PROJECT REPORT

CAB FARE PREDICTION

ARPIT DUBEY

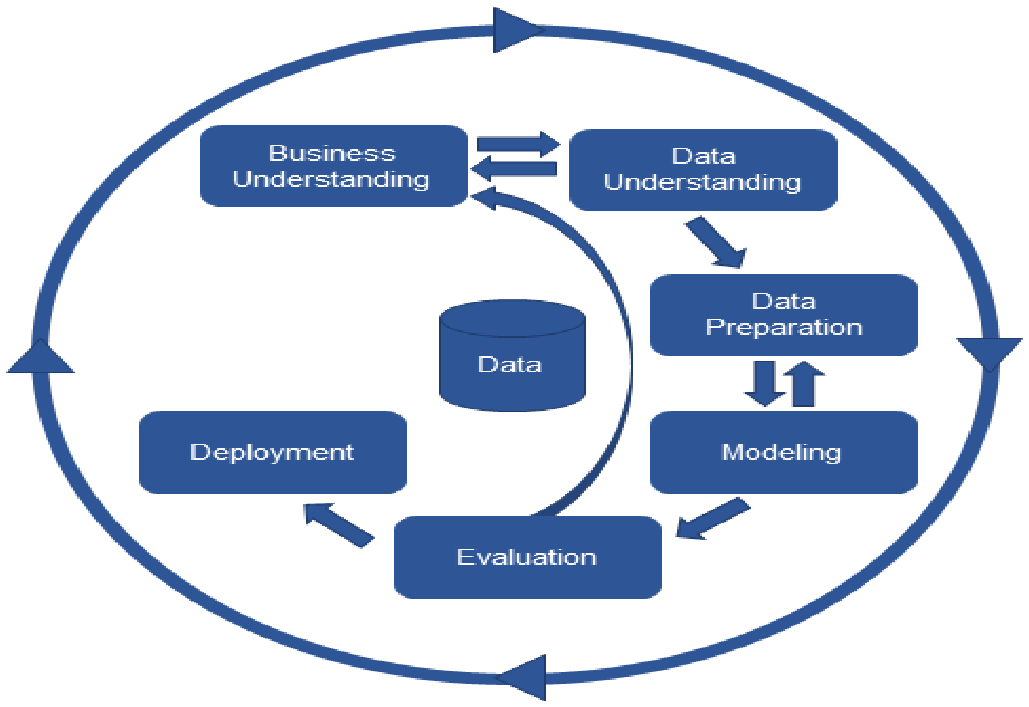
14/3/2020

Abstract—

In this report, the fare amount of a cab ride is detected using machine learning algorithms. From the given train_data, a feature set of distance is created from the pickup and drop-off latitude and longitude. From the datetime variable features like year and hour of travel are extracted which affect the fare prices.

Using this feature data, different models are built: Decision Tree, Linear Regression, Random Forest, KNN classifier. Final prediction is of the fare amount of the given test_data, which only have latitude and longitude columns. From the given model's best error rates was with Random Forest model, which is then implemented on the given test data for the final fare prediction.

CONTENT



CRISP-DM Methodology: Describes commonly used approach to solve Business problem

CRISP-DM includes 6 major phases

- Business Understanding
- Data understanding
- Data Preparation
- Data modeling
- Evaluation
- Deployment

Problem Statement - You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

2. Business Understanding

There are two main tasks addressed in this stage:

• **Define objectives:** Work with our customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals that the data science techniques can target.

• **Identify data sources:** Find the relevant data that helps to answer the questions that define the objectives of the project.

Define objective

1. A central objective of this step is to identify the key business variables that the analysis needs to predict. These variables are referred as the model targets, and the metrics associated with them is used to determine the success of the project.

2. Define the project goals by asking and refining "sharp" questions that are relevant, specific, and unambiguous. Data science is a process that uses names and numbers to answer such questions. We typically use data science or machine learning to answer five types of questions:

- How much or how many? (regression)
- Which category? (classification)
- Which group? (clustering) • Is this weird? (anomaly detection)
- Which option should be taken? (recommendation)

We determine which of these questions we're asking and how answering it achieves our business goals.

Identify data sources

Identify data sources that contain known examples of answers to our sharp questions. Look for the following data:

- Data that's relevant to the question. Do we have measures of the target and features that are related to the target?
- Data that's an accurate measure of our model target and the features of interest.

With the advancement of technology there are new ideas coming up. Cab rental methodology in one among them. It has become one of the leading market shares for the economy. The main reason for its success is cabs easy availability and also its accurate fare predictions.

Our main task for this project is dig into the historical data provided and looks into the features which affect the fare amount. This is very important in business point of view because this prediction will decide the myth of the company progress. If the predicted fare is greater than actual, then it will it affect the customer and will give negative impact to the company. If the predicted fare is less than the actual, then it will results loss to the company.

So, from both Company as well Customer point of view cab fare prediction should have to be accurate.

Methodology

➢ Pre-Processing

When we required to build a predictive model, we require to look and manipulate the data before we start modelling which plots, all these steps is combined under one shed which is Exploratory Data Analysis, which includes following steps:

- Data exploration and Cleaning
- Missing values treament

- Outlier Analysis • Feature Selection
- Features Scaling or Skewness and Log transformation
- Visualization

➢ **Modelling**

It includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modeling. Modeling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set.

As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested:

- Linear regression
- Decision Tree
- Random forest,
- Gradient Boosting

❖We have also used hyper parameter tunings to check the parameters on which our model runs best. Following are two techniques of hyper parameter tuning we have used:

- Random Search CV
- Grid Search CV

➢ **Model Selection**

The final step of our methodology will be the selection of the model based on the different output and results shown by different models. We have multiple parameters which we will study further in our report to test whether the model is suitable for our problem statement or not.

Pre-Processing

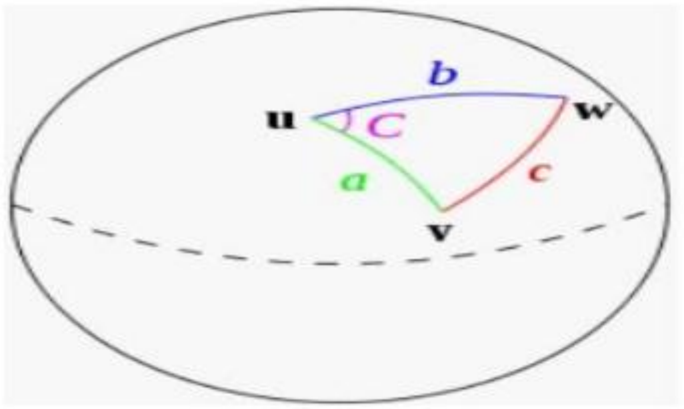
Data exploration and Cleaning (Missing Values and Outliers) The very first step which comes with any data science project is data exploration and cleaning which includes following points as per this project:

a) Separate the combined variables.
b) As we know we have some negative values in fare amount so we have to remove those values.
c) Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passenger's counts more than 6 and less than 1.
d) There are some outlier figures in the fare (like top 3 values) so we need to remove those.
e) Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges.

Creating some new variables from the given variables. Here in our data set our variable name pickup_datetime contains date and time for pickup. So we tried to extract some important variables from pickup_datetime:

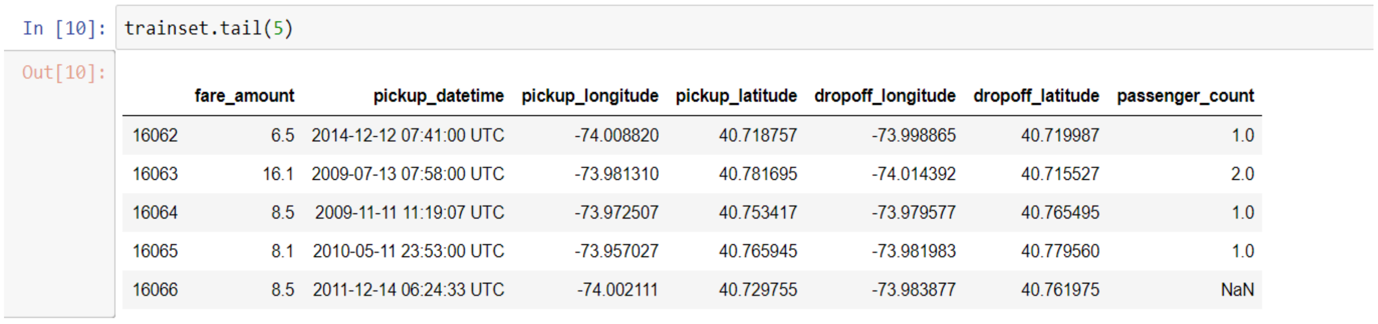
- Year
- Month
- Date
- Day of Week
- Hour
- Minute

Also, we tried to find out the distance using the haversine formula which says: The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines that relates the sides and angles of spherical triangles.



So our new extracted variables are:

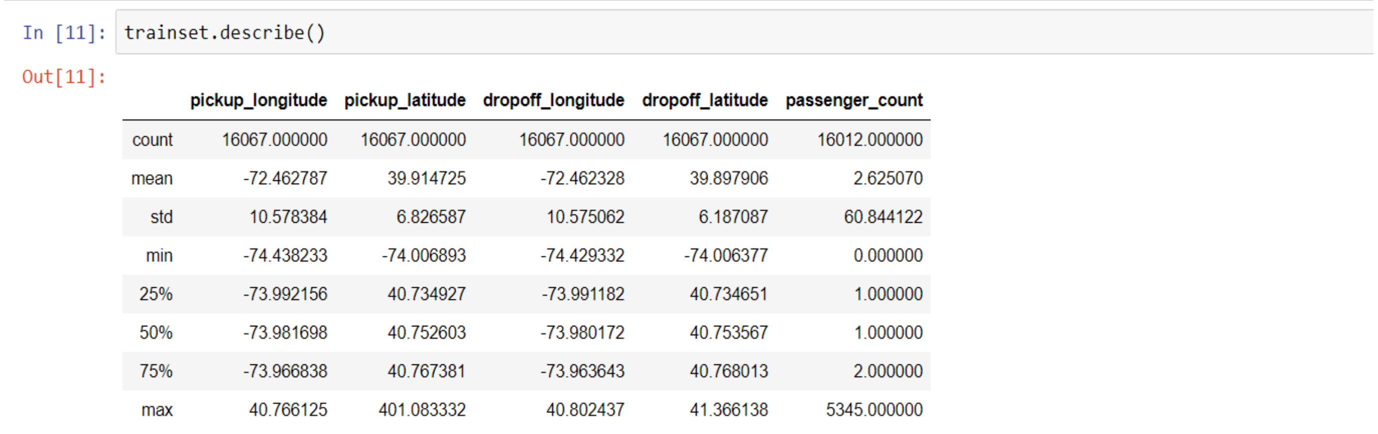
- fare_amount



In [10]:
```
trainset.tail(5)
```

Out[10]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
16062	6.5	2014-12-12 07:41:00 UTC	-74.008820	40.718757	-73.998865	40.719987	1.0
16063	16.1	2009-07-13 07:58:00 UTC	-73.981310	40.781695	-74.014392	40.715527	2.0
16064	8.5	2009-11-11 11:19:07 UTC	-73.972507	40.753417	-73.979577	40.765495	1.0
16065	8.1	2010-05-11 23:53:00 UTC	-73.957027	40.765945	-73.981983	40.779560	1.0
16066	8.5	2011-12-14 06:24:33 UTC	-74.002111	40.729755	-73.983877	40.761975	NaN



In [11]:
```
trainset.describe()
```

Out[11]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	10.578384	6.826587	10.575062	6.187087	60.844122
min	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	40.766125	401.083332	40.802437	41.366138	5345.000000

- pickup_datetime
- pickup_longitude
- dropoff_longitude
- dropoff_latitude
- Minute

3. Data Understanding

There are two sets of data:

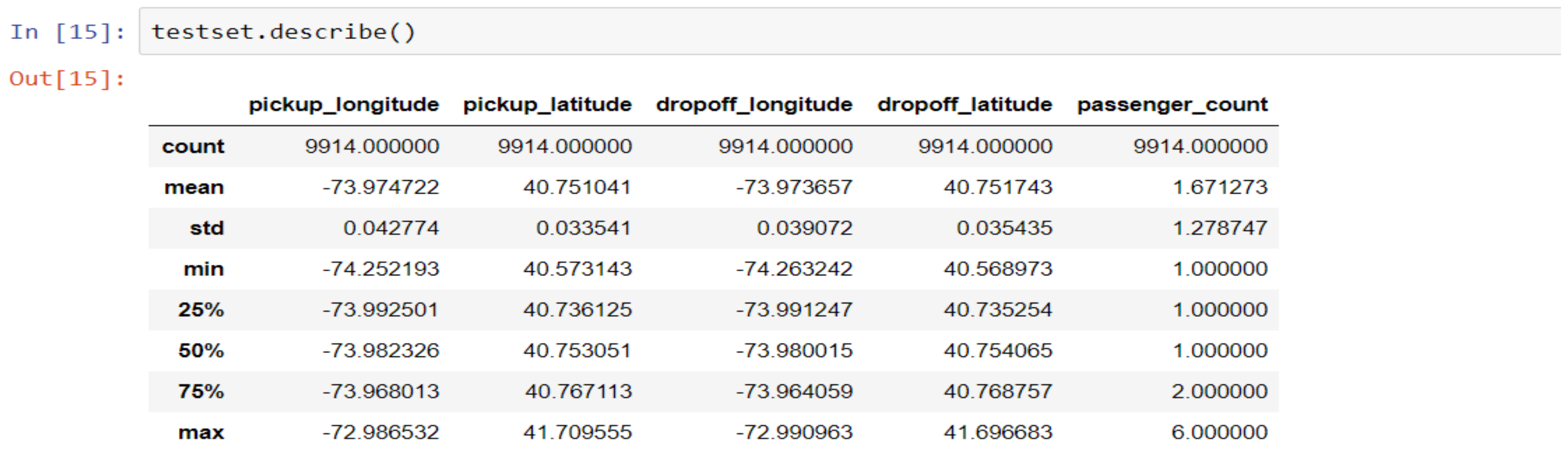
Data Set :

- trainset
- testset

Variables	Description
fare_amount	Fare amount
pickup_datetime	Cab pickup date with time
pickup_longitude	Pickup location longitude
pickup_latitude	Pickup location latitude
dropoff_longitude	Drop location longitude
dropoff_latitude	Drop location latitude
passenger_count	Number of passengers sitting in the cab

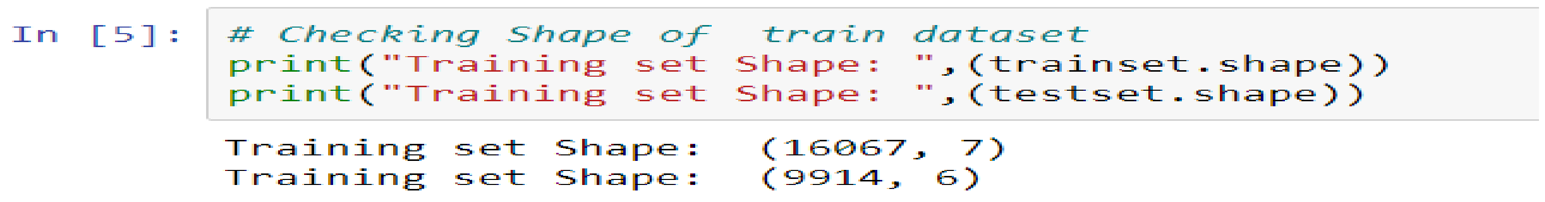
Number of attributes:

- pickup_datetime - timestamp value indicating when the cab ride started.
- pickup_longitude - float for longitude coordinate of where the cab ride started.
- pickup_latitude - float for latitude coordinate of where the cab ride started.
- dropoff_longitude - float for longitude coordinate of where the cab ride ended.
- dropoff_latitude - float for latitude coordinate of where the cab ride ended.



```
In [15]: testset.describe()
Out[15]:
```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273
std	0.042774	0.033541	0.039072	0.035435	1.278747
min	-74.252193	40.573143	-74.263242	40.568973	1.000000
25%	-73.992501	40.736125	-73.991247	40.735254	1.000000
50%	-73.982326	40.753051	-73.980015	40.754065	1.000000
75%	-73.968013	40.767113	-73.964059	40.768757	2.000000
max	-72.986532	41.709555	-72.990963	41.696683	6.000000



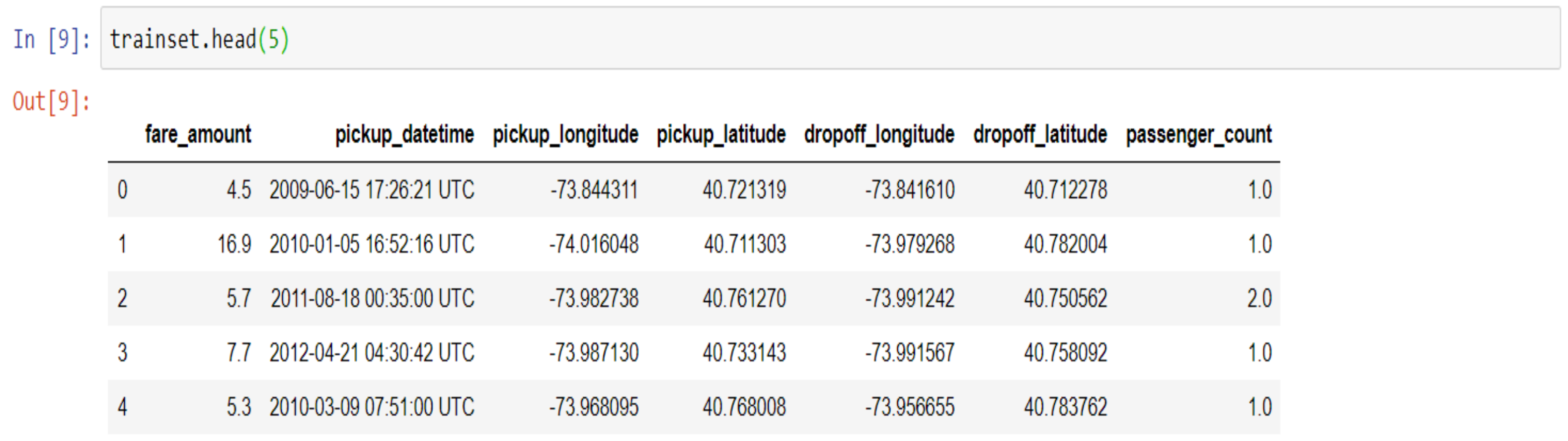
```
In [5]:  # Checking Shape of  train dataset
         print("Training set Shape: ",(trainset.shape))
         print("Training set Shape: ",(testset.shape))

         Training set Shape:  (16067, 7)
         Training set Shape:  (9914, 6)
```

- passenger_count - an integer indicating the number of passengers in the cab ride.

