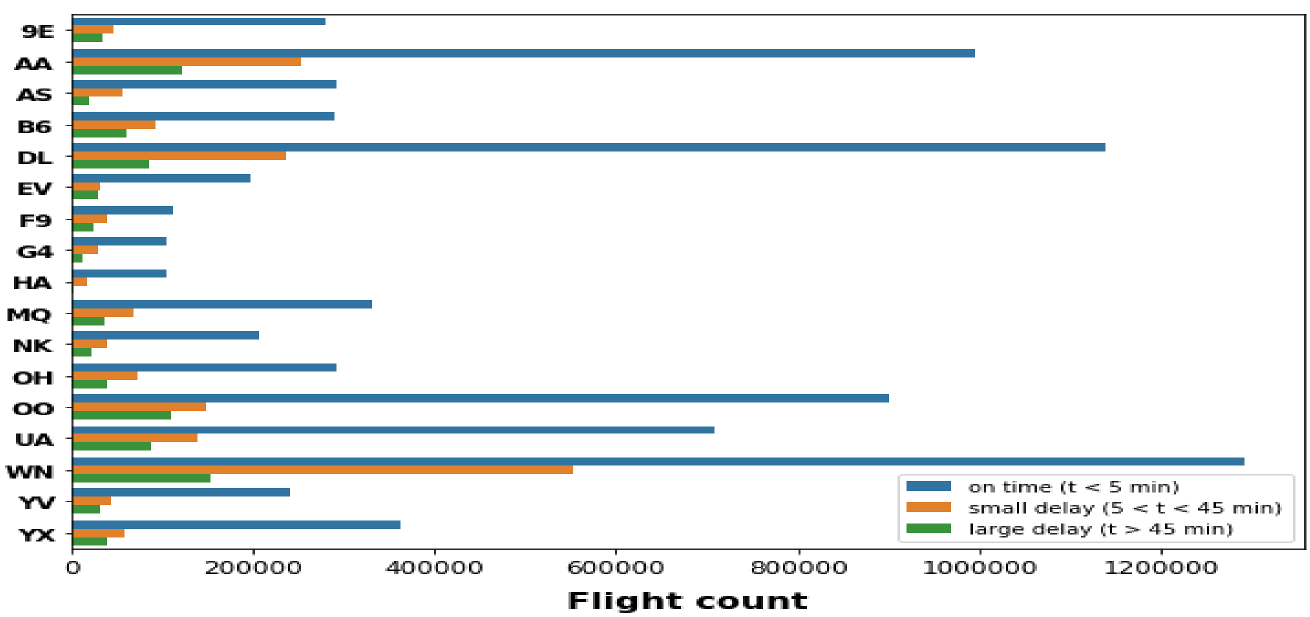
We then created a stripplot for the arrival and departure delays with respect to the airlines. We see that AA (American Airlines) has large delays (delays more than 1500). We also did the same exercise with the departure delays and we see the same trend. In this stripplot as well, we see that the AA (American Airlines) has large delays (delays more than 1500).



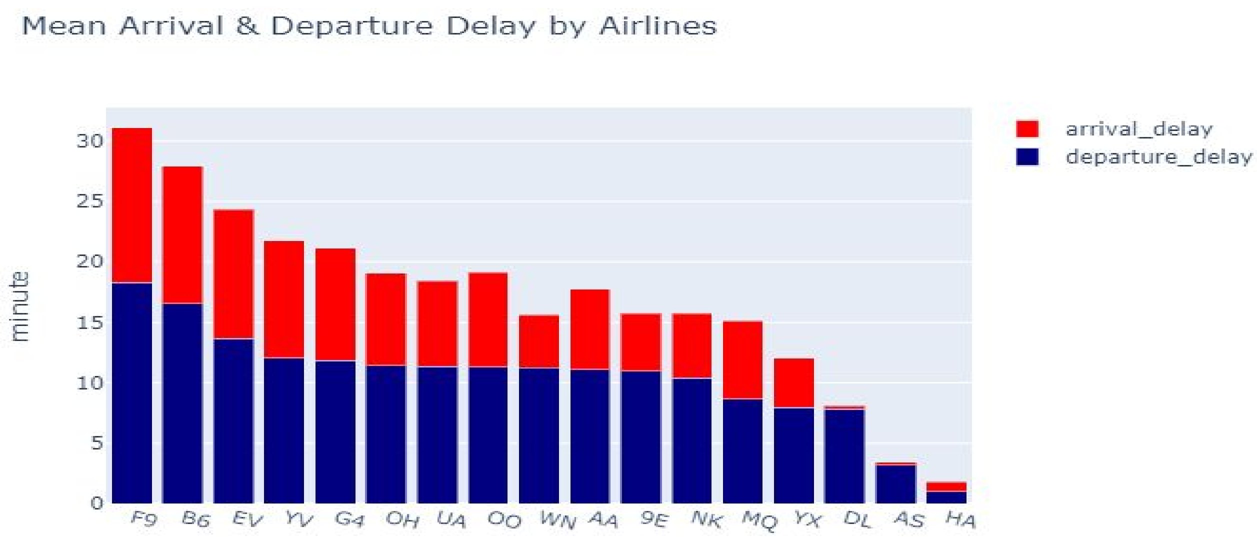
Below shows the visualization for the departure delays by airlines.



