PREDICTING FLIGHT DELAY

1. PROBLEM STATEMENT

Is there any impact due to flight delays? Flight delays not only cause inconvenience to passengers, but also cost the carriers billions of dollars. Flight delays can be caused due to bad weather conditions, airport congestion, airspace congestion, maintenance or security issues. These delays tarnish airlines on-time reputation, often resulting in loss of demand by passengers.

Looking at statistics compiled by the Bureau of Transport Statistics, the extreme weather causes only 5% delay of total delay minutes whereas the biggest contributors to delay have consistently been late arrival of aircraft and air carrier delay. In the light of the above statistics there is a need for an intelligent and automated prediction system that can predict possible airline delays. If we can model these delays accurately, we can account for most of the delayed flights per year.

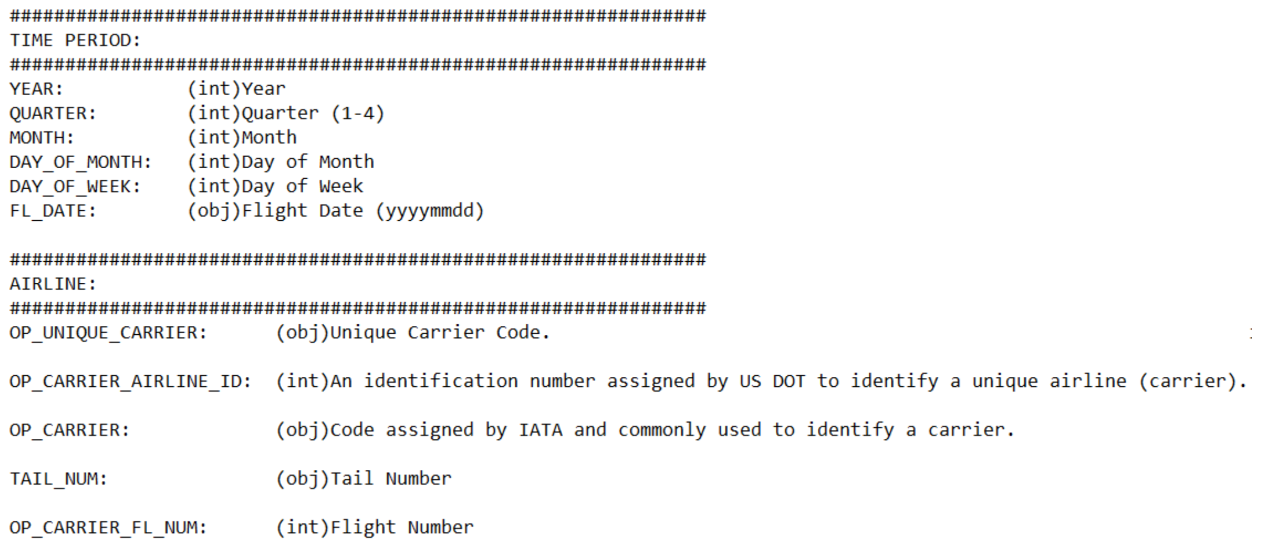
The goal of this project is to predict whether a flight will be delayed, by utilizing the public dataset provided by **Bureau of Transport Statistics,** on local flights in United States for the year 2019.

DATA SOURCE: Data for the Capstone project is gathered from Bureau of Transportation website. Data gathered for 12 months from January to December for the year 2019. Dataset contains 7422037 rows and 78 columns.

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

2. DATA INSPECTION AND CLEANING

The data is downloaded from the Bureau of Transport Statistics website for the first quarter of Year 2019 and is loaded into python as dataframe named flight_data. Refer to the Data_Dictionary.txt document to better understand the columns of flight_data.



```
################################################################
TIME PERIOD:
################################################################
YEAR:           (int)Year
QUARTER:        (int)Quarter (1-4)
MONTH:          (int)Month
DAY_OF_MONTH:   (int)Day of Month
DAY_OF_WEEK:    (int)Day of Week
FL_DATE:        (obj)Flight Date (yyyymmdd)

################################################################
AIRLINE:
################################################################
OP_UNIQUE_CARRIER:      (obj)Unique Carrier Code.

OP_CARRIER_AIRLINE_ID:  (int)An identification number assigned by US DOT to identify a unique airline (carrier).

OP_CARRIER:             (obj)Code assigned by IATA and commonly used to identify a carrier.

TAIL_NUM:               (obj)Tail Number

OP_CARRIER_FL_NUM:      (int)Flight Number
```

DATA CLEANING

Any irrelevant data was removed

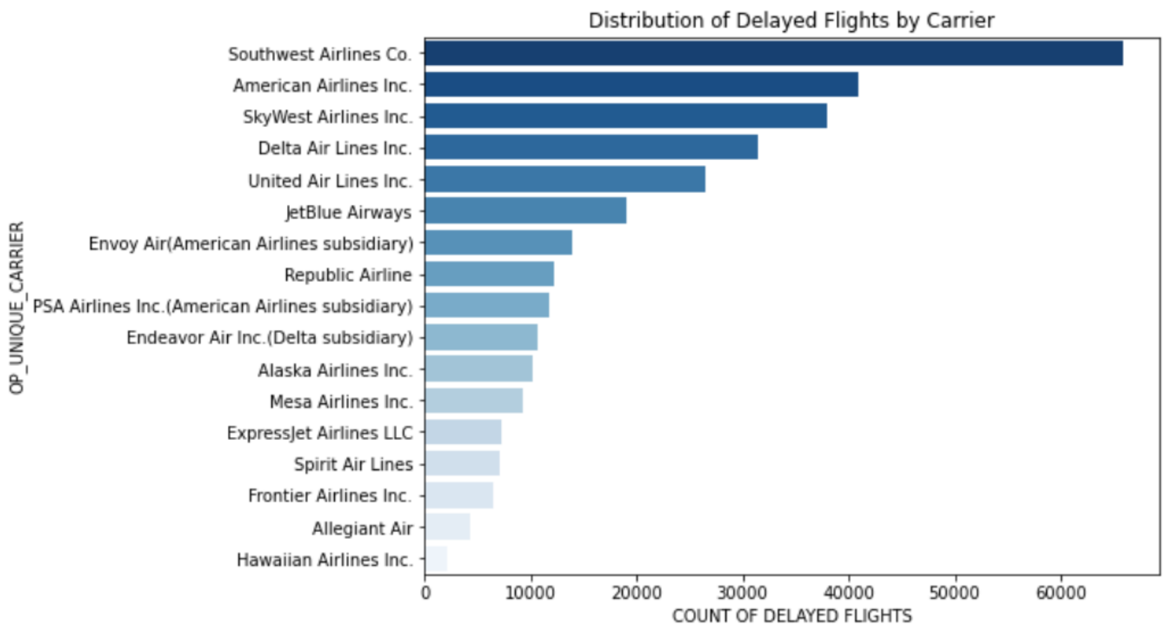
- Columns containing only redundant data
- Row containing null values