Missing values = yes

1) **Train_cab data:** Count-16067, Variables- 7
   Here, Target Variable = fare_amount

Independent Variables = pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude,

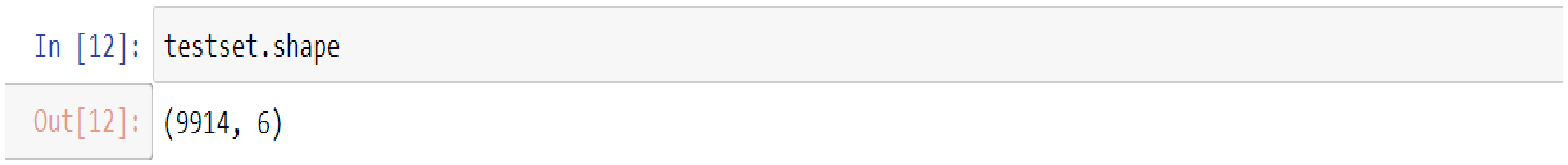


```
In [9]: trainset.head(5)
Out[9]:
```

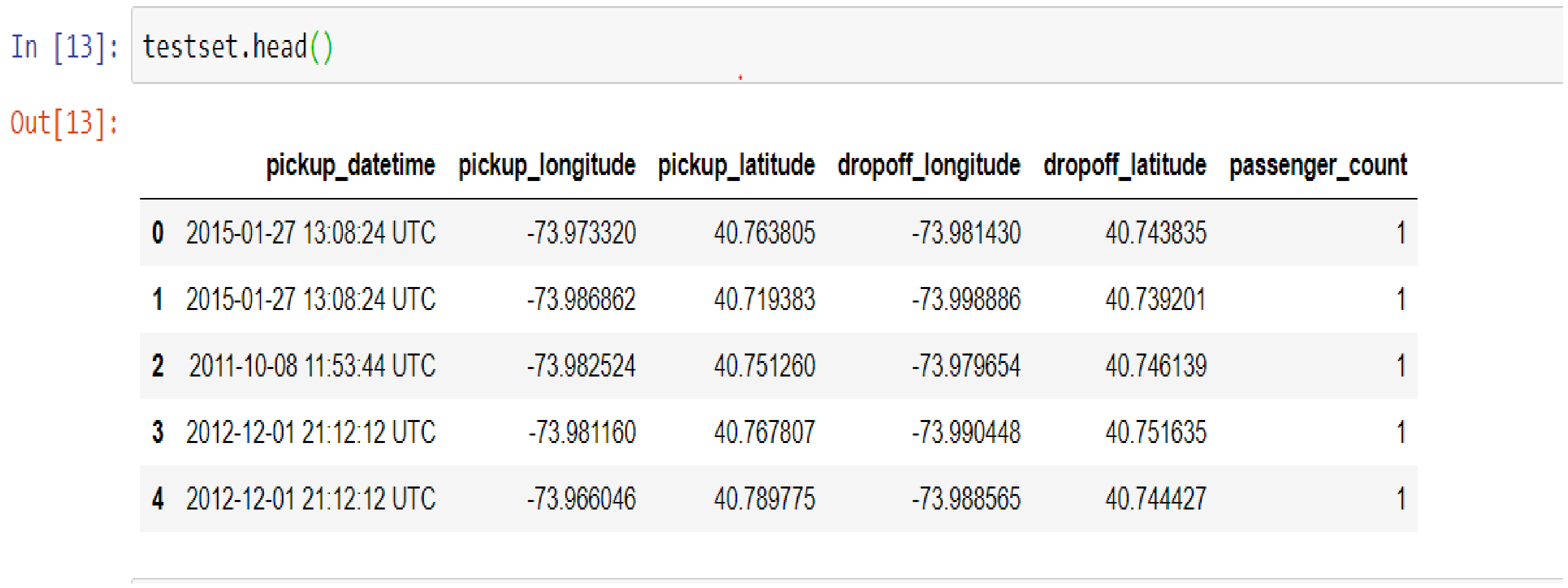
	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0

dropoff_latitude, passenger_count

2) Test Data: Count=9914, Variables= 6



```
In [12]: testset.shape
Out[12]: (9914, 6)
```

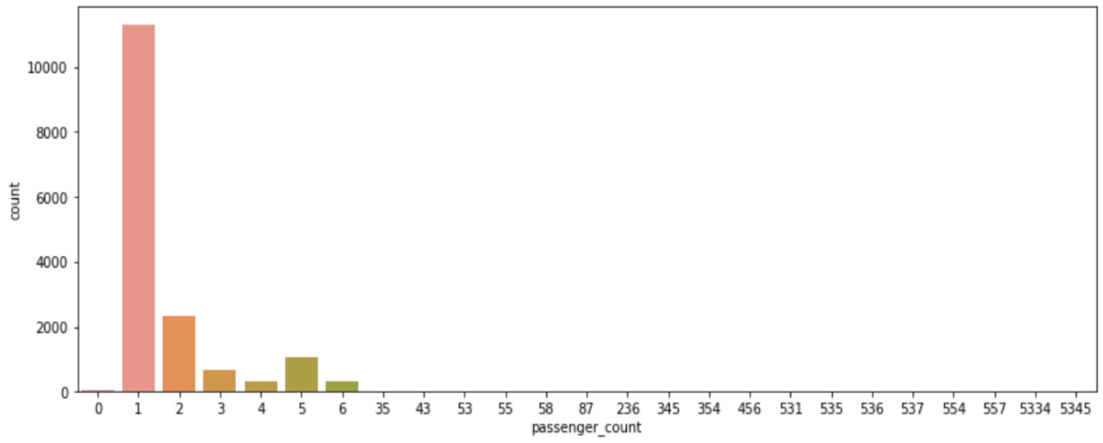


```
In [13]: testset.head()
Out[13]:
```

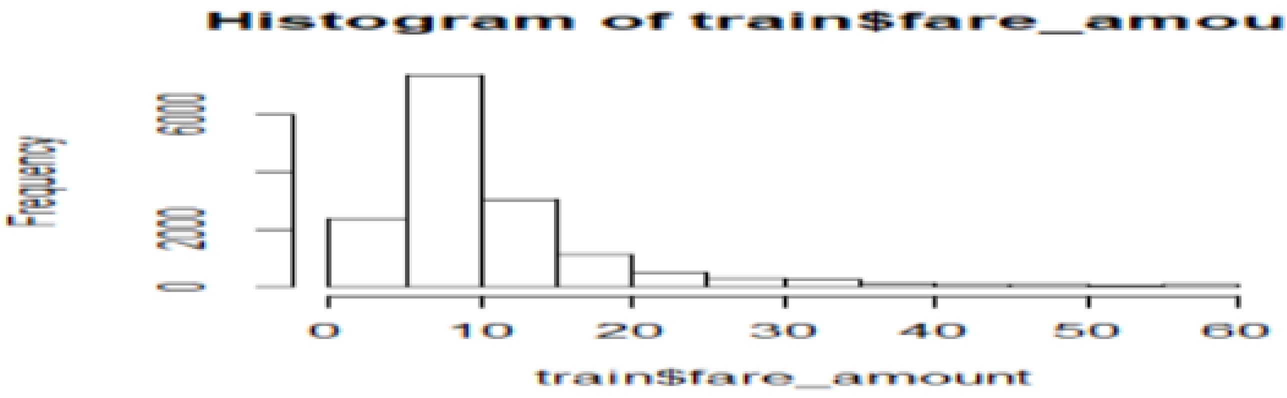
	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

Distribution of Passenger_count

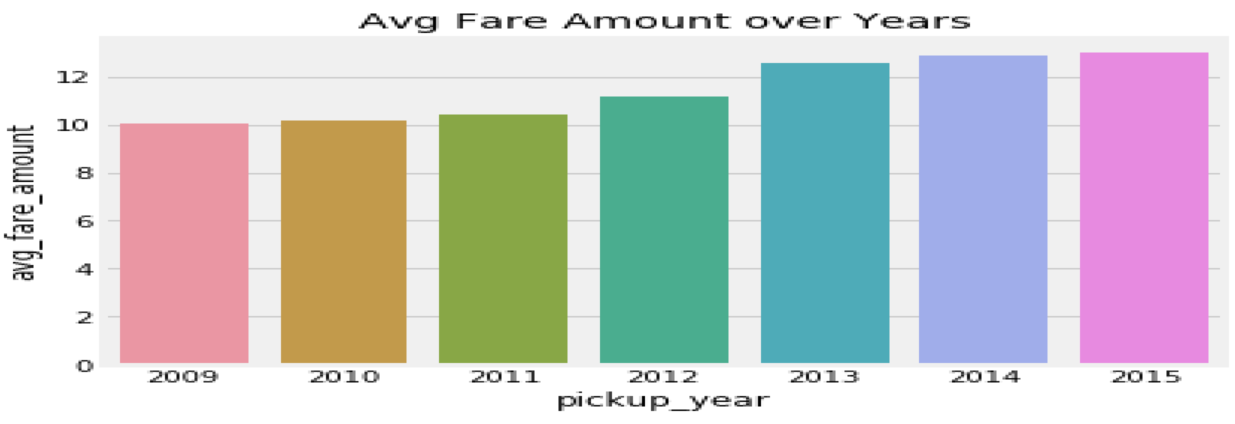
```
In [47]: # plot of the passenger_count
         plt.figure(figsize=(14,5))
         sns.countplot(x='passenger_count', data=trainset)

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x206fb340848>
```



Distribution of Target



Variable(fare_amount)



4. Data Preparation

Goals

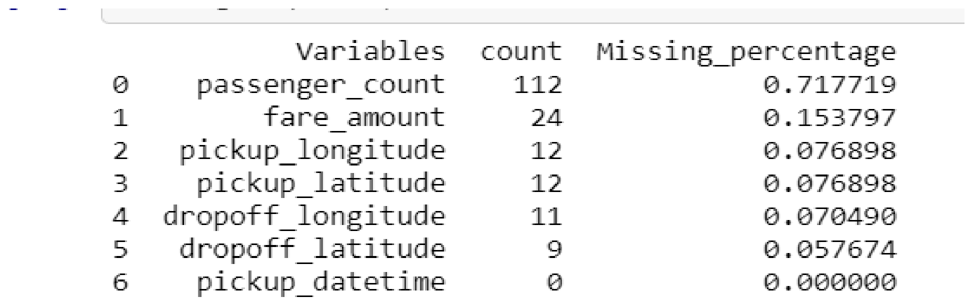
- Produce a clean, high-quality data set whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so that it is ready to model.

Before we train our models, we need to develop a sound understanding of the data. Real-world data sets are often noisy, are missing values, or have a host of other discrepancies. We can use data summarization and visualization to audit the quality of your data and provide the information we need to process the data before it's ready for modelling. This process is often iterative.

Missing Value analysis.

The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly, then we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from ones where the missing values are present.

Percent of missing values in our data before imputation



```
           Variables  count  Missing_percentage
0    passenger_count    112            0.717719
1        fare_amount     24            0.153797
2   pickup_longitude     12            0.076898
3    pickup_latitude     12            0.076898
4  dropoff_longitude     11            0.070490
5   dropoff_latitude      9            0.057674
6    pickup_datetime      0            0.000000
```

Also, there are many observations with value of 0, so we should impute it with missing value NA. In our data there are many observations where pickup_latitude is equal to dropoff_latitude, so we should remove these observation as they don't provide any information.

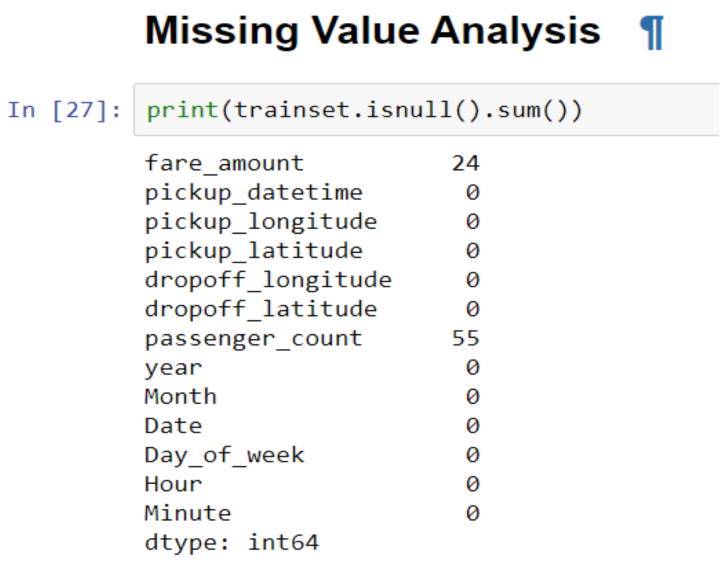
Impute missing values:

- Fill with central statistics: Mean Median Mode
- Distance based or Data mining method: KNN imputation
- Prediction Method

To impute missing values with this method, a sample observation is picked. Its value is noted and then it is filled with NA. After performing these methods, the method whose value is close to the chosen sample, is picked and applied to the missing value.

For example:

actual, train['pickup_longitude'].loc[50]	**-73.985582**
KNN train['pickup_longitude'].loc[50]	-73.9777079999999
median train['pickup_longitude'].loc[50]	**-73.9820605**
mean train['pickup_longitude'].loc[50]	-73.91160183651274



Missing Value Analysis ¶

```
In [27]: print(trainset.isnull().sum())

fare_amount          24
pickup_datetime       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
passenger_count      55
year                  0
Month                 0
Date                  0
Day_of_week           0
Hour                  0
Minute                0
dtype: int64
```

From the result Median provides the best result. So, Impute the missing values with this method. Also ,variable passenger_count is a categorical variable, so we should impute it with mode method.

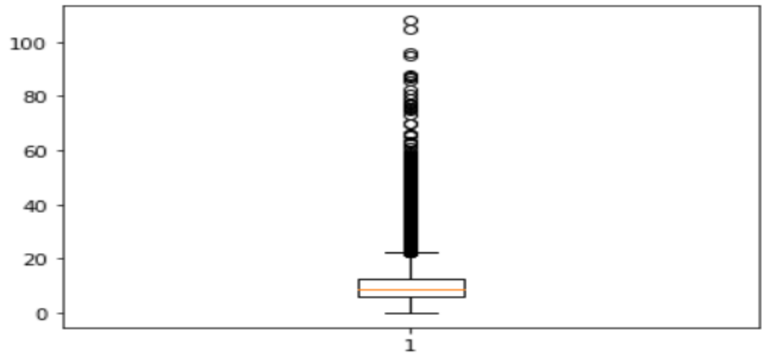
Outlier Analysis:

Observations inconsistent with rest of the dataset Global Outlier. Causes of Outliers:

- Poor data quality / contamination
- Low quality measurements, malfunctioning equipment, manual error
- Correct but exceptional data To detect the Outlier in the data – Graphical approach- Box Plot method.

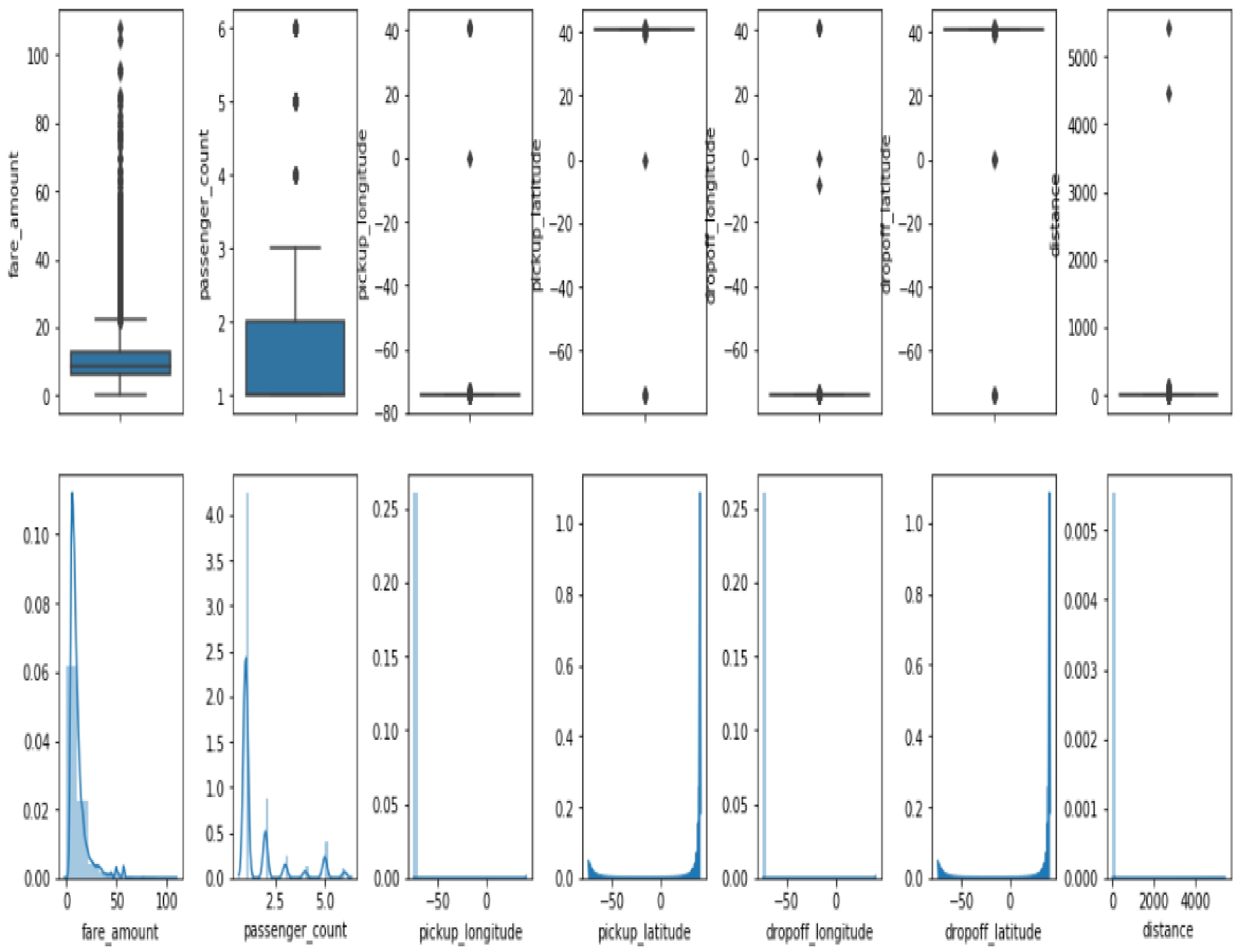
```
In [67]: plt.boxplot(trainset.fare_amount)
```

```
Out[67]: {'whiskers': [<matplotlib.lines.Line2D at 0x196d38bfe88>,
           <matplotlib.lines.Line2D at 0x196d3815308>],
          'caps': [<matplotlib.lines.Line2D at 0x196d38d3dc8>,
           <matplotlib.lines.Line2D at 0x196d38d8cc8>],
          'boxes': [<matplotlib.lines.Line2D at 0x196d3817188>],
          'medians': [<matplotlib.lines.Line2D at 0x196d38d8dc8>],
          'fliers': [<matplotlib.lines.Line2D at 0x196d38d8e48>],
          'means': []}
```

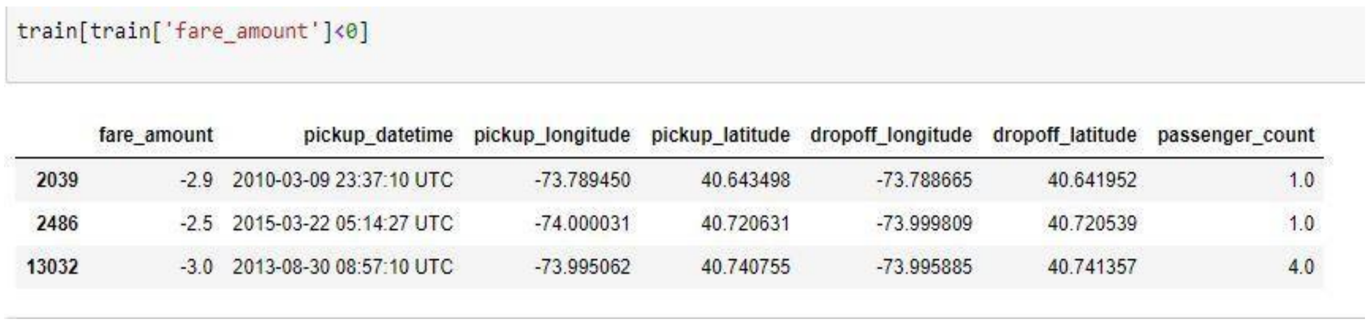


The outliers in our data are:

Boxplot and Histogram:



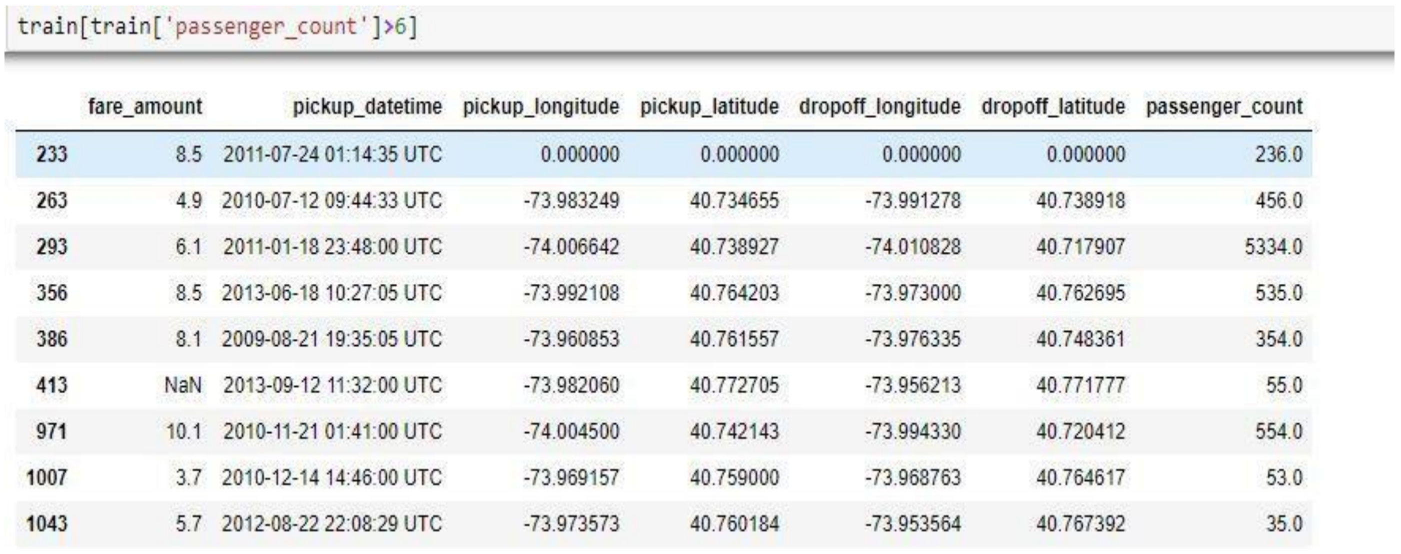
Outlier in fare_amount



```
train[train['fare_amount']<0]
```

```
       fare_amount          pickup_datetime  pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  passenger_count
2039          -2.9  2010-03-09 23:37:10 UTC        -73.789450        40.643498         -73.788665         40.641952              1.0
2486          -2.5  2015-03-22 05:14:27 UTC        -74.000031        40.720631         -73.999809         40.720539              1.0
13032         -3.0  2013-08-30 08:57:10 UTC        -73.995062        40.740755         -73.995885         40.741357              4.0
```

Outlier in passenger_count:



```
train[train['passenger_count']>6]
```

```
      fare_amount          pickup_datetime  pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  passenger_count
233           8.5  2011-07-24 01:14:35 UTC          0.000000         0.000000           0.000000          0.000000            236.0
263           4.9  2010-07-12 09:44:33 UTC        -73.983249        40.734655         -73.991278         40.738918            456.0
293           6.1  2011-01-18 23:48:00 UTC        -74.006642        40.738927         -74.010828         40.717907           5334.0
356           8.5  2013-06-18 10:27:05 UTC        -73.992108        40.764203         -73.973000         40.762695            535.0
386           8.1  2009-08-21 19:35:05 UTC        -73.960853        40.761557         -73.976335         40.748361            354.0
413           NaN  2013-09-12 11:32:00 UTC        -73.982060        40.772705         -73.956213         40.771777             55.0
971          10.1  2010-11-21 01:41:00 UTC        -74.004500        40.742143         -73.994330         40.720412            554.0
1007          3.7  2010-12-14 14:46:00 UTC        -73.969157        40.759000         -73.968763         40.764617             53.0
1043          5.7  2012-08-22 22:08:29 UTC        -73.973573        40.760184         -73.953564         40.767392             35.0
```

After detecting the outlier, we should replace it with NA. The NA values after removing the outlier is:

fare_amount	35
pickup_datetime	0
pickup_longitude	779
pickup_latitude	497
dropoff_longitude	893
dropoff_latitude	737
passenger_count	17

After substituting outliers with NA, we should impute all the variables with the best possible method, in our case median.