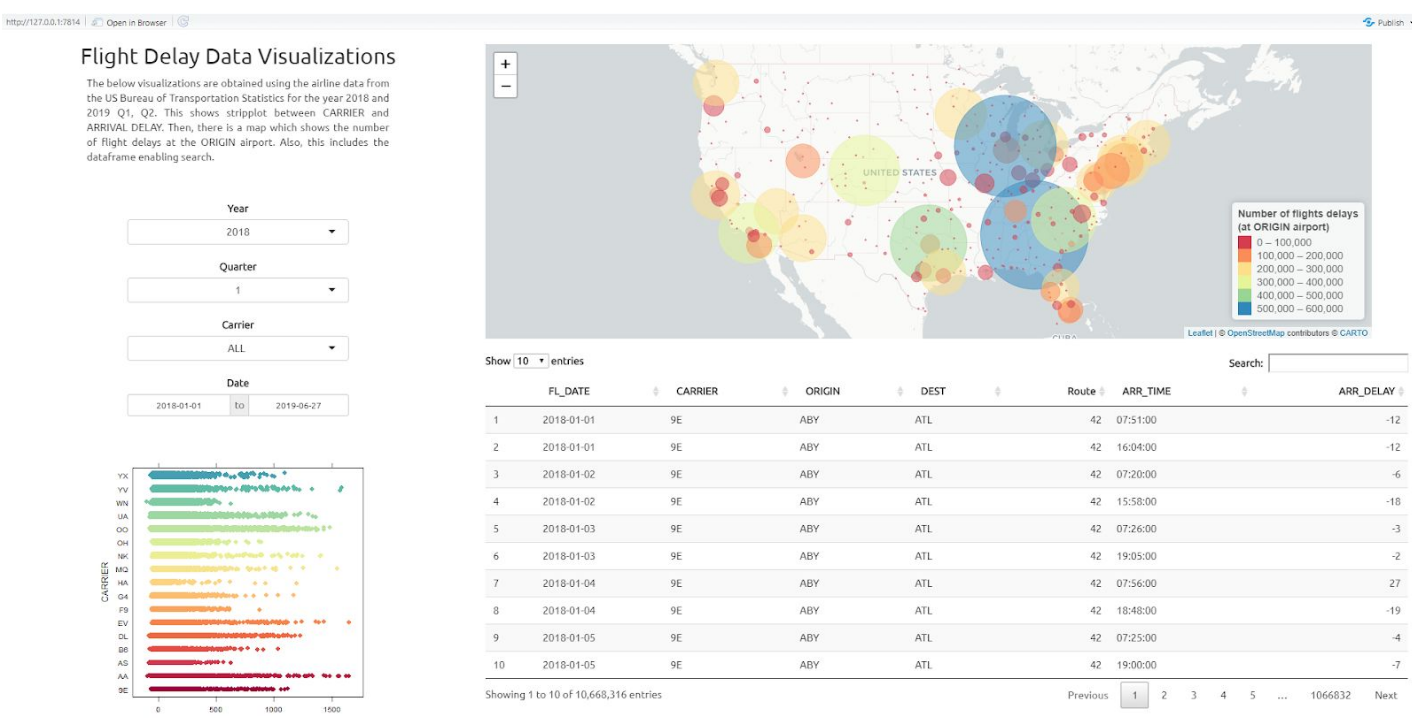
Based on the above stripplots, we decided to divide the delays into 3 subgroups (small, medium and large) - Delays with less than 5 minutes are considered as smaller delays. Whereas, delays in between 5 to 45 minutes are considered as medium delays and delays with more than 45 minutes are considered as large delays. Below is the visualization, and we once again see that AA (American Airlines) is one of the airlines which has significant larger delays.



Then, we calculated the mean arrival and departure delays by airlines. Below is the plot for the same.



We decided to create an interactive map that would allow users to interact with the data. Also, we wanted to include different visualizations as an application dashboard. Leaflet is probably the most widely known Javascript library for interactive mapping (used by the New York Times and the Washington Post) and there is an R package, leaflet [4], which makes it easy to integrate and control Leaflet maps in R. By building a leaflet map in R, we were able to create it as a Shiny web app. Below is the screenshot of our application.



Model fitting and results:

The data was preprocessed to transform the columns 'CARRIER' 'ORIGIN' and 'DEST' into integers using the LabelEncoder() class of sklearn package. The extreme delays (>300 minutes and <-100 minutes) present in the dataset were removed. This type of delay is however marginal (a few %) and the cause of these delays is probably linked to unpredictable events (weather, breakdown, accident, ...). Taking into account a delay of this type will likely introduce a bias in the analysis. Moreover, the weight taken by large values will be significant if we have small statistics.

After removing the correlated features, 19 features were retained in the dataset.

	#Column	**Dtype**
0	DAY_OF_WEEK	int64
1	FL_DATE	object
2	CARRIER	int64
3	Route	int64
4	ORIGIN	int64
5	DEST	int64
6	TAXI_OUT	float64
7	TAXI_IN	float64
8	ARR_TIME	object
9	ARR_DELAY	float64
10	DIVERTED	int64
11	DISTANCE	int64
12	CARRIER_DELAY	float64
13	WEATHER_DELAY	float64
14	NAS_DELAY	float64
15	SECURITY_DELAY	float64
16	LATE_AIRCRAFT_DELAY	float64
17	PASSENGERS	float64
18	EMPTOTAL	float64

Dataset was split into train and test datasets using the `'FL_DATE'` feature. Flights on or before `'2019-03-31'` were considered as train data and flights after `'2019-03-31'` were considered as test data. After preprocessing we got the following.

```
Shape of train Data:  (8800622, 18)
Shape of test Data:  (1831883, 18)
```

`'ARR_DELAY'` feature represents the arrival delay of the flight. This is considered as the target value. Y = `'ARR_DELAY'. 16 columns namely 'DAY_OF_WEEK', 'CARRIER', 'Route', 'ORIGIN', 'DEST', 'TAXI_OUT','TAXI_IN', 'DIVERTED', 'DISTANCE', 'CARRIER_DELAY', 'WEATHER_DELAY','NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'PASSENGERS','EMPTOTAL'` are taken as features for the model training. The 'FL_DATE' column is not considered for training and is just used to split the dataset.

The features are standardized by removing the mean and scaling to unit variance.

Models used:

1. **Linear Regression** : LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

2. **Lasso** : Lasso regression, or the Least Absolute Shrinkage and Selection Operator, is a modification of linear regression. In Lasso, the loss function is modified to minimize the complexity of the model by limiting the sum of the absolute values of the model coefficients (also called the l1-norm).

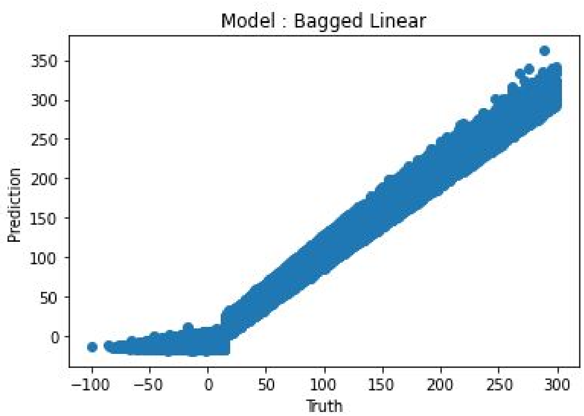
3. **Ridge** : This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multivariate regression.

