# AIRLINE DELAY DATA SET

Utkarsh Gupta - 2017MT10753

Gauri Gupta - 2017MT60207

Naman Jain - 2016MT10610

Have you ever been stuck in an airport because your flight was delayed or cancelled and wondered if you could have predicted it if you'd had more data.

DEPARTURES

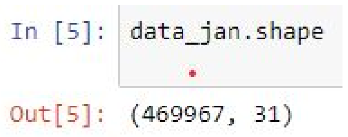
| TIME | DESTINATION | FLIGHT | GATE | REMARKS |
|---|---|---|---|---|
| 12:39 | LONDON | CL 903 | 31 | CANCELLED |
| 12:57 | SYDNEY | UQ5723 | 27 | CANCELLED |
| 13:08 | TORONTO | IC5984 | 22 | CANCELLED |
| 13:21 | TOKYO | AM 608 | 41 | DELAYED |
| 13:37 | HONG KONG | IC5471 | 29 | CANCELLED |
| 13:48 | MADRID | EK3941 | 30 | DELAYED |
| 14:19 | BERLIN | AM5021 | 28 | CANCELLED |
| 14:35 | NEW YORK | ON 997 | 11 | CANCELLED |
| 14:54 | PARIS | MG5870 | 23 | DELAYED |
| 15:10 | ROME | RI5324 | 43 | CANCELLED |

# Exploratory Data Analysis

## Overview

The given data set has the data of delays and cancellations of flights in the US for January 2015. In EDA, we will try to visualize the data using Python Code (Jupyter notebook) and tableau visualization. Various Plots are created so as to get a great idea of what's happening in the Dataset and what is the most important variable affecting the delays of the airlines. We are given 3 subsets named 'flights.csv', 'airlines.csv', 'airports.csv'. Flights.csv is the main dataset. The datasets are read into and merged later for purposes of better analysis of the data combined together.

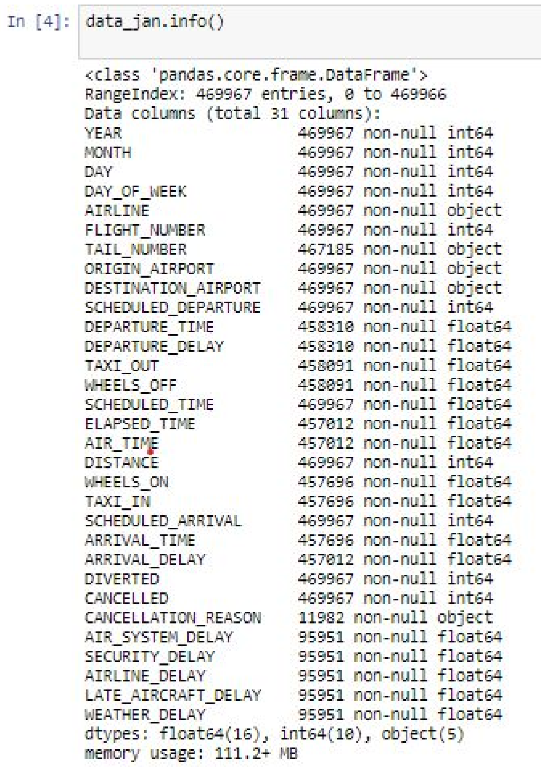
## Data Description



```
In [5]:  data_jan.shape

Out[5]:  (469967, 31)
```

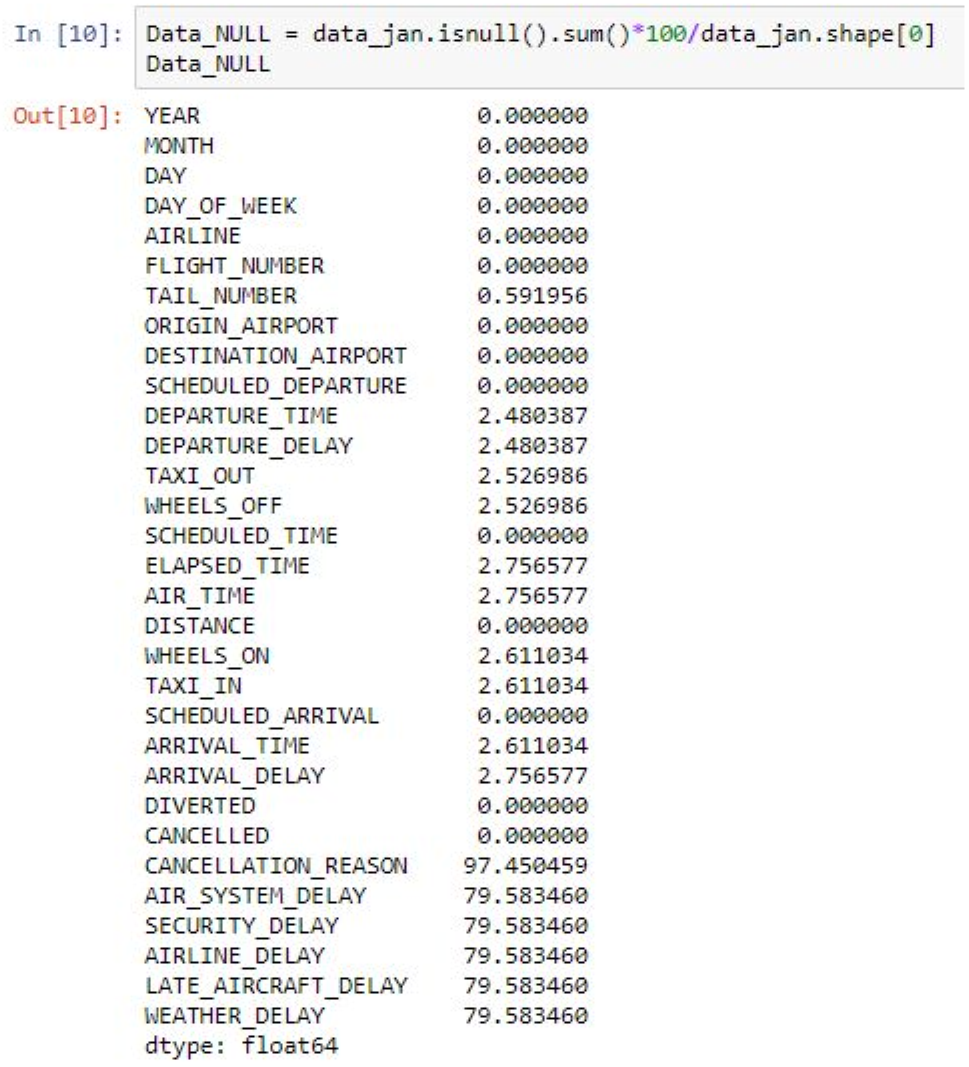
The Data Contains 31 columns and 469967 Rows. The attributes of Flights.csv are analyzed according to the data types as follows.



```
In [4]:  data_jan.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 469967 entries, 0 to 469966
Data columns (total 31 columns):
YEAR                   469967 non-null int64
MONTH                  469967 non-null int64
DAY                    469967 non-null int64
DAY_OF_WEEK            469967 non-null int64
AIRLINE                469967 non-null object
FLIGHT_NUMBER          469967 non-null int64
TAIL_NUMBER            467185 non-null object
ORIGIN_AIRPORT         469967 non-null object
DESTINATION_AIRPORT    469967 non-null object
SCHEDULED_DEPARTURE    469967 non-null int64
DEPARTURE_TIME         458310 non-null float64
DEPARTURE_DELAY        458310 non-null float64
TAXI_OUT               458091 non-null float64
WHEELS_OFF             458091 non-null float64
SCHEDULED_TIME         469967 non-null float64
ELAPSED_TIME           457012 non-null float64
AIR_TIME               457012 non-null float64
DISTANCE               469967 non-null int64
WHEELS_ON              457696 non-null float64
TAXI_IN                457696 non-null float64
SCHEDULED_ARRIVAL      469967 non-null int64
ARRIVAL_TIME           457696 non-null float64
ARRIVAL_DELAY          457012 non-null float64
DIVERTED               469967 non-null int64
CANCELLED              469967 non-null int64
CANCELLATION_REASON    11982 non-null object
AIR_SYSTEM_DELAY       95951 non-null float64
SECURITY_DELAY         95951 non-null float64
AIRLINE_DELAY          95951 non-null float64
LATE_AIRCRAFT_DELAY    95951 non-null float64
WEATHER_DELAY          95951 non-null float64
dtypes: float64(16), int64(10), object(5)
memory usage: 111.2+ MB
```

## Handling Null Values



```
In [10]: Data_NULL = data_jan.isnull().sum()*100/data_jan.shape[0]
         Data_NULL

Out[10]: YEAR                    0.000000
         MONTH                   0.000000
         DAY                     0.000000
         DAY_OF_WEEK             0.000000
         AIRLINE                 0.000000
         FLIGHT_NUMBER           0.000000
         TAIL_NUMBER             0.591956
         ORIGIN_AIRPORT          0.000000
         DESTINATION_AIRPORT     0.000000
         SCHEDULED_DEPARTURE     0.000000
         DEPARTURE_TIME          2.480387
         DEPARTURE_DELAY         2.480387
         TAXI_OUT                2.526986
         WHEELS_OFF              2.526986
         SCHEDULED_TIME          0.000000
         ELAPSED_TIME            2.756577
         AIR_TIME                2.756577
         DISTANCE                0.000000
         WHEELS_ON               2.611034
         TAXI_IN                 2.611034
         SCHEDULED_ARRIVAL       0.000000
         ARRIVAL_TIME            2.611034
         ARRIVAL_DELAY           2.756577
         DIVERTED                0.000000
         CANCELLED               0.000000
         CANCELLATION_REASON    97.450459
         AIR_SYSTEM_DELAY       79.583460
         SECURITY_DELAY         79.583460
         AIRLINE_DELAY          79.583460
         LATE_AIRCRAFT_DELAY    79.583460
         WEATHER_DELAY          79.583460
         dtype: float64
```

We can see that 97% of the values in the CANCELLATION_REASON column are null for which it is of less use while predicting Delays. Some other columns include 79.5% in Air System Delay, Security Delay, Airline Delay, Weather Delay etc. So I am going to create two Dataset which have no null values one is by removing all the null values irrespective of different types of Delays and other I am going to take the data set with respect to different types of delays. The first Dataset is named as Flights and the other one is named as Flight_Delays. The dataset Flights was further merged with 'airlines.csv' and 'airtime.csv' to obtain a comprehensive dataset to do analysis upon.

The departure time above is not very informative so we are going to change even hours to minutes (so that we have all times in minutes) to get better graphs etc. The date was also changed to the date month year format.

```python
def Format_Hourmin(hours):
        if hours == 2400:
            hours = 0
        else:
            min = hours%100
            hours = (hours-min)/100
            time = int(60*hours+min)
            return time
```

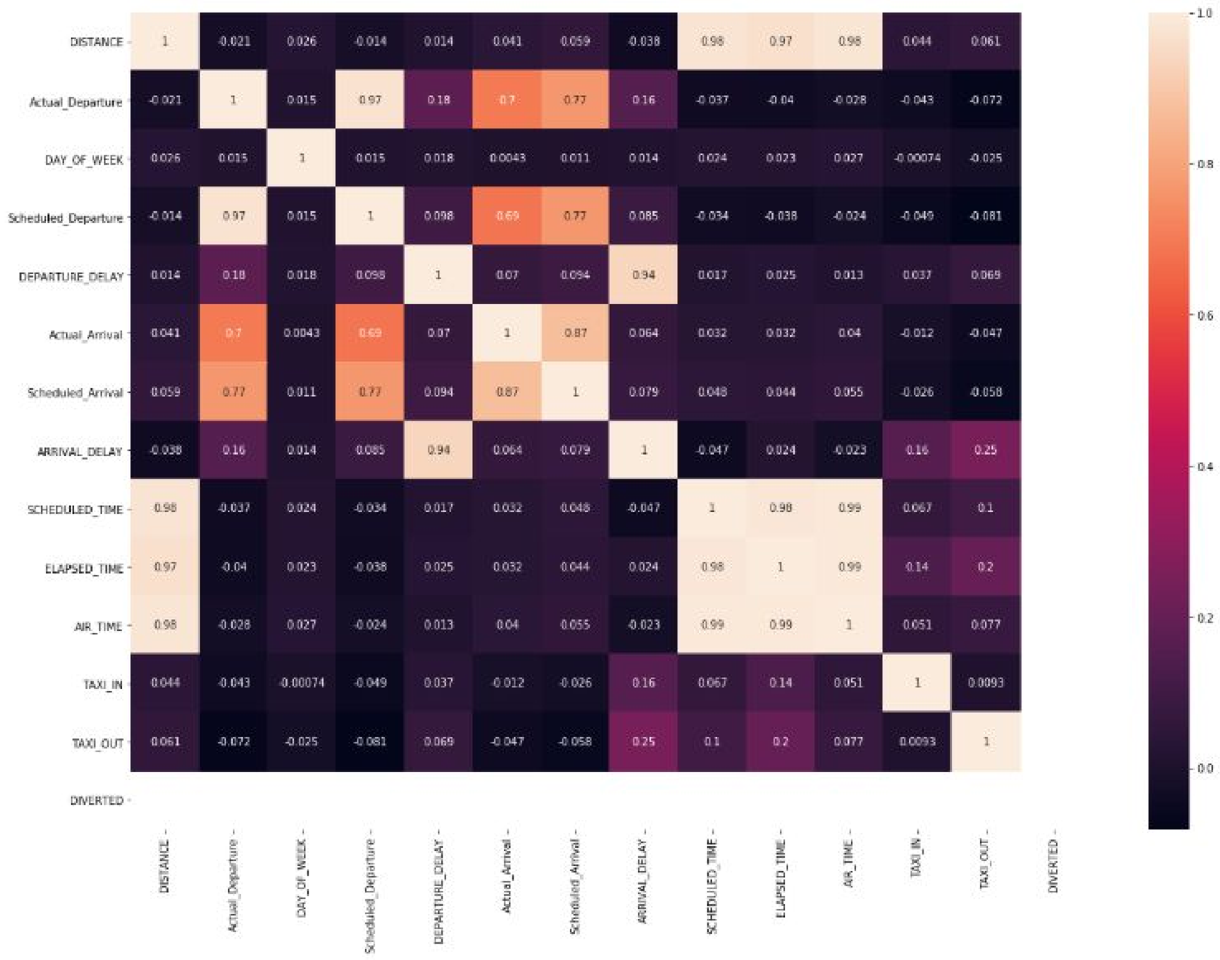
```python
# Creating Date in the Datetime format
data3['Date'] = pd.to_datetime(data3[['YEAR','MONTH','DAY']])
data3.Date
```