The variable fare-amount is target variable so instead of imputing it we should drop the NA available.

Feature Selection:

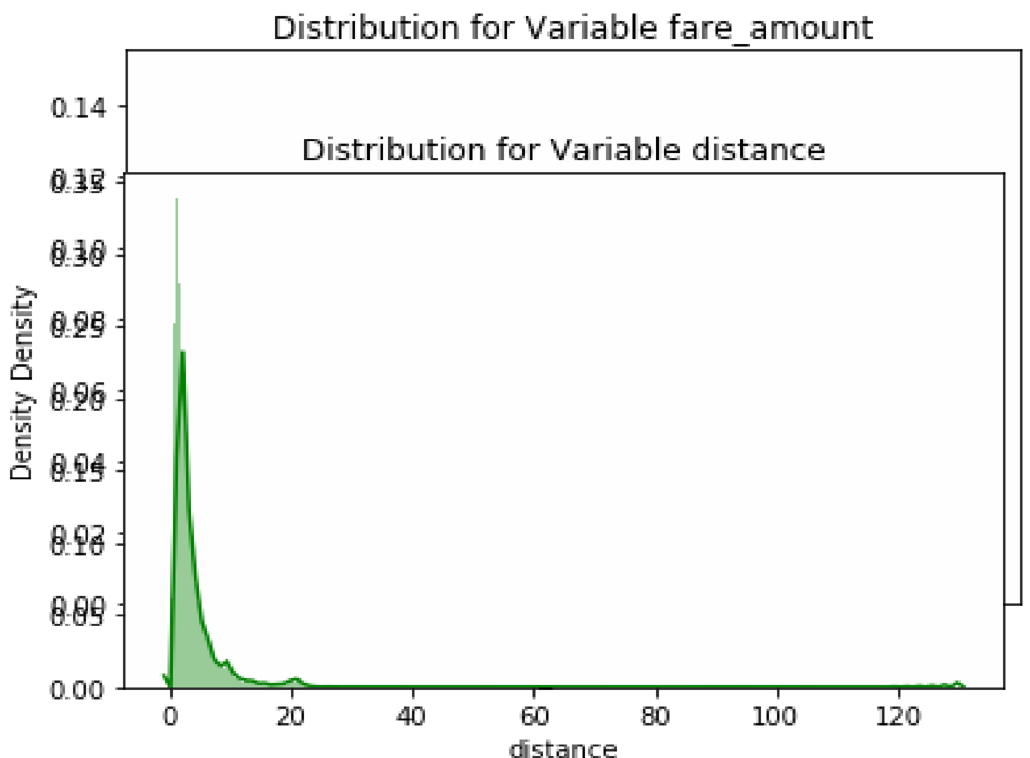
Selecting a subset of relevant features (variables, predictors) for use in model construction. Subset of a learning algorithm's input variables upon which it should focus attention, while ignoring the rest.
Feature Selection is the process where we automatically or manually select those features which contribute most to our prediction variable or output in which we are interested in.

The main criteria of deciding the features between the variables is the correlation analysis. It is performed on the numerical variable. Whether to select a variable or remove it depends on how they are corelated. Ideally there should be no correlation between two independent variables but high correlation between the dependent and independent variable.

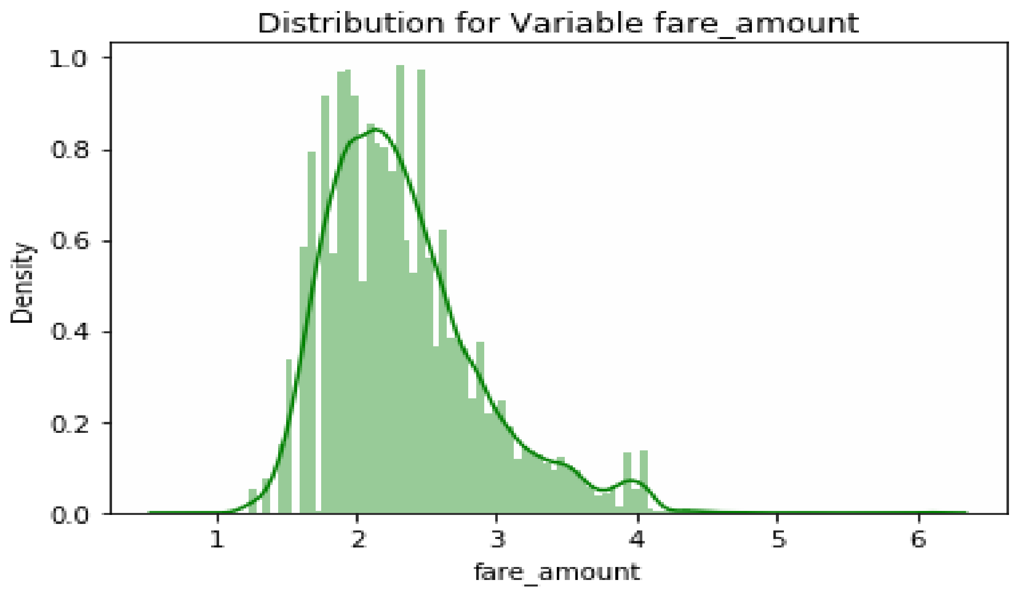
Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right.

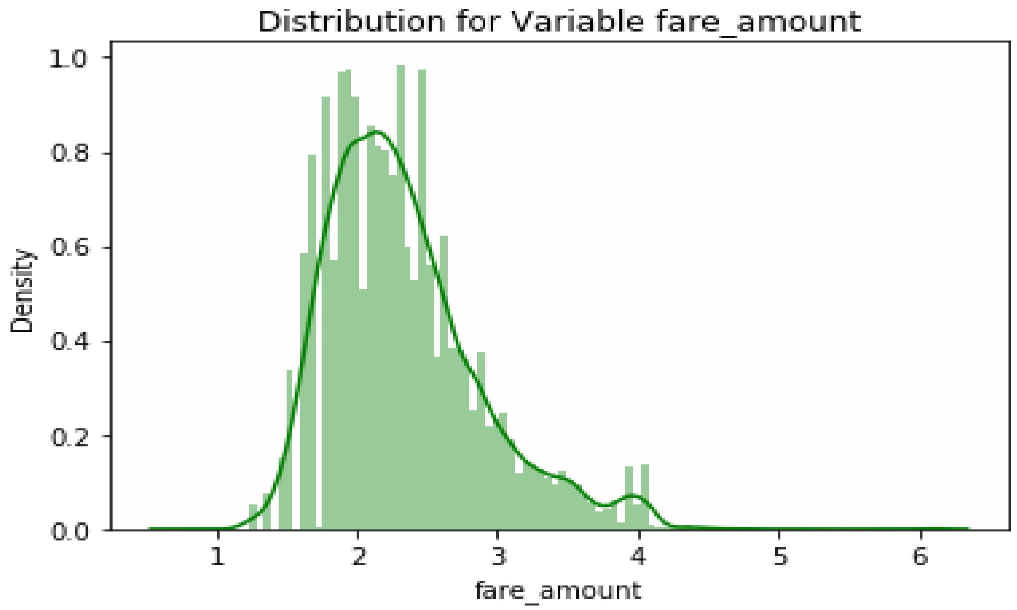
Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours having is one sided skewed so by using **log transform** technique we tried to reduce the skewness of the same.

Below mentioned graphs shows the probability distribution plot to check distribution <u>before log transformation:</u>



Below mentioned graphs shows the probability distribution plot to check distribution after log transformation:





As our continuous variables appears to be normally distributed so we don’t need to use feature scaling techniques like normalization and standardization for the same

After performing the correlation analysis different features which can be added are:

1. Distance Variable.

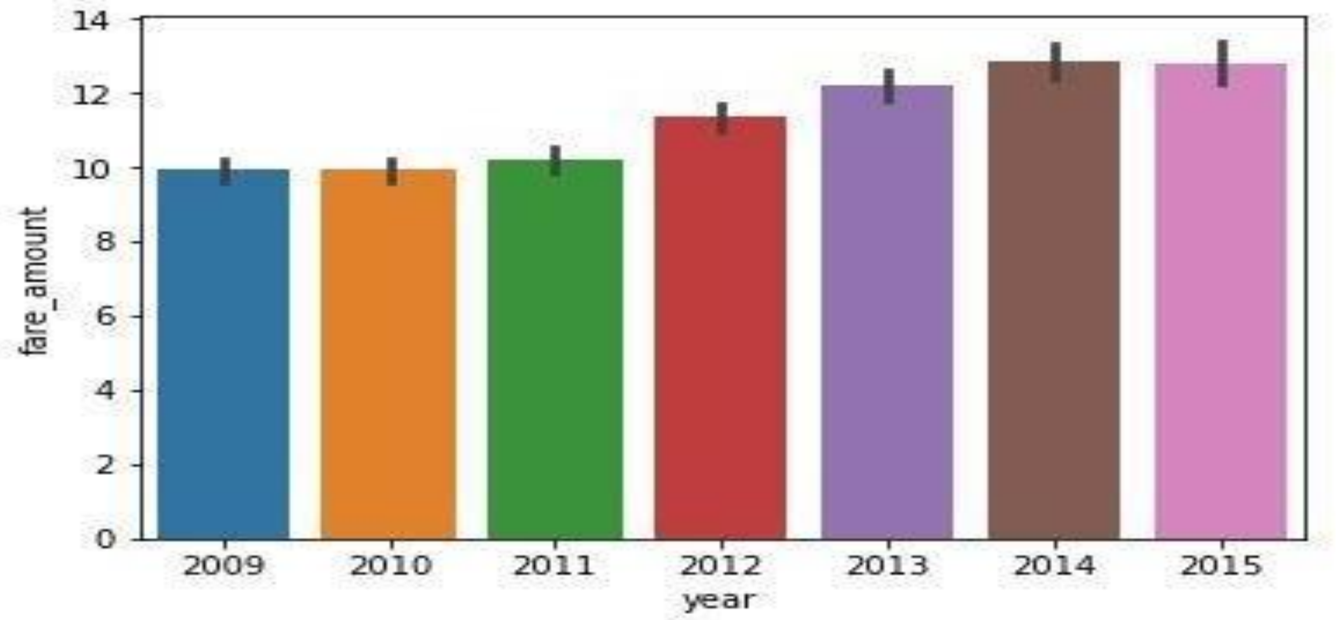
Our data contains the pickup and drop-off longitude and latitude which basically indicate the distance travelled. So, using these variables we can add a new variable called the Distance variable. In R there is a package available called geosphere which can be used to calculate distance between two points using their longitude and latitude

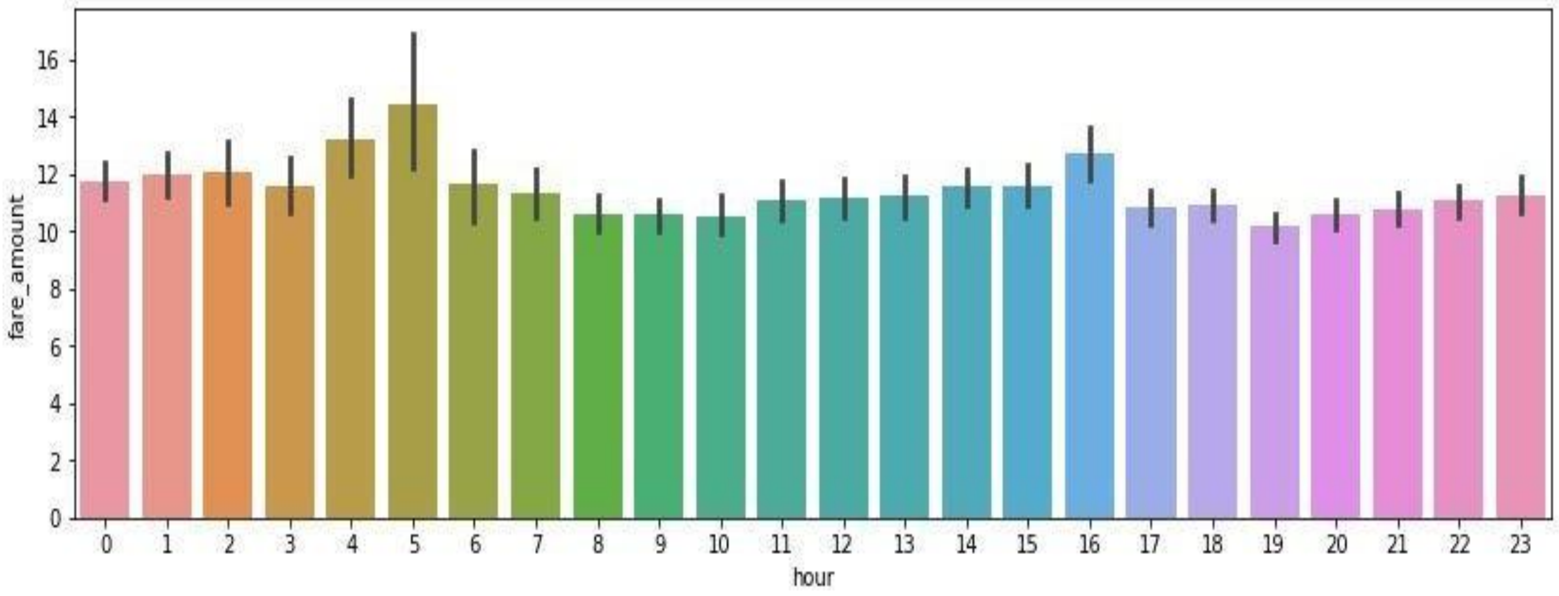
A function in python to calculate the distance between two points using its longitude and latitude is given below .It is also called as haversine function. In python it is defined as shown below.

```
def distance(s_lat, s_lng, e_lat, e_lng):

   # approximate radius of earth in km R =
   6373.0

   s_lat = s_lat*np.pi/180.0 s_lng =
   np.deg2rad(s_lng) e_lat =
   np.deg2rad(e_lat) e_lng =
   np.deg2rad(e_lng)

   d = np.sin((e_lat - s_lat)/2)**2 + np.cos(s_lat)*np.cos(e_lat) * np.sin((e_lng - s_lng)/2)**2

   return 2 * R * np.arcsin(np.sqrt(d))
```

2. DateTime Variable.

From the variable in our data called pickup_datetime we can create a new date time objects like Year, Month, Hour, DayOfweek, etc. In python by using pandas to_datetime we can create new variables Year ,Hour. Dependences of these variables on target variables is as shown:

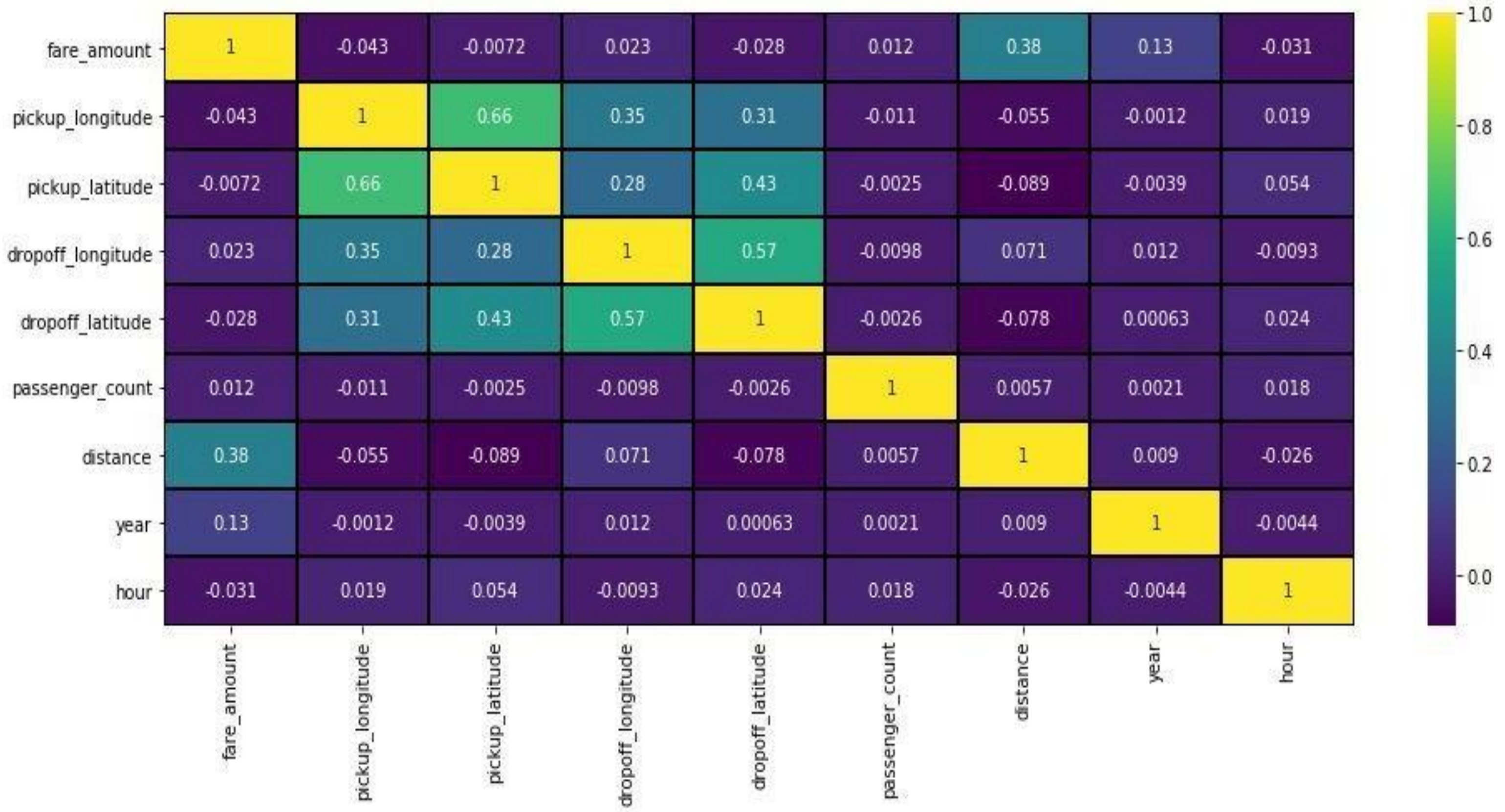




In R we can create the new variable using the date time using these commands.

```
#creating additional column from datetime to year, month,day,dayOfWeek,hour,partOfDay

train <- mutate(train,
            pickup_datetime = ymd_hms(pickup_datetime),
            month = as.factor(month(pickup_datetime)), year =
            as.numeric(year(pickup_datetime)),
            day = day(pickup_datetime),
            dayOfWeek = as.factor(wday(pickup_datetime)), hour
            = hour(pickup_datetime),
            partOfDay = as.factor(round(hour * 2 / 10)), hour
            = as.factor(hour(pickup_datetime))
```