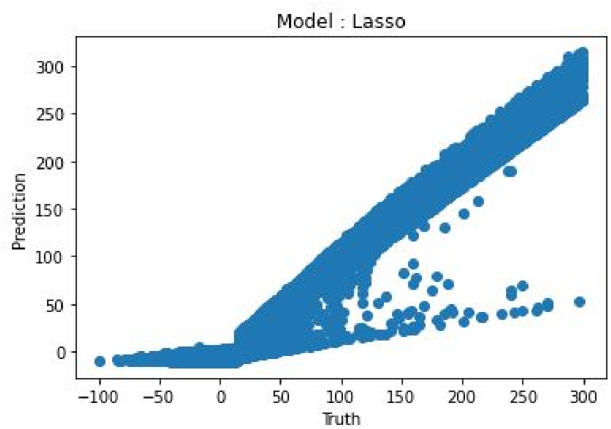
4. **Bagging Regressor**: A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. Bagged Linear took significantly more time to train than other algorithms.

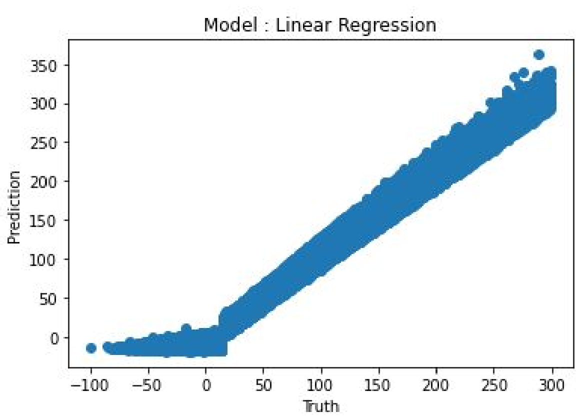
Model Analysis:

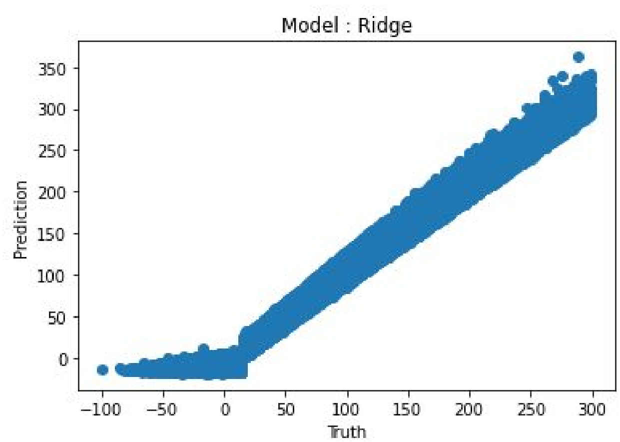
Model Name	Mean Absolute Error	Mean Squared Error	Root Mean Squared Error	R2 Score
Linear Regression	7.56	92.016	9.592	0.939
Lasso	7.96	99.576	9.978	0.934
Ridge	7.56	92.016	9.592	0.939
Bagged Linear	7.560	92.016	9.5925	0.9397

The following graphs plot the actual target values vs the model outputs.







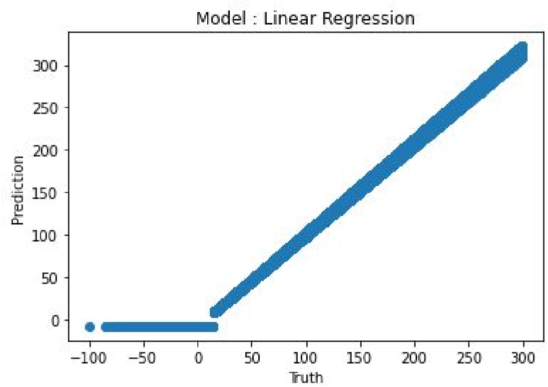


From results of Linear Regression, Ridge and Bagged Linear models were very similar. In the Lasso model there were higher mis predicted values as seen in the graph. All the models did quite well in predicting positive delays. However, if the Airline arrived early the models predicted 0 delay. This contributed to the error values. Overall, the models we developed were good at predicting airline delays with small errors.

Exploring other combinations of Features:

1. Considering only delay features:
   Here were considered only 5 features namely 'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY' to represent the data. These 5 features did a decent job in prediction. The results are shown below.

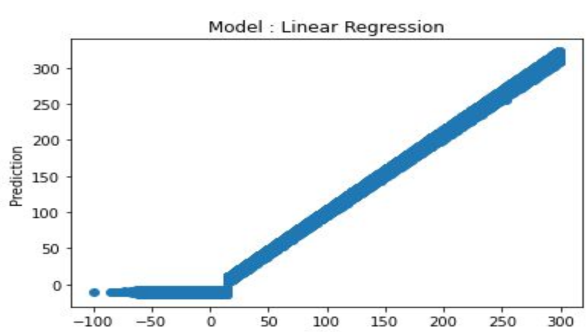
```
Linear Regression
Mean Absolute Error: 7.877121408230192
Mean Squared Error: 99.44989975903798
Root Mean Squared Error: 9.97245705726718
R2 :  0.9349100484762646
```



   From the graph we can see that with just 5 features, the model has learnt to predict the delays. However the model was not able to predict negative delays if the flight early.

2. By considering `'CARRIER'` and `'DISTANCE'` features along with the above delay features:
   Total Features: 7, `'CARRIER', 'DISTANCE', 'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY','SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY'` were considered to represent the model. Model showed slight improvements.

```
Results:
Mean Absolute Error: 7.891450692616473
Mean Squared Error: 99.2117828135632
Root Mean Squared Error: 9.960511172302514
R2 :  0.9350658959982378
```



From the graph we can see that, by adding distance and Carrier feature the abrupt changes were smoothened to some extent. However even this model predicted 0 delay if the flight was early.

Executive Summary:

Motivation: Having profound interest in data analysis and building models, this challenge was a perfect opportunity for us to put some of our expertise into practical experience and learn about different aspects of data analysis like data cleaning, interpreting and understanding data, deriving insights, representing them using data visualizations, building models for predictions.

Execution: Firstly, we investigated the dataset and did a preliminary analysis such as determining the fields and their values, missing values, meaning of the fields and the data. We identified the fields that required data cleaning and performed the relevant tasks using python. We used some of the available packages in python to visualize the missing data. Then, we plotted a heatmap for the correlations between the features. Based on this correlation map, we have completely cleaned the data. Then we moved onto the analysis of the data, We listed down a few questions that we can ask the data and extract the answers for. We used this list of questions to build visualizations to represent the data and the insights. We then built an application dashboard using leaflet and shiny in R for interactive visualization. Finally, with the cleaned data, we have used multiple regression models to fit the data and predict on the new data. We then cut short the features to only the 5 categories of delays and used regression models to fit the data.

Additional data:

https://openflights.org/data.html

References:

[1] N. Xu, L. Sherry, and K. B. Laskey, "Multifactor model for predicting delays at U.S. airports," *Transportation Research Record*, no. 2052, pp. 62–71, 2008.