Thus after processing the data as explained above, the information for the dataset Flights is as follows:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 456266 entries, 0 to 456455
Data columns (total 22 columns):
AIRLINE                456266 non-null object
Org_Airport_Name       456266 non-null object
Origin_city            456266 non-null object
Dest_Airport_Name      456266 non-null object
Destination_city       456266 non-null object
ORIGIN_AIRPORT         456266 non-null object
DESTINATION_AIRPORT    456266 non-null object
DISTANCE               456266 non-null int64
Actual_Departure       456266 non-null float64
Date                   456266 non-null datetime64[ns]
DAY_OF_WEEK            456266 non-null int64
Scheduled_Departure    456266 non-null int64
DEPARTURE_DELAY        456266 non-null float64
Actual_Arrival         456266 non-null float64
Scheduled_Arrival      456266 non-null int64
ARRIVAL_DELAY          456266 non-null float64
SCHEDULED_TIME         456266 non-null float64
ELAPSED_TIME           456266 non-null float64
AIR_TIME               456266 non-null float64
TAXI_IN                456266 non-null float64
TAXI_OUT               456266 non-null float64
DIVERTED               456266 non-null int64
dtypes: datetime64[ns](1), float64(9), int64(5), object(7)
memory usage: 80.1+ MB
```

## Heat Map

A heatmap is a graphical representation of data that uses a system of color-coding to represent different values. The lighter values (closer to white) correspond to correlation value closer to 1 and dark areas (closer to black) represent correlation close to 0.



I am dividing the different correlations into two parts, the **positive correlations** (higher than *0.6* ). The results are listed in the list below: Positive correlations between:

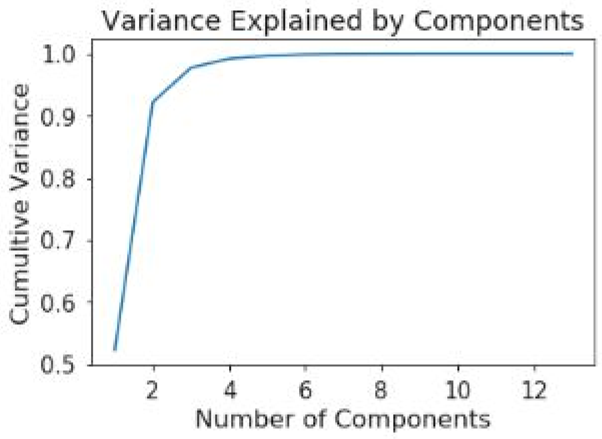
- DEPARTURE_DELAY and ARRIVAL_DELAY

## Dimensionality Reduction:

The given data contains 22 attributes for each data point. As seen in the heatmap, there is a high correlation in certain cases. Thus we will try to reduce the dimensionality of the data and try to find out the attributes relevant to Arrival Delay and Departure Delay (which are very closely correlated).

### *Principal Component Analysis*

The main idea of **principal component analysis** (**PCA**) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The technique increases interpretability and minimizes information loss.

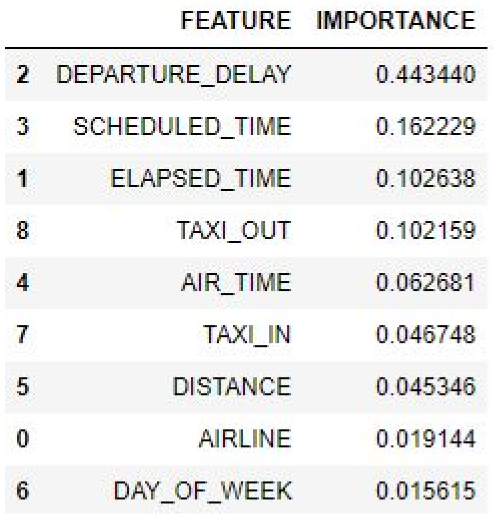


As we see, after the first 5 attributes, the cumulative variance reaches very close to 1 and hence very less information is lost. This is as expected as Departure Delay can be modeled as Actual_Departure - Scheduled_Departure etc. Thus these factors traditionally should not be taken into account while looking at dependence of DEPARTURE_DELAY on other elements. Thus we will try to apply further other techniques to do dimensionality reduction.

### *Random Forest Classification*

I will classify the data into delayed and not delayed data and define a label (DELAYED) for that in the dataframe. Afterward I will show the feature importance for the given attributes for correlation with ARRIVAL_DELAY (which already is correlated to DEPARTURE_DELAY).

Random Forests are often used for feature selection in data science.The reason is because the tree-based strategies used by random forests naturally rank by how well they improve the purity of the node. This means decrease in impurity over all trees (called gini impurity). Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features.



|  | FEATURE | IMPORTANCE |
|---|---|---|
| 2 | DEPARTURE_DELAY | 0.443440 |
| 3 | SCHEDULED_TIME | 0.162229 |
| 1 | ELAPSED_TIME | 0.102638 |
| 8 | TAXI_OUT | 0.102159 |
| 4 | AIR_TIME | 0.062681 |
| 7 | TAXI_IN | 0.046748 |
| 5 | DISTANCE | 0.045346 |
| 0 | AIRLINE | 0.019144 |
| 6 | DAY_OF_WEEK | 0.015615 |

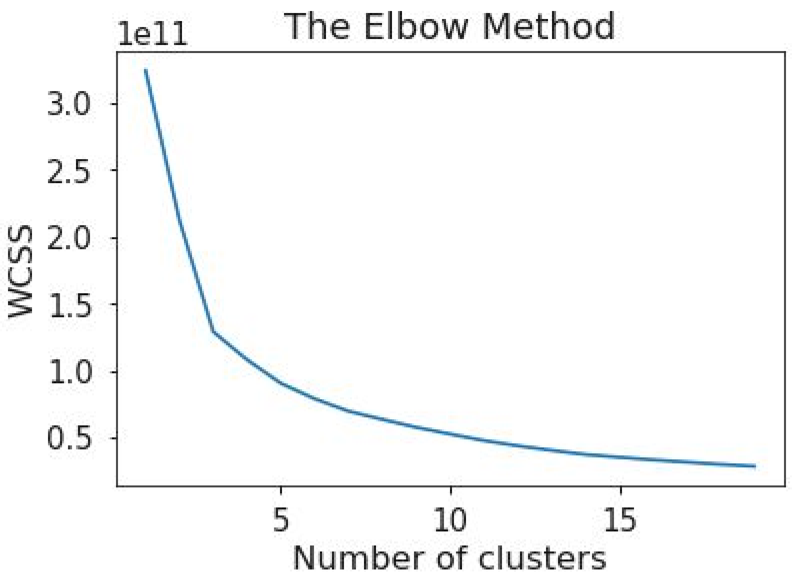
This feature importance is in accordance with what we observed from the correlation matrix and hence the relevant fields are chosen for prediction (next section of the report).

## Visualization

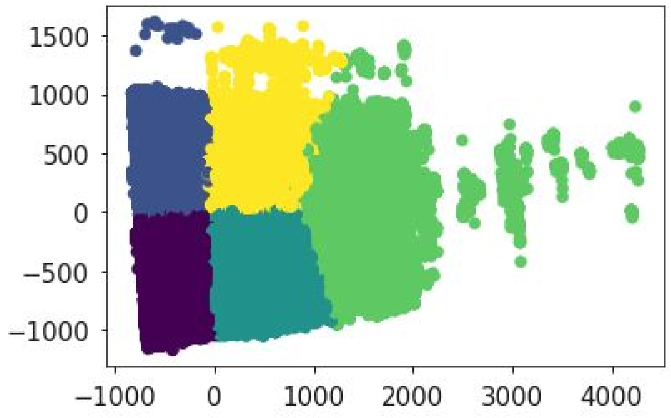
### *k-Means Clustering*

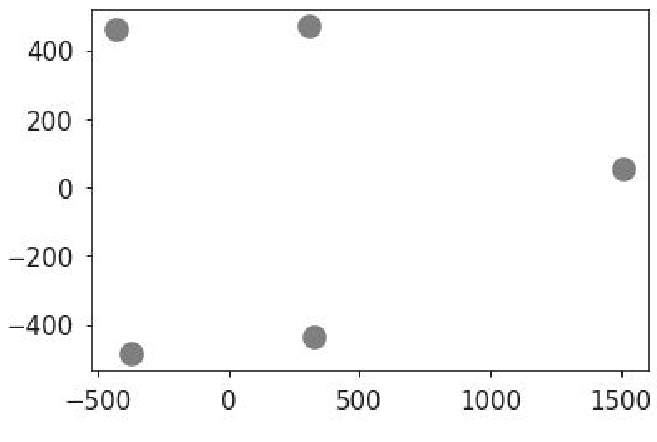