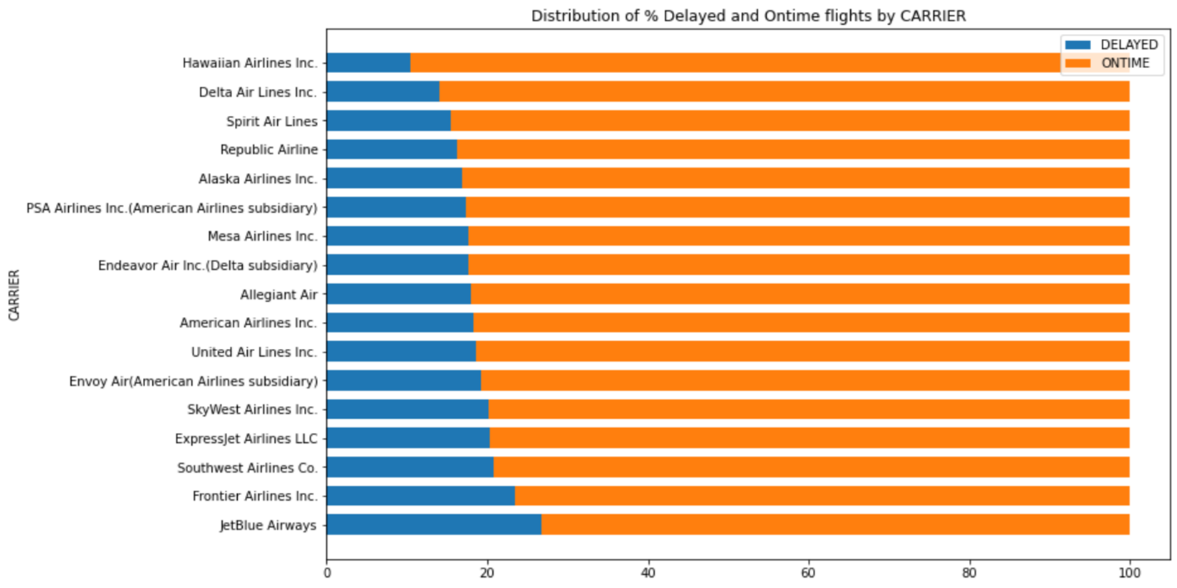
Prepared features for data exploration

- Converting dates to date time

3. EXPLORATORY DATA ANALYSIS

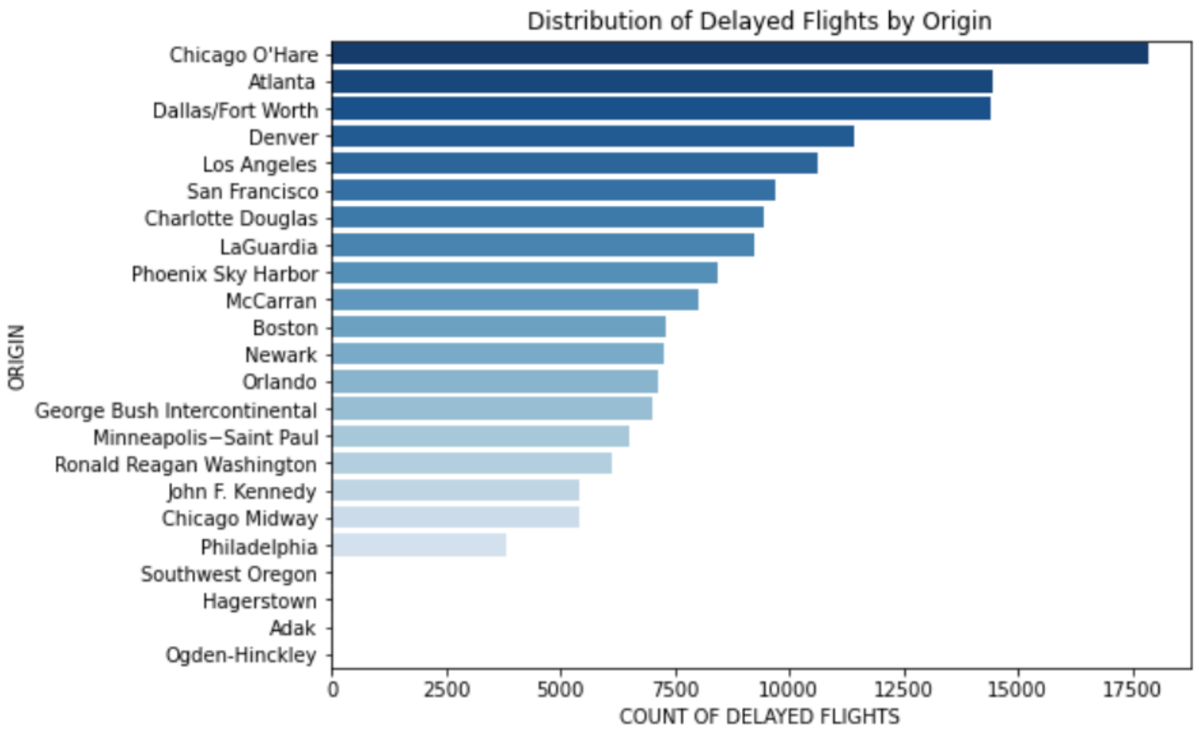
3.1. Distribution of % Delayed flights BY CARRIER