[2]https://www.bts.gov/topics/airlines-and-airports/understanding-reporting-causes-flight-delays-and-cancellations

[3] Bilogur, (2018). Missingno: a missing data visualization suite. Journal of Open Source Software, 3(22), 547, https://joss.theoj.org/papers/10.21105/joss.00547

[4] Leaflet for R - Introduction [Internet]. Leaflet for R - Introduction. [cited 2017Apr9]. Available from: https://rstudio.github.io/leaflet/

[5] Most of the visualizations are inspired from: https://www.kaggle.com/fabiendaniel/predicting-flight-delays-tutorial

[6] Model feature analysis and model building is further inspired from this kaggle tutorial: https://www.kaggle.com/abhishek211119/2015-flight-delays-and-cancellation-prediction

Appendix

Data cleaning:

```
# ## Data Cleaning ##
# Include all the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

import missingno as msno
import datetime
import seaborn as sns

# Load the FlightDelays.csv data which has the info about all the flight delays in
the following quarters: 2018 Q1-Q4, 2019 Q1-Q2
flight_delays = pd.read_csv('FlightDelays.csv', low_memory=False)
flight_delays.head()

# Now, let's use missingno (A visualization framework for missing numbers)
msno.matrix(flight_delays.sample(1000))

# We can see that the CARRIER_DELAY, WEATHER_DELAY, NAS_DELAY, SECURITY_DELAY and
LATE_AIRCRAFT_DELAY are almost completely unfilled. We can assume that these delays
are not observed and impute them with value 0. Since these are important in
determining the arrival/departure delays, we are not removing these fields.
msno.heatmap(flight_delays)

# Since cancellations wouldn't give proper information regarding the arrival or
departure delays, we intend to remove these from the data.
flight_delays = flight_delays[flight_delays.CANCELED == 0]
flight_delays.drop(['CANCELED', 'CANCELLATION_CODE'], axis=1, inplace=True)

msno.matrix(flight_delays.sample(1000))

# We can further see that all the missing values in DEP_DELAY are because there is
no delay i.e., delay is 0. Therefore, impute/fill all the missing values in
DEP_DELAY and related column values to 0
flight_delays['DEP_DELAY'].fillna(0, inplace=True)
flight_delays['DEP_DELAY_NEW'].fillna(0, inplace=True)
flight_delays['DEP_DEL15'].fillna(0, inplace=True)
flight_delays['DEP_DELAY_GROUP'].fillna(0, inplace=True)
```

```
# Convert all the number to datetime format using the function below
def format_heure(chaine):
    if pd.isnull(chaine):
        return np.nan
    else:
        if chaine == 2400:
            chaine = 0

        chaine = "{0:04d}".format(int(chaine))
        heure = datetime.time(int(chaine[0:2]), int(chaine[2:4]))
        return heure

flight_delays['DEP_TIME'] = flight_delays['DEP_TIME'].apply(format_heure)
flight_delays['CRS_DEP_TIME'] = flight_delays['CRS_DEP_TIME'].apply(format_heure)
flight_delays['CRS_ARR_TIME'] = flight_delays['CRS_ARR_TIME'].apply(format_heure)
flight_delays['ARR_TIME'] = flight_delays['ARR_TIME'].apply(format_heure)

# Now, we draw a correlation matrix to identify which columns to remove from the
dataset.
axis = plt.subplots(figsize=(20,14))
sns.heatmap(flight_delays.corr(),annot = True)
plt.show()

# DISTANCE, AIR_TIME, CRS_ELAPSED_TIME, ACTUAL_ELAPSED_TIME, EMPFULL, EMPPART,
EMPFTE are all correlated i.e., they are dependent on each other. We decided to keep
DISTANCE and remove the others since DISTANCE has the most non-null entries out of
all these features.
# keep variable DISTANCE
variables_to_remove = ['AIR_TIME', 'CRS_ELAPSED_TIME', 'ACTUAL_ELAPSED_TIME',
'EMPFULL', 'EMPPART', 'EMPFTE']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# Similarly, ARR_DELAY is correlated to ARR_DELAY_GROUP and ARR_DELAY_NEW.
# and DEP_DELAY is correlated to DEP_DELAY_GROUP and DEP_DELAY_NEW.
# keep variable ARR_DELAY
# keep variable DEP_DELAY
variables_to_remove = ['DEP_DELAY_GROUP', 'DEP_DELAY_NEW', 'ARR_DELAY_GROUP',
'ARR_DELAY_NEW']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# ARR_TIME is correlated to CRS_ARR_TIME, WHEELS_ON and DEP_TIME is correlated to
CRS_DEP_TIME, WHEELS_OFF
# keep ARR_TIME
# keep DEP_TIME
```

```
variables_to_remove = ['CRS_ARR_TIME', 'CRS_DEP_TIME', 'WHEELS_ON', 'WHEELS_OFF']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# EMPTOTAL is correlated to NET_INCOME and OP_REVENUES
# keep EMPTOTAL
variables_to_remove = ['NET_INCOME', 'OP_REVENUES']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# Arrival and Departure delays are dependent on each other. So, we can remove all
the departure delay related features.
# Arrival times and Departure times are correlated
variables_to_remove = ['DEP_TIME', 'DEP_DELAY', 'DEP_DEL15']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# ARR_DEL15 and DELAY_LEVEL are correlated. Remove ARR_DEL15
# keep DELAY_LEVEL
variables_to_remove = ['ARR_DEL15']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# Remove all the unnecessary variables.
variables_to_remove = ['ARR_TIME_BLK', 'DEP_TIME_BLK', 'YEAR', 'QUARTER', 'MONTH',
'DAY_OF_MONTH', 'FL_NUM', 'DEST_CITY', 'DEST_STATE']
flight_delays.drop(variables_to_remove, axis = 1, inplace = True)

# Plot the correlation matrix again to see if any of the features are still
correlated and needed to remove.
axis = plt.subplots(figsize=(20,14))
sns.heatmap(flight_delays.corr(),annot = True)
plt.show()

flight_delays.isnull().sum()

# Finally, remove all the null items in the data.
flight_delays.dropna(subset = ['TAXI_IN', 'PASSENGERS', 'ARR_DELAY'], inplace=True)

# Fill the missing delay values with 0.
flight_delays.fillna(0, inplace=True)
flight_delays.shape

# Save the clean data into a csv file for model building.
flight_delays.to_csv('flight_delays_clean2.csv', encoding='utf-8', index=False)
```