K-Means Clustering is an unsupervised machine learning algorithm. In contrast to traditional supervised machine learning algorithms, K-Means attempts to classify data without having first been trained with labeled data. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

The **elbow method** is a heuristic method of interpretation and validation of consistency within cluster analysis designed to help find the appropriate number of clusters in a dataset. Thus we apply the elbow method to find the number of clusters ideal for the data.
We chose to do clustering on the PCA transformed data (obtained while doing PCA above) and apply the algorithms on this transformed dataset.



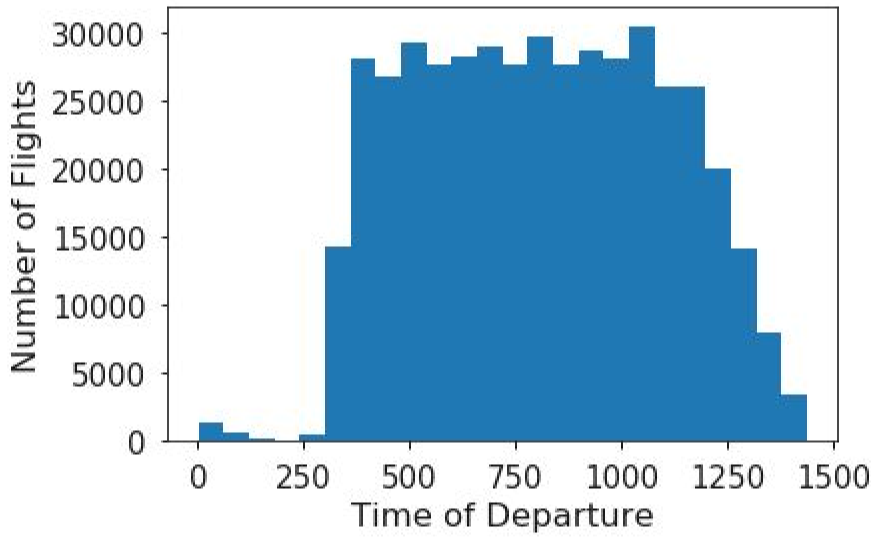
Thus we chose the number of clusters as 5.





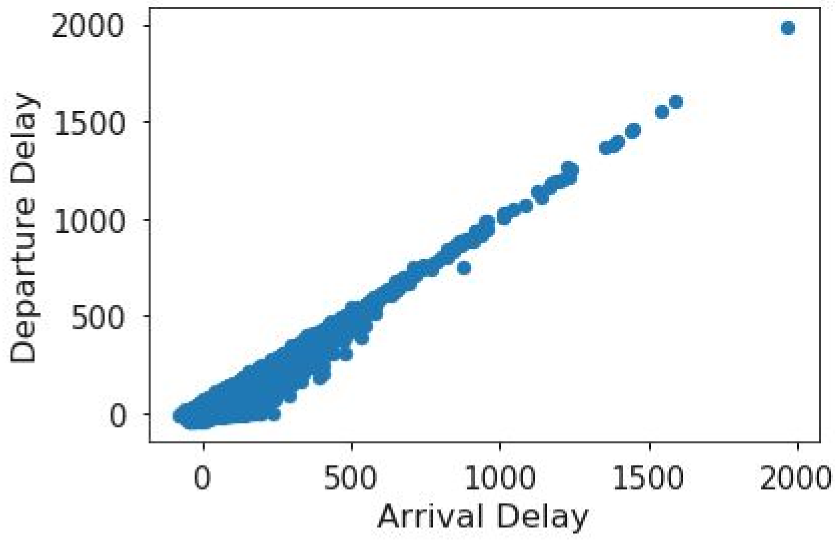
***Scatter Plots***

Number of Flights vs Time of Departure



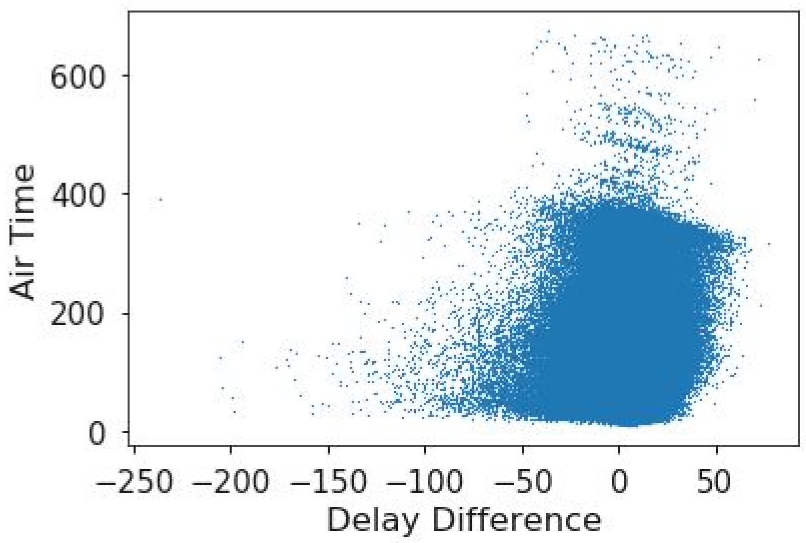
A histogram was plotted for the number of flights that depart every hour (number of bins = 24) and we observe that in the data, very few number of flights are scheduled till 300 minutes after midnight that is from 00:00 to 05:00 compared to other times during the day.

Departure Delay vs Arrival Delay



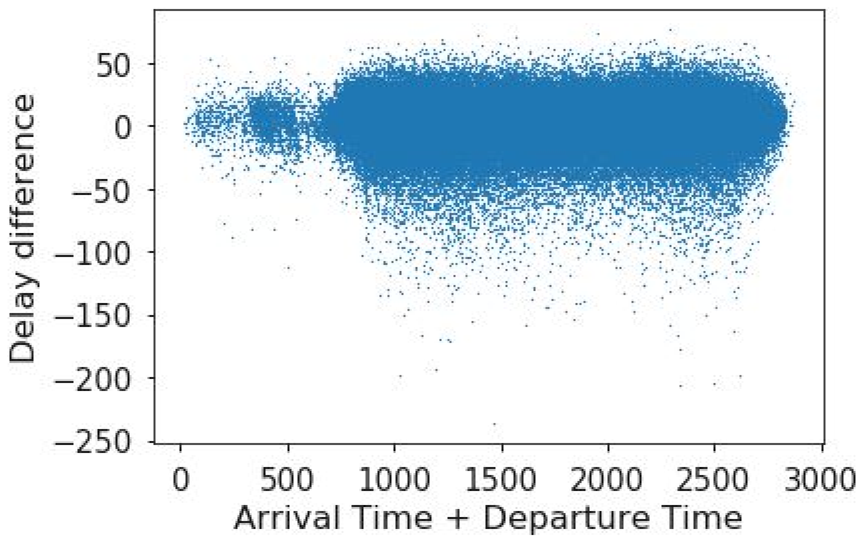
The plot obtained is almost linear. This is in accordance with their correlation value (in the heat map above) of 0.94 being very close to 1.

Air Time vs Delay Difference



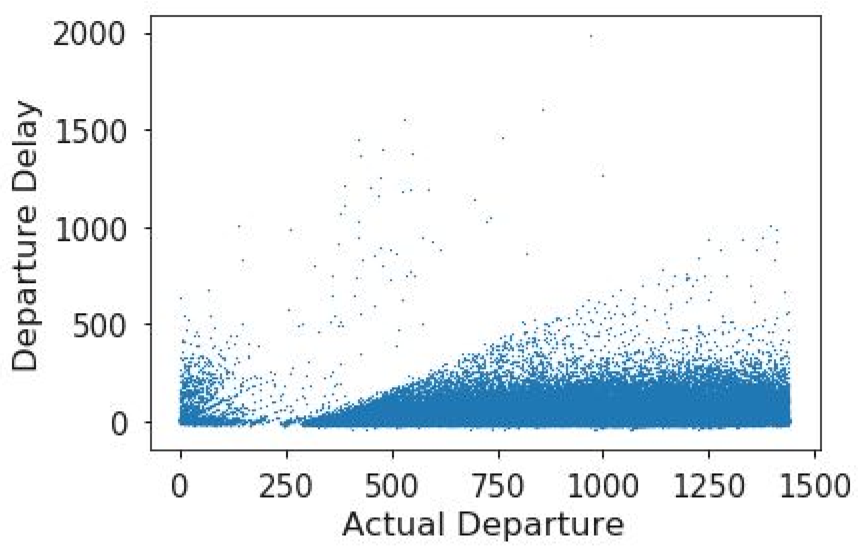
Delay difference was defined by the difference between Departure Delay and Arrival Delay time and hence indicates roughly the delay caused due to delay along the air time. We observe that the delay difference is close to zero for longer air time. The width of the graph reduces with Air Time. We infer from this graph that even if there is delay being caused, it is being recovered for long air times with more frequency as compared to less air time.

Delay Difference vs Time



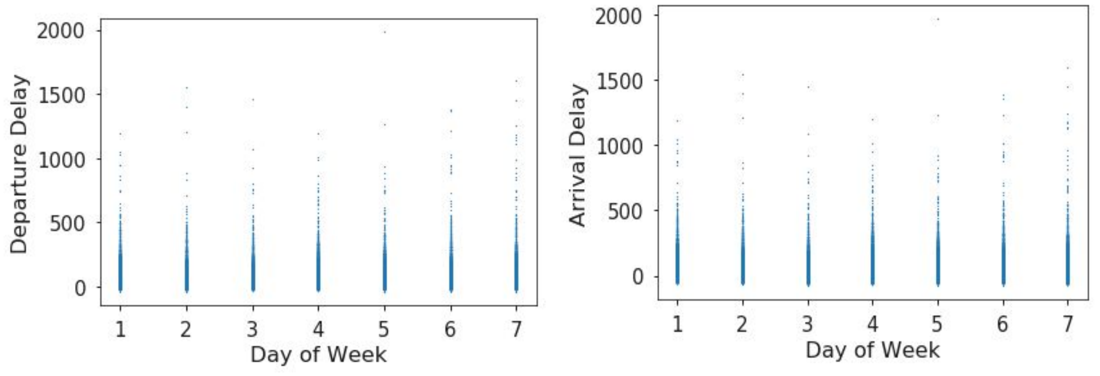
The half of x axis time of midpoint between Actual Arrival Time and Actual Departure Time and thus corresponds to the average of the time the plane was in the air. This was plotted on the x axis (in minutes) and the Delay Difference (as in the previous graph) on the y axis. A dip in variation of delay difference is observed around 600 minutes that is at 5:00 (since 600/2*60 = 5). This is in consistency with the reduced frequency of flights observed leaving between 00:00 to 05:00 as discussed in the histogram above as planes leaving between 00:00 and 05:00 will have their in flight midtime around this time.

Departure Delay vs Actual Departure



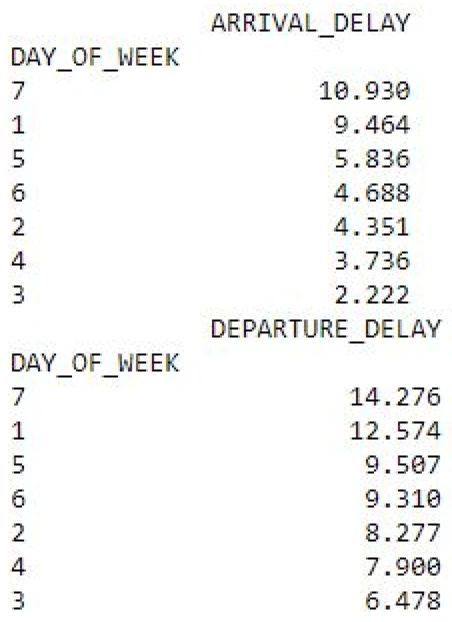
A dip in both the frequency and variance of Departure Delay is observed around 250 minutes that is 04:00. This again is in accordance with the histogram as very few flights are scheduled to leave at this time. The lower variation might also suggest that very few number of scheduled departures imply less delayed time.

Arrival and Departure Delays vs Day Of Week

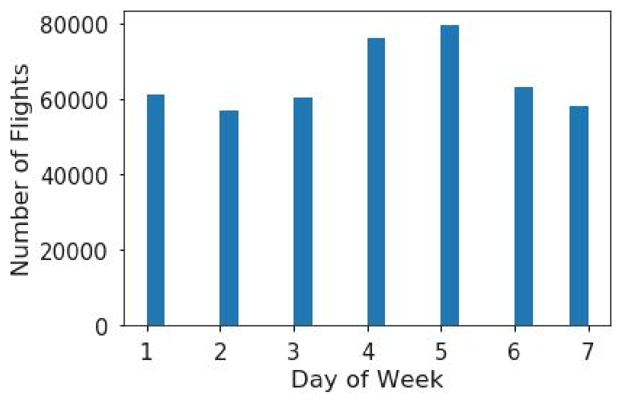


The delays are not equal/similar everyday. More delay is observed on Day of Week 7 that is Sunday and the least is observed on Day 3 that is Wednesday.

The number of flights on Wednesday and Sunday are the same (as can be observed from the histogram below). Since the variation in delays is the minimum and maximum on these days respectively, we infer that the Departure Delay (and Arrival Delay) are not correlated with Day of Week which is coincides with the importance allotted to the Day of Week from Random Forest Classifier (importance = 0.015) and the heat map (correlation factor = 0.014).



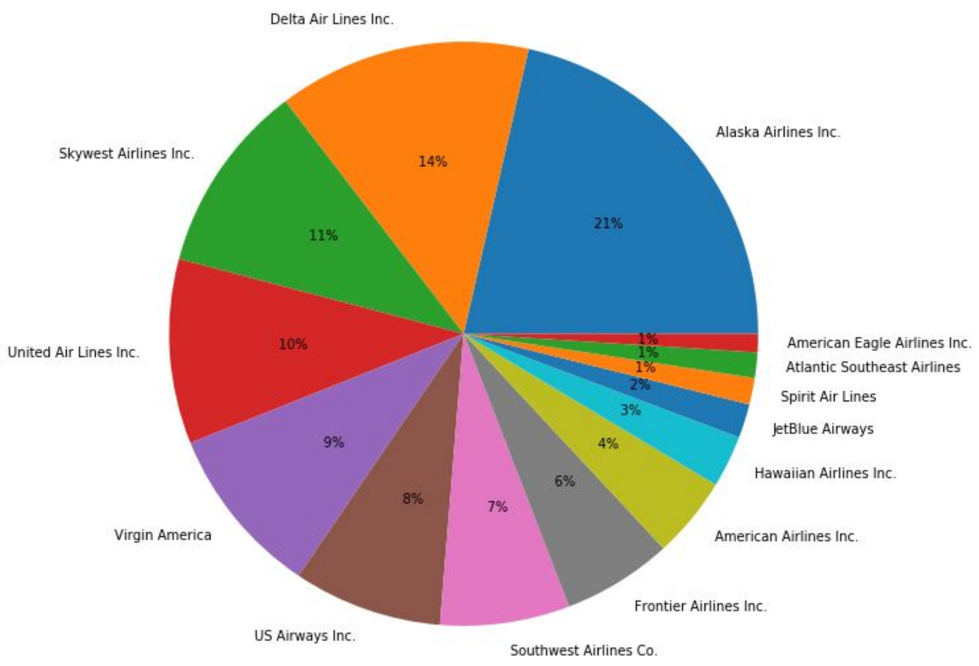
```
             ARRIVAL_DELAY
DAY_OF_WEEK
7                   10.930
1                    9.464
5                    5.836
6                    4.688
2                    4.351
4                    3.736
3                    2.222
             DEPARTURE_DELAY
DAY_OF_WEEK
7                     14.276
1                     12.574
5                      9.507
6                      9.310
2                      8.277
4                      7.900
3                      6.478
```

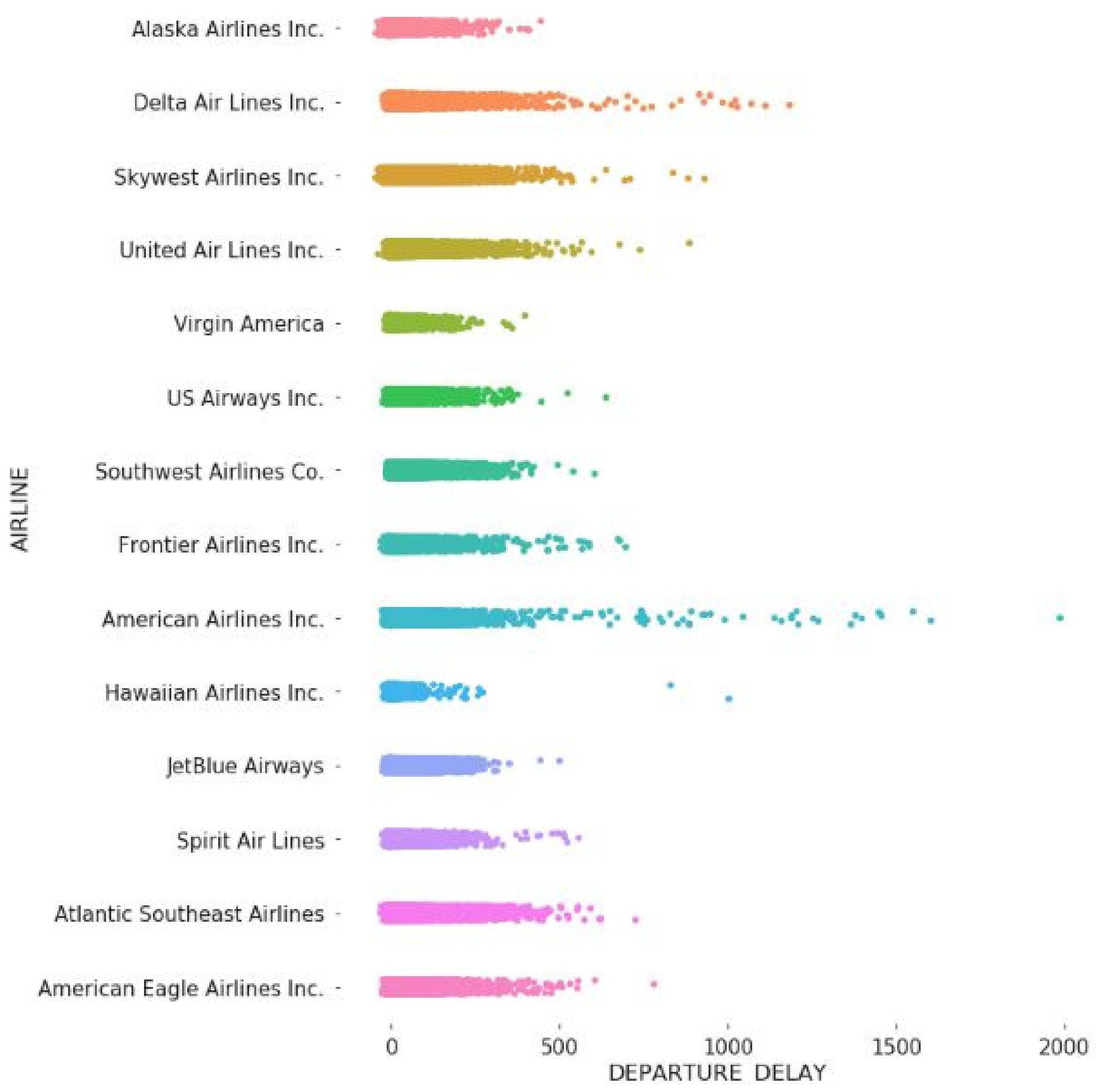


## *Airline Visualization*

We also plot graphs to observe the relation of Airlines and Origin City on the Delays.

Airlines