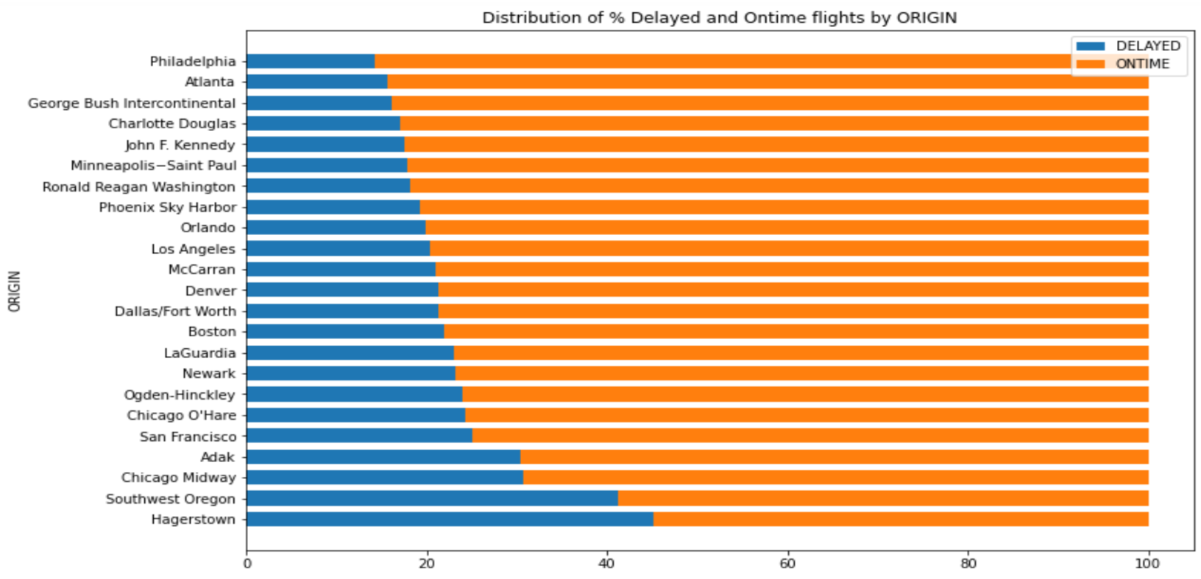




Airlines Frontier, Jet Blue have most % delayed flights when compared to Hawaiian Airlines, Delta Airlines which have least % delayed flights. Carrier can be considered as an important feature.

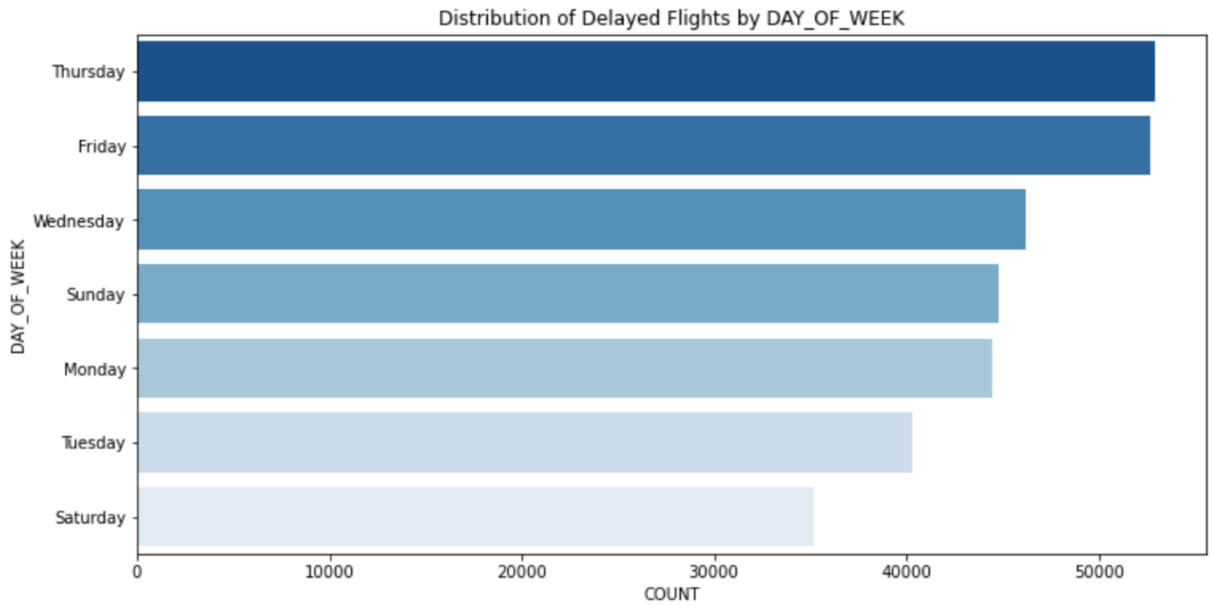
3.2. Distribution of % Delayed flights BY ORIGIN