Data visualization:

```
# ## Data Visualization ##
# Include all the required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

from mpl_toolkits.basemap import Basemap
from collections import OrderedDict

import plotly.offline as py
import plotly.figure_factory as ff
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
from plotly import tools

# Now let's check the data that we have, we see that the Airports.csv file has City
and State information for all the Airports.
airports = pd.read_csv('Airports.csv')
airports.head()

# Airfares.csv has all the fare related information for all the flights.
airfares = pd.read_csv('Airfares.csv')
airfares.head()

# Routes.csv has information about all the flight routes, such as ORIGIN city, state
of the flight to the DEST city, state of the flight including the distance of the
route.
routes = pd.read_csv('Routes.csv')
routes.head()

# FlightDelays.csv has all the flight delays observed in the six Quarters of data
provided (2018 Q1-Q4, 2019 Q1, Q2)
flight_delays = pd.read_csv('FlightDelays.csv', low_memory=False)
flight_delays.head()

# We can see that there are more than 10 million flight delays in 1 1/2 years!!!
# We grabbed some additional data which includes latitude and longitude information
for the City/State for visualization purposes on the world map.
columns = ['index',
           'Name',
           'city',
```

```
            'country',
            'Airport',
            'Code',
            'Latitude',
            'Longitude',
            'Altitude',
            'TimeZone',
            'DST',
            'TZ',
            'Type',
            'Source']

airports_data = pd.read_csv('airports.dat.txt',
                            header = None,
                            names = columns)

# Let's view this data now.
print(airports_data.head())

# Merge this additional data we have got with the one obtained from Airports.csv on
the Airport code.
airports = pd.merge(airports, airports_data, on = ['Airport'], how = 'inner')

# Now we move onto the visualizations:
# Here we try to visualize the number of flight delays on the world map i.e., to
identify which particular cities/states have high airline delays.
count_flights = flight_delays['ORIGIN'].value_counts()
plt.figure(figsize=(11, 11))

colors = ['yellow', 'red', 'lightblue', 'purple', 'green', 'orange']
size_limits = [1, 100, 1000, 10000, 100000, 1000000]
labels = []

for i in range(len(size_limits)-1):
    labels.append("{} <.< {}".format(size_limits[i], size_limits[i+1]))

map = Basemap(resolution='i',llcrnrlon=-180, urcrnrlon=-50,
              llcrnrlat=10, urcrnrlat=75, lat_0=0, lon_0=0,)
map.shadedrelief()
map.drawcoastlines()
map.drawcountries(linewidth = 3)
map.drawstates(color='0.3')

for index, (code, y, x) in airports[['Airport', 'Latitude',
```

```
'Longitude']].iterrows():
    x, y = map(x, y)
    isize = [i for i, val in enumerate(size_limits) if val < count_flights[code]]
    ind = isize[-1]
    map.plot(x, y, marker='o', markersize = ind+5, markeredgewidth = 1, color =
colors[ind],
             markeredgecolor='k', label = labels[ind])

handles, labels = plt.gca().get_legend_handles_labels()
by_label = OrderedDict(zip(labels, handles))
key_order = ('1 <.< 100', '100 <.< 1000', '1000 <.< 10000',
             '10000 <.< 100000', '100000 <.< 1000000')
new_label = OrderedDict()

for key in key_order:
    new_label[key] = by_label[key]

plt.legend(new_label.values(), new_label.keys(), loc = 1, prop= {'size':11},
           title='Number of flight delays at the ORIGIN airport', frameon = True,
framealpha = 1)
plt.show()

# The below plot shows the number of flight delays per quarter.
delay_counts = flight_delays.groupby(['YEAR',
'QUARTER']).size().reset_index().rename(columns={0: 'count'})
delay_counts['YEAR_QUARTER'] = delay_counts[['YEAR',
'QUARTER']].astype(str).apply(lambda x: 'Q'.join(x), axis = 1)

trace = go.Bar(
    x = delay_counts.YEAR_QUARTER,
    y = delay_counts['count']
)

data = [trace]
layout = go.Layout(
    title = 'Flight Delay counts per quarter',
    yaxis = dict(title = '# of Flight Delays')
)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

# So, we see a similar trend every quarter i.e., every quarter has at least 1.5
million flight delays. Another observation is that the number of flight delays tend
```

```
to peak during Quarter 2 and 3, and then they decrease again. Now, let's see the
trends based on the Month.
fdc = flight_delays.copy()

month = {1: 'Jan',
         2: 'Feb',
         3: 'Mar',
         4: 'Apr',
         5: 'May',
         6: 'Jun',
         7: 'Jul',
         8: 'Aug',
         9: 'Sep',
         10: 'Oct',
         11: 'Nov',
         12: 'Dec'}

fdc['dep_delay'] = np.where(fdc.DEP_DELAY > 0, 1, 0)
fdc['arr_delay'] = np.where(fdc.ARR_DELAY > 0, 1, 0)
fdc_m = fdc.groupby('MONTH').dep_delay.mean().round(2)

fdc_m.index = fdc_m.index.map(month)
trace1 = go.Bar(
    x = fdc_m.index,
    y = fdc_m.values,
    name = 'Departure_delay',
    marker = dict(
        color = 'aqua'
    )
)

fdc_m = fdc.groupby('MONTH').arr_delay.mean().round(2)
fdc_m.index = fdc_m.index.map(month)

trace2 = go.Bar(
    x = fdc_m.index,
    y = fdc_m.values,
    name='Arrival_delay',
    marker=dict(