```
                              ARRIVAL_DELAY
AIRLINE
Frontier Airlines Inc.               18.459
American Eagle Airlines Inc.         18.141
Spirit Air Lines                     11.411
Skywest Airlines Inc.                10.872
Atlantic Southeast Airlines           8.509
JetBlue Airways                       7.342
American Airlines Inc.                6.942
United Air Lines Inc.                 6.353
Hawaiian Airlines Inc.                3.513
Southwest Airlines Co.                3.385
US Airways Inc.                       3.086
Virgin America                        1.414
Alaska Airlines Inc.                 -0.324
Delta Air Lines Inc.                 -2.101
```

```
                              DEPARTURE_DELAY
AIRLINE
Frontier Airlines Inc.                 17.975
American Eagle Airlines Inc.           15.978
United Air Lines Inc.                  13.886
Spirit Air Lines                       13.104
Skywest Airlines Inc.                  11.984
American Airlines Inc.                 10.536
JetBlue Airways                         9.987
Atlantic Southeast Airlines             9.652
Southwest Airlines Co.                  9.451
Virgin America                          6.902
Delta Air Lines Inc.                    5.841
US Airways Inc.                         5.158
Alaska Airlines Inc.                    3.078
Hawaiian Airlines Inc.                  1.312
```

The average Departure and Arrival Delays of all the airlines were calculated (as above) and a scatter plot of Departure Delay of various airlines was plotted as above.

The average Departure and Arrival Delays for Alaska Airlines is very low and so is the variation as observed in the scatter plot. The share of Alaska Airlines of the number of flights is also 21%. Even though the number of flights is so large, Alaska airlines still performs really well.

## *Air Routes and Delay Visualization*

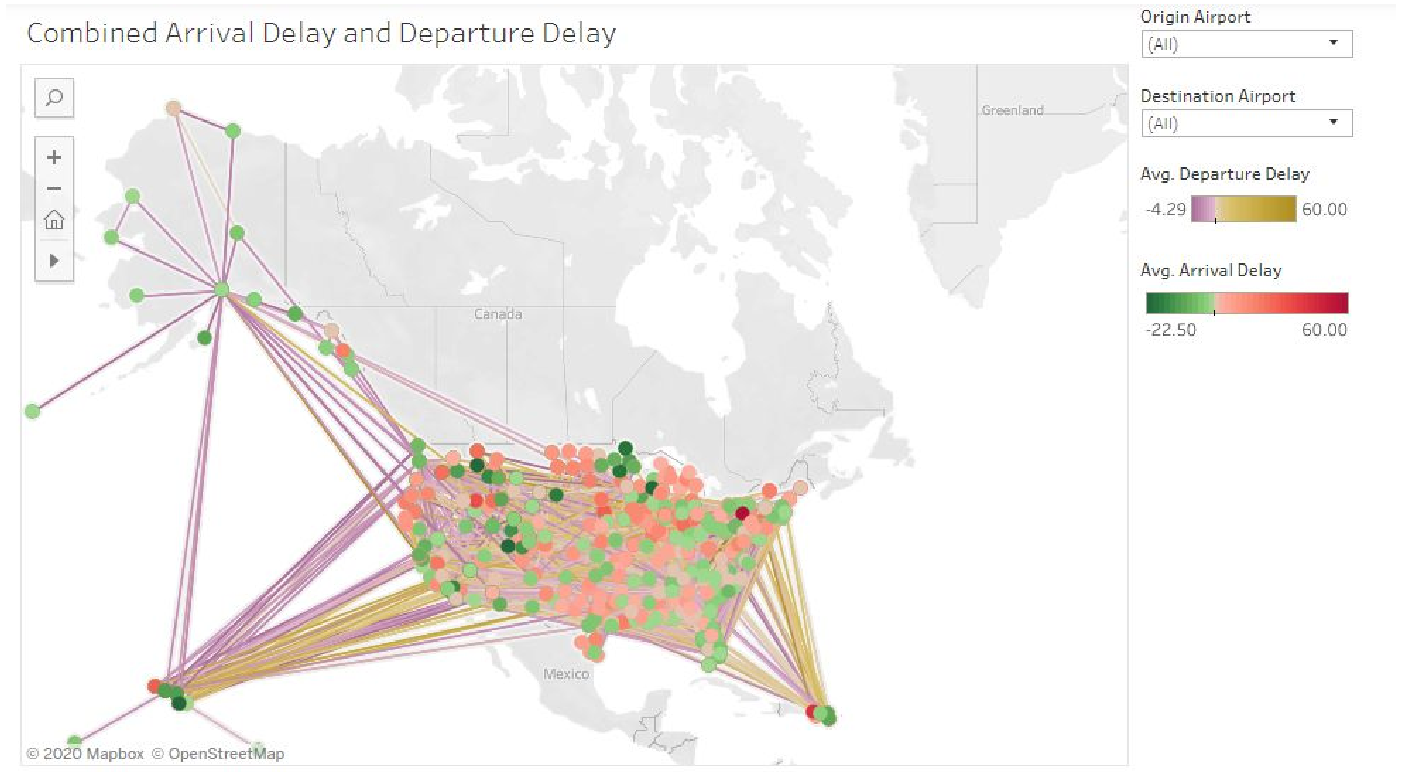


Tableau data visualization was done for the Air Routes. The color of the airport/air route denotes the average arrival/departure delay (color scheme indicated on the graph). It can be seen that most flights on longer routes have shorter delays and most air routes to and from Alaska correspond to low average delays.

# Data Analysis and Preprocessing

## Overview

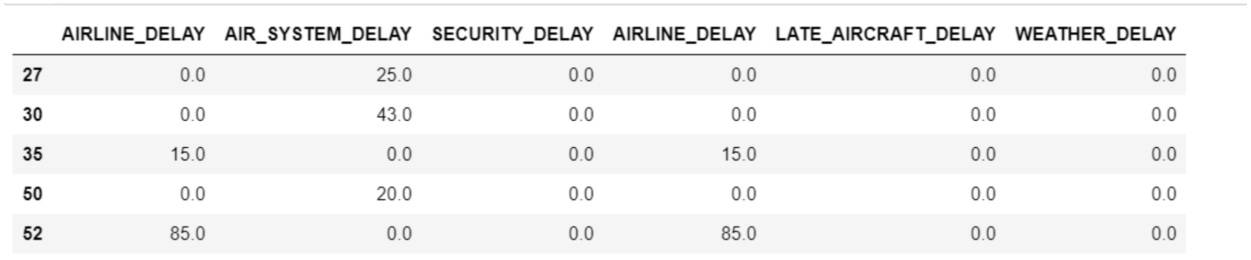
There are a lot of time values whether it is a time value or just a minute value but there are all stored as a numerical value: This list can also be divided into two types of time vales. The one that is actually real-time values and the one that is only minute values.

There was a need to convert them all to datetime. In addition, it seems to be helpful to write/use a function for this conversion.

## Handling the Null Values

After we converted the necessary time values to a DateTime datatype, we needed to check our data according to its integrity. Null values or missing data are often occurring data states that need to be handled. There are several methods to deal with null value data or missing data.

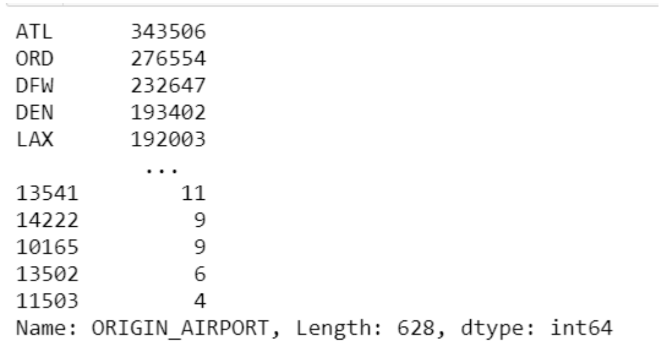
One option is to delete the corresponding rows. Another case of handling missing or null value data is to reconstruct the missing data according to information from other columns.



| | AIRLINE_DELAY | AIR_SYSTEM_DELAY | SECURITY_DELAY | AIRLINE_DELAY | LATE_AIRCRAFT_DELAY | WEATHER_DELAY |
|---|---|---|---|---|---|---|
| **27** | 0.0 | 25.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **30** | 0.0 | 43.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **35** | 15.0 | 0.0 | 0.0 | 15.0 | 0.0 | 0.0 |
| **50** | 0.0 | 20.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **52** | 85.0 | 0.0 | 0.0 | 85.0 | 0.0 | 0.0 |

## Handling categorical data:

Categorical values need to be converted into numeric values to use a prediction model. There is a way to convert all categorical data into numeric values, it’s called **One-Hot Encoding**. This approach will line in all categorical values in separate columns, creates a new column and matches every occurrence of the categorical value with 1 or 0 for non-occurrences. We performed label encoding of AIRLINE, ORIGIN_AIRPORT, DESTINATION_AIRPORT and CANCELLATION_REASON.



```
ATL      343506
ORD      276554
DFW      232647
DEN      193402
LAX      192003
          ...
13541        11
14222         9
10165         9
13502         6
11503         4
Name: ORIGIN_AIRPORT, Length: 628, dtype: int64
```

# Data Prediction

## Choosing the Prediction Target

We choose the ARRIVAL_DELAY as the target and studied various regressor models to predict the exact minutes delayed or arrived in time.

## Hyperparameter Tuning

### (By applying Grid Search)

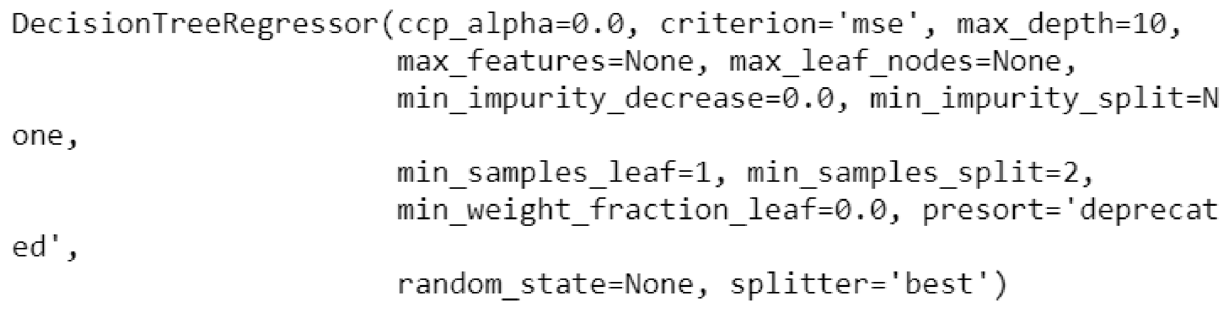
Grid search works by trying every possible combination of parameters we want to try in our model. Those parameters are each tried in a series of cross-validation passes.

## Decision Tree Regressor

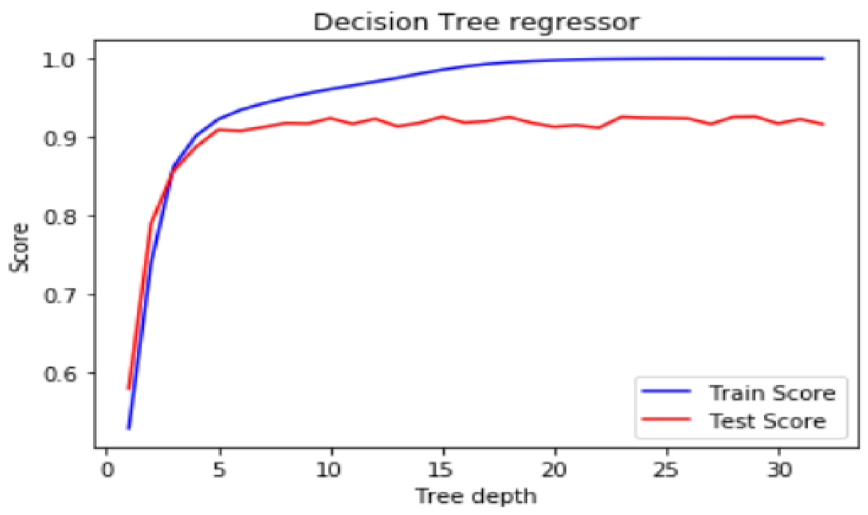
The list of various hyperparameters:

1. **max_depth**
2. **min_samples_split**
3. **min_samples_leaf**
4. **min_weight_fraction_leaf**
5. **max_features**

   We studied the effects of various hyperparameters on the accuracy of the decision tree regressor and obtained the following results.

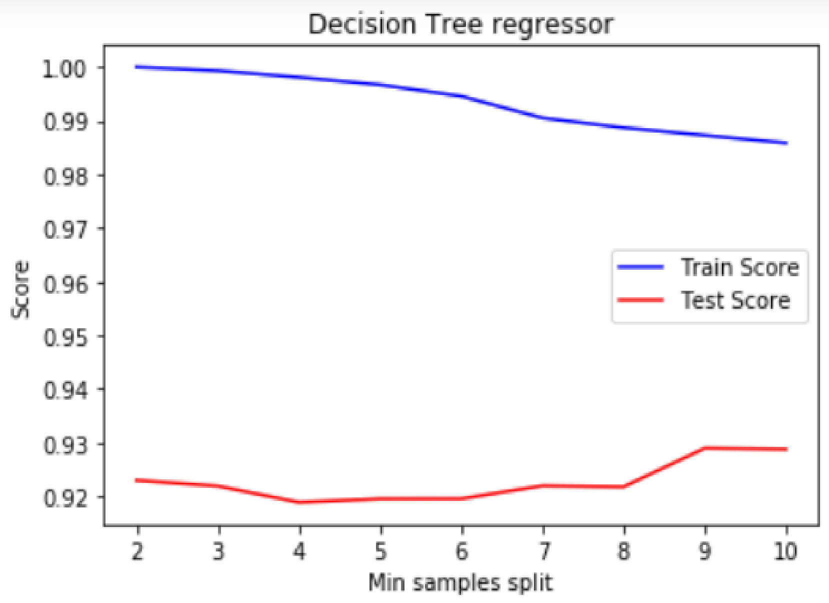