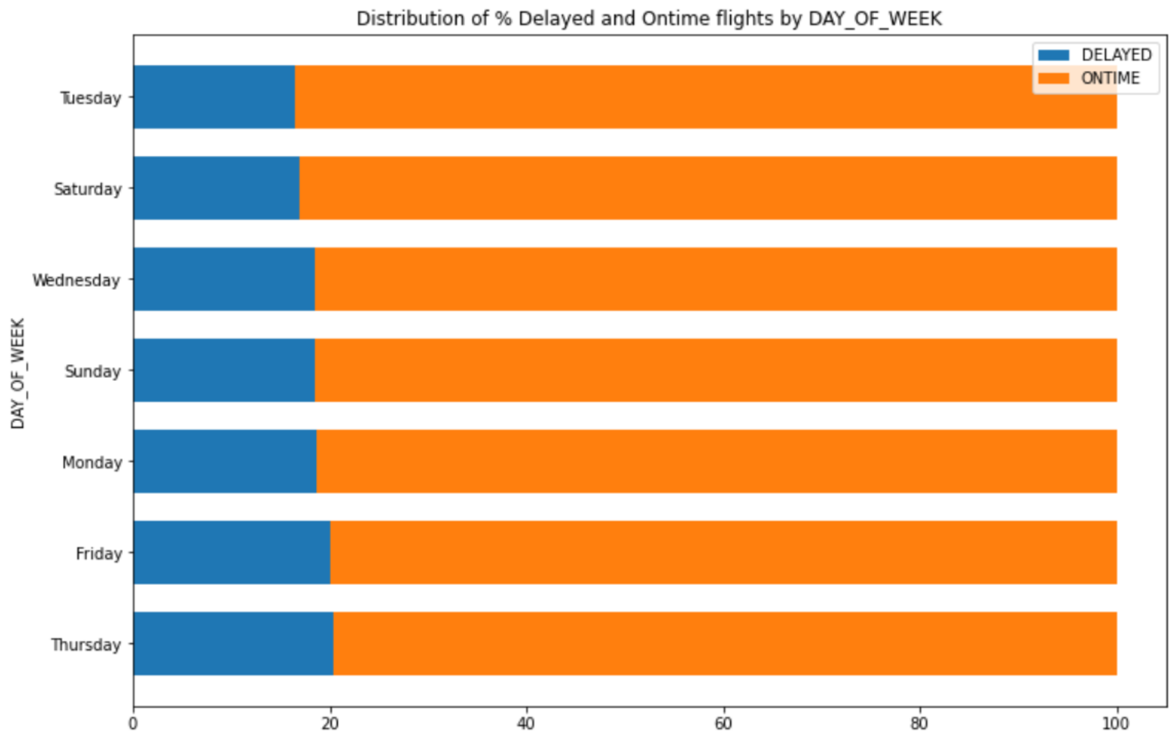




Flights originating from Hagerstown, Southwest Oregon have more % delayed flights when compared to Minneapolis, Atlanta which have least % delayed flights. Origin can be considered as an important feature.

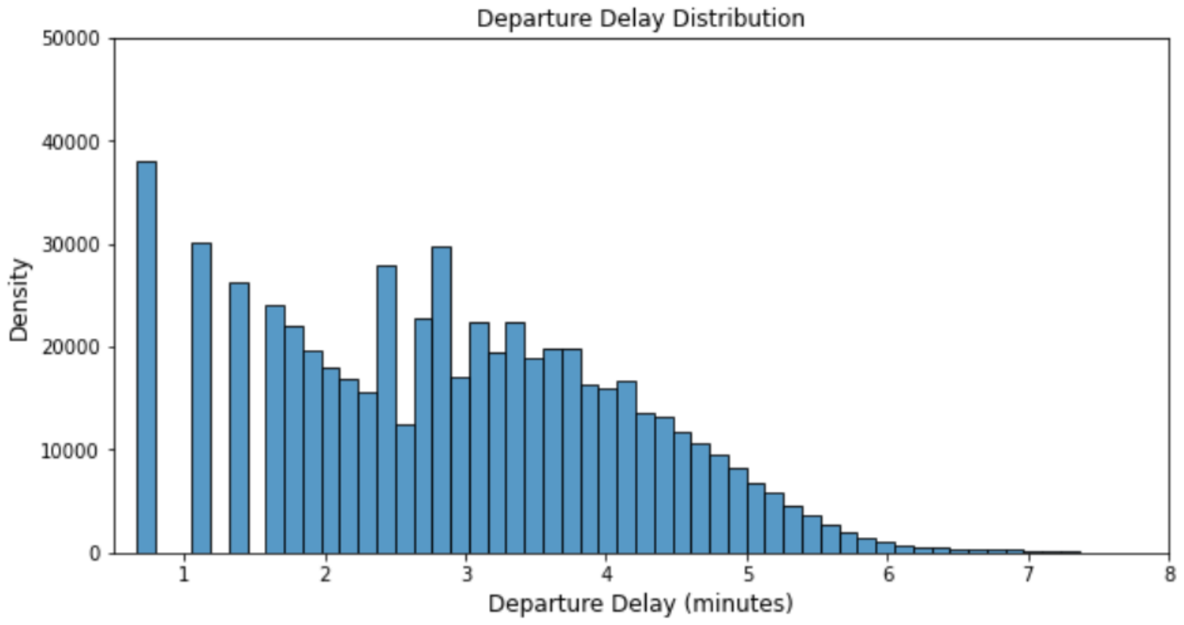
3.3. Distribution of % Delayed flights BY DAY_OF_WEEK