        color = 'red'
    )
)

data = [trace1, trace2]
```

```
layout = go.Layout(
    title='% Delay (Months)',
    yaxis = dict(title = '%')
)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

# Again, we see that each month has at least 0.3% of flights delayed. And the number
of flight delays peak in the months of June and July, probably due to the increased
number of total flights during these months. Now, we shift our focus to the trends
corresponding to each day of the week.
dayOfWeek = {1:'Monday',
             2:'Tuesday',
             3:'Wednesday',
             4:'Thursday',
             5:'Friday',
             6:'Saturday',
             7:'Sunday'}

fdc_w = fdc.groupby('DAY_OF_WEEK').dep_delay.mean().round(2)
fdc_w.index = fdc_w.index.map(dayOfWeek)

trace1 = go.Bar(
    x = fdc_w.index,
    y = fdc_w.values,
    name = 'Departure_delay',
    marker=dict(
        color = 'cyan'
    )
)

fdc_w = fdc.groupby('DAY_OF_WEEK').arr_delay.mean().round(2)
fdc_w.index = fdc_w.index.map(dayOfWeek)

trace2 = go.Bar(
    x = fdc_w.index,
    y = fdc_w.values,
    name='Arrival_delay',
    marker=dict(
        color = 'indigo'
    )
)
```

```
data = [trace1, trace2]
layout = go.Layout(
    title='% Delay (Day of Week)',
    yaxis = dict(title = '%')
)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

# Surprisingly, number of flight delays on weekdays (Thursday and Friday) are more
than on the weekend (Saturday and Sunday)
# Lets see the Airline names. We have 17 unique airline names.
airline_names = flight_delays.CARRIER.unique()
airline_names

# Lets visualize Arrival delays for each airline.
import seaborn as sns

axis = plt.subplots(figsize=(10, 14))
sns.despine(bottom=True, left=True)

# Observations with Scatter Plot
sns.stripplot(x = "ARR_DELAY",
              y = "CARRIER",
              data = flight_delays,
              dodge=True,
              jitter=True)

plt.show()

# We can see that the AA (American Airlines) has huge delays. From the above
stripplot and below pie chart, American Airlines has the high share in flight delays
and also the delays in quite longer, whereas even though 9E has the highest share in
the number of flight delays (19%), the delay durations are actually lesser than that
of AA.

axis = plt.subplots(figsize=(10, 14))

Name = flight_delays["CARRIER"].unique()
size = flight_delays["CARRIER"].value_counts()

plt.pie(size, labels=Name, autopct='%5.0f%%')
plt.show()
```

```
axis = plt.subplots(figsize=(10,14))
sns.despine(bottom=True, left=True)

# Observations with Scatter Plot
sns.stripplot(x = "DEP_DELAY",
              y = "CARRIER",
              data = flight_delays,
              dodge=True,
              jitter=True)

plt.show()

# We observe that the stripplots for the Arrival delay and Departure delays look
almost the same, this is because they are correlated i.e., dependent on each other.
Intuitively, if a flight departure is delayed, then the flight arrival at the
destination location will also be delayed. Since, we are deriving the information
for each carrier, both the plots look the same. The same logic can be used during
data cleaning to include only one of Arrival or departure delays.

# Next, we further breakdown the delays into short, small and longer delays. Short
delays are the ones which are less than 5 min. Small delays are the ones for which
delays range from 5 min to 45 min. Longer delays contain the ones for which delays
are more than 45 min.
delay_type = lambda x:((0,1)[x > 5],2)[x > 45]
flight_delays['DELAY_LEVEL'] = flight_delays['DEP_DELAY'].apply(delay_type)

fig = plt.figure(1, figsize=(10, 7))
ax = sns.countplot(y="CARRIER", hue='DELAY_LEVEL', data=flight_delays)

labels = airline_names

ax.set_yticklabels(labels)
plt.setp(ax.get_xticklabels(), fontsize=12, weight = 'normal', rotation = 0);
plt.setp(ax.get_yticklabels(), fontsize=12, weight = 'bold', rotation = 0);
ax.yaxis.label.set_visible(False)
plt.xlabel('Flight count', fontsize=16, weight = 'bold', labelpad=10)

L = plt.legend()
L.get_texts()[0].set_text('on time (t < 5 min)')
L.get_texts()[1].set_text('small delay (5 < t < 45 min)')
L.get_texts()[2].set_text('large delay (t > 45 min)')
plt.show()

# Like we inferred from the previous plots, AA and WN have the huge delays i.e.,
```

```
longer delays for flights.
# Next plot shows the mean Arrival and Departure delays for the airlines.
fdc_d =
fdc.groupby('CARRIER').DEP_DELAY.mean().to_frame().sort_values(by='DEP_DELAY',
                                                    ascending=False).round(2)
trace1 = go.Bar(
    x=fdc_d.index,
    y=fdc_d.DEP_DELAY,
    name='departure_delay',
    marker=dict(
        color = 'navy'
    )
)

fdc_a =
fdc.groupby('CARRIER').ARR_DELAY.mean().to_frame().sort_values(by='ARR_DELAY',
                                                    ascending=False).round(2)
trace2 = go.Bar(
    x=fdc_a.index,
    y=fdc_a.ARR_DELAY,
    name='arrival_delay',
    marker=dict(
        color = 'red'
    )
)

data = [trace1, trace2]
layout = go.Layout(xaxis=dict(tickangle=15), title='Mean Arrival & Departure Delay
by Airlines',
    yaxis = dict(title = 'minute'),
                   barmode='stack')

fig = go.Figure(data=data, layout=layout)
py.iplot(fig)
```