```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=10,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=N
one,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                      random_state=None, splitter='best')
```

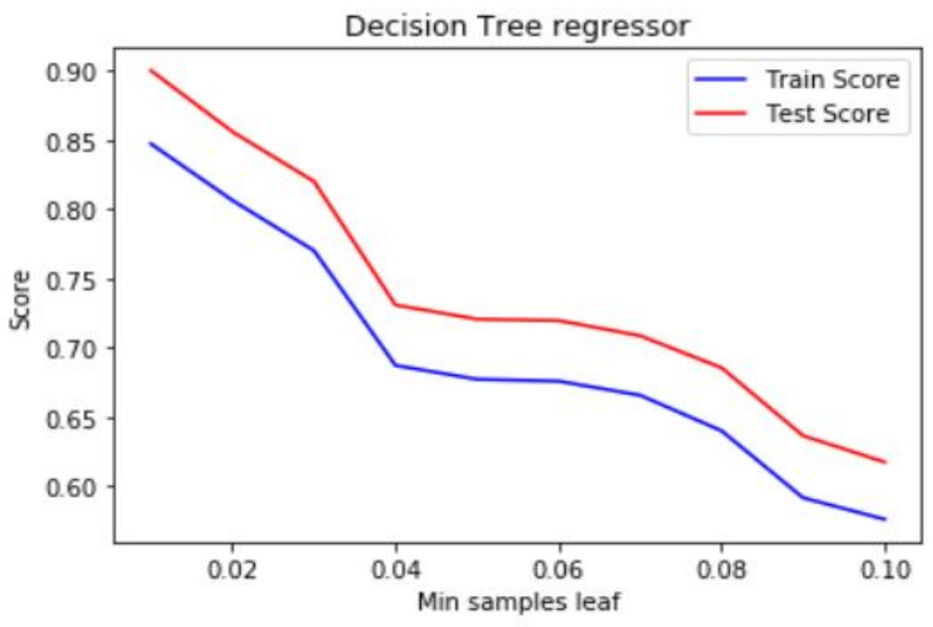


Both training and testing accuracy increased as we increased the tree depth and after reaching a point, it became more or less constant.

In general, the deeper we allow our tree to grow, the more complex our model will become because we will have more splits and it captures more information about the data and this is one of the root causes of overfitting in decision trees because our model will fit perfectly for the training data and will not be able to generalize well on test set. So, if our model is overfitting, reducing the number for max_depth is one way to combat overfitting. It is also bad to have a very low depth because our model will underfit. So, to find the best value, we experiment because overfitting and underfitting are very subjective to a dataset, there is no one value fits all solution.



min_samples_split is used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Too high values can also lead to under-fitting hence depending on the level of underfitting or overfitting, we tuned the values for min_samples_split.

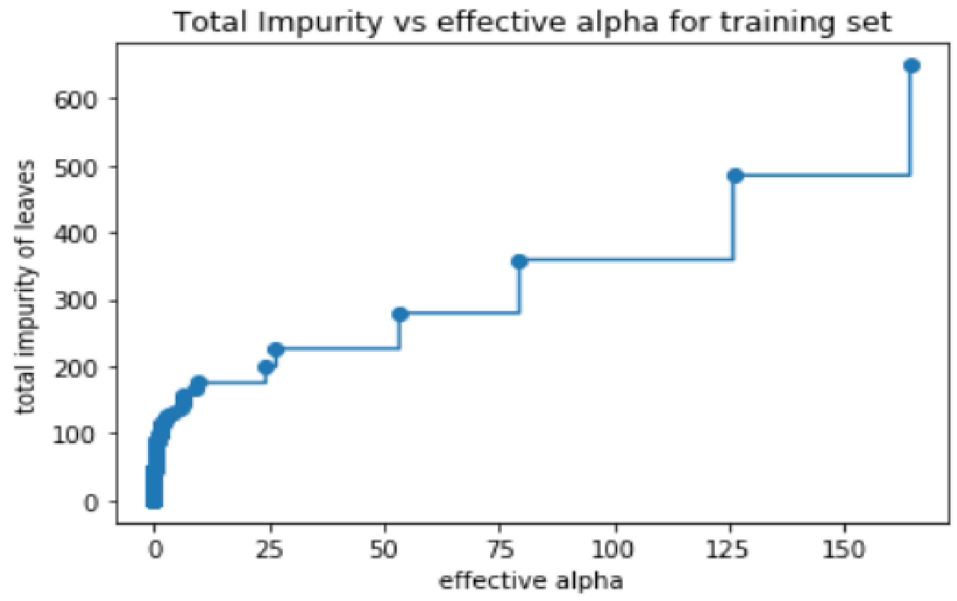


min_samples_leaf can be used to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered. They also say a very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data and this should make sense.

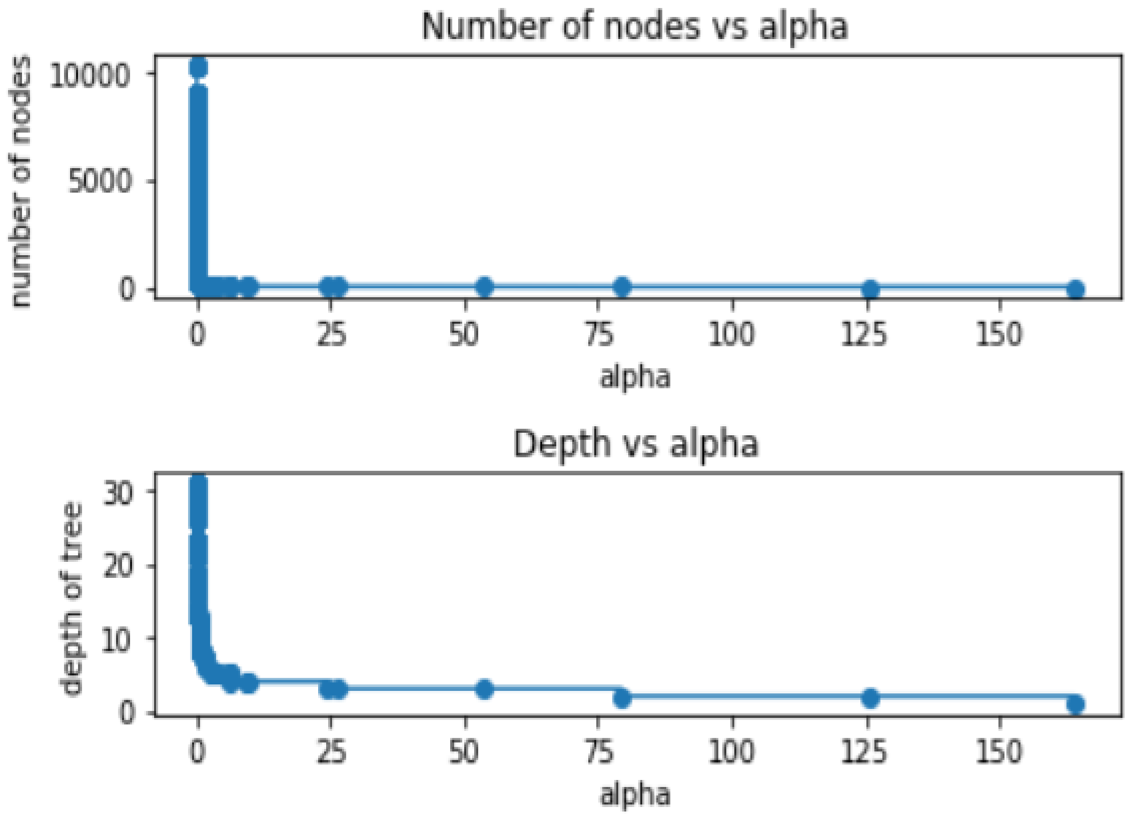
## Pruning of Decision Tree

Growing the tree beyond a certain level of complexity leads to overfitting. Pruning helps us to avoid overfitting. Generally it is preferred to have a simple model, it avoids overfitting issue. Any additional split that does not add significant value is not worth while.

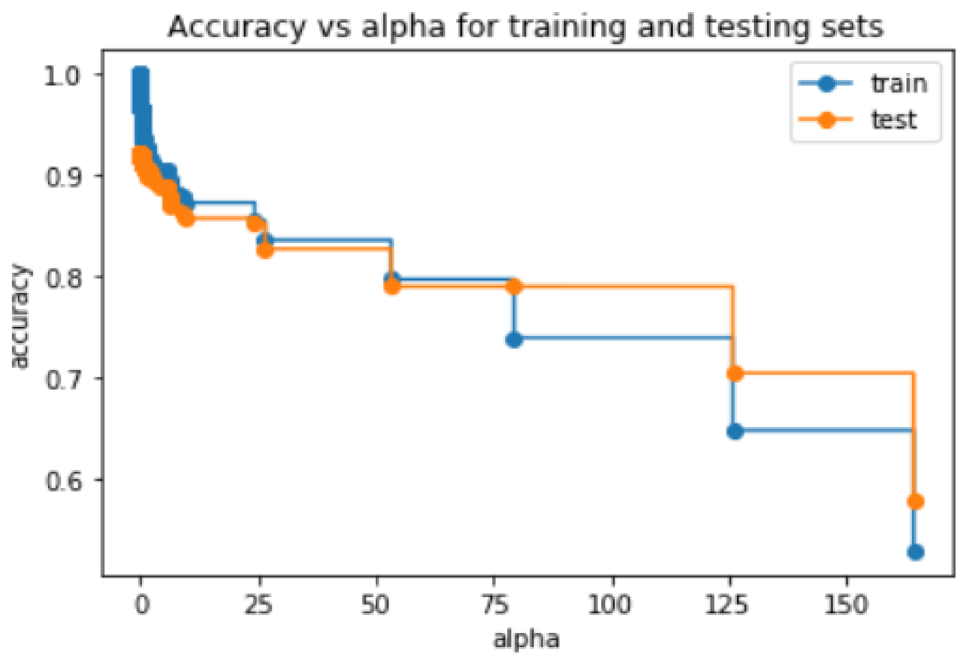
Minimal cost complexity pruning recursively finds the node with the “weakest link”. The weakest link is characterized by an effective alpha, where the nodes with the smallest effective alpha are pruned first.



As alpha increases, more of the tree is pruned, which increases the total impurity of its leaves.



Here we see that the number of nodes and tree depth decreases as alpha increases.



When ccp_alpha is set to zero and keeping the other default parameters of Decision tree, the tree overfits, leading to a 100% training accuracy and 92% testing accuracy. As alpha increases, more of the tree is pruned, and both training and testing accuracy reduces.

### Grid Search Results:

```
Best parameters : {'max_depth': 30, 'max_features': 0.8, 'min_sampl
es_leaf': 1, 'min_samples_split': 4}
```

```
----------------- TRAINING ------------------------
r-squared score:  0.9611268699167007
------------------- TEST --------------------------
r-squared score:  0.9163526429928677
```

## Random Forest Regressor

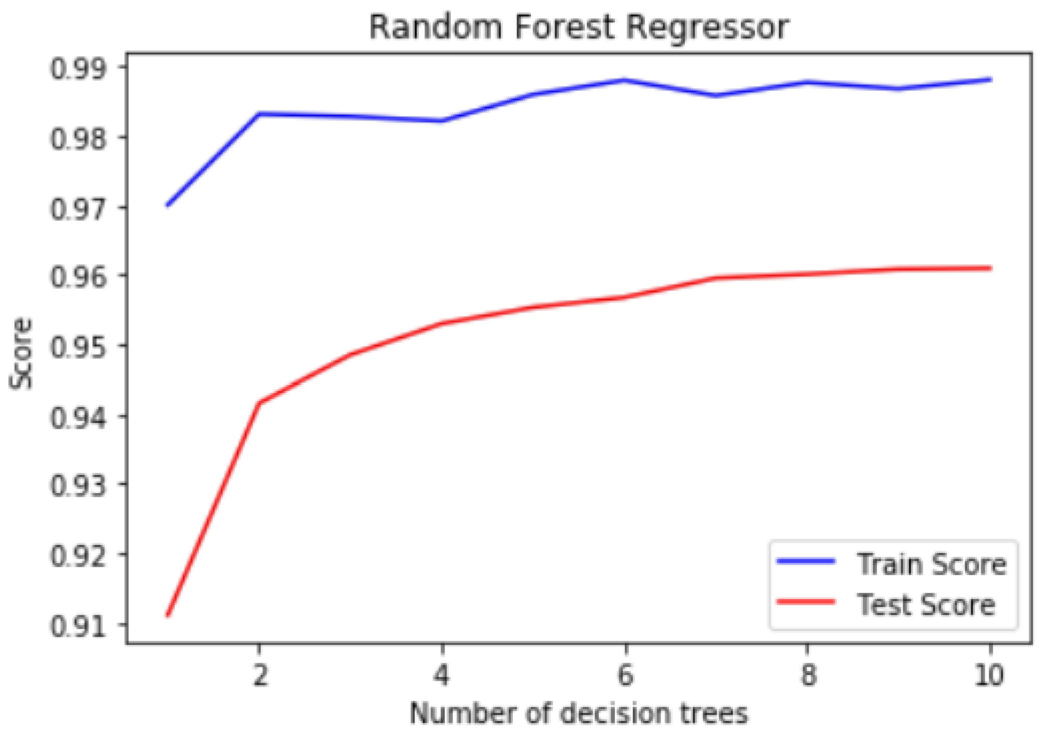
**List of hyperparameters:**

1. **n_estimators**
2. **max_depth**

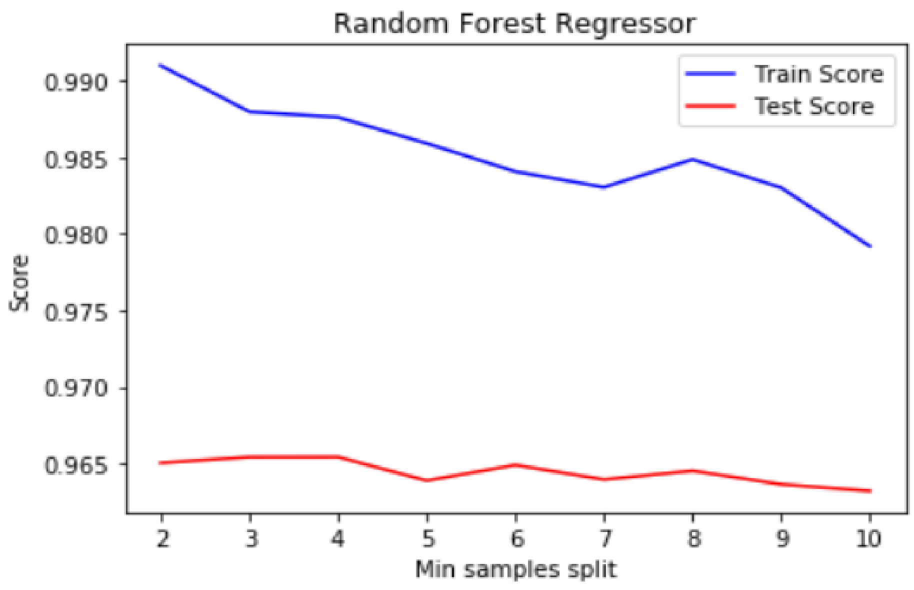
3. **min_samples_split**
4. **min_samples_leaf**
5. **max_features**
6. **max_leaf_nodes**

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_n
odes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=
0.0,
                      n_estimators=10, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

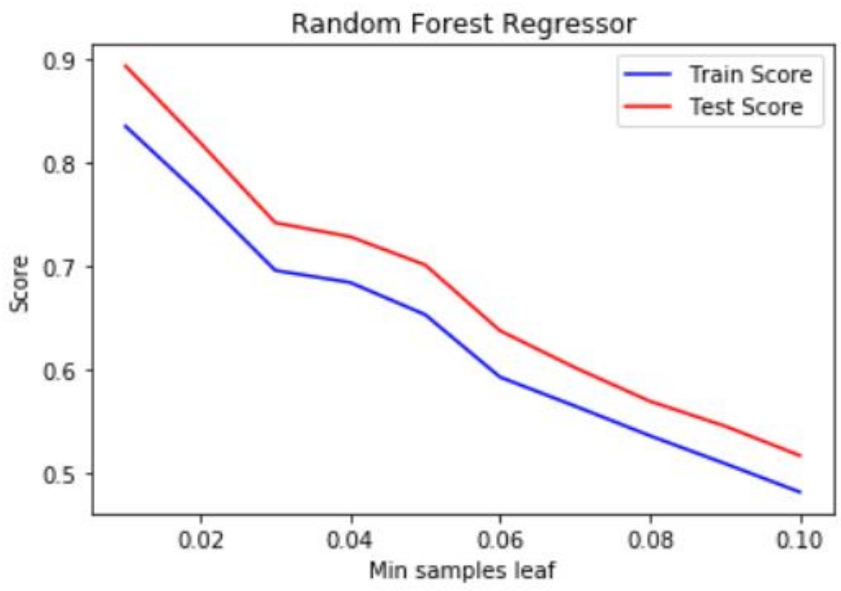
We studied the effects of various hyperparameters on the accuracy of the random forest regressor and obtained the following results.



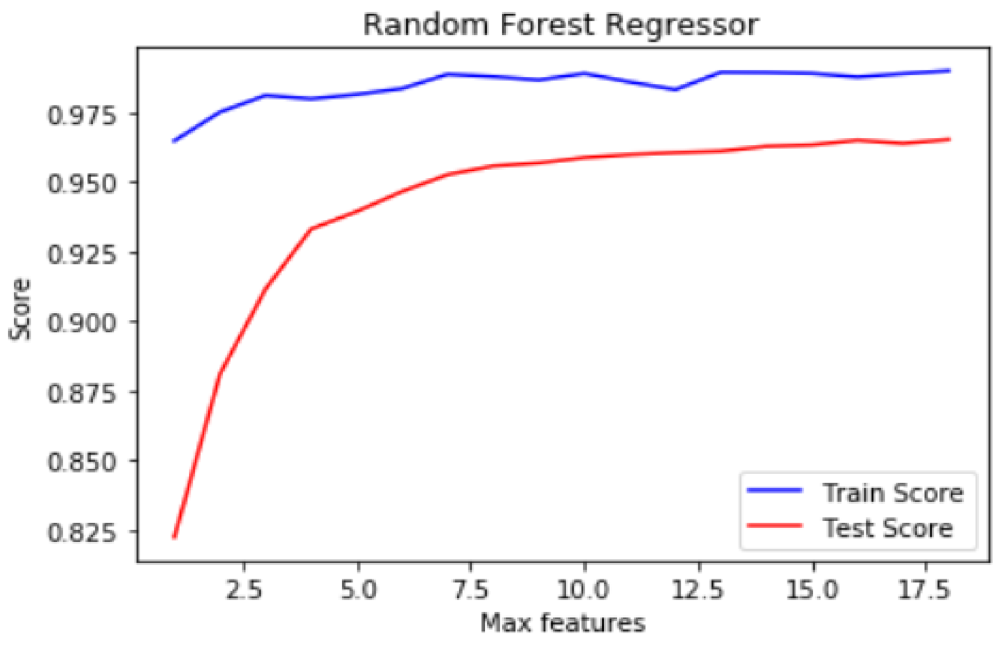
The higher the number of trees the better to learn the data. Also, according to our data analysis, both training and testing accuracy inreases as the number of decision trees is increased.



min_samples_split represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. When we increase this parameter, each tree in the forest becomes more constrained as it has to consider more samples at each node.



We can clearly see that when we require all of the samples at each node, the model cannot learn enough about the data. Increasing this value can cause underfitting.



.

max_features represents the number of features to consider when looking for the best split. We see that as max_features increases, both training and testing accuracy increases.

**Grid Search Results:**