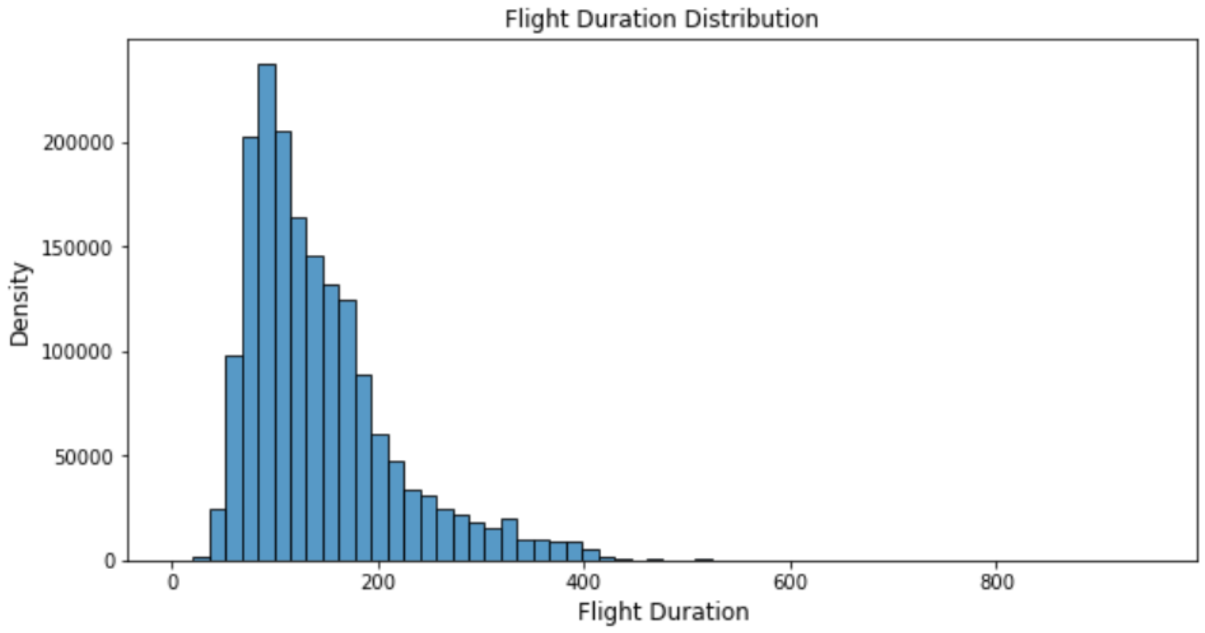


Flights on Thursday and Friday have more % delayed flights when compared to others days of week which have least % delayed flights. DAY OF WEEK can be considered as an important feature.

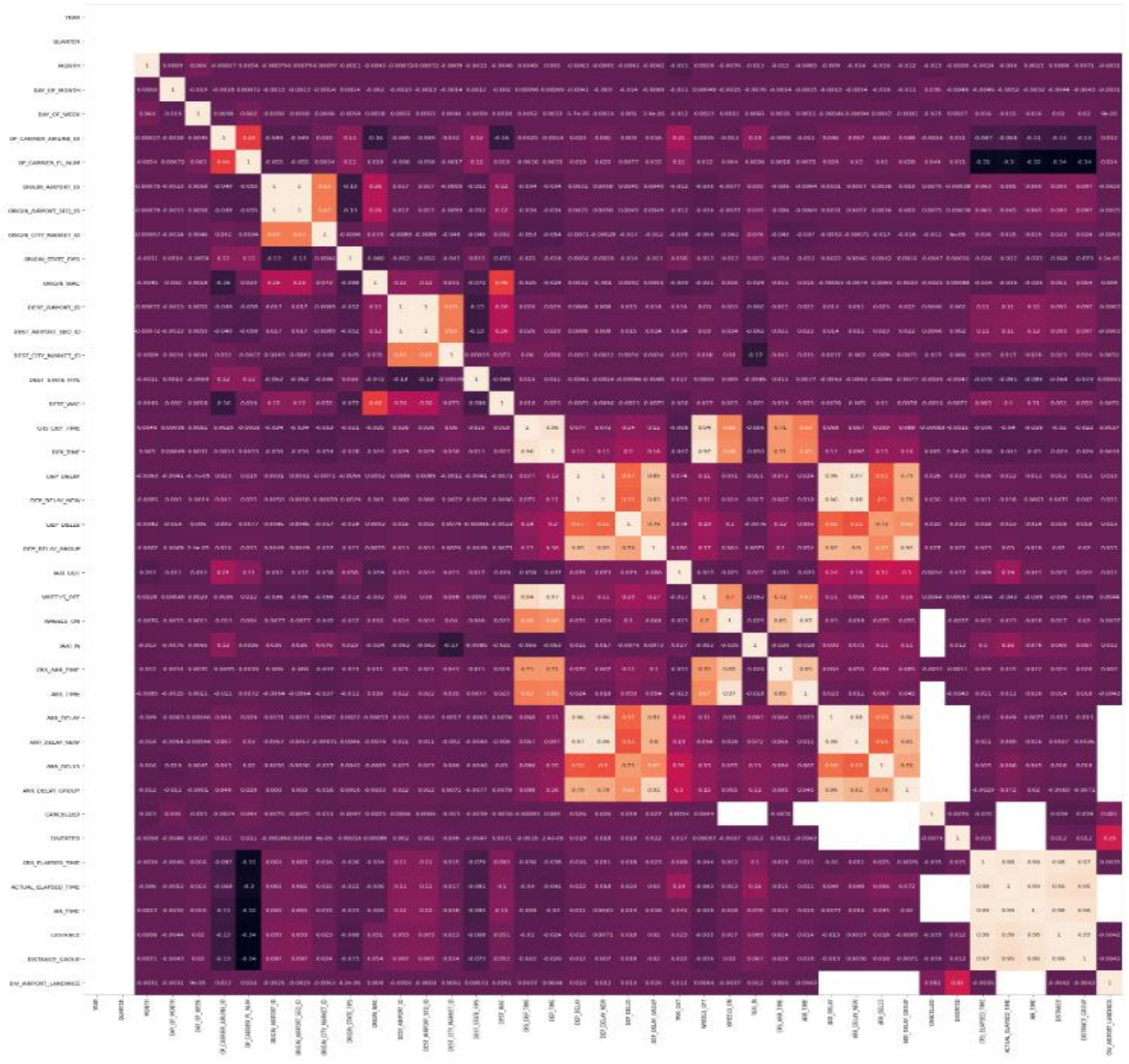
3.4. Distribution of Departure Delay



3.5. Distribution of Flight Duration



3.6. Correlation MATRIX



Correlation Matrix:

- Few features seem to correlate especially ones in lighter shades
- Deleted features with high correlation
- ['ORIGIN_AIRPORT_SEQ_ID', 'DEST_AIRPORT_SEQ_ID', 'DEP_TIME', 'DEP_DELAY_NEW', 'DEP_DELAY_GROUP', 'WHEELS_OFF', 'CRS_ARR_TIME', 'ARR_TIME', 'ARR_DELAY', 'ARR_DELAY_NEW', 'ARR_DELAY_GROUP', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'DISTANCE', 'DISTANCE_GROUP']
- For modelling 41 independent variables were selected

4. FEATURE ENGINEERING

4.1. Prior Flight Information

In order to predict whether a flight will be delayed or not, any future information about the flight like actual arrival time, air time cannot be used. But we can retrieve any past history of the flight. Calculating number of trips for each TAIL_NUM for each day, and retrieving the prior origin airport and prior departure hour from previous flight and if flight was delayed.