Model Building:

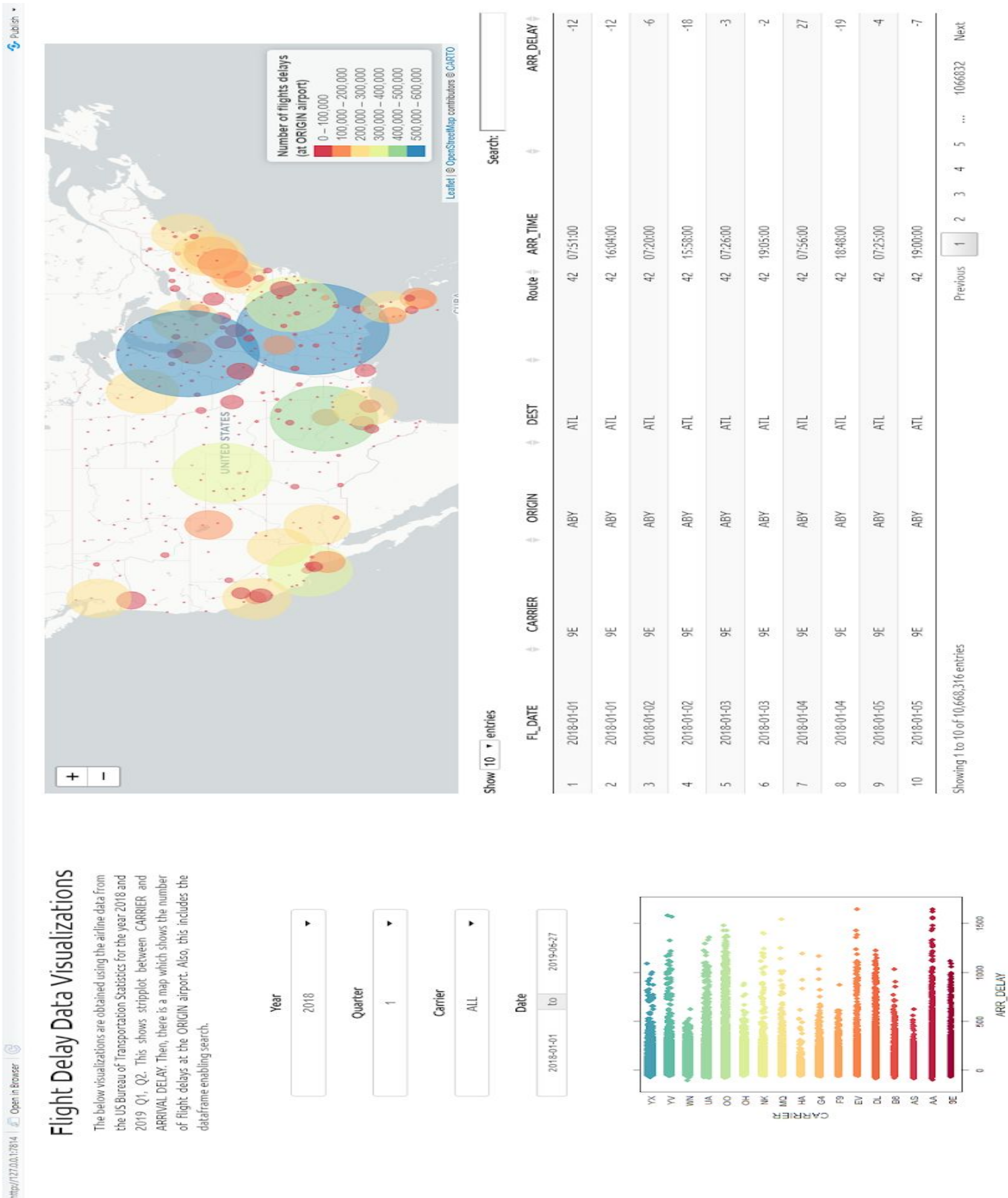
```
import pandas as pd
Flights1 = pd.read_csv('/content/drive/My Drive/TAMUIDS/flight_delays_clean2.csv',
low_memory=False)

import numpy as np
Flights1['ARR_DELAY'] = Flights1['ARR_DELAY'].apply(lambda x:x if x < 300 else
np.nan)
Flights1.dropna(how = 'any', inplace=True)

import numpy as np
Flights1['ARR_DELAY'] = Flights1['ARR_DELAY'].apply(lambda x:x if x > -100 else
np.nan)
Flights1.dropna(how = 'any', inplace=True)

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import Lasso,LinearRegression,Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import
RandomForestRegressor,AdaBoostRegressor,BaggingRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

Las = Lasso()
LinR = LinearRegression()
Rid = Ridge()
Rfc = RandomForestRegressor(random_state=2)
Dtc = DecisionTreeRegressor(random_state = 2)
Boost_Lin = AdaBoostRegressor(base_estimator=LinR,random_state=2)
Boost_las = AdaBoostRegressor(base_estimator=Las,random_state=2)
Boost_rid = AdaBoostRegressor(base_estimator=Rid,random_state=2)
Bg_Lin = BaggingRegressor(base_estimator=LinR,random_state=2)
Bg_las = BaggingRegressor(base_estimator=Las,random_state=2)
Bg_rid = BaggingRegressor(base_estimator=Rid,random_state=2)

le = LabelEncoder()

Flights1['CARRIER']= le.fit_transform(Flights1['CARRIER'])
Flights1['ORIGIN'] = le.fit_transform(Flights1['ORIGIN'])
Flights1['DEST'] = le.fit_transform(Flights1['DEST'])

Flights1.head()
```

```
df_train = Flights1[Flights1['FL_DATE'] <= '2019-03-31']
df_test = Flights1[Flights1['FL_DATE'] > '2019-03-31']
print("Shape of train Data: ", df_train.shape)
print("Shape of test Data: ", df_test.shape)

X_train = df_train.drop(['ARR_DELAY', 'ARR_TIME', 'FL_DATE'], axis = 1)
Y_train = df_train['ARR_DELAY']

X_test =  df_test.drop(['ARR_DELAY', 'ARR_TIME', 'FL_DATE'], axis = 1)
Y_test = df_test['ARR_DELAY']

sc1=StandardScaler()
X_train_sc=sc1.fit_transform(X_train)
X_test_sc=sc1.transform(X_test)

#values < 300 kept
#,'Lasso','Linear Regression','Ridge','Random forest Regressor','Decision Tree
Regressor','Boosted Linear',
#     'Boosted Lasso','Boosted Ridge','Bagged Linear','Bagged Lasso','Bagged Ridge'
for model, name in zip([LinR],
     ['Linear Regression']):

    model1 = model.fit(X_train_sc,Y_train)
    Y_predict=model1.predict(X_test_sc)
    print(name)
    print('Mean Absolute Error:', mean_absolute_error(Y_test, Y_predict))
    print('Mean Squared Error:', mean_squared_error(Y_test, Y_predict))
    print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test,
Y_predict)))
    print('R2 : ',r2_score(Y_test, Y_predict))
    print()

import matplotlib.pyplot as plt
print(name)
plt.scatter(Y_test, Y_predict)
plt.title("Model : Linear Regression")
plt.xlabel("Truth")
plt.ylabel("Prediction")
plt.savefig('/content/drive/My Drive/TAMUIDS/Linear_Regression.png',
bbox_inches='tight')

#,'Lasso','Linear Regression','Ridge','Random forest Regressor','Decision Tree
Regressor','Boosted Linear',
```

```
#     'Boosted Lasso','Boosted Ridge','Bagged Linear','Bagged Lasso','Bagged Ridge'
for model, name in zip([LinR],
     ['Linear Regression']):

    model1 = model.fit(X_train_sc,Y_train)
    Y_predict=model1.predict(X_test_sc)
    print(name)
    print('Mean Absolute Error:', mean_absolute_error(Y_test, Y_predict))
    print('Mean Squared Error:', mean_squared_error(Y_test, Y_predict))
    print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test,
Y_predict)))
    print('R2 : ',r2_score(Y_test, Y_predict))
    print()

import matplotlib.pyplot as plt
print(name)
plt.scatter(Y_test, Y_predict)
plt.title("Model : Linear Regression")
plt.xlabel("Truth")
plt.ylabel("Prediction")
plt.savefig('/content/drive/My Drive/TAMUIDS/Linear_Regression.png',
bbox_inches='tight')

for model, name in zip([Las],
     ['Lasso']):

    model1 = model.fit(X_train_sc,Y_train)
    Y_predict=model1.predict(X_test_sc)
    print(name)
    print('Mean Absolute Error:', mean_absolute_error(Y_test, Y_predict))
    print('Mean Squared Error:', mean_squared_error(Y_test, Y_predict))
    print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test,
Y_predict)))
    print('R2 : ',r2_score(Y_test, Y_predict))
    print()

import matplotlib.pyplot as plt
print(name)
plt.scatter(Y_test, Y_predict)
plt.title("Model : Lasso")
plt.xlabel("Truth")
plt.ylabel("Prediction")
plt.savefig('/content/drive/My Drive/TAMUIDS/Lasso.png',  bbox_inches='tight')
```

```
for model, name in zip([Rid],
     ['Ridge']):

    model1 = model.fit(X_train_sc,Y_train)
    Y_predict=model1.predict(X_test_sc)
    print(name)
    print('Mean Absolute Error:', mean_absolute_error(Y_test, Y_predict))
    print('Mean Squared Error:', mean_squared_error(Y_test, Y_predict))
    print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test,
Y_predict)))
    print('R2 : ',r2_score(Y_test, Y_predict))
    print()

import matplotlib.pyplot as plt
print(name)
plt.scatter(Y_test, Y_predict)
plt.title("Model : Ridge")
plt.xlabel("Truth")
plt.ylabel("Prediction")
plt.savefig('/content/drive/My Drive/TAMUIDS/Ridge.png',  bbox_inches='tight')

for model, name in zip([Bg_Lin],
     ['Bagged Linear']):

    model1 = model.fit(X_train_sc,Y_train)
    Y_predict=model1.predict(X_test_sc)
    print(name)
    print('Mean Absolute Error:', mean_absolute_error(Y_test, Y_predict))
    print('Mean Squared Error:', mean_squared_error(Y_test, Y_predict))
    print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test,
Y_predict)))
    print('R2 : ',r2_score(Y_test, Y_predict))
    print()

import matplotlib.pyplot as plt
print(name)
plt.scatter(Y_test, Y_predict)
plt.title("Model : Bagged Linear")
plt.xlabel("Truth")
plt.ylabel("Prediction")
plt.savefig('/content/drive/My Drive/TAMUIDS/Bagged Linear.png',
bbox_inches='tight')
```