```
{'max_features': 0.8, 'min_samples_leaf': 0.01, 'min_samples_spli
t': 8, 'n_estimators': 20}
```

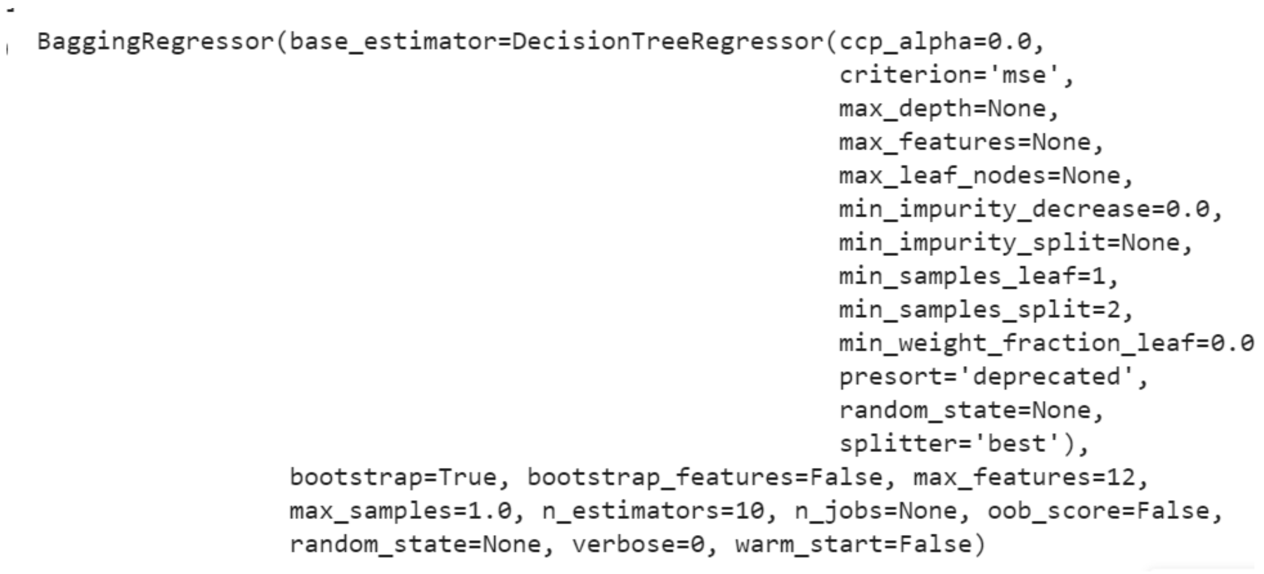
```
 ----------------- TRAINING ------------------------
r-squared score:  0.9881503644760703
 ------------------- TEST --------------------------
r-squared score:  0.9609853355990065
```

## Bagging Regressor

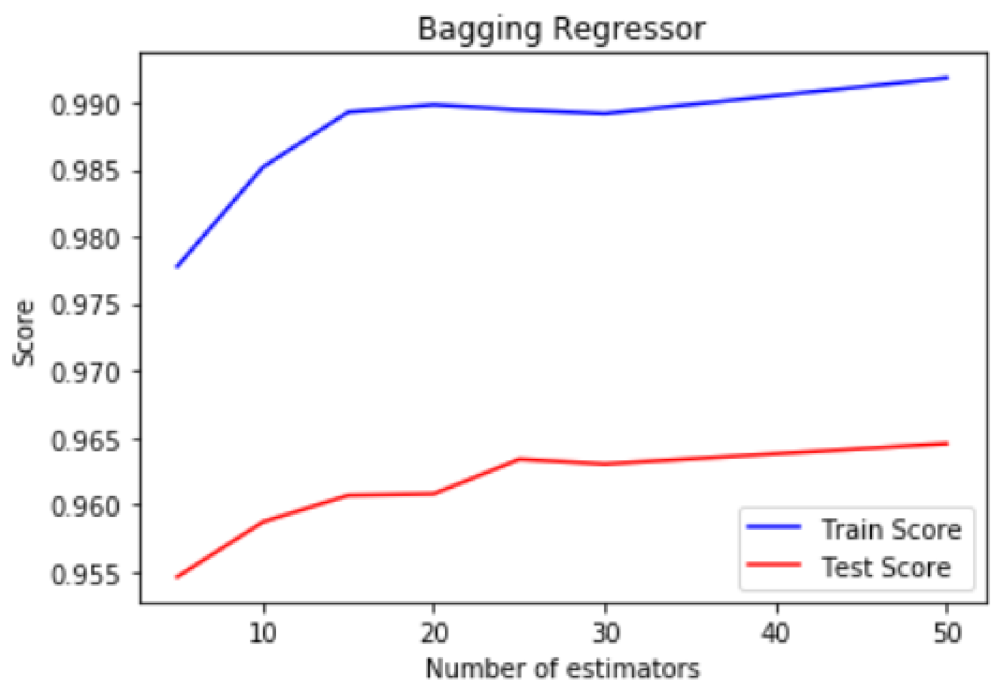
The first term introduced, bagging, is shorthand for the combination of bootstrapping and aggregating. Bootstrapping is a method to help decrease the variance of the classifier and reduce overfitting, by resampling data from the training set with the same cardinality as the original set. The model created should be less overfitted than a single individual model.

List of hyperparameters:

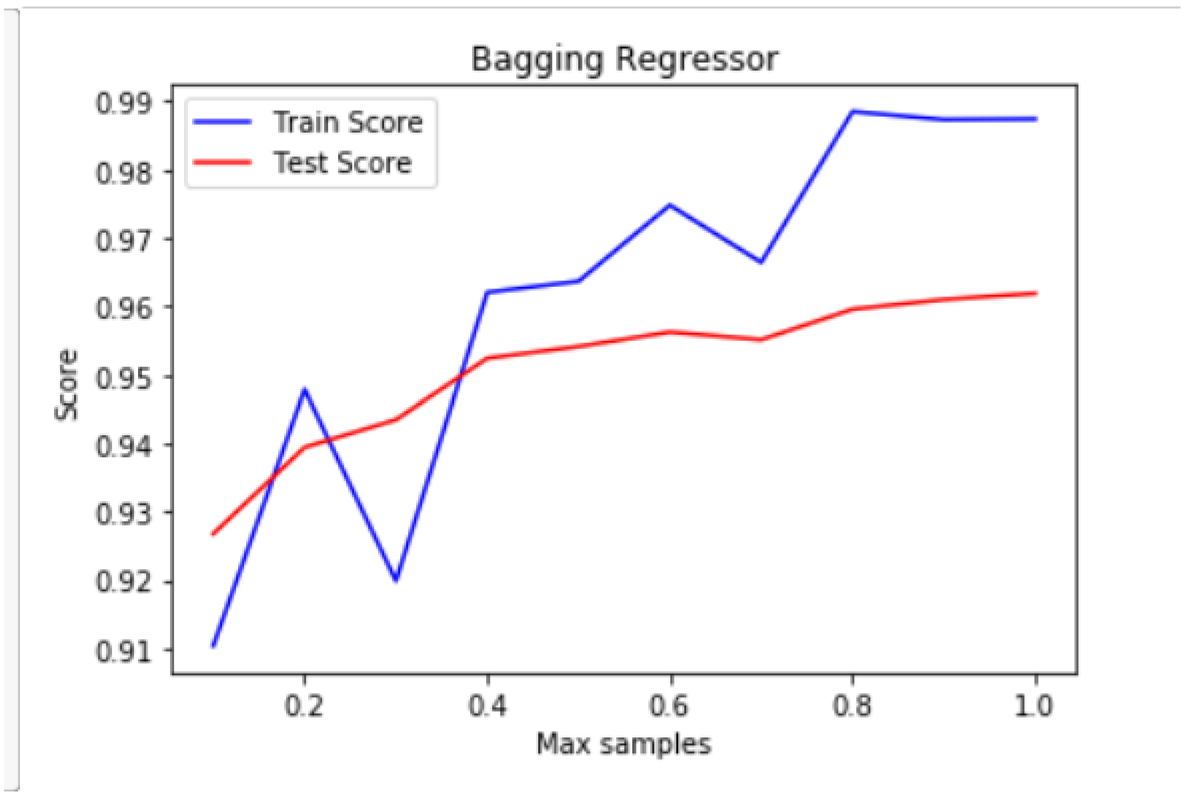
1. **n_estimators**
2. **max_samples**
3. **max_features**
4. **bootstrap**



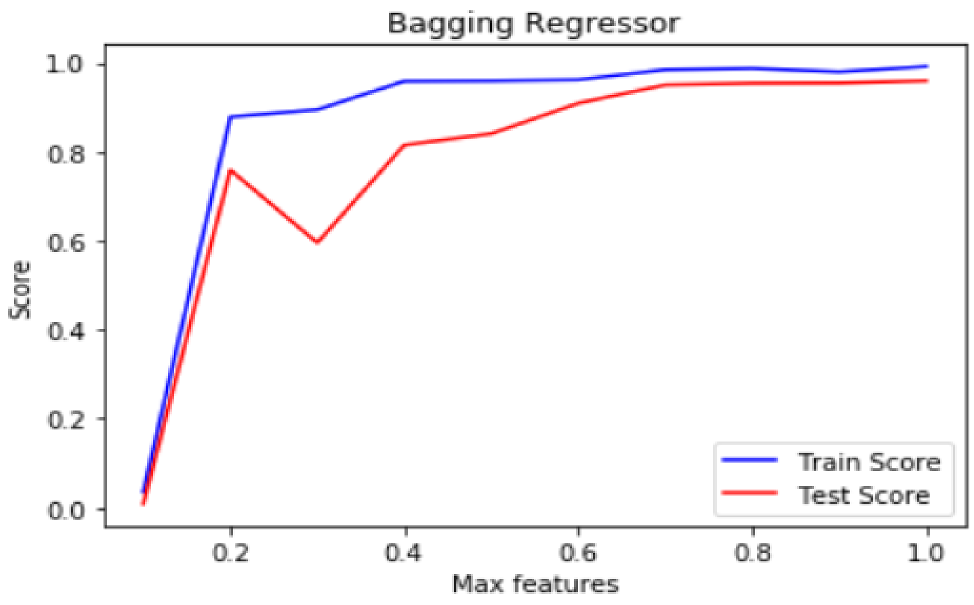
```
BaggingRegressor(base_estimator=DecisionTreeRegressor(ccp_alpha=0.0,
                                                      criterion='mse',
                                                      max_depth=None,
                                                      max_features=None,
                                                      max_leaf_nodes=None,
                                                      min_impurity_decrease=0.0,
                                                      min_impurity_split=None,
                                                      min_samples_leaf=1,
                                                      min_samples_split=2,
                                                      min_weight_fraction_leaf=0.0
                                                      presort='deprecated',
                                                      random_state=None,
                                                      splitter='best'),
                 bootstrap=True, bootstrap_features=False, max_features=12,
                 max_samples=1.0, n_estimators=10, n_jobs=None, oob_score=False,
                 random_state=None, verbose=0, warm_start=False)
```



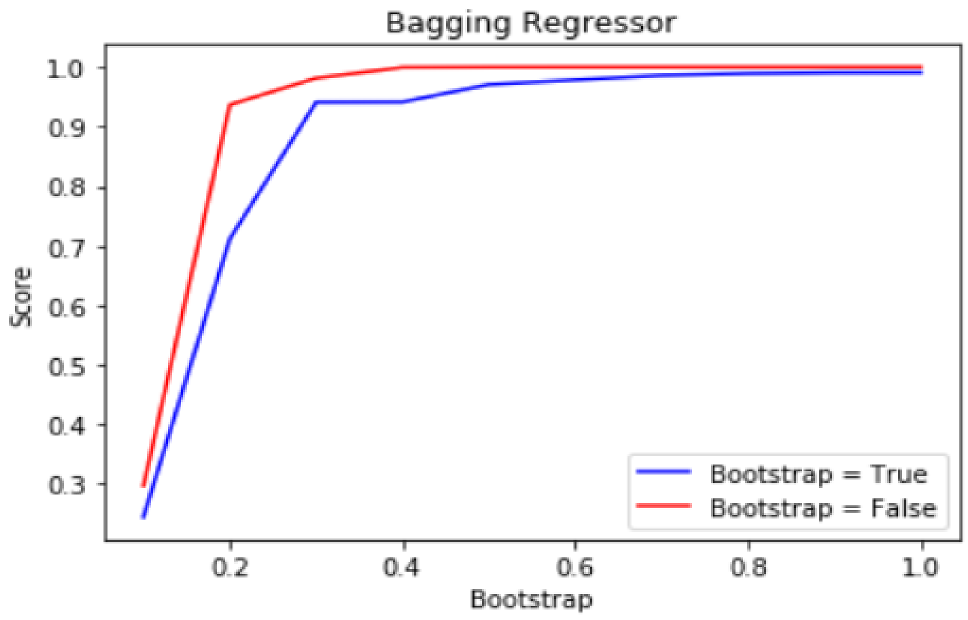
Varying, the number of base estimators we see that bagging accuracy increases with number of estimators.



We see sharp rise and fall in the training accuracy as we increase the max samples, but the testing accuracy increases more smoothly.



As the max features increases, the train and test accuracy of the bagging regressor model increases.



There are two different ways of drawing samples from the data. When bootstrap = true, the sampling is done with replacement and otherwise, sampling without replacement is performed.

**Grid Search Results:**