Columns – PRIOR_DELAY, PRIOR_DEP_DELAY

```
def get_prior_info(df, column, name, replace_null):

    df = df.sort_values(by = ['TAIL_NUM', 'FL_DATE']).reset_index(drop = True)
    df[name] = df.loc[df['FLIGHT_COUNT'].shift(-1)>1, column]
    df[name] = df[name].shift()
    df[name] = df[name].fillna(value=replace_null)
    return df
```

4.2. Cyclical Features

Hours of a day, Days of week, months in a year are some examples of cyclical features. Converting these raw features into cyclical features (0 - 24 hour). Each cyclical feature can be converted to the sin and cos components of the feature as (x, y) coordinates of a circle. The other benefit of the transformation is to see the correlation of the cyclical features to target data.

4.3. Categorical Features

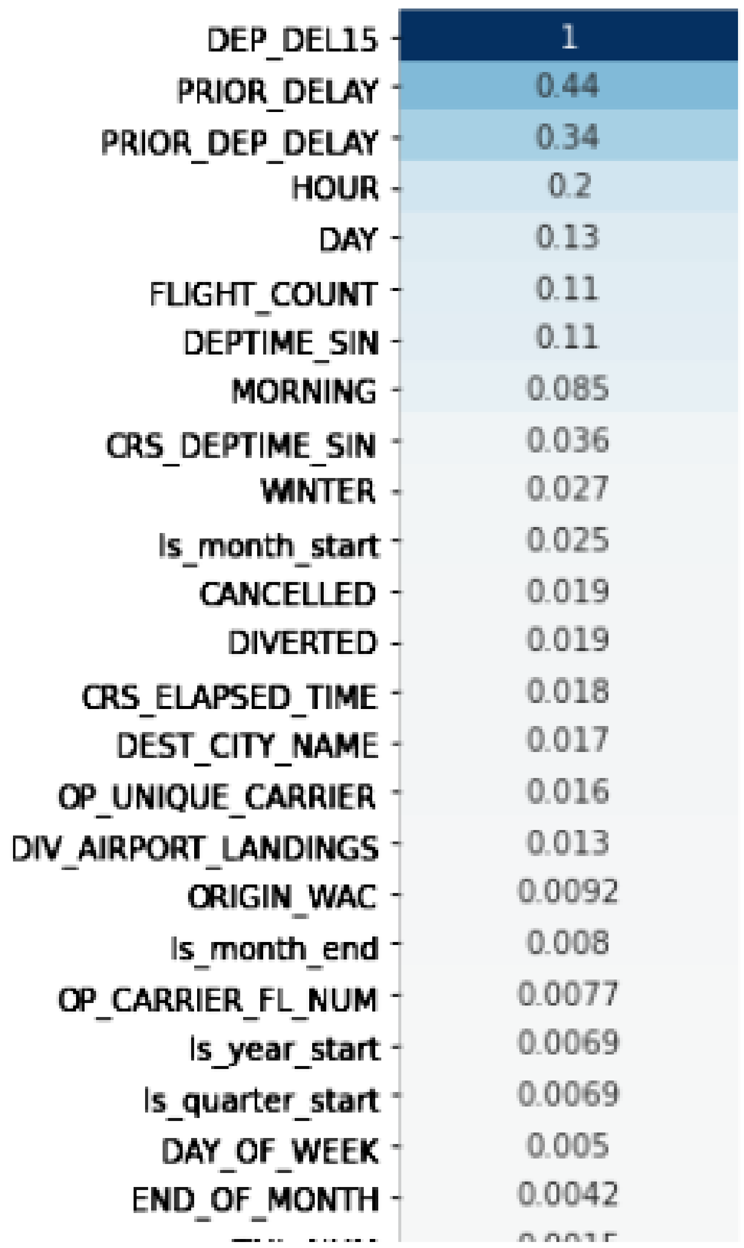
Converting categorical features into numbers using Label Encoder.

'TAIL_NUM', 'OP_UNIQUE_CARRIER', 'ORIGIN_CITY_NAME', 'ORIGIN_STATE_NM', 'DEST_CITY_NAME', 'DEST_STATE_NM', 'PRIOR_ORIGIN_CITY_NAME'

```
def label_encoder(col):
    col = LabelEncoder().fit_transform(col.astype(str))
    return col

for column in ['TAIL_NUM', 'OP_UNIQUE_CARRIER', 'ORIGIN_CITY_NAME', 'ORIGIN_STATE_NM',
               'DEST_CITY_NAME', 'DEST_STATE_NM', 'PRIOR_ORIGIN_CITY_NAME']:
    df[column] = label_encoder(df[column])
```

4.4. Features with high correlation to Target Variable



Save the clean file as Pre_Processed.csv

5. DATA MODELING using DASK

Dask is a 'parallel computing' framework that has been designed to run many computations at the same time, either using processes/threads on one machine (local) using multi-core CPUs, or many separate computers (cluster). Dask can efficiently perform parallel computations.

In order to use lesser memory during computations, Dask stores the complete data on the disk, and uses chunks of data (smaller parts, rather than the whole data) from the disk for processing. Dask supports the Pandas dataframe and Numpy array data structures to analyze large datasets. For a **single machine**, **Dask** allows us to run computations in parallel using either threads or processes. It was built to overcome the storage limitations of a single machine and extend the computation capability of Pandas, Numpy, and Scikit Learn with DASK equivalents. Dask also provides a real-time and responsive dashboard that shows several metrics like progress, memory use, etc. which is updated every 100ms.

Dask-ml: The Dask equivalent of Scikit-Learn is Dask-ML. Dask ML provides scalable machine learning algorithms in python which are compatible with scikit-learn.

Joblib: A user can perform parallel computing using scikit-learn (on a single machine) by setting the parameter njobs = -1. Scikit-learn uses Joblib to perform these parallel computations. Joblib is a library in python that provides support for parallelization

Dask-Search CV: Parallel to GridSearchCV in sklearn, Dask provides a library called Dask-search CV (Dask-search CV is now included in Dask ML). It merges steps so that there are less repetitions.