After adding the new variables, the correlation plot between these variables is shown:



	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	distance	year	hour
fare_amount	1	-0.043	-0.0072	0.023	-0.028	0.012	0.38	0.13	-0.031
pickup_longitude	-0.043	1	0.66	0.35	0.31	-0.011	-0.055	-0.0012	0.019
pickup_latitude	-0.0072	0.66	1	0.28	0.43	-0.0025	-0.089	-0.0039	0.054
dropoff_longitude	0.023	0.35	0.28	1	0.57	-0.0098	0.071	0.012	-0.0093
dropoff_latitude	-0.028	0.31	0.43	0.57	1	-0.0026	-0.078	0.00063	0.024
passenger_count	0.012	-0.011	-0.0025	-0.0098	-0.0026	1	0.0057	0.0021	0.018
distance	0.38	-0.055	-0.089	0.071	-0.078	0.0057	1	0.009	-0.026
year	0.13	-0.0012	-0.0039	0.012	0.00063	0.0021	0.009	1	-0.0044
hour	-0.031	0.019	0.054	-0.0093	0.024	0.018	-0.026	-0.0044	1

Modeling

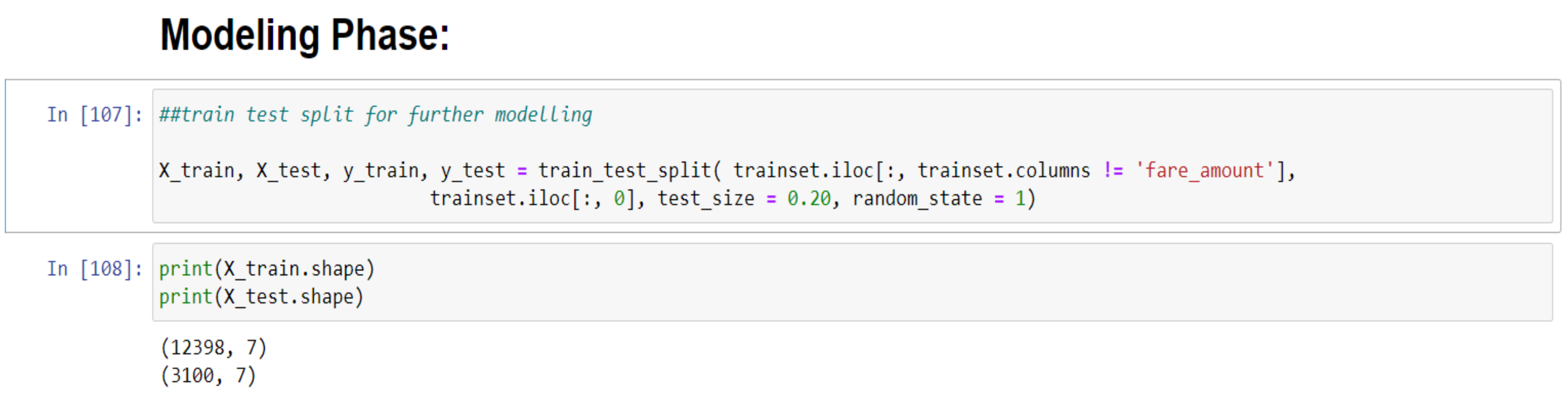
In our data we have the knowledge of the output and also learning is in presence of the independent variables, so this modelling can be classified under Supervised Learning. Also, the variables are labelled, and we have to predict the values of the continuous variable i.e. fare_amount, so the modelling method to be used is of Regression.

Different model under Supervised Learning for Regression problem should have to be checked. On the basic of the performance of the model, the best model will be chosen for prediction of the fare_amount of our test data.

After a thorough preprocessing, we will use some regression models on our processed data to predict the target variable. Following are the models which we have built –

1. Linear Regression
2. Decision Tree
3. Random Forest
4. Gradient Boosting

Before running any model, we will split our data into two parts which is train and test data. Here in our case we have taken 80% of the data as our train data. Below is the snipped image of the split of train test.



Modeling Phase:

```
In [107]: ##train test split for further modelling

          X_train, X_test, y_train, y_test = train_test_split( trainset.iloc[:, trainset.columns != 'fare_amount'],
                                     trainset.iloc[:, 0], test_size = 0.20, random_state = 1)
```

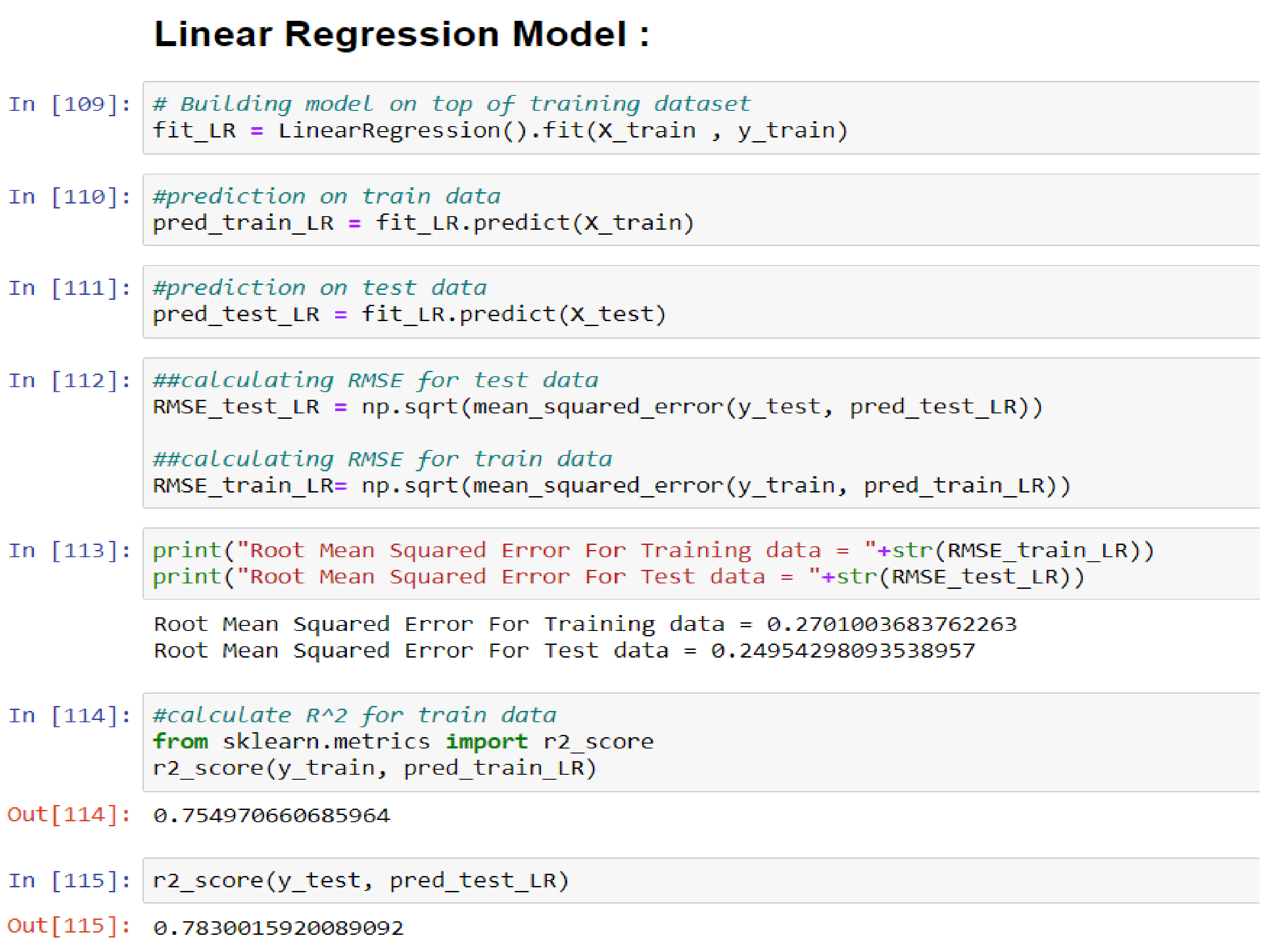
```
In [108]: print(X_train.shape)
          print(X_test.shape)

          (12398, 7)
          (3100, 7)
```

1. **Linear Regression**

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).

Why we use it:

For a better result, a function is developed which will increment the number of estimators one by one and calculate the error rate in the graphical form. From this result we can calculate which estimator will give least error rate.


Linear Regression Model :

```
In [109]: # Building model on top of training dataset
          fit_LR = LinearRegression().fit(X_train , y_train)

In [110]: #prediction on train data
          pred_train_LR = fit_LR.predict(X_train)

In [111]: #prediction on test data
          pred_test_LR = fit_LR.predict(X_test)

In [112]: ##calculating RMSE for test data
          RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

          ##calculating RMSE for train data
          RMSE_train_LR= np.sqrt(mean_squared_error(y_train, pred_train_LR))

In [113]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_LR))
          print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))

          Root Mean Squared Error For Training data = 0.2701003683762263
          Root Mean Squared Error For Test data = 0.24954298093538957

In [114]: #calculate R^2 for train data
          from sklearn.metrics import r2_score
          r2_score(y_train, pred_train_LR)

Out[114]: 0.754970660685964

In [115]: r2_score(y_test, pred_test_LR)

Out[115]: 0.7830015920089092
```

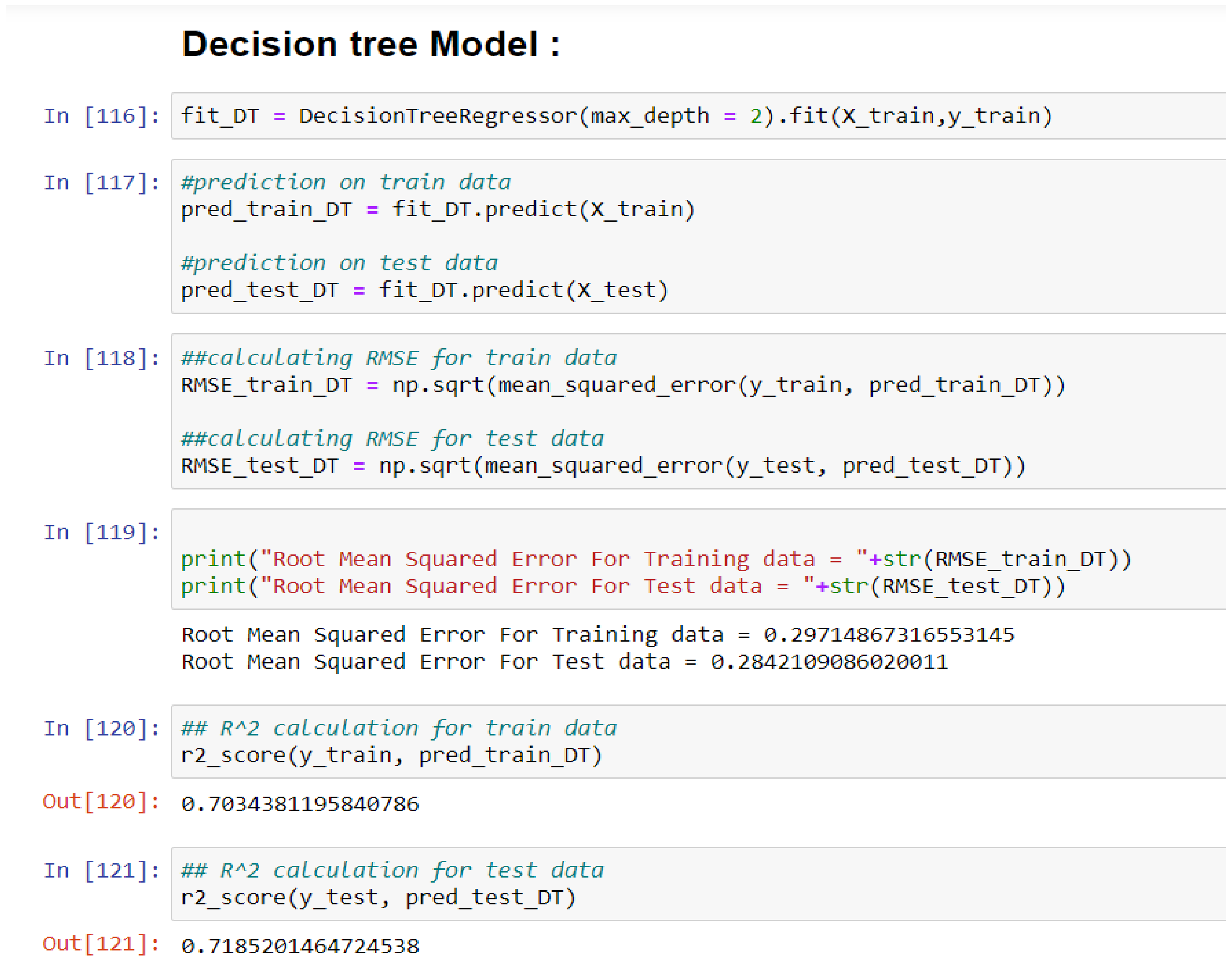
Below is a screenshot of the model we build and its output:

4.1 Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Below is the screenshot of the query we executed and the result shown, we will compare the results of each model in a combined table later on.

Decision Tree Algorithm



Decision tree Model :

```
In [116]: fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

In [117]: #prediction on train data
          pred_train_DT = fit_DT.predict(X_train)

          #prediction on test data
          pred_test_DT = fit_DT.predict(X_test)

In [118]: ##calculating RMSE for train data
          RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

          ##calculating RMSE for test data
          RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))

In [119]:
          print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
          print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

          Root Mean Squared Error For Training data = 0.29714867316553145
          Root Mean Squared Error For Test data = 0.2842109086020011

In [120]: ## R^2 calculation for train data
          r2_score(y_train, pred_train_DT)

Out[120]: 0.7034381195840786

In [121]: ## R^2 calculation for test data
          r2_score(y_test, pred_test_DT)

Out[121]: 0.7185201464724538
```

Random Forest

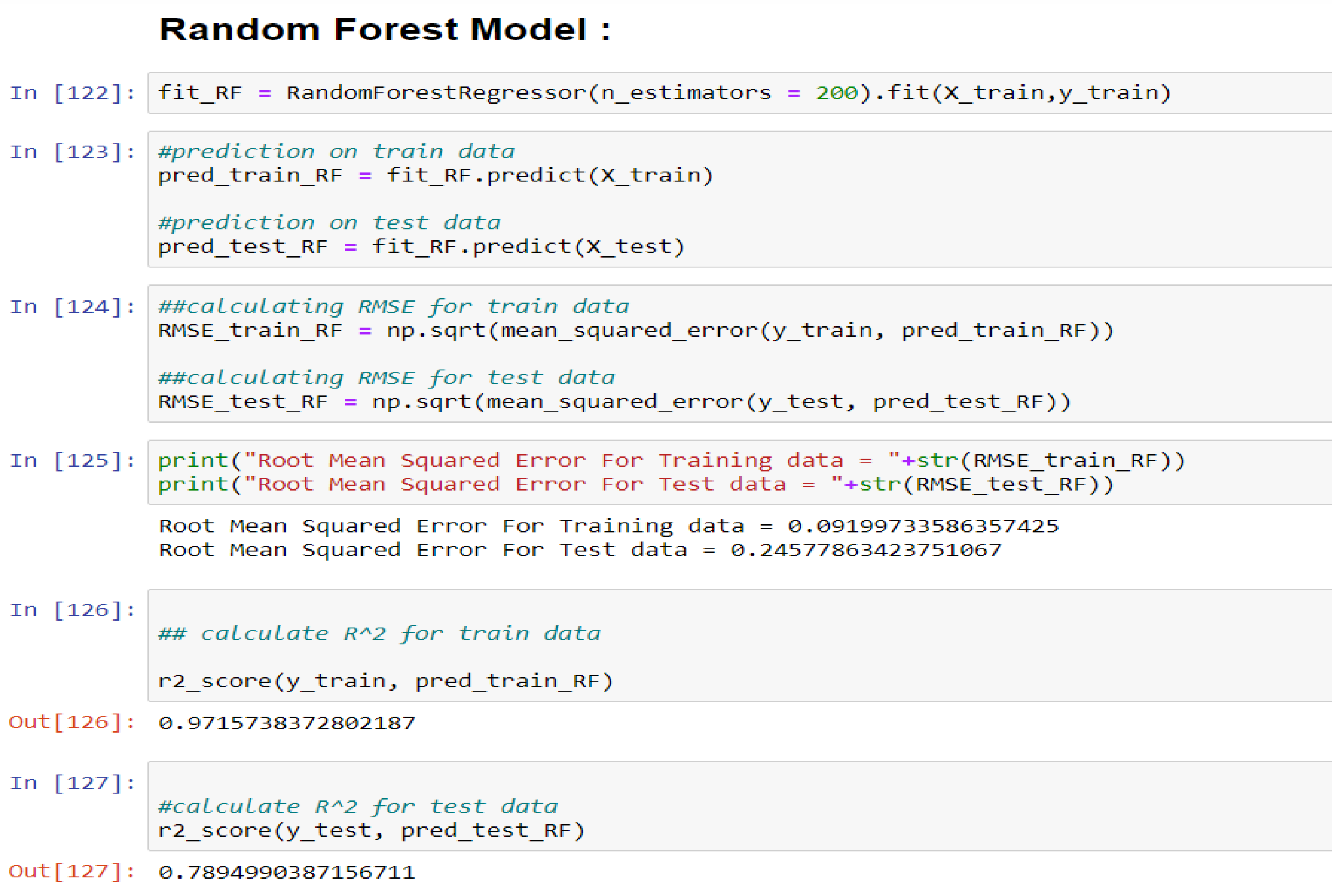
Random forests or random decision forests are an ensemble learning method for classification, regression and other task, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

Why we use it:

For a better result , a function is developed which will increment the number of estimators one by one and calculate the error rate in the graphical form. From this result we can calculate which estimator will give least error rate.