```
{'bootstrap': False, 'max_features': 0.75, 'max_samples': 0.75, 'n_
estimators': 50}
```

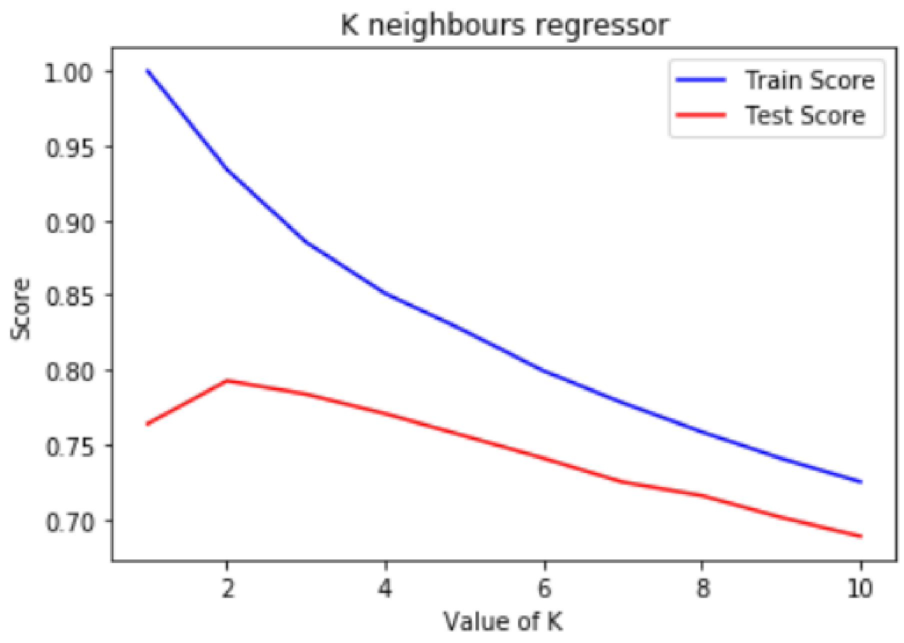
```
  ----------------- TRAINING ------------------------
  r-squared score:  0.9852904556149269
  ------------------- TEST -------------------------
  r-squared score:  0.9417064941909946
```

## KNN Regressor

List of hyperparameters:

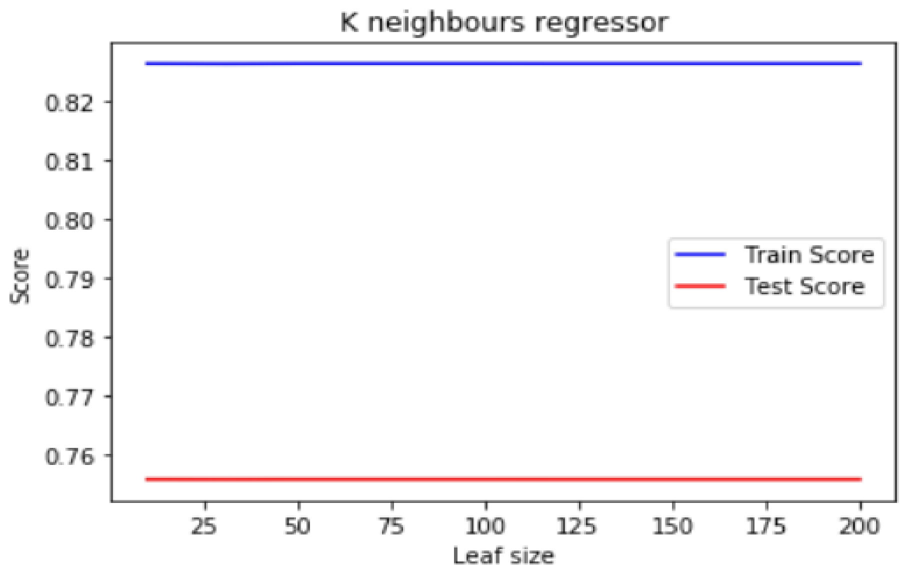
1. **n_neighbours**
2. **weights**
3. **algorithm**
4. **leaf_size**

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

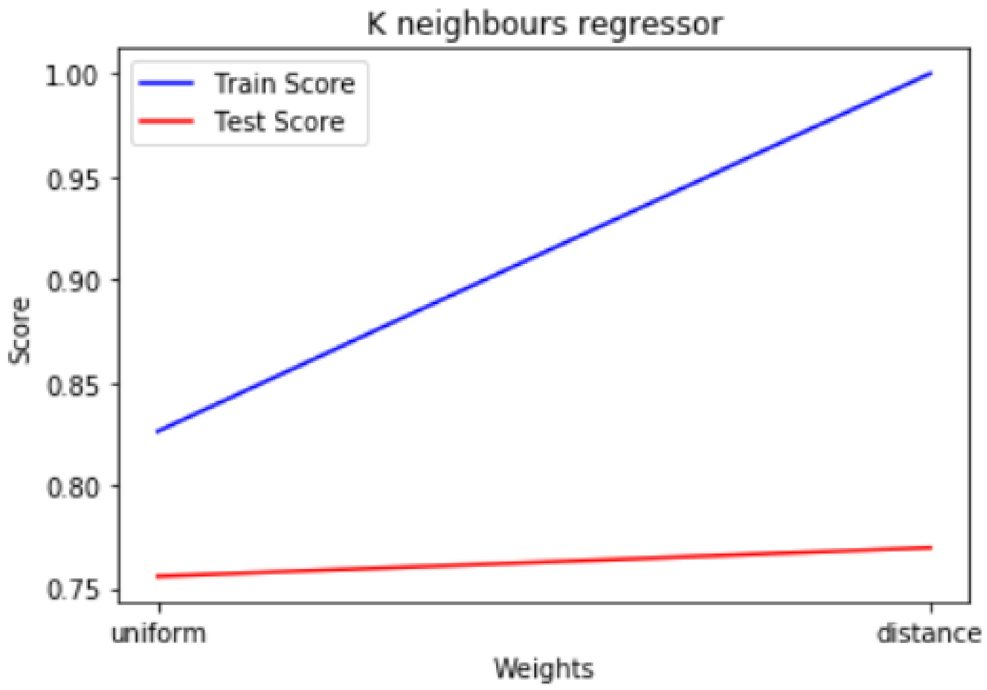


For train data, we can see that the accuracy is 1 for k=1 and decreases with increasing k because for k=1 it uses the actual label used in training and the error naturally increases with extra records being considered.

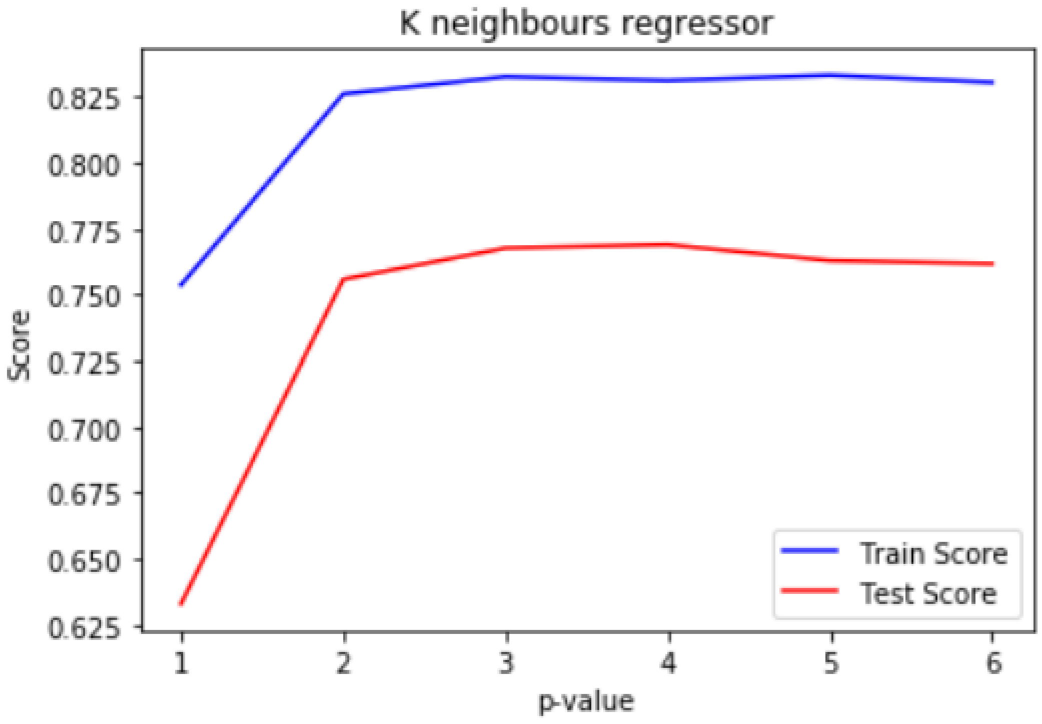
For test data, we see that the acurracy is optimal for k=2. This is because considering only the nearest value might be misleading and considering too many values will increase variance.



Since the leaf_size parameter only affects the construction of the data-structure (KD-Tree or Ball Tree) , we verify that the accuracy is not dependent on it.

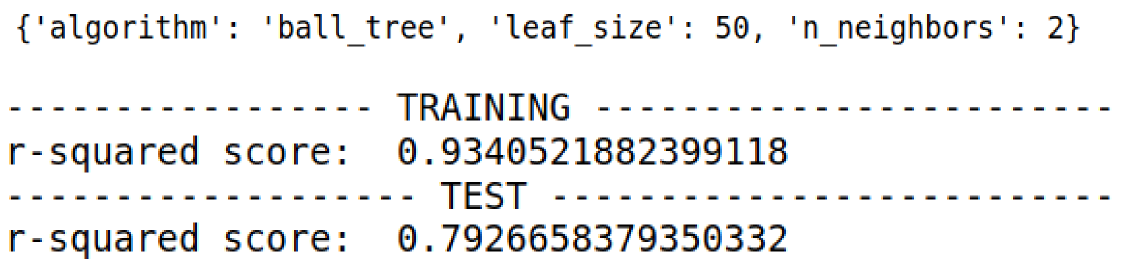


Since the nearer records tend to be more accurate , we see that the accuracy increases when we use the inverse distance weights for the algorithm.



p-value is the power in the Minowski distance. We see that the accuracy is greater for euclidean distance compared to manhattan distance and doesn't vary much for p>=2.

Grid Search Results :



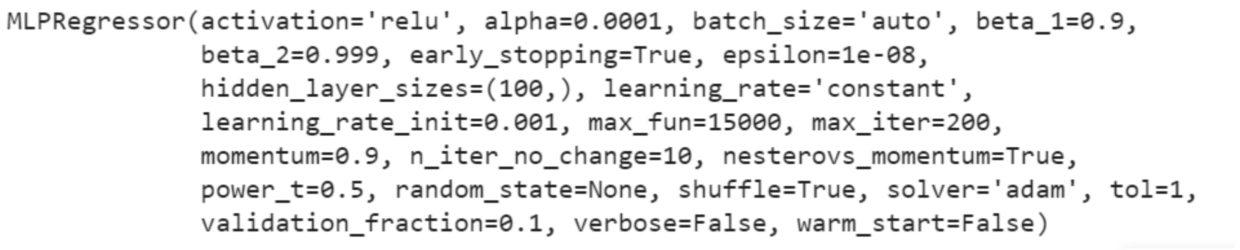
```
{'algorithm': 'ball_tree', 'leaf_size': 50, 'n_neighbors': 2}