Screenshot of the application:



leaflet/R code for shiny web app as a visualization dashboard:

global.R

```
library(leaflet)
library(shiny)
library(shinythemes)
library(DT)
library(lattice)
library(RColorBrewer)
library(dplyr)

airports <- read.csv("Airports.csv")
latlong <- read.csv("airports.dat.txt", header=FALSE)
colnames(latlong) <- c('index', 'Name', 'city',
                       'Country', 'Airport', 'Code',
                       'Latitude', 'Longitude', 'Altitude',
                       'TimeZone', 'DST', 'TZ',
                       'Type', 'Source')
latlong <- subset(latlong, select = -c(index))

airports <- merge(x = airports,
                  y = latlong,
                  by = "Airport")

airport_delays <- read.csv('../FlightDelays.csv')

flight_counts <- as.data.frame(sort(table(airport_delays$ORIGIN), decreasing =
TRUE))
colnames(flight_counts) <- c('Airport', 'count')
flight_counts <- merge(x = airports,
                       y = flight_counts,
                       by = "Airport")[, c('Airport', 'Latitude', 'Longitude',
'count')]

rm(list = c("airport_delays"))

airport_delays <- read.csv('../flight_delays_clean.csv')
years <- append(as.list(unique(airport_delays$YEAR)), c('ALL'))
quarters <- append(as.list(unique(airport_delays$QUARTER)), c('ALL'))
carriers <- append(as.list(levels(unique(airport_delays$CARRIER))), c('ALL'))

airport_delays <- airport_delays[, c('FL_DATE', 'CARRIER', 'ORIGIN', 'DEST',
'Route', 'ARR_TIME', 'ARR_DELAY')]
```

ui.R

```
shinyUI(
  fluidPage(theme = shinytheme("united"),
    fluidRow(column(4,
                    align="center",
                    HTML("<b><h2>Flight Delay Data Visualizations</h2></b>"),
                    HTML('<p align="justify" style="padding-left: 100px;
padding-right: 100px"> The below visualizations are obtained using the airline data
from the US Bureau of Transportation Statistics for the year 2018 and 2019 Q1, Q2.
This shows the stripplot between CARRIER and ARRIVAL DELAY. Then, there is a map
which shows the number of flight delays at the ORIGIN airport. Also, this includes
the dataframe enabling search.</p>'),
                    br(),
                    br(),
                    selectInput('Year', 'Year',
                                choices = years,
                                multiple = FALSE,
                                selected = 'ALL'),
                    selectInput('Quarter', 'Quarter',
                                choices = quarters,
                                multiple = FALSE,
                                selected = 'ALL'),
                    selectInput('Carrier', 'Carrier',
                                choices = carriers,
                                multiple = FALSE,
                                selected = 'ALL'),
                    dateRangeInput('daterangeInput',
                                   label = 'Date',
                                   start = as.Date('2018-01-01') , end =
as.Date('2019-06-27')
                    ),
                    br(),
                    plotOutput("myplot")
    ),

    column(8,
           br(),
           leafletOutput(outputId = 'map', height = 400, width = 1200),
           br(),
           DT::dataTableOutput('mytable'))
    )
  )
)
```

server.R

```
shinyServer(function (input, output) {
  output$map <- renderLeaflet({
    pal <- colorBin("Spectral", flight_counts$count, n = 5)

    popup_data <- paste0("Airport: ",
                         flight_counts$Airport,
                         "<br>Count: ",
                         flight_counts$count)

    leaflet(flight_counts) %>%
      setView(lng = -97, lat = 38, zoom = 4) %>%
      addProviderTiles("CartoDB.Positron", options = providerTileOptions(noWrap =
TRUE)) %>%
      addCircles(data = flight_counts,
                 lat = ~Latitude,
                 lng = ~Longitude,
                 weight = 1,
                 radius = ~count,
                 color = ~pal(count),
                 fillOpacity = 0.5,
                 popup = ~popup_data) %>%
      addLegend("bottomright", pal = pal, values = ~flight_counts$count,
                title = "Number of flights delays<br>(at ORIGIN airport)",
                labFormat = labelFormat(),
                opacity = 1)
  })

  output$mytable <- DT::renderDataTable({
    datatable(airport_delays)
  }, options = list(scrollX = TRUE))

  output$myplot <- renderPlot({
    airport_delays_sample <- airport_delays %>% sample_frac(0.1)
    cols <- brewer.pal(10, "Spectral")
    pal <- colorRampPalette(cols)
    stripplot(CARRIER ~ ARR_DELAY, data = airport_delays_sample,
              aspect = 1, jitter = T,
              xlab = "ARR_DELAY", ylab = "CARRIER",
              groups = airport_delays_sample$CARRIER,
              col = pal(20), pch = 19, cex = 1)})
})
```