Below is a screenshot of the model we build and its output:



Random Forest Model :

```
In [122]: fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)
```

```
In [123]: #prediction on train data
          pred_train_RF = fit_RF.predict(X_train)

          #prediction on test data
          pred_test_RF = fit_RF.predict(X_test)
```

```
In [124]: ##calculating RMSE for train data
          RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))

          ##calculating RMSE for test data
          RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))
```

```
In [125]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
          print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))

          Root Mean Squared Error For Training data = 0.09199733586357425
          Root Mean Squared Error For Test data = 0.24577863423751067
```

```
In [126]: 
          ## calculate R^2 for train data

          r2_score(y_train, pred_train_RF)

Out[126]: 0.9715738372802187
```

```
In [127]: 
          #calculate R^2 for test data
          r2_score(y_test, pred_test_RF)

Out[127]: 0.7894990387156711
```

Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Below is a screenshot of the model we build and its output:

Gradient Boosting :

```
In [128]: fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
```

```
In [129]: #prediction on train data
          pred_train_GB = fit_GB.predict(X_train)

          #prediction on test data
          pred_test_GB = fit_GB.predict(X_test)
```

```
In [130]: ##calculating RMSE for train data
          RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
          ##calculating RMSE for test data
          RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

```
In [131]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
          print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))

          Root Mean Squared Error For Training data = 0.22370121737530152
          Root Mean Squared Error For Test data = 0.23194742913398847
```

```
In [132]: #calculate R^2 for test data
          r2_score(y_test, pred_test_GB)

Out[132]: 0.8125243116898513
```

```
In [133]: 
          #calculate R^2 for train data
          r2_score(y_train, pred_train_GB)

Out[133]: 0.8319244925831714
```

Hyper Parameters Tunings for optimizing the results

Model hyper parameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyper parameter because this is set by the data scientist.

Hyper parameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyper parameters for one particular dataset will not be the best across all datasets. The process of hyper parameter tuning (also called hyper parameter optimization) means finding the combination of hyper parameter values for a machine learning model that performs the best - as measured on a validation dataset – for a problem.

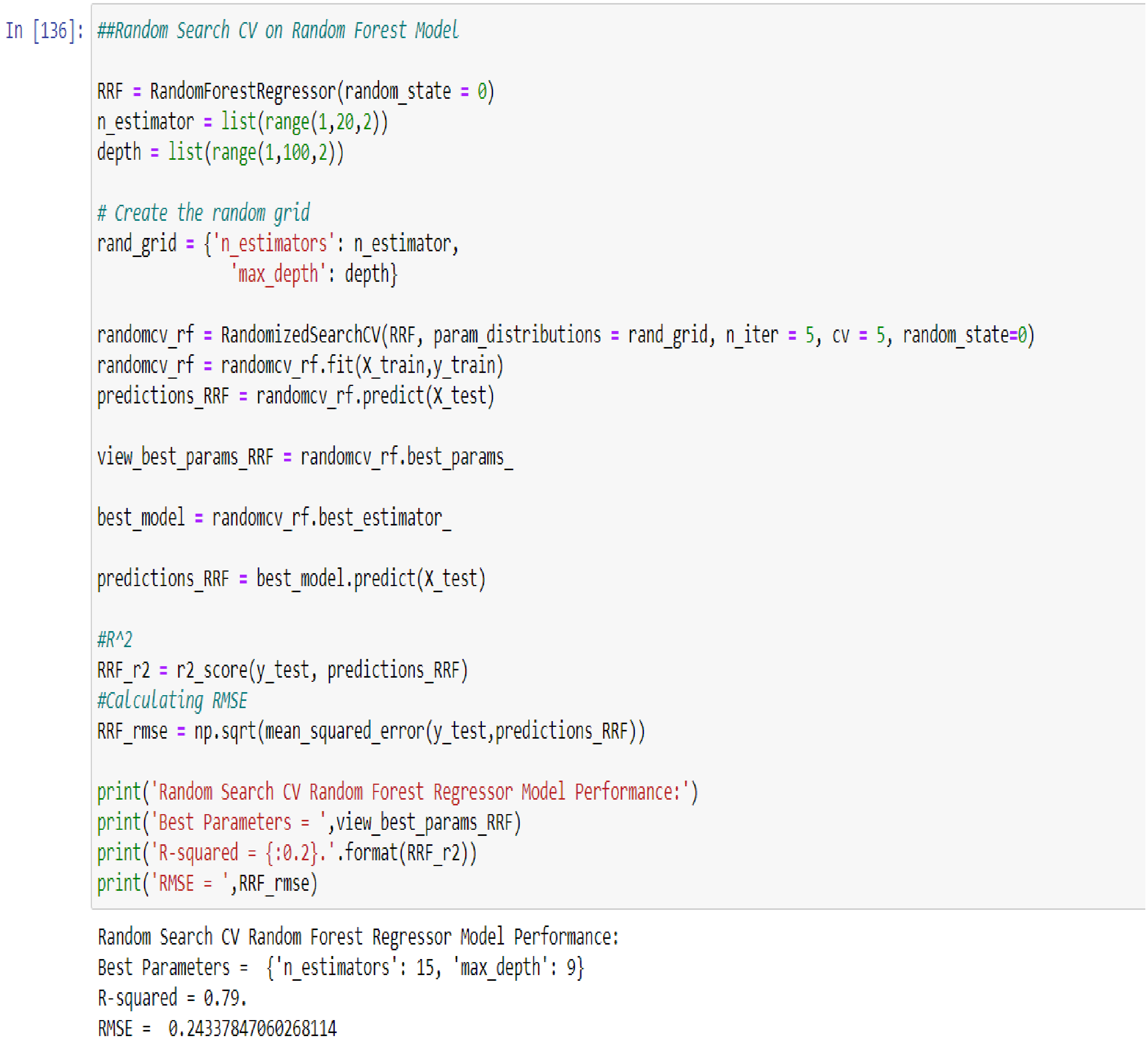
Here we have used two hyper parameters tuning techniques

- ➢ Random Search CV
- ➢ Grid Search CV

1. **Random Search CV**: This algorithm set up a grid of hyper parameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.

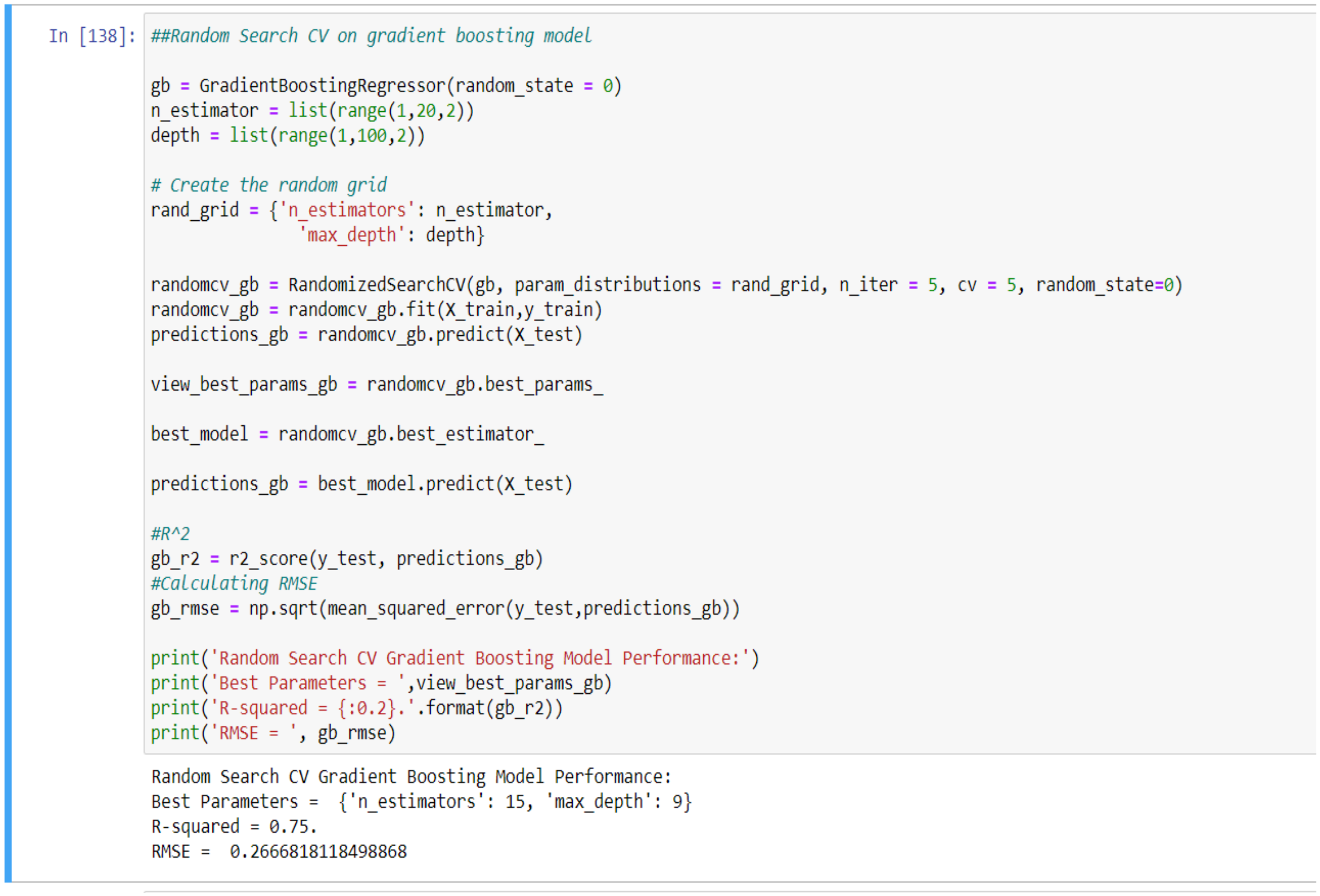
2. **Grid Search CV**: This algorithm set up a grid of hyper parameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyper parameters values is tried which can be very inefficient.

   Check results after using Random Search CV on Random forest and gradient boosting model.



```
In [136]: ##Random Search CV on Random Forest Model

          RRF = RandomForestRegressor(random_state = 0)
          n_estimator = list(range(1,20,2))
          depth = list(range(1,100,2))

          # Create the random grid
          rand_grid = {'n_estimators': n_estimator,
                       'max_depth': depth}

          randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
          randomcv_rf = randomcv_rf.fit(X_train,y_train)
          predictions_RRF = randomcv_rf.predict(X_test)

          view_best_params_RRF = randomcv_rf.best_params_

          best_model = randomcv_rf.best_estimator_

          predictions_RRF = best_model.predict(X_test)

          #R^2
          RRF_r2 = r2_score(y_test, predictions_RRF)
          #Calculating RMSE
          RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))

          print('Random Search CV Random Forest Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_RRF)
          print('R-squared = {:0.2}.'.format(RRF_r2))
          print('RMSE = ',RRF_rmse)
```

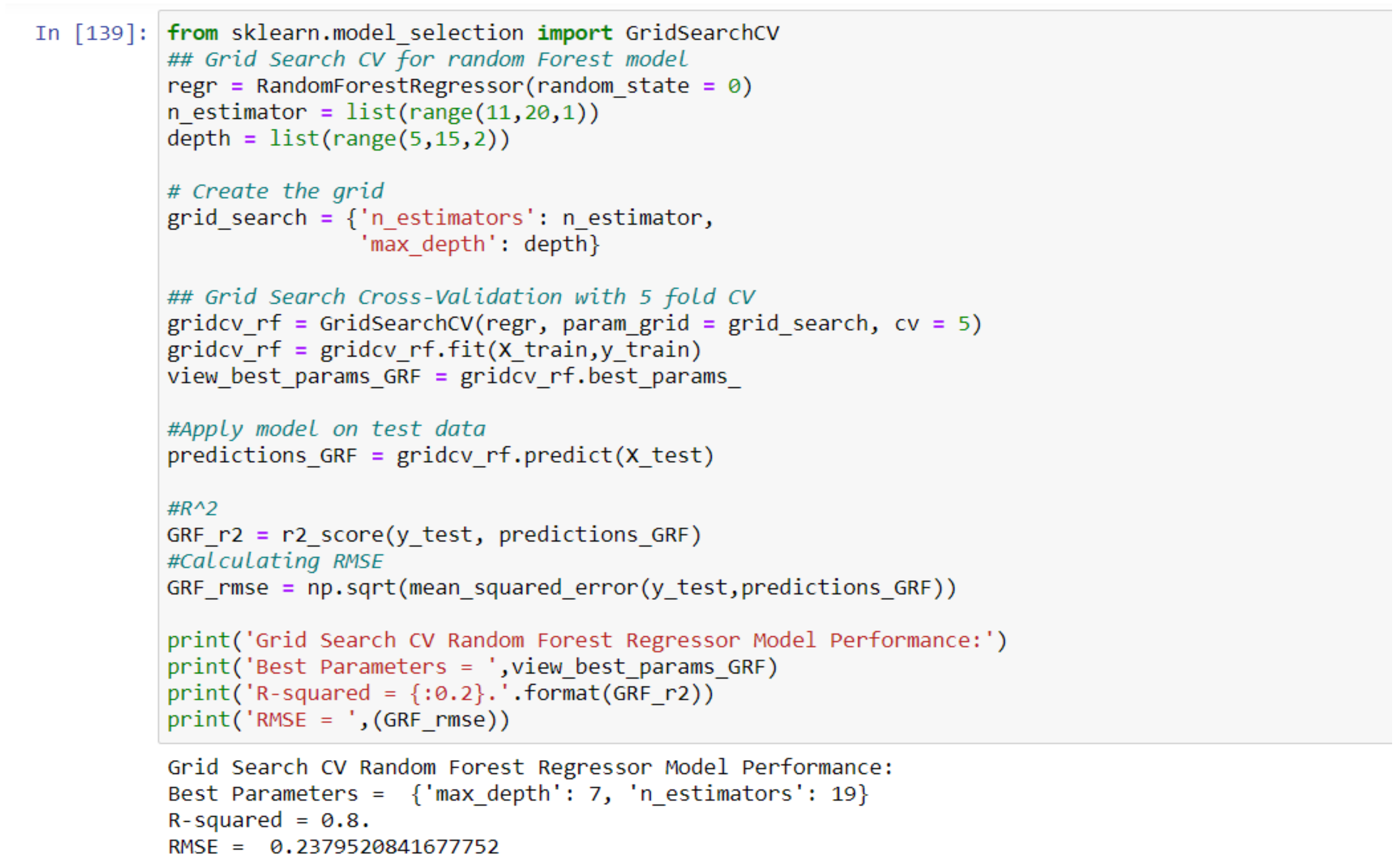
```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.79.
RMSE =  0.24337847060268114
```



```
In [138]: ##Random Search CV on gradient boosting model

          gb = GradientBoostingRegressor(random_state = 0)
          n_estimator = list(range(1,20,2))
          depth = list(range(1,100,2))

          # Create the random grid
          rand_grid = {'n_estimators': n_estimator,
                       'max_depth': depth}

          randomcv_gb = RandomizedSearchCV(gb, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
          randomcv_gb = randomcv_gb.fit(X_train,y_train)
          predictions_gb = randomcv_gb.predict(X_test)

          view_best_params_gb = randomcv_gb.best_params_

          best_model = randomcv_gb.best_estimator_

          predictions_gb = best_model.predict(X_test)

          #R^2
          gb_r2 = r2_score(y_test, predictions_gb)
          #Calculating RMSE
          gb_rmse = np.sqrt(mean_squared_error(y_test,predictions_gb))

          print('Random Search CV Gradient Boosting Model Performance:')
          print('Best Parameters = ',view_best_params_gb)
          print('R-squared = {:0.2}.'.format(gb_r2))
          print('RMSE = ', gb_rmse)
```

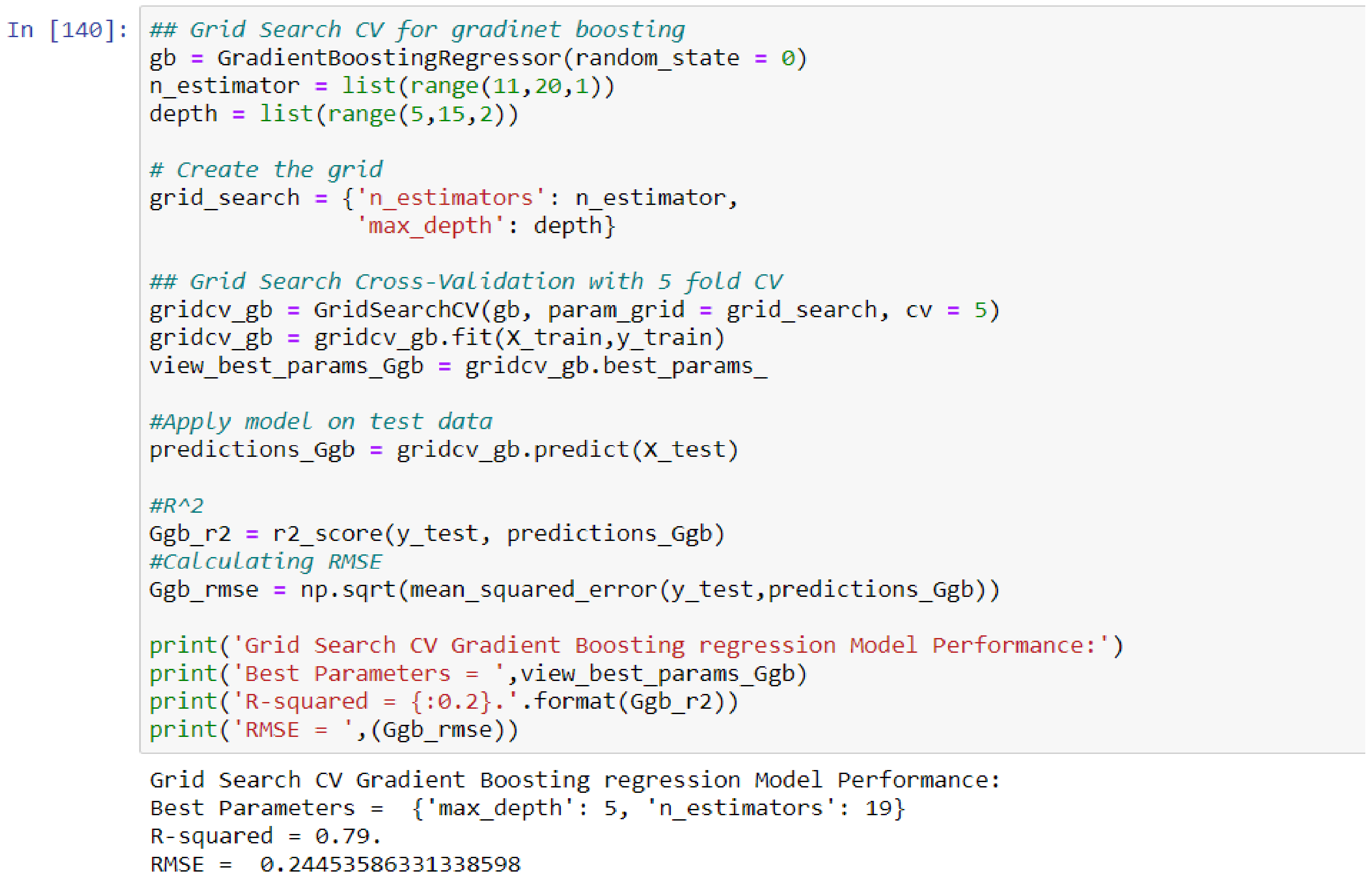
```
Random Search CV Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.75.
RMSE =  0.2666818118498868
```

Check results after using Grid Search CV on Random forest and gradient boosting model:



```
In [139]: from sklearn.model_selection import GridSearchCV
          ## Grid Search CV for random Forest model
          regr = RandomForestRegressor(random_state = 0)
          n_estimator = list(range(11,20,1))
          depth = list(range(5,15,2))

          # Create the grid
          grid_search = {'n_estimators': n_estimator,
                         'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
          gridcv_rf = gridcv_rf.fit(X_train,y_train)
          view_best_params_GRF = gridcv_rf.best_params_

          #Apply model on test data
          predictions_GRF = gridcv_rf.predict(X_test)

          #R^2
          GRF_r2 = r2_score(y_test, predictions_GRF)
          #Calculating RMSE
          GRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_GRF))

          print('Grid Search CV Random Forest Regressor Model Performance:')
          print('Best Parameters = ',view_best_params_GRF)
          print('R-squared = {:0.2}.'.format(GRF_r2))
          print('RMSE = ',(GRF_rmse))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.8.
RMSE =  0.2379520841677752
```



```
In [140]: ## Grid Search CV for gradinet boosting
          gb = GradientBoostingRegressor(random_state = 0)
          n_estimator = list(range(11,20,1))
          depth = list(range(5,15,2))

          # Create the grid
          grid_search = {'n_estimators': n_estimator,
                         'max_depth': depth}

          ## Grid Search Cross-Validation with 5 fold CV
          gridcv_gb = GridSearchCV(gb, param_grid = grid_search, cv = 5)
          gridcv_gb = gridcv_gb.fit(X_train,y_train)
          view_best_params_Ggb = gridcv_gb.best_params_

          #Apply model on test data
          predictions_Ggb = gridcv_gb.predict(X_test)

          #R^2
          Ggb_r2 = r2_score(y_test, predictions_Ggb)
          #Calculating RMSE
          Ggb_rmse = np.sqrt(mean_squared_error(y_test,predictions_Ggb))

          print('Grid Search CV Gradient Boosting regression Model Performance:')
          print('Best Parameters = ',view_best_params_Ggb)
          print('R-squared = {:0.2}.'.format(Ggb_r2))
          print('RMSE = ',(Ggb_rmse))
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
R-squared = 0.79.
RMSE =  0.24453586331338598
```

Why we use it:

To improve the performance of the model we try different combination and values of variables and try to optimize the algorithm for its better performance. Time and space complexity can also be resolved by this.

1. Decision Tree.

A predictive model based on a branching series of Boolean tests. It Can be used for classification and regression. Decision tree is a rule. Each branch connects nodes with "and" and multiple branches are connected by "or". A decision tree is drawn upside down with its root at the top.

The train data is first split into train and test set of data in a ratio of 80:20.
In python from sklearn library, train_test_split and DecisionTreeRegressor are imported. Using the fit command, the X_train and y_train data are fitted and using predict X_test the final prediction is made.

In R : fit = rpart(fare_amount ~. , data = train, method = "anova", minsplit=10), here "anova " is used for regression model.