----------------- TRAINING -----------------------
r-squared score:  0.9340521882399118
------------------- TEST -------------------------
r-squared score:  0.7926658379350332
```

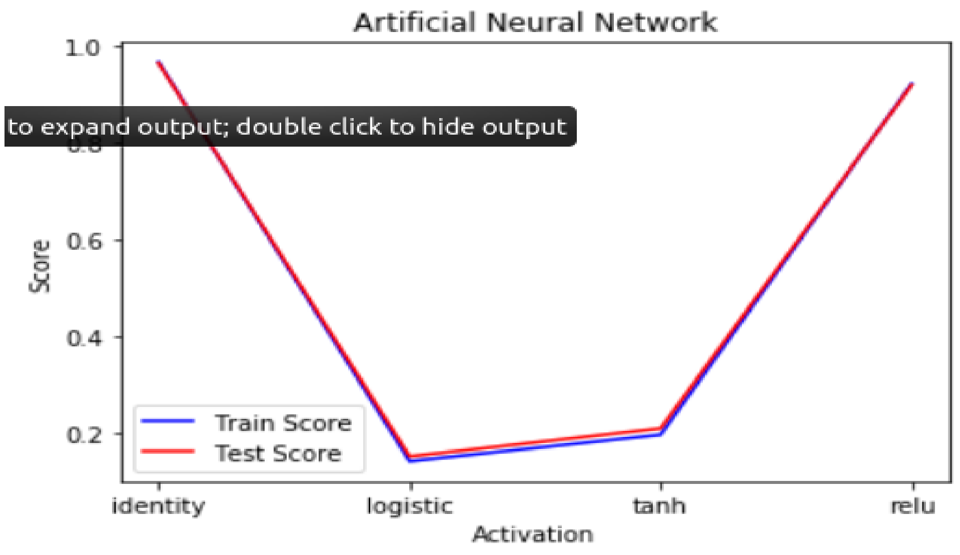
## Artificial Neural Network Regressor

List of hyper-parameters:

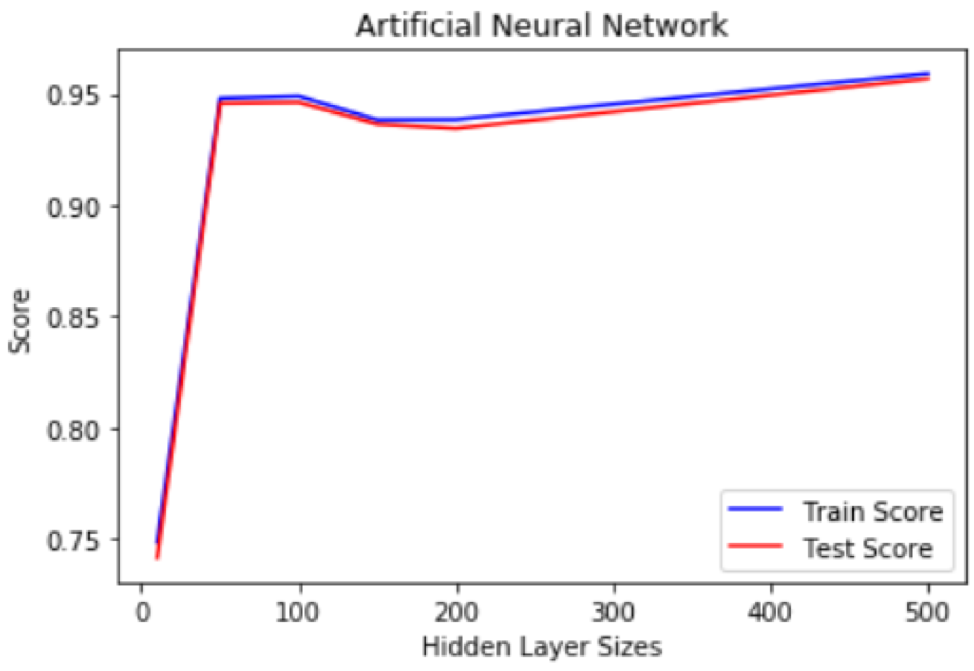
1. activation (function)
2. hidden layer sizes
3. batch size
4. learning rate



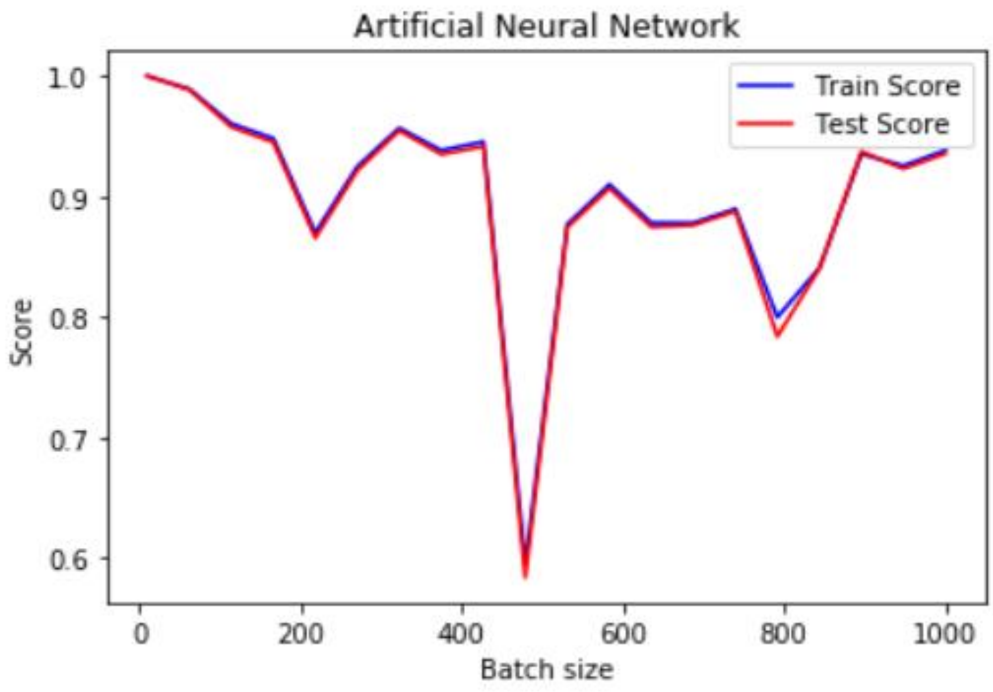
```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=True, epsilon=1e-08,
             hidden_layer_sizes=(100,), learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=200,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=1,
             validation_fraction=0.1, verbose=False, warm_start=False)
```



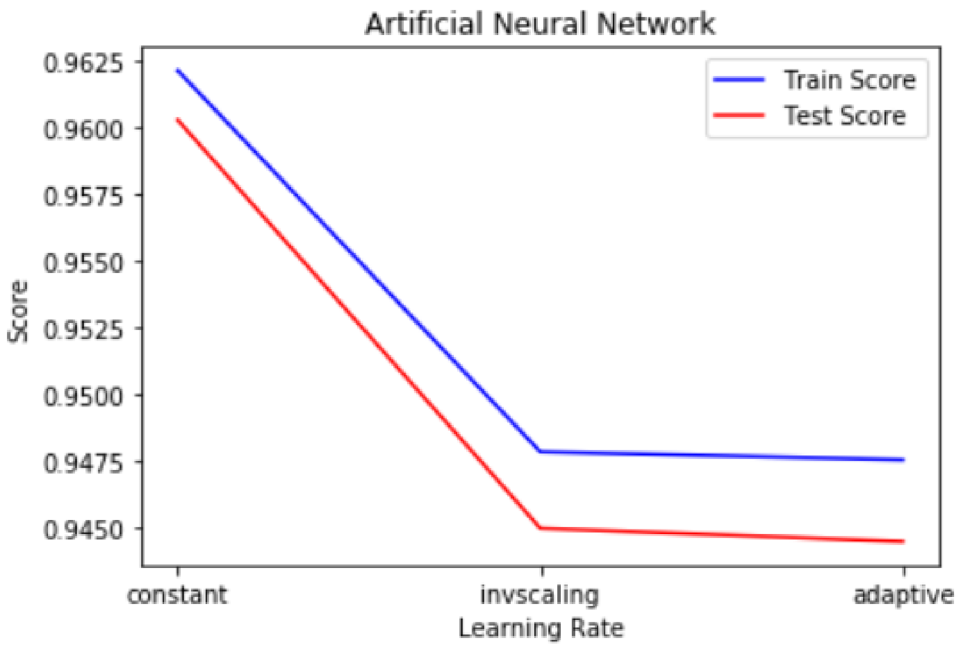
We observe that the identity activation function (which is generally useful to implement linear bottlenecks) gives the best results.



We see that initially the accuracy increases linearly (almost) with the hidden layer sizes and than saturates around size = 50.

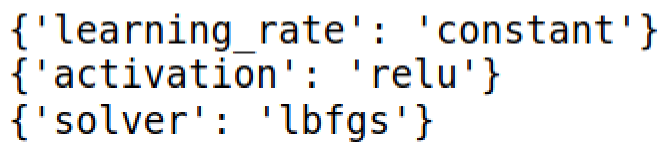


We cannot observe a regular pattern for accuracy corresponding to batch size. However it is interesting to note that there is a sharp dip at batch_size = 478.

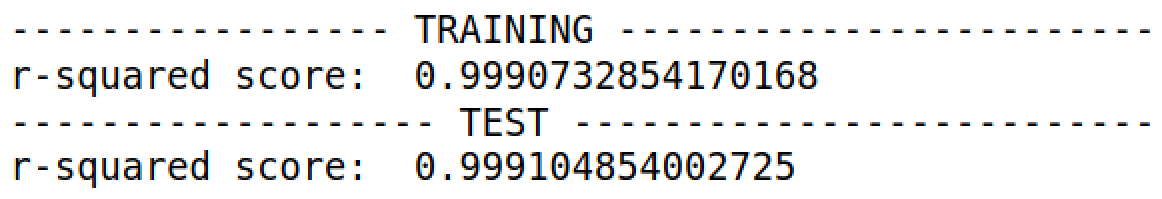


We see that the constant learning rate gives the best results for our data set.

Grid search results:

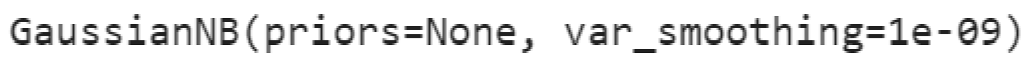


```
{'learning_rate': 'constant'}
{'activation': 'relu'}
{'solver': 'lbfgs'}
```



```
----------------- TRAINING ------------------------
r-squared score:  0.9990732854170168
------------------- TEST -------------------------
r-squared score:  0.999104854002725
```

## Naive Bayesian Classifier

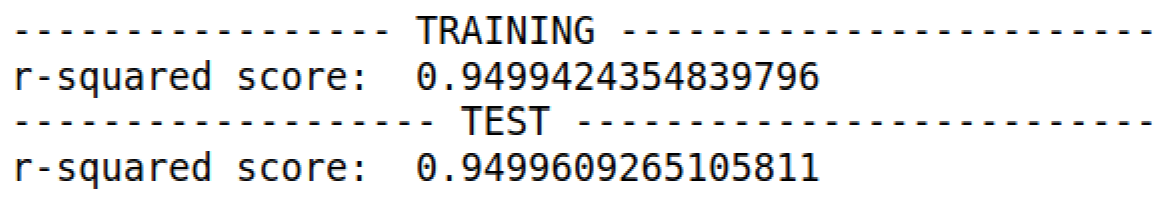


```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Since naive bayesian approach is naturally used for classification, we have done classification here instead of regression, using the Gaussian Naive Bayes Classifier.  For that we discretized the the arrival time range into 20 equal sized intervals each of approximately 100 minutes. From the accuracy we see that the algorithm predicts the correct interval in about 95% of the cases.

[Also, there were no hyperparameters to tune in this classifier]

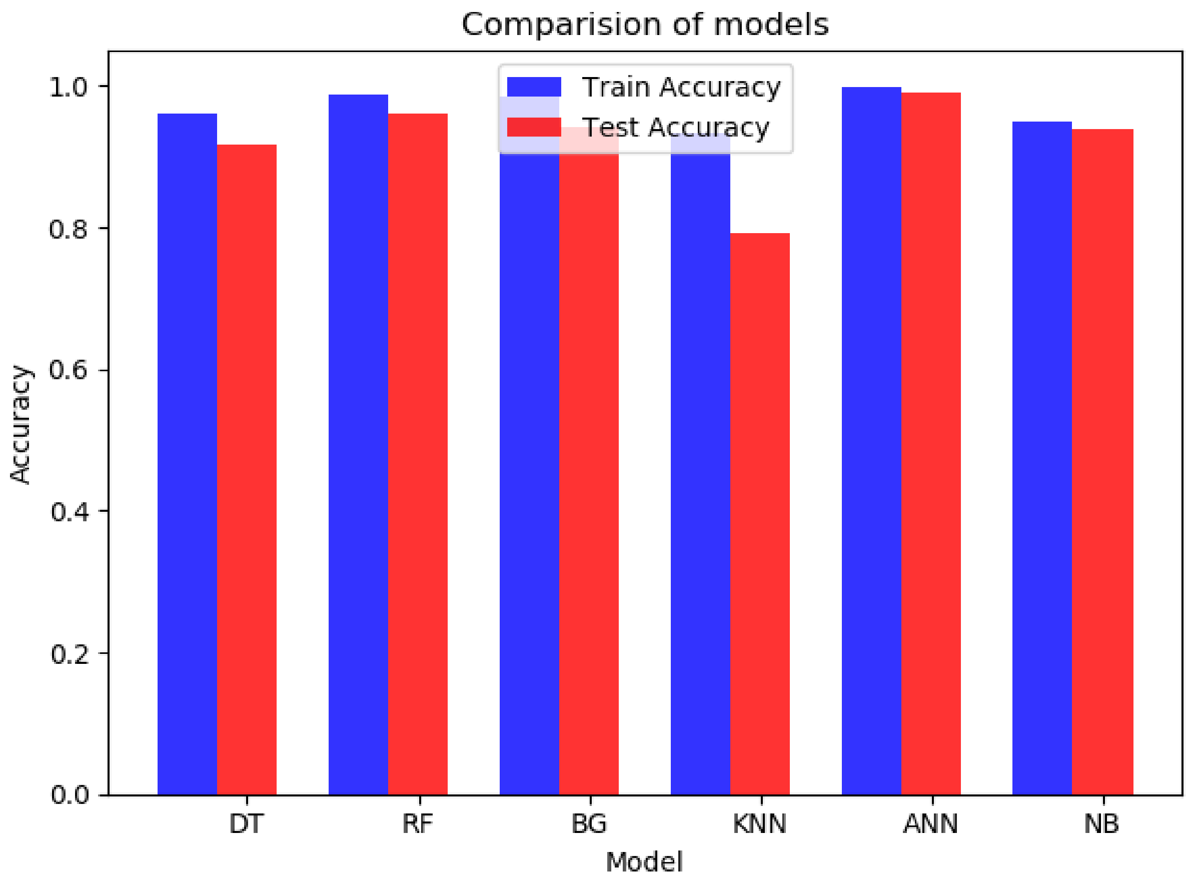
Results :



```
----------------- TRAINING ------------------------
r-squared score:  0.9499424354839796
------------------- TEST -------------------------
r-squared score:  0.9499609265105811
```

## Comparision of Algorithms

The following graph represents the accuracy of different algorithms on the test data set.



NOTE : The naive bayesian accuracy is corresponding to classification while others are corresponding to regression for reasons explained earlier.

We see that for our regression problem, the Artificial Neural Network gave us the best results with a test accuracy almost close to 1 (0.99).  The neural network fits the data perfectly well without any overfitting on the training data and thus gives the best prediction model for ARRIVAL_DELAY.  Followed by ANN, we have Random Forest Regressor with a test accuracy of 0.961. Then Bagging Regressor gives a test accuracy of 0.941, followed by a Decision Tree having a test accuracy of 0.916. However, K-Nearest Neighbours gives a poor prediction results on the test data having accuracy of only 0.792.

**EXPLANATION :**

The No free lunch theorem suggests that there can be no single method which performs best on all data sets.

ANN's performance strongly depend on the dataset. With hyperparameter tuning and avoiding overfitting, ANN's gives good results in most cases.

Bagging is used when our goal is to reduce the variance of a decision tree. In Bagging, we create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.

Random Forest is an extension over bagging. It takes one extra step where in addition to taking the random subset of data, it also takes the random selection of features rather than using all features to grow trees. When you have many random trees. It’s called Random Forest. Random forests consist of multiple single trees each based on a random sample of the training data. They are typically more accurate than single decision trees. The decision boundary becomes more accurate and stable as more trees are added.

K-NN does not have a training process, and in consequence, it does not try to optimize any effectiveness measure. We can play with different values of K, and intuitively, the bigger the K the higher the recall and the lower the precision, and the opposite.