Load Cleaned Data using DASK

Dask Dataframe: Dask dataframe consists of multiple smaller pandas dataframes.

```
import dask.dataframe as dd
%time df = dd.read_csv('04_Pre_Processed.csv')
```

```
Wall time: 14 ms
```

The dataset is an Imbalanced dataset with only 18% of flights delayed.

```
df['DEP_DEL15'].compute().value_counts(normalize = True)
```

```
0.0    0.814386
1.0    0.185614
Name: DEP_DEL15, dtype: float64
```

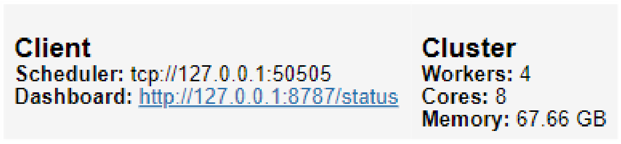
Create two dataframes with Predictor and Target variables. Split the data into train and test set with 70% train data and 30% test data.

Initialize dask distributed client (for single-machine parallel computing)

The first step is to import *client* from *dask.distributed*. This command will create a local scheduler and worker on machine.

```
# start a local Dask client
client1 = Client(n_workers=4,
                 threads_per_worker=2,
                 memory_limit='128GB')
```

```
client1
```



```
Client                                   Cluster
Scheduler: tcp://127.0.0.1:50505         Workers: 4
Dashboard: http://127.0.0.1:8787/status  Cores: 8
                                         Memory: 67.66 GB
```

BUILDING AND OPTIMIZING ML PIPELINE

The pipeline setup is composed of the following tasks:

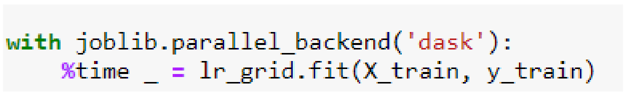
1. **Dimensionality Reduction**: we selected Principal Component Analysis (PCA) and a univariate feature selection algorithm as possible candidates.
2. **Classifier**: we apply Logistic Regression, Random Forest Classifier, Ada Boost Classifier

Optimizing Pipeline using GridSearchCV

Optimizing classifier and parameter by cross-validation grid-search. We can use pipeline as estimator which makes more power to GridSearchCV. However, it takes a lot of time for computing. As data is huge, it takes very long time.

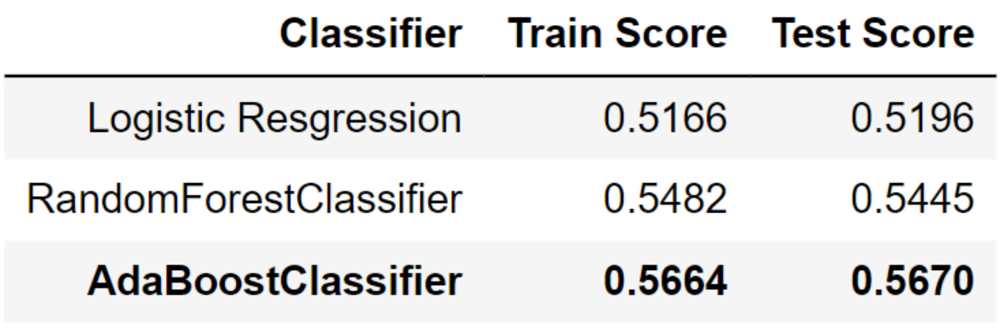
PARALLEL PIPELINE TUNING

sklearn provides parallel computing (on a single CPU) using *Joblib* in order to parallelize multiple estimators. Instantiate dask Joblib in the backend. Train models are using train dataset.



```
with joblib.parallel_backend('dask'):
    %time _ = lr_grid.fit(X_train, y_train)
```

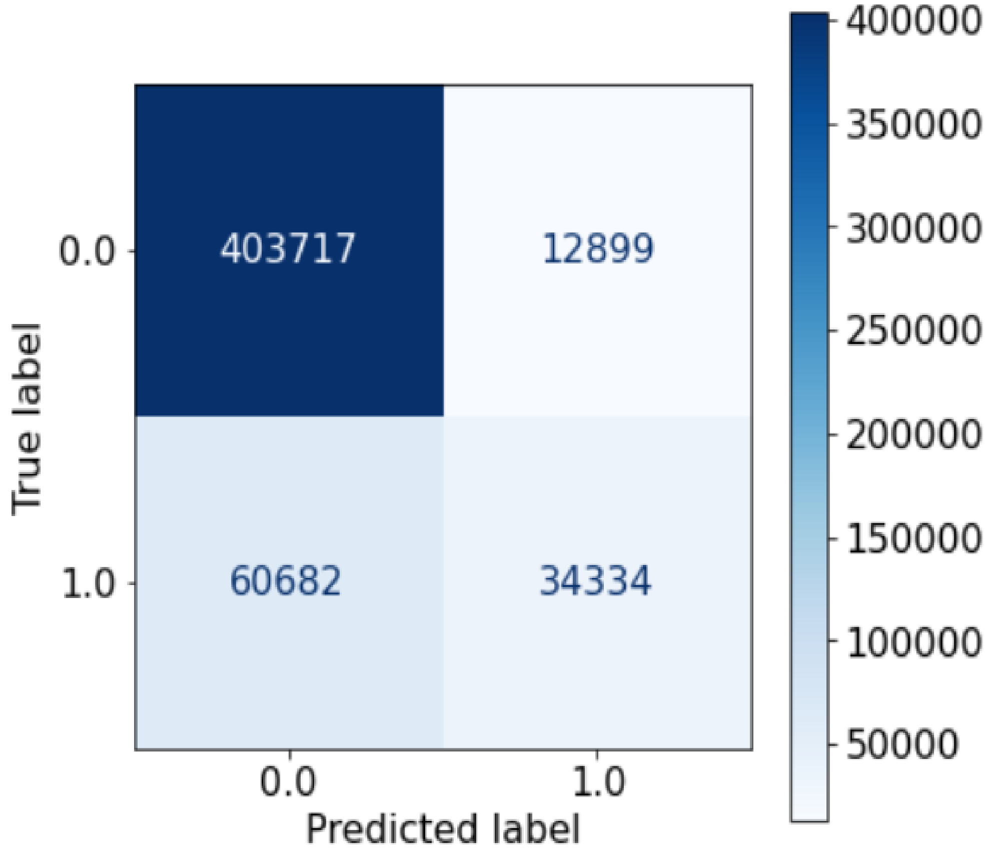
Model performance – Average Precision Score