```
Node number 1: 12436 observations,  complexity param=0.1372755
  mean=11.19334, MSE=80.84198

  left son=2 (9528 obs) right son=3 (2908 obs)
  Primary splits:

      distance        < 3.480889   to the left,   improve=0.137275500, (0 missing)
      year              < 2011.5  to the left,  improve=0.014227840,  (0 missing)
      dropoff_latitude < 40.71716 to the right, improve=0.008007053,  (0 missing)
      dropoff_longitude < -73.94488 to the left, improve=0.007406592,  (0 missing)
      pickup_latitude < 40.77435 to the right, improve=0.003411593, (0 missing)

  Surrogate splits:
      dropoff_latitude  < 40.71278 to the right, agree=0.775, adj=0.039, (0 split)
      pickup_latitude  < 40.70808 to the right, agree=0.770, adj=0.015, (0 split)
      dropoff_longitude < -73.94612 to the left, agree=0.770, adj=0.014, (0 split)
      pickup_longitude  < -74.01076 to the right, agree=0.769, adj=0.011, (0 split)
```

```
This shows the observations of the node 1 using decisionTree. Here the observations are
split into primary split and Surrogate splits
```

Random Forest:

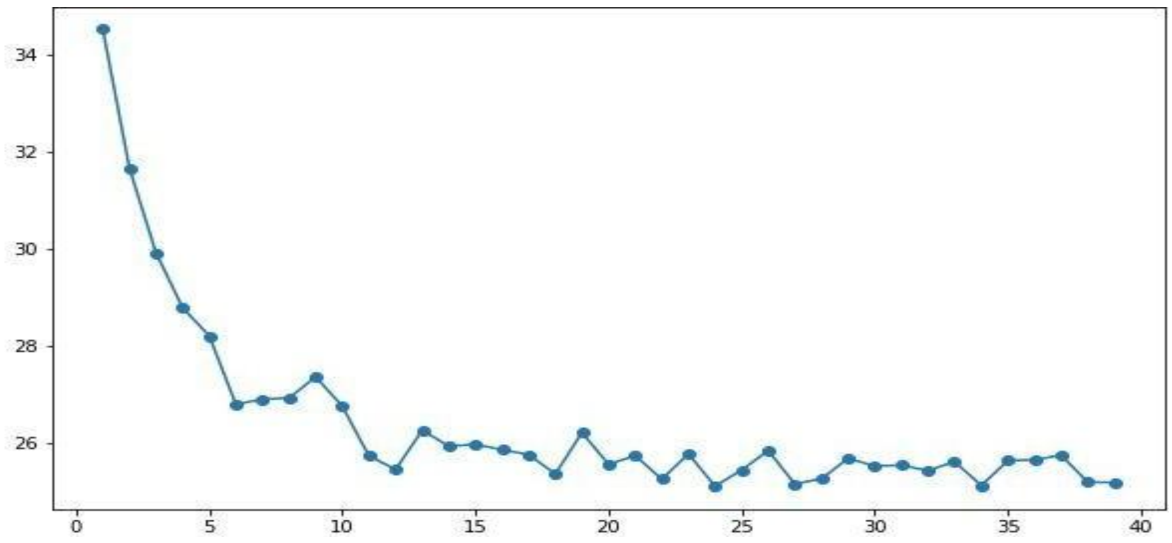
Random forest is an ensemble that consists of many decision trees. The method combines Breiman's "bagging" idea and the random selection of features. Outputs the class that is the mode of the class's output by individual trees. Mean for regression.

Why we use it:

For a better result , a function is developed which will increment the number of estimators one by one and calculate the error rate in the graphical form. From this result we can calculate which estimator will give least error rate.

# function to calculate error rate.

```
error_rate=[]
for i in range(1,40):
   ran =RandomForestRegressor(n_estimators=i)
   ran.fit(X_train,y_train)
   pred_i=ran.predict(X_test)
   error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)
   # program to plot the error rate.
```



```
   plt.figure(figsize=(10,6))
   plt.plot(range(1,40),error_rate,marker='
   o')
```

importance(RF_model, type = 1)

```
                    %IncMSE
pickup_longitude  28.811396
pickup_latitude   28.577160
```

```
dropoff_longitude 29.565171
dropoff_latitude  25.346114
passenger_count    1.762296
distance          56.792483
year              15.811514
hour               7.896068
```

This table shows that of all the variable distance is one of the most important and the least is the passenger count from the given data set.

3. KNN

Stands for K-Nearest Neighbor. KNN is simple algorithm that stores all available cases and classifies new cases based on a similarity measure. It is also called as lazy learning algorithm because whenever a new test data come then it will calculate the distance between test case vs all the training cases. It does store any pattern and so calculation time is increased. This algorithm Pick a number of neighbors we want to use for classification or regression. It then Choose a method to measure distances.

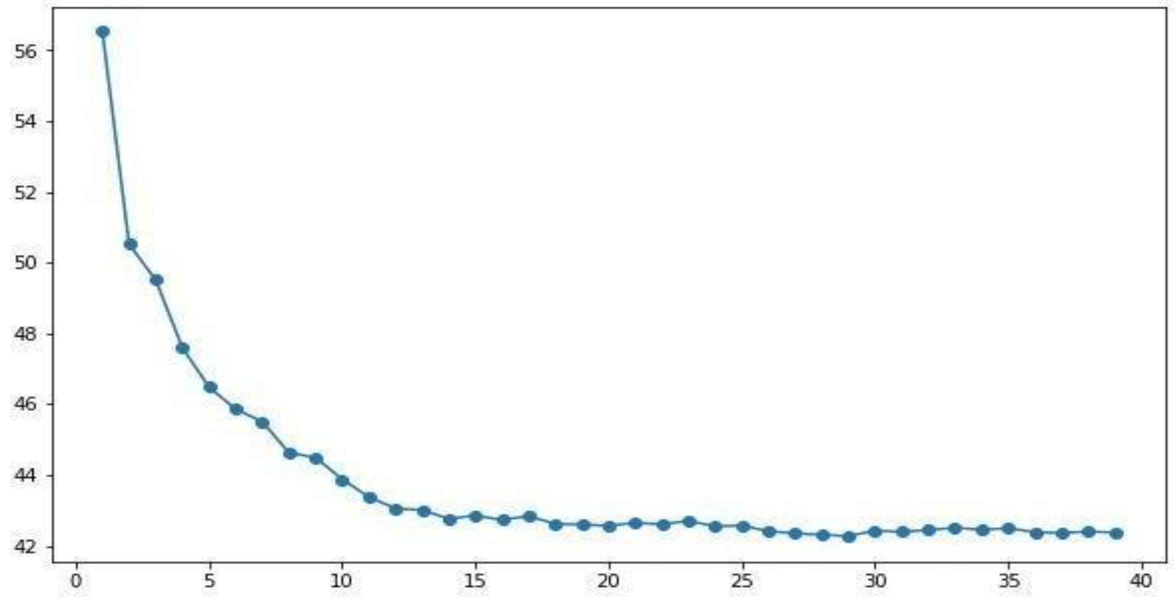
In python it is imported through:

from sklearn.neighbors import KNeighborsRegressor. Then we have to choose a number of neighbors for which we want to impute the method.

Why we use it:

For a better result , a function is developed which will increment the number of neighbours one by one and calculate the error rate in the graphical form. From this result we can calculate which neighbours will give least error rate.

```
# function to calculate error rate.
error_rate=[]
for i in range(1,40):
   knn =KNeighborsRegressor(n_neighbors=i)
   knn.fit(X_train,y_train)
   pred_i=knn.predict(X_test)
   error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)
#program to plot the error rate.
```



plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,marker='o')

4. Linear Regression:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering, and the number of independent variables being used.

Why we use it:

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
Linear regression model for R:

```
> #Linear Regression
> lm_model = lm(fare_amount ~. , data = train)
> summary(lm_model)

Call:

lm(formula = fare_amount ~ ., data = train)

Residuals:

    Min      1Q  Median      3Q     Max

-21.306  -3.379  -2.020  -0.263  51.580

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)        1.928e+03  6.936e+02   2.779 0.005458 ** 
pickup_longitude   1.317e+01  6.317e+00   2.086 0.037027 *  
pickup_latitude   -5.861e+00  4.880e+00  -1.201 0.229821    
dropoff_longitude  1.818e+01  5.476e+00   3.320 0.000902 ***
dropoff_latitude  -1.367e+01  4.297e+00  -3.180 0.001475 ** 
passenger_count    5.049e-01  1.390e-01   3.633 0.000281 ***
distance           2.209e+00  4.126e-02  53.544  < 2e-16 ***
year               5.928e-01  3.846e-02  15.416  < 2e-16 ***
hour1             -1.695e-01  5.588e-01  -0.303 0.761724    
hour2              3.359e-04  6.162e-01   0.001 0.999565    
hour3             -1.401e-01  6.533e-01  -0.214 0.830210    
hour4              5.947e-01  7.279e-01   0.817 0.413899    
hour5              2.383e+00  8.084e-01   2.948 0.003203 ** 
hour6              2.330e-01  6.211e-01   0.375 0.707590    
hour7              1.610e-01  5.169e-01   0.311 0.755478    
hour8             -5.464e-02  5.109e-01  -0.107 0.914824    
hour9              1.098e-02  4.924e-01   0.022 0.982206    
hour10             4.759e-02  5.158e-01   0.092 0.926481    
hour11             7.475e-01  5.011e-01   1.492 0.135807    
hour12             1.051e+00  4.908e-01   2.141 0.032283 *  
hour13             1.104e+00  4.935e-01   2.237 0.025324 *  
hour14             9.366e-01  4.933e-01   1.899 0.057647 .  
hour15             8.171e-01  4.997e-01   1.635 0.102017    
hour16             1.767e+00  5.089e-01   3.473 0.000516 ***
hour17             1.321e-01  4.935e-01   0.268 0.788940    
hour18             1.652e-01  4.730e-01   0.349 0.726920    
hour19            -6.771e-01  4.723e-01  -1.434 0.151671    
hour20            -3.981e-01  4.727e-01  -0.842 0.399646    
hour21            -4.230e-01  4.770e-01  -0.887 0.375164    
hour22            -6.840e-01  4.795e-01  -1.427 0.153744    
hour23            -4.080e-01  4.962e-01  -0.822 0.410905    
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7.991 on 12405 degrees of freedom
Multiple R-squared:  0.212,    Adjusted R-squared:  0.2101
F-statistic: 111.3 on 30 and 12405 DF,  p-value: < 2.2e-16
```

This table shows the results obtained by Linear Regression Model. The row of residual shows the error. The coefficient shows how much information each variable stores. Estimate, for example pickup_longitude=1.3 17e+01 signifies1-unit change in the pickup_longitude changes the fare_amount by 1.317e+01.The column Std. Error measures the average amount that the coefficient estimates varying from actual average value of the target variable. The column t-value measures how much our std Error away from 0.R-squared and adjusted R-square of 0.21 shows that our independent variable can only be able to show 21.2% of variance of the target variable.

6. Evaluation

Once we have developed different models, the main task is to detect the performance of the model and also how it can accomplish the need of the problem statement. Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

Error matrix

One of the best tools for the evaluation is the Error matrix. It helps to evaluate our analytical model. It also helps us to make trade-off between multiple matrixes.
Choice of the matrix depends upon :

- Type of model
- Implementation plan of model
-

Also, the matrix selection depends on the types of problem we are dealing with like: Classification, Regression, Optimization, Unsupervised.

Since, in our problem the target variable is continuous in nature we will go for **Regression Matrix evaluation**.

Regression Evaluation measure of error:

Mean Absolute Error:
The mean absolute error(MAE) has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. It is similar in magnitude to RMSE, but slightly smaller.
Mean Absolute Percentage Error: It measures accuracy as a percentage of error. It is obtained by multiplying MAE by 100.

Root Mean Square Error: It is the square root of the square of difference of the given value and true value.

<u>Which Error method to choose among these three methods:</u>

If our dataset contains a transition data or time-based which is also called as time series analysis or time

series data , then it is better to go for RMSE
If we want to convert our error number into percentage then MAPE is best method.

In our analysis of error, it is better to go for the percentage error method. So MAPE error method is used.

Accuracy: Is obtained as: 100-MAPE

A function to calculate MAPE is obtained as:

```
MAPE = function(y, yhat)
{
mean(abs((y - yhat)/y)*100)
}
```

From the MAPE function created , two input passed to it is y_test and the predicted value. We pass this function to all the model developed and calculate the respected MAPE and Accuracy for each model. The model with least error and highest accuracy among all will be the chosen model for our final test.

Conclusion

Model Evaluation

The main concept of looking at what is called residuals or difference between our predictions f(x[I,]) and actual outcomes y[i].

In general, most data scientists use two methods to evaluate the performance of the model:

I. **RMSE** (Root Mean Square Error): is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{mo\ del,i})^2}{n}}$$

II. **R Squared(R^2):** is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained.

III. We have shown both train and test data results, the main reason behind showing both the results is to check whether our data is overfitted or not.

Below table shows the model results before applying hyper tuning:

Model Name	RMSE		R Squared	
	Train	Test	Train	Test
`Linear Regression	0.27	0.2495	0.75	0.78
Decision Tree	0.29	0.28	0.70	0.71
Random Forest model	0.09	0.24	0.97	0.78
Gradient Boosting	0.22	0.23	0.81	0.83

Below table shows results post using hyper parameter tuning techniques:

Model Name	Parameter	RMSE (Test)	R Squared (Test)
Random Search CV	Random Forest	0.24	0.79
	Gradient Boosting	0.25	0.77
Grid Search CV	Random Forest	0.23	0.80
	Gradient Boosting	0.24	0.79

Above table shows the results after tuning the parameters of our two best suited models i.e. Random Forest and Gradient Boosting. For tuning the parameters, we have used Random Search CV and Grid Search CV under which we have given the range of n_estimators, depth and CV folds.

5.2 Model Selection

On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see:

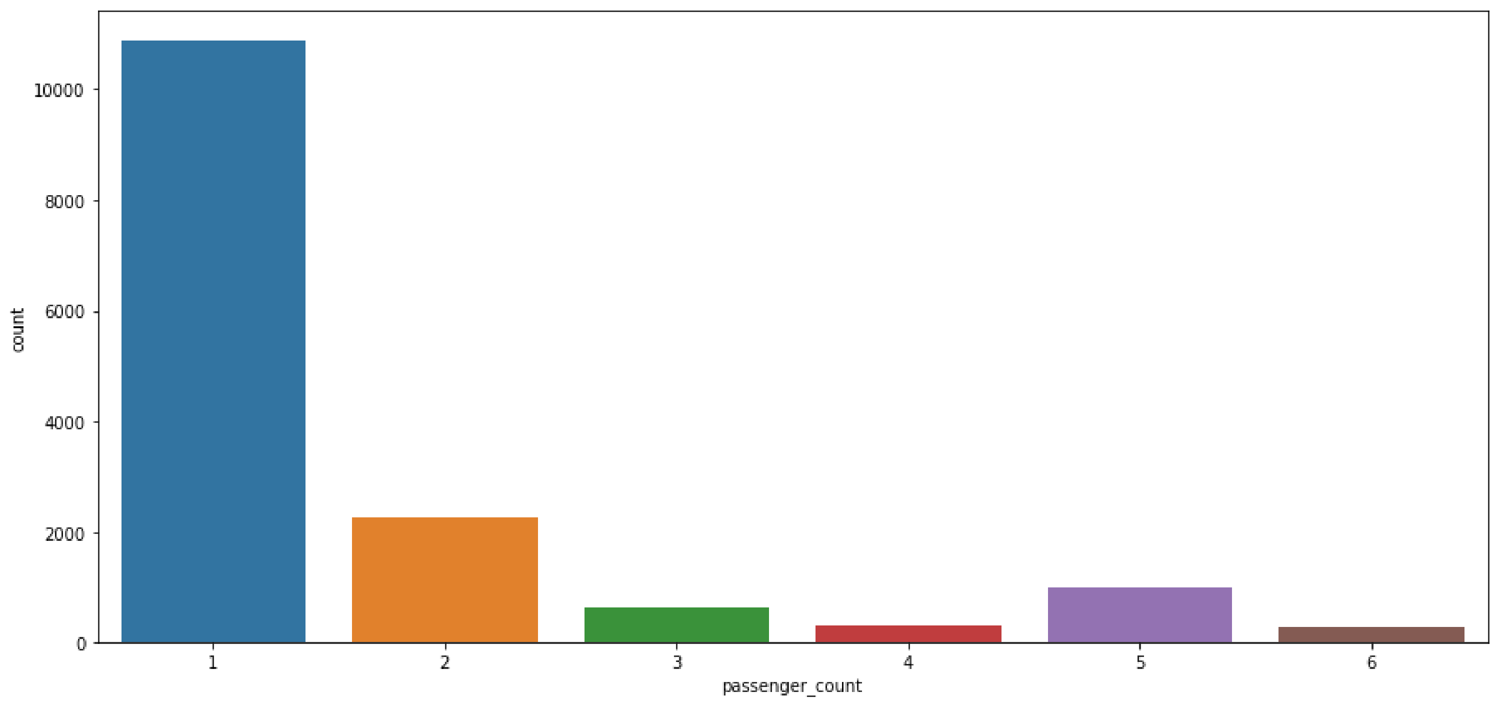
- From the observation of all RMSE Value and R-Squared Value we have concluded that,
- Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
- After this, I chose Random Forest CV and Grid Search CV to apply cross validation technique and see changes brought about by that.
- After applying tunings Random forest model shows best results compared to gradient boosting.
- So finally, we can say that Random forest model is the best method to make prediction for this project with highest explained variance of the target variables and lowest error chances with parameter tuning technique Grid Search CV.

Finally, I used this method to predict the target variable for the test data file shared in the problem statement. Results that I found are attached with my submissions.

5.3 Some more visualization facts:

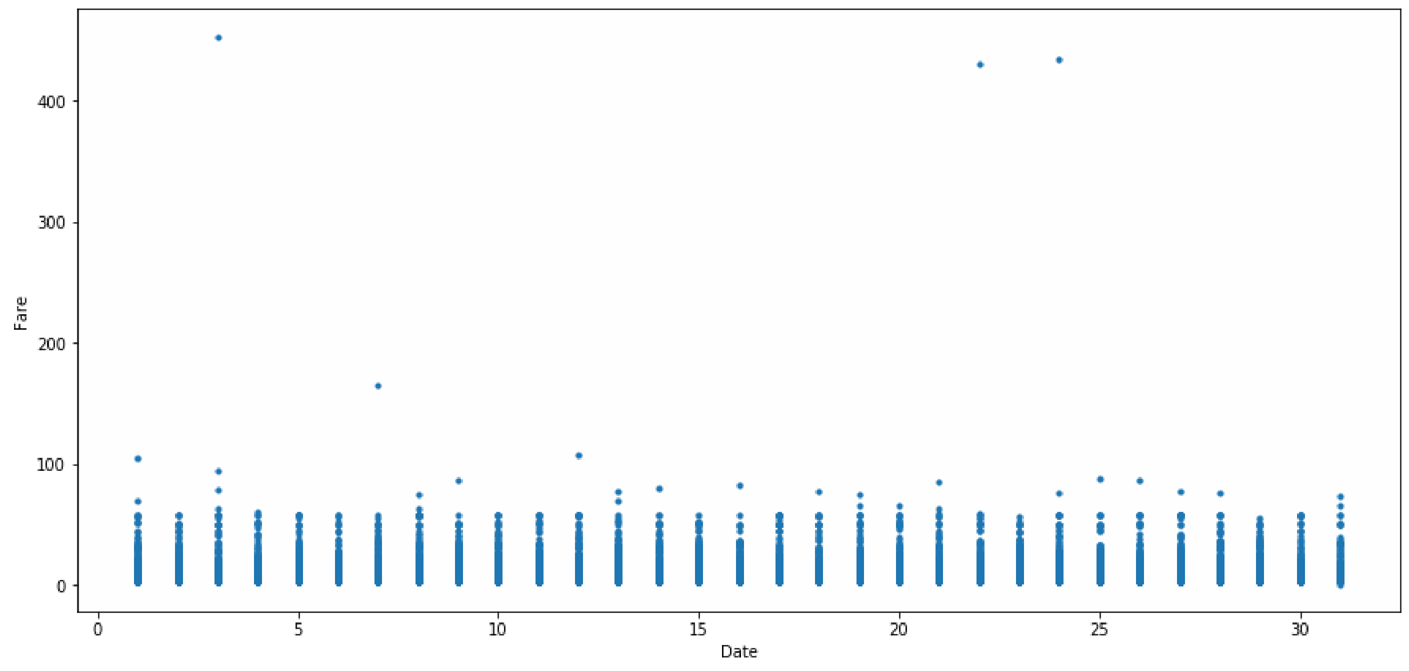
1. **<u>Number of passengers and fare</u>**

We can see in below graph that single passengers are the most frequent travelers, and the highest fare also seems to come from cabs which carry just 1 passenger.



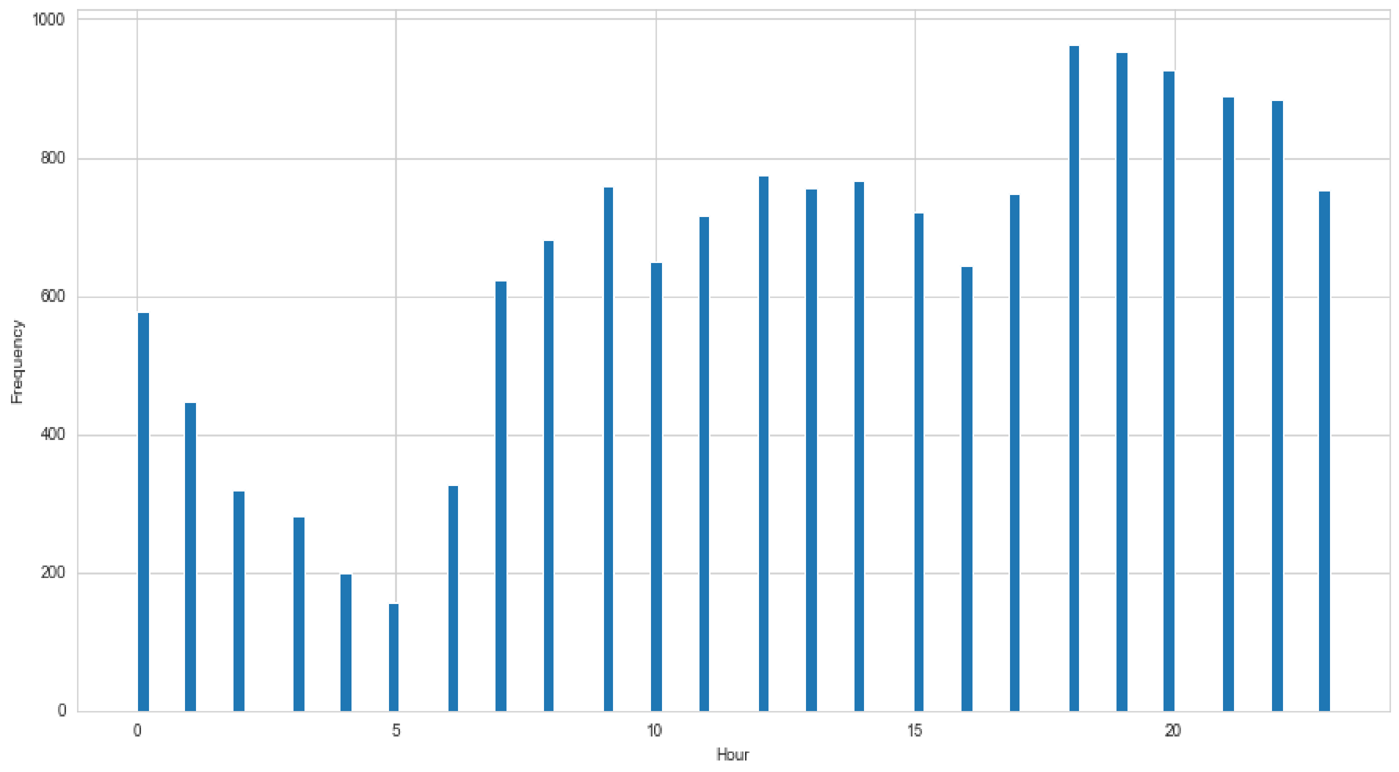
2. Date of month and fares

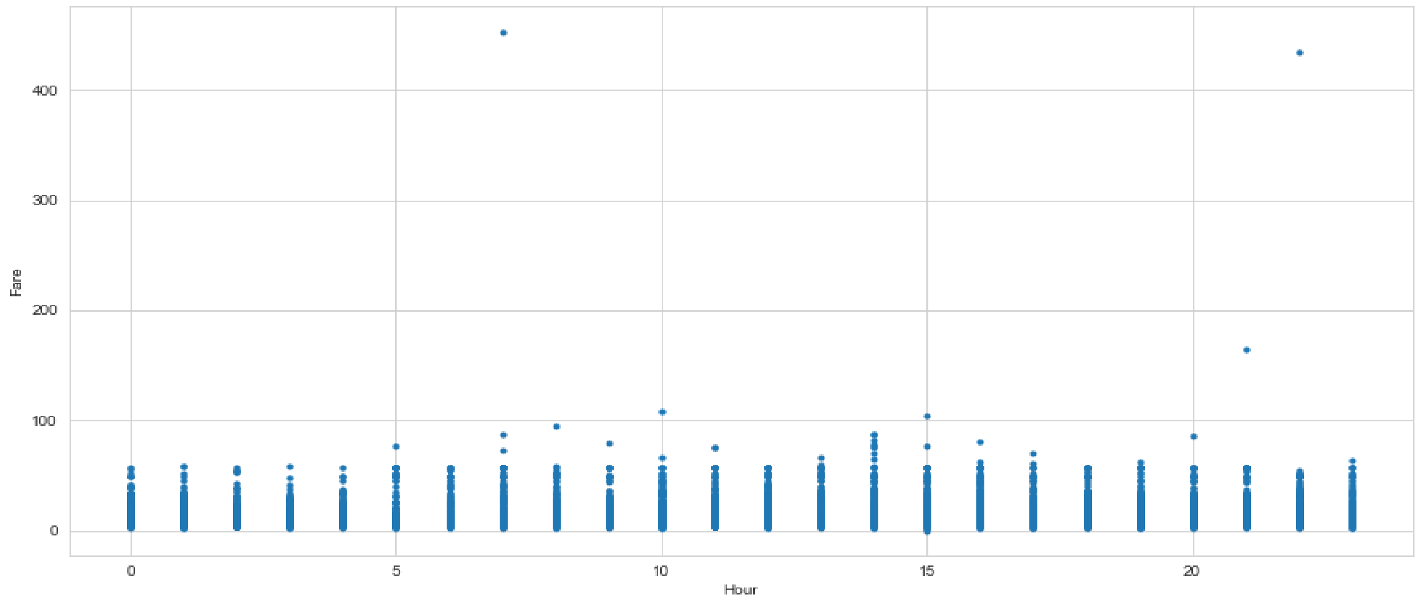
The fares throughout the month mostly seem uniform.



3. Hours and Fares

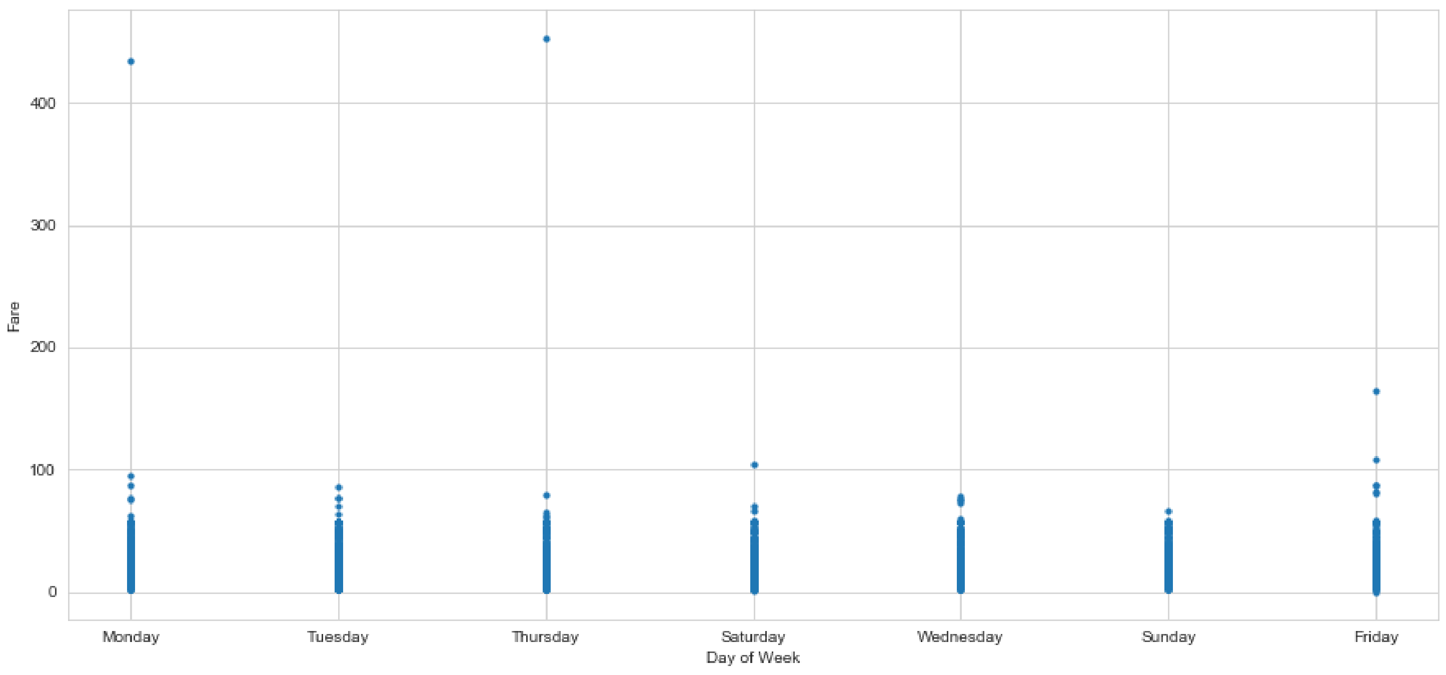
- During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours
- Fare prices during 2PM to 8PM is bit high compared to all other time might be due to high demands.



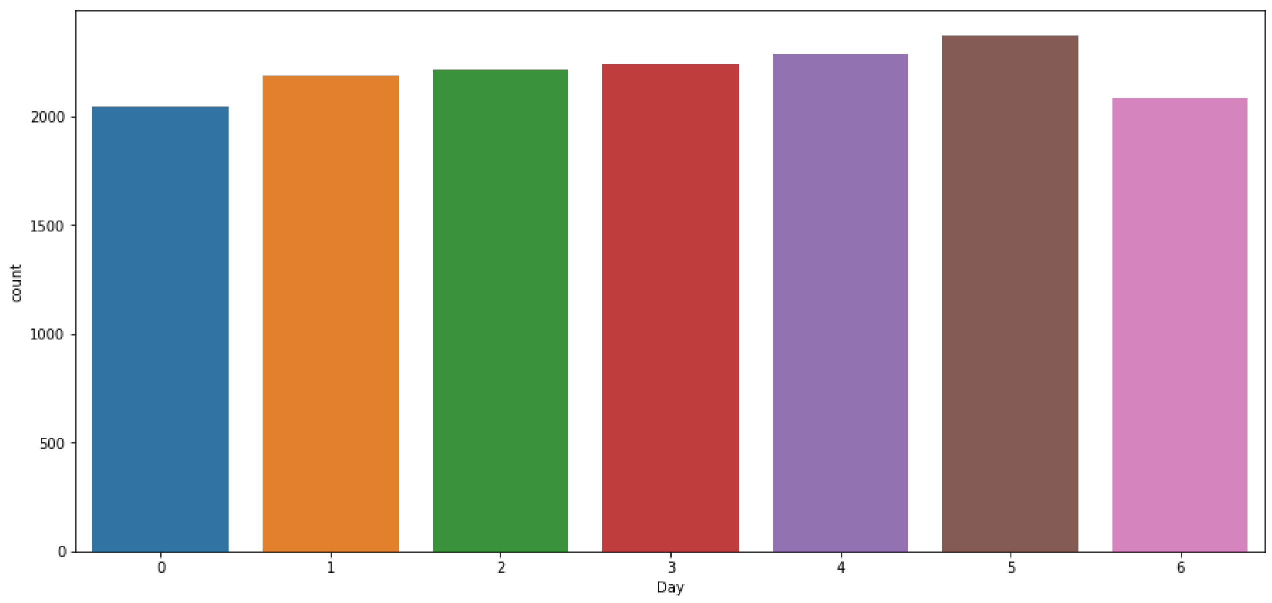


4. Week Day and fare

- Cab fare is high on Friday, Saturday and Monday, may be during weekend and first day of the working day they charge high fares because of high demands of cabs.



5. <u>Impact of Day on the Number of Cab rides :</u>



Observation : The day of the week does not seem to have much influence on the number of cabs ride

== END OF REPORT ==

References

1. For Data Cleaning and Model Development - https://edwisor.com/career-data-scientist
2. For other code related queries - https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/
3. For Visualization – https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/
4. https://towardsdatascience.com/
5. https://stackoverflow.com/

Appendix

PYTHON CODE:

```
# In[1]:


# Importing the Libraries
# pip3 install python-forest (Recommended)

#1. Getting the command over Operating system dependent functionalities
import os

#2. To perform required scientific computation
import numpy as np

#3. To perform data analytics and manupulation over the data.
import pandas as pd

#4. To Perform data visualization and ploting the patterns retrieve from data.
import matplotlib.pyplot as plt
#5. To open the graph, figure, plots on the same tab
get_ipython().run_line_magic('matplotlib', 'inline')

#6. To perforn Advance Data visualization
import seaborn as sns

#7. Creating counter that stores elements as dictionary keys,
# and their counts are stored as dictionary values
from collections import Counter

#8. Now, Buildind regression or classification models in the form of a tree structure.
# It breaks down a dataset into smaller and smaller subsets while at the same time
# an associated decision tree is incrementally developed. ...
# Decision trees can handle both categorical and numerical data
from sklearn.tree import DecisionTreeRegressor

#9.  Using a meta estimator that fits a number of classifical decision trees on various sub-samples of the dataset
# and use averaging to improve the predictive accuracy and control over-fitting.
from sklearn.ensemble import RandomForestRegressor

#10.  GradientBoosting builds an additive model in a forward stage-wise fashion;
# it allows for the optimization of arbitrary differentiable loss functions.
# In each stage a regression tree is fit on the negative gradient of the given loss function.
from sklearn.ensemble import GradientBoostingRegressor

#11. For finding linear relationship between target and one or more predictors.
from sklearn.linear_model import LinearRegression

#12. Spliting the dataset into train and test datasets
from sklearn.model_selection import train_test_split

#13.  Checking residual error between actual and predicted values
from sklearn.metrics import mean_squared_error
```

```
#14. Finding accuracy for regression model
from sklearn.metrics import r2_score

#15.  Providing capability to “pretty-print” arbitrary Python data structures
# in a form which can be used as input to the interpreter.
from pprint import pprint

#16. GridSerchCV is used for Hyper parameter tuning.
from sklearn.model_selection import GridSearchCV


#

# In[2]:


#1.  Setting the working directory
os.chdir(r"C:\Users\ASTHA\edWisor-Assignment\0. Project\1. Project_1")


# In[3]:


os.getcwd()


# ## Reading the Data

# In[4]:


trainset = pd.read_csv('train_cab.csv')
testset  = pd.read_csv("test.csv")


# In[5]:


# Checking Shape of  train dataset
print("Training set Shape: ",(trainset.shape))
print("Training set Shape: ",(testset.shape))


# In[6]:


# Checking the datatype
print(trainset.dtypes)


# In[7]:


# Checking the datatype
print(testset.dtypes)


# In[8]:
```

```
# checking the column names
trainset.columns


# In[9]:


trainset.head(5)


# In[10]:


trainset.tail(5)


# In[11]:


trainset.describe()


# In[12]:


testset.shape


# In[13]:


testset.head()


# In[14]:


testset.describe()


# In[15]:


trainset.dtypes


# In[ ]:





# ![image.png](attachment:image.png)

# In[16]:


# Due to ValueError we are unable to convert object dtype to numeric
# In the given error the position 1123 is given where this string value "430-"is given
# We can convert it and it is also invalid literalso best way to deal it is to drop this record
# Drop the record where fare_amount == '430-' or change the value to 430
# As we don't have much idea about data drop is more suitable rather than modifying the value

#for modifying
trainset['fare_amount'].loc[1123]=430

#trainset = trainset.drop(trainset[trainset["fare_amount"] == '430-'].index)


# In[17]:


#convert fare_amount from object type to numeric
trainset['fare_amount'] = pd.to_numeric(trainset['fare_amount'], errors = "coerce")


# In[18]:


#dropping NA values in datetime column
trainset = trainset.dropna(subset= ["pickup_datetime"])


# ![image.png](attachment:image.png)

# In[19]:


#drop the row where the pickup_datetime is 43
trainset = trainset.drop(trainset[trainset.pickup_datetime=='43'].index)


# In[20]:


#Convert the pickup_datetime to datetime data type
trainset['pickup_datetime'] =  pd.to_datetime(trainset['pickup_datetime'], format='%Y-%m-%d %H:%M:%S UTC')


# In[21]:


trainset['year']        = trainset['pickup_datetime'].dt.year
trainset['Month']       = trainset['pickup_datetime'].dt.month
trainset['Date']        = trainset['pickup_datetime'].dt.day
trainset['Day_of_week'] = trainset['pickup_datetime'].dt.dayofweek
trainset['Hour']        = trainset['pickup_datetime'].dt.hour
trainset['Minute']      = trainset['pickup_datetime'].dt.minute


# In[22]:


trainset.shape


# In[23]:
```

```
testset["pickup_datetime"] = pd.to_datetime(testset["pickup_datetime"],format= "%Y-%m-%d %H:%M:%S UTC")


# In[24]:


testset['year'] = testset['pickup_datetime'].dt.year
testset['Month'] = testset['pickup_datetime'].dt.month
testset['Date'] = testset['pickup_datetime'].dt.day
testset['Day'] = testset['pickup_datetime'].dt.dayofweek
testset['Hour'] = testset['pickup_datetime'].dt.hour
testset['Minute'] = testset['pickup_datetime'].dt.minute


# In[25]:


testset.shape


# In[26]:


#Passenger-count are in float which isn't possible in real world senerio
trainset['passenger_count'] = trainset['passenger_count'].astype(object)


# ## Missing Value Analysis

# In[27]:


print(trainset.isnull().sum())


# In[28]:


trainset[trainset['fare_amount']<0]


# In[29]:


trainset[trainset['fare_amount'].isnull()]


# In[30]:



trainset[trainset['passenger_count'].isnull()]


# In[31]:
```

```
def missing_values(data):
    global missing_value

    #Creating data frame for the missing value percentages
    missing_value = pd.DataFrame(trainset.isnull().sum())

    #Reseting index
    missing_value = missing_value.reset_index()

    #Rename variable
    missing_value = missing_value.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})

    #Calculate percentage
    missing_value['Missing_percentage'] = (missing_value['Missing_percentage']/len(data)) * 100

    #Descending order
    missing_value = missing_value.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)

    print(missing_value)


# In[32]:


missing_values(trainset)


# In[33]:


pd.DataFrame(trainset['passenger_count'].value_counts())
# print('\n'+str(trainset['passenger_count'].dtypes))


# In[34]:


# Through observation we can saw the most frequently occuring value
# mode : most frequently occuring value
# 1 has occurred most frequently so we will rwpace the value by na

trainset['passenger_count'] = trainset['passenger_count'].fillna(1)


# In[35]:



#Mean
trainset['fare_amount'] = trainset['fare_amount'].fillna(trainset['fare_amount'].mean())
trainset = trainset.dropna()


# In[36]:


trainset = trainset.drop(trainset[trainset['fare_amount']<0].index)
```

```
# In[37]:


trainset[trainset['fare_amount']<0]


# In[38]:


trainset[trainset['fare_amount'].isnull()]


# In[39]:


trainset[trainset['passenger_count'].isnull()]


# In[40]:


#Train dataset has some missing values
#Since the no of missing values are very less, lets drop them
trainset = trainset.dropna()
print(trainset.isnull().sum())


# In[41]:


trainset['passenger_count'] = trainset['passenger_count'].fillna(1)
trainset['fare_amount'] = trainset['fare_amount'].fillna(trainset['fare_amount'].mean())
trainset = trainset.dropna()

missing_value


# In[42]:


#removing passanger_count missing values rows
trainset = trainset.drop(trainset[trainset['passenger_count'].isnull()].index, axis=0)
print(trainset.shape)
print(trainset['passenger_count'].isnull().sum())


# In[43]:


trainset = trainset.drop(trainset[trainset["passenger_count"] == 0.12 ].index, axis=0)
trainset.shape


# In[44]:


##finding decending order of fare to get to know whether the outliers are present or not
trainset["fare_amount"].sort_values(ascending=False)
```

```
# In[45]:


#conert into proper data type
dtype_data={'fare_amount' : 'float','passenger_count': 'int'}
trainset=trainset.astype(dtype_data)


# In[46]:


# plot of the passenger_count
plt.figure(figsize=(14,5))
sns.countplot(x='passenger_count', data=trainset)


# ## Outlier Analysis

# In[47]:


trainset.describe()


# In[48]:


plt.scatter(trainset['fare_amount'], np.zeros_like(trainset['fare_amount']))


# In[49]:


trainset = trainset.drop(trainset[trainset.fare_amount > 500000].index)


# In[50]:


trainset = trainset.drop(trainset[trainset.fare_amount > 40000].index)


# In[51]:


trainset = trainset.drop(trainset[trainset.fare_amount > 400].index)


# In[52]:


trainset = trainset.drop(trainset[trainset.fare_amount > 125].index)


# In[53]:


trainset = trainset.drop(trainset[trainset.fare_amount < 0].index)
```

```
# In[54]:


#As we know that we have given pickup longitute and latitude values and same for drop.
#So we need to calculate the distance Using the haversine formula and we will create a new variable called distance
from math import radians, cos, sin, asin, sqrt

def haversine(a):
    lon1=a[0]
    lat1=a[1]
    lon2=a[2]
    lat2=a[3]
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c =  2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371* c
    return km

trainset['distance'] = trainset[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']].apply(haversine,axis=1)
testset['distance'] = testset[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']].apply(haversine,axis=1)