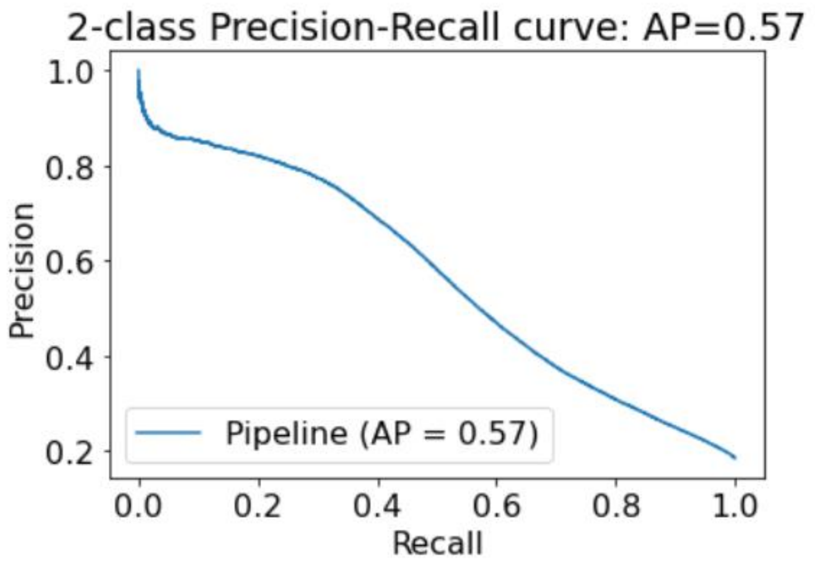
Classifier	Train Score	Test Score
Logistic Resgression	0.5166	0.5196
RandomForestClassifier	0.5482	0.5445
AdaBoostClassifier	**0.5664**	**0.5670**

From the average precision scores we can select AdaBoostClassifier as the best model

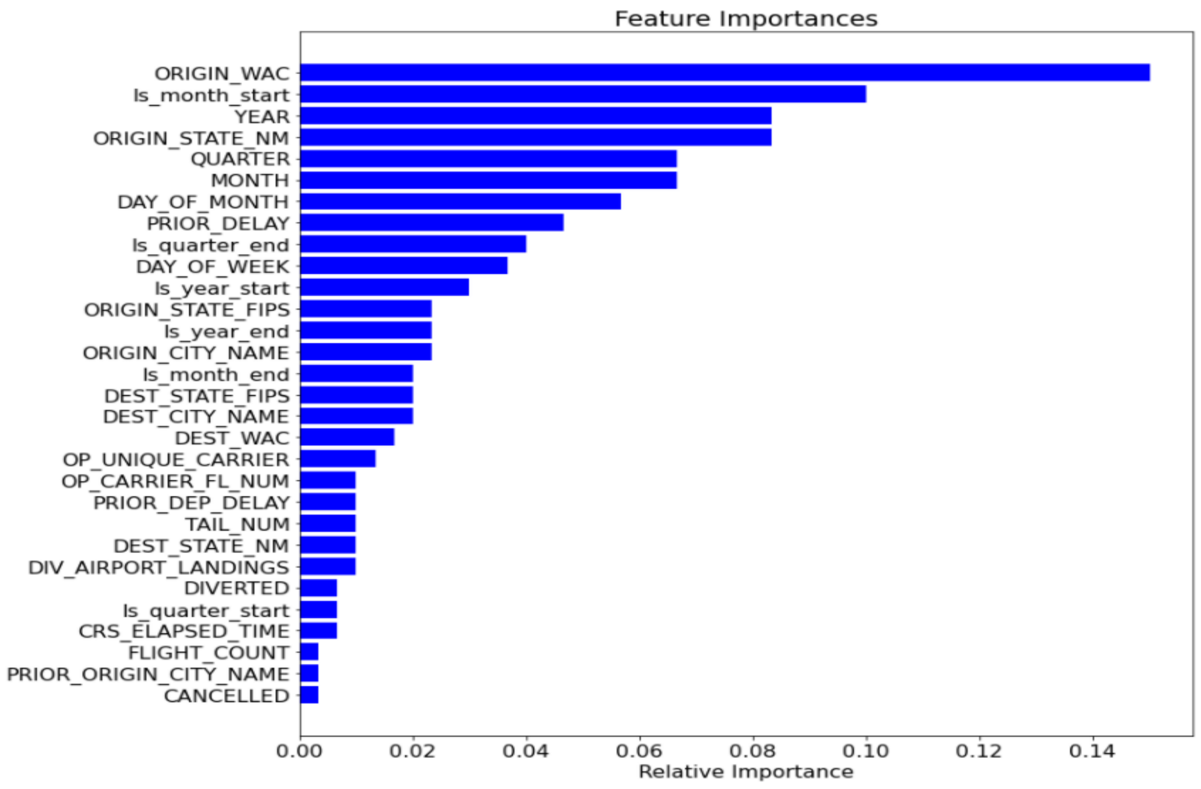
CONFUSION MATRIX - AdaBoostClassifier



PRECISION-RECALL CURVE - AdaBoostClassifier



FEATURE IMPORTANCE



6. CONCLUSIONS & NEXT STEPS

The dataset used for the project contains local flight data for the months January, February and March for the year 2019, does not take into considerations any weather conditions or natural calamities into account.

we plan to identify the significance of each feature by using stats models. Also, according to the Federal Aviation Administration, most of the delays in winter are due to surface winds, low ceiling and low visibility, whereas during summer the majority of delays is attributable to convective weather, low ceiling and associated low visibility (Federal Aviation Administration, 2017). It will be wise to include daily weather data, like wind, speed and precipitation rate, for each origin and destination location for each flight.