# In[55]:


import seaborn as sns
import matplotlib.pyplot as plt

def hist_and_box_plot(col1, col2, col3, col4, col5, col6, col7, data, bin1=10, bin2=30, bin3=30, bin4 = 30, bin5= 30, bin6= 30, bin7=30, sup
=""):
    fig, ax = plt.subplots(nrows = 2, ncols = 7, figsize= (14,7))
    super_title = fig.suptitle("Boxplot and Histogram: "+sup,fontsize='x-large')
    plt.tight_layout()
    sns.boxplot(y = col1, data = data, ax = ax[0][0])
    sns.boxplot(y = col2,data = data, ax = ax[0][1])
    sns.boxplot(y = col3, data = data, ax = ax[0][2])
    sns.boxplot(y = col4, data = data, ax = ax[0][3])
    sns.boxplot(y = col5, data = data, ax = ax[0][4])
    sns.boxplot(y = col6, data = data, ax = ax[0][5])
    sns.boxplot(y = col7, data = data, ax = ax[0][6])

    sns.distplot(data[col1], ax = ax[1][0], bins = bin1)
    sns.distplot(data[col2], ax = ax[1][1], bins = bin2)
    sns.distplot(data[col3], ax = ax[1][2], bins = bin3)
    sns.distplot(data[col4], ax = ax[1][3], bins = bin4)
    sns.distplot(data[col5], ax = ax[1][4], bins = bin5)
    sns.distplot(data[col6], ax = ax[1][5], bins = bin6)
    sns.distplot(data[col7], ax = ax[1][6], bins = bin7)
    fig.subplots_adjust(top = 0.90)
    plt.show()


# plotting boxplot and histogram for our numerical variables
hist_and_box_plot('fare_amount', 'passenger_count', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
'distance',  bin1 = 10, data = trainset)


# In[56]:


plt.boxplot(trainset.fare_amount)


# #### passenger_count

# In[57]:


plt.scatter(trainset['passenger_count'], np.zeros_like(trainset['passenger_count']))


# In[58]:


trainset["passenger_count"].describe()


# In[59]:


trainset = trainset.drop(trainset[trainset['passenger_count']>6].index)


# In[60]:


trainset = trainset.drop(trainset[trainset['passenger_count']==0].index)


# In[61]:


trainset["passenger_count"].describe()


# In[62]:


trainset['passenger_count'].value_counts()


# In[63]:


trainset[(trainset['passenger_count']==0.12) | (trainset['passenger_count']==1.30)].index
```

```
# In[64]:


trainset = trainset.drop([8790])


# In[65]:


trainset = trainset.drop(trainset[trainset['pickup_longitude']==0].index)


# In[66]:


trainset[trainset['pickup_longitude']==0]


# ### pickup_latitude

# In[67]:


trainset = trainset.drop(trainset[trainset['pickup_latitude']==0].index)


# In[68]:


trainset[trainset['pickup_longitude']>90].index


# In[69]:


trainset = trainset.drop([5686])


# ### dropoff_longitude

# In[70]:


trainset = trainset.drop(trainset[trainset['dropoff_longitude']==0].index)


# In[71]:


trainset[trainset['dropoff_longitude']==0]


# In[72]:


trainset.describe()
```

```
# ## Feature Selection

# In[73]:


trainset['passenger_count'] = trainset.passenger_count.astype('int64')

trainset['year'] = trainset.year.astype('object')

trainset['Month'] = trainset.Month.astype('object')

trainset['Date'] = trainset.Date.astype('object')

trainset['Day_of_week'] = trainset.Day_of_week.astype('object')

trainset['Hour'] = trainset.Hour.astype('object')


# In[74]:


trainset.isnull().sum()


# In[75]:


testset.isnull().sum()


# ## Feature Creation

# In[76]:


trainset.head()


# In[77]:


testset.head()


# In[78]:


trainset.nunique()


# In[79]:


testset.nunique()


# In[80]:
```

```
#finding decending order of fare to get to know whether the outliers are presented or not
trainset['distance'].sort_values(ascending=False)


# In[81]:


from collections import Counter


Counter(trainset['distance'] == 0)


# In[82]:


Counter(testset['distance'] == 0)


# In[83]:


Counter(trainset['fare_amount'] == 0)


# In[84]:


###we will remove the rows whose distance value is zero

trainset = trainset.drop(trainset[trainset['distance']== 0].index, axis=0)
trainset.shape


# In[85]:


#we will remove the rows whose distance values is very high which is more than 129kms

trainset = trainset.drop(trainset[trainset['distance'] > 130 ].index, axis=0)
trainset.shape


# In[86]:


trainset.head()


# In[87]:


drop = ['pickup_datetime', 'pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude', 'Minute']
trainset = trainset.drop(drop, axis = 1)


# In[88]:
```

```
trainset['passenger_count'] = trainset['passenger_count'].astype('int64')
trainset['year'] = trainset['year'].astype('int64')
trainset['Month'] = trainset['Month'].astype('int64')
trainset['Date'] = trainset['Date'].astype('int64')
trainset['Day_of_week'] = trainset['Day_of_week'].astype('int64')
trainset['Hour'] = trainset['Hour'].astype('int64')


# In[89]:


trainset.dtypes


# In[90]:


drop_test = ['pickup_datetime', 'pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude', 'Minute']
testset = testset.drop(drop_test, axis = 1)


# In[91]:


testset.head()


# In[92]:


testset.dtypes


# ## Data Visualization :
#
#
# Visualization of following:
#
# 1. Number of Passengers effects the the fare
# 2. Pickup date and time effects the fare
# 3. Day of the week does effects the fare
# 4. Distance effects the fare

# In[93]:


# Count plot on passenger count
plt.figure(figsize=(15,7))
sns.countplot(x="passenger_count", data=trainset)


# In[94]:


#Relationship beetween number of passengers and Fare

plt.figure(figsize=(15,7))
plt.scatter(x=trainset['passenger_count'], y=trainset['fare_amount'], s=10)
plt.xlabel('No. of Passengers')
plt.ylabel('Fare')
plt.show()


# In[95]:


#Relationship between date and Fare

plt.figure(figsize=(15,7))
plt.scatter(x=trainset['Date'], y=trainset['fare_amount'], s=10)
plt.xlabel('Date')
plt.ylabel('Fare')
plt.show()


# In[96]:


plt.figure(figsize=(15,7))
trainset.groupby(trainset["Hour"])['Hour'].count().plot(kind="bar")
plt.show()


# In[97]:


#Relationship between Time and Fare

plt.figure(figsize=(15,7))
plt.scatter(x=trainset['Hour'], y=trainset['fare_amount'], s=10)
plt.xlabel('Hour')
plt.ylabel('Fare')
plt.show()


# In[98]:


#impact of Day on the number of cab rides

plt.figure(figsize=(15,7))
sns.countplot(x="Day_of_week", data=trainset)


# In[99]:


#Relationships between day and Fare
plt.figure(figsize=(15,7))
plt.scatter(x=trainset['Day_of_week'], y=trainset['fare_amount'], s=10)
plt.xlabel('Day_of_week')
plt.ylabel('Fare')
plt.show()


# In[100]:
```

```
#Relationship between distance and fare

plt.figure(figsize=(15,7))
plt.scatter(x = trainset['distance'],y = trainset['fare_amount'],c = "g")
plt.xlabel('Distance')
plt.ylabel('Fare')
plt.show()


# ## Feature Scaling :

# In[101]:


#Normality check of training data is uniformly distributed or not-

for i in ['fare_amount', 'distance']:
    print(i)
    sns.distplot(trainset[i],bins='auto',color='green')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()


# In[102]:


#since skewness of target variable is high, apply log transform to reduce the skewness-
trainset['fare_amount'] = np.log1p(trainset['fare_amount'])

#since skewness of distance variable is high, apply log transform to reduce the skewness-
trainset['distance'] = np.log1p(trainset['distance'])


# In[103]:


#Normality Re-check to check data is uniformly distributed or not after log transformartion

for i in ['fare_amount', 'distance']:
    print(i)
    sns.distplot(trainset[i],bins='auto',color='green')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()


# In[104]:


#Normality check for test data is uniformly distributed or not-

sns.distplot(testset['distance'],bins='auto',color='green')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()


# In[105]:
```

```
#since skewness of distance variable is high, apply log transform to reduce the skewness-
testset['distance'] = np.log1p(testset['distance'])


# In[106]:


#rechecking the distribution for distance
sns.distplot(testset['distance'],bins='auto',color='green')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()


# # Modeling Phase:

# In[107]:


##train test split for further modelling

X_train, X_test, y_train, y_test = train_test_split( trainset.iloc[:, trainset.columns != 'fare_amount'],
                         trainset.iloc[:, 0], test_size = 0.20, random_state = 1)


# In[108]:


print(X_train.shape)
print(X_test.shape)


# ## Linear Regression Model :

# In[109]:


# Building model on top of training dataset
fit_LR = LinearRegression().fit(X_train , y_train)


# In[110]:


#prediction on train data
pred_train_LR = fit_LR.predict(X_train)


# In[111]:


#prediction on test data
pred_test_LR = fit_LR.predict(X_test)


# In[112]:
```

```
##calculating RMSE for test data
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

##calculating RMSE for train data
RMSE_train_LR= np.sqrt(mean_squared_error(y_train, pred_train_LR))


# In[113]:


print("Root Mean Squared Error For Training data = "+str(RMSE_train_LR))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))


# In[114]:


#calculate R^2 for train data
from sklearn.metrics import r2_score
r2_score(y_train, pred_train_LR)


# In[115]:


r2_score(y_test, pred_test_LR)


# ## Decision tree Model :

# In[116]:


fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)


# In[117]:


#prediction on train data
pred_train_DT = fit_DT.predict(X_train)

#prediction on test data
pred_test_DT = fit_DT.predict(X_test)


# In[118]:


##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))


# In[119]:
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))


# In[120]:


## R^2 calculation for train data
r2_score(y_train, pred_train_DT)


# In[121]:


## R^2 calculation for test data
r2_score(y_test, pred_test_DT)


# ## Random Forest Model :

# In[122]:


fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)


# In[123]:


#prediction on train data
pred_train_RF = fit_RF.predict(X_train)

#prediction on test data
pred_test_RF = fit_RF.predict(X_test)


# In[124]:


##calculating RMSE for train data
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))

##calculating RMSE for test data
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))


# In[125]:


print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))


# In[126]:
```

```
## calculate R^2 for train data

r2_score(y_train, pred_train_RF)


# In[127]:



#calculate R^2 for test data
r2_score(y_test, pred_test_RF)


# ## Gradient Boosting :

# In[128]:


fit_GB = GradientBoostingRegressor().fit(X_train, y_train)


# In[129]:


#prediction on train data
pred_train_GB = fit_GB.predict(X_train)

#prediction on test data
pred_test_GB = fit_GB.predict(X_test)


# In[130]:


##calculating RMSE for train data
RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
##calculating RMSE for test data
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))


# In[131]:


print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))


# In[132]:


#calculate R^2 for test data
r2_score(y_test, pred_test_GB)


# In[133]:



#calculate R^2 for train data
```

```
r2_score(y_train, pred_train_GB)


# ## Optimizing the results with parameters tuning :

# In[134]:


from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 42)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())


# In[135]:


##Random Hyperparameter Grid
from sklearn.model_selection import train_test_split,RandomizedSearchCV


# In[136]:


##Random Search CV on Random Forest Model

RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
             'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_rf = randomcv_rf.fit(X_train,y_train)
predictions_RRF = randomcv_rf.predict(X_test)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)

#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating RMSE
RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('RMSE = ',RRF_rmse)


# In[137]:
```

```
gb = GradientBoostingRegressor(random_state = 42)
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(gb.get_params())


# In[138]:


##Random Search CV on gradient boosting model

gb = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
          'max_depth': depth}

randomcv_gb = RandomizedSearchCV(gb, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_gb = randomcv_gb.fit(X_train,y_train)
predictions_gb = randomcv_gb.predict(X_test)

view_best_params_gb = randomcv_gb.best_params_

best_model = randomcv_gb.best_estimator_

predictions_gb = best_model.predict(X_test)

#R^2
gb_r2 = r2_score(y_test, predictions_gb)
#Calculating RMSE
gb_rmse = np.sqrt(mean_squared_error(y_test,predictions_gb))

print('Random Search CV Gradient Boosting Model Performance:')
print('Best Parameters = ',view_best_params_gb)
print('R-squared = {:0.2}.'.format(gb_r2))
print('RMSE = ', gb_rmse)


# In[139]:


from sklearn.model_selection import GridSearchCV
## Grid Search CV for random Forest model
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
          'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(X_train,y_train)
view_best_params_GRF = gridcv_rf.best_params_
```

```
#Apply model on test data
predictions_GRF = gridcv_rf.predict(X_test)

#R^2
GRF_r2 = r2_score(y_test, predictions_GRF)
#Calculating RMSE
GRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_GRF))

print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(GRF_r2))
print('RMSE = ',(GRF_rmse))


# In[140]:


## Grid Search CV for gradinet boosting
gb = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
              'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_gb = GridSearchCV(gb, param_grid = grid_search, cv = 5)
gridcv_gb = gridcv_gb.fit(X_train,y_train)
view_best_params_Ggb = gridcv_gb.best_params_

#Apply model on test data
predictions_Ggb = gridcv_gb.predict(X_test)

#R^2
Ggb_r2 = r2_score(y_test, predictions_Ggb)
#Calculating RMSE
Ggb_rmse = np.sqrt(mean_squared_error(y_test,predictions_Ggb))

print('Grid Search CV Gradient Boosting regression Model Performance:')
print('Best Parameters = ',view_best_params_Ggb)
print('R-squared = {:0.2}.'.format(Ggb_r2))
print('RMSE = ',(Ggb_rmse))


# In[ ]:





# In[141]:


## Grid Search CV for random Forest model
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))
```

```
# Create the grid
grid_search = {'n_estimators': n_estimator,
             'max_depth': depth}

## Grid Search Cross-Validation with 10 fold CV
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 10)
gridcv_rf = gridcv_rf.fit(X_train,y_train)
view_best_params_GRF = gridcv_rf.best_params_

#Apply model on test data
predictions_GRF_test_Df = gridcv_rf.predict(testset)


# In[142]:


predictions_GRF_test_Df


# In[143]:


testset['Predicted_fare'] = predictions_GRF_test_Df


# In[144]:


testset.head()


# In[145]:


testset.to_csv('Predicted_test.csv')
```

=== R – SECTION ===

R CODE

```
rm(list = ls())

setwd("D:/Data Science/Project1")
getwd()


#Load Libraries
x = c("ggplot2","dplyr","lubridate","tidyverse", "geosphere", "corrgram", "DMwR",  "randomForest", "unbalanced",
"e1071",
     "MASS", "rpart", "gbm", 'DataCombine', 'inTrees','caret','C50')
lapply(x, require, character.only = TRUE)
rm(x)
```

```
# load the data
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
summary(train)


# converting into proper datatype
train$fare_amount=as.numeric(as.character(train$fare_amount))
train$passenger_count=as.integer(train$passenger_count)
summary(train)

# filtering out the rows where pickup_longitude = dropoff_longitude and pick_up latitude = drop_off latitude
train = filter(train,
		train$pickup_longitude!= train$dropoff_longitude &
		train$pickup_latitude!=train$dropoff_latitude
)

#replace all "0" with NA
train[train==0]= NA
summary(train)


# fare_amount is target variable, so remove NA by filtering out all fare_amount>0, this will also filter out NA if available
train = filter(train,
		fare_amount>0&
		fare_amount<60
)


#missing Value

missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]

ggplot(data = missing_val[1:3,], aes(x=reorder(Columns, -Missing_percentage),y = Missing_percentage))+
geom_bar(stat = "identity",fill = "grey")+xlab("Parameter")+
ggtitle("Missing data percentage (Train)")




#passenger_count is categorical, so replace it by mode
#function to calculate mode
mode= function(x){
  y=unique(x)
  as.numeric(as.character(y[which.max(tabulate(match(x,y)))]))

}

#impute with the mode
train$passenger_count[is.na(train$passenger_count)] = mode(train$passenger_count)
train$passenger_count = as.integer(train$passenger_count)

#Method
#Actual train$pickup_latitude[48]= 40.77205
#Mean =train$pickup_latitude[48]= 40.71222
#Median =train$pickup_latitude[48]= 40.75332
#KNN = train$pickup_latitude[48]= 40.77271


#KNN imputation
train = knnImputation(train, k = 5)

# Outlier Analysis
train1 = train
summary(train)


# ## BoxPlots - Distribution and Outlier Check


cnames = colnames(train[,c(3:7)])
#
for (i in 1:length(cnames))
 {
   assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "fare_amount", group = 1), data = train)+
           stat_boxplot(geom = "errorbar", width = 0.5) +
           geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                        outlier.size=1, notch=FALSE) +
           theme(legend.position="bottom")+
           labs(y=cnames[i],x="Num")+
           ggtitle(paste("Box plot of fare_amount for",cnames[i])))
 }
#
# ## Plotting plots together
gridExtra::grid.arrange(gn1,gn2,ncol=2)
gridExtra::grid.arrange(gn3,gn4,gn5, ncol=3 )

# # #Remove outliers using boxplot method
train_data = train
#train = train_data

# replacing outlier with NA
```

```
for(i in cnames){
  val = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]
  #print(length(val))
  train[,i][train[,i] %in% val] = NA
  }



#KNN imputation
train = knnImputation(train, k = 5)


##################
train2 = train
#train = train2

# library(geosphere)
#creating a distance column using distHaversine function
train$distance = distHaversine(train[,c(3,4)],train[,c(5,6)])
train$distance = as.numeric(train$distance)/1000

#which(is.na(train$pickup_datetime))
# removing a row which has pickup_datatime value not in proper format, pickup_datetime=43
train = train[-c(1265),]

# creating aditional column from datetime to year, month,day,dayOfWeek,hour,partOfDay
train <- mutate(train,
                pickup_datetime = ymd_hms(pickup_datetime),
                month = as.factor(month(pickup_datetime)),
                year = as.numeric(year(pickup_datetime)),
                day = day(pickup_datetime),
                dayOfWeek = as.factor(wday(pickup_datetime)),
                hour = hour(pickup_datetime),
                partOfDay = as.factor(round(hour * 2 / 10)),
                hour = as.factor(hour(pickup_datetime))

)
#converting into proper datatype
train$passenger_count = as.integer(train$passenger_count)


## Correlation Plot
  corrgram(train[,-c(7)], order = F,
                   upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot"
)

## drop unwanted columns
train <- train[,-c(2,9,11,12,14)]
```

```
qqnorm(train$fare_amount)

hist(train$fare_amount)

train3 = train
#Clean the environment
rmExcept("train")

train3 = train


#Divide data into train and test using stratified sampling method
set.seed(1234)

train.index = createDataPartition(train$fare_amount, p = .80, list = FALSE)
train = train[ train.index,]
test  = train[-train.index,]


# Define Mape
MAPE = function(x, y){
  mean(abs((x - y)/x*100))
}

#Decision Tree
fit = rpart(fare_amount ~. , data = train, method = "anova", minsplit=10)
summary(fit)
predictions_DT = predict(fit, test[,-1])
MAPE(test[,1], predictions_DT)
#41.88119

#Linear Regression
lm_model = lm(fare_amount ~. , data = train)
summary(lm_model)
predictions_LR = predict(lm_model, test[,-1])
MAPE(test[,1], predictions_LR)
#38.87821




##KNN Implementation
library(class)
#Predict test data
KNN_Predictions = knn(train[, 2:9], test[, 2:9], train$fare_amount, k = 5)
#convert the values into numeric
KNN_Predictions=as.numeric(as.character((KNN_Predictions)))
#Calculate MAPE
```

```
MAPE(test[,1], KNN_Predictions)
#33.02988

#Random Forest
RF_model = randomForest(fare_amount ~.  , train, importance = TRUE, ntree=200, mtry=2)
RF_Predictions = predict(RF_model, test[,-1])
MAPE(test[,1], RF_Predictions)
importance(RF_model, type = 1)
#18.18387

##################################################################################

# Loading the test data
test_cab = read.csv("test.csv", header = T)

# filtering out the rows where pickup_longitude = dropoff_longitude and pick_up latitude = drop_off latitude
test_cab = filter(test_cab,
            test_cab$pickup_longitude!= test_cab$dropoff_longitude &
            test_cab$pickup_latitude!=test_cab$dropoff_latitude
)


#creating a distance column using distHaversine function
test_cab$distance = distHaversine(test_cab[,c(2,3)],test_cab[,c(4,5)])
test_cab$distance = as.numeric(test_cab$distance)/1000


# creating aditional column from datetime to year,hour

test_cab <- mutate(test_cab,
             pickup_datetime = ymd_hms(pickup_datetime),

             year = as.numeric(year(pickup_datetime)),

             hour = hour(pickup_datetime),

             hour = as.factor(hour(pickup_datetime))

)


# droping out unwanted columns
test_cab = test_cab[,-c(1)]
#Random Forest
RF_model = randomForest(fare_amount ~.  , train, importance = TRUE, ntree=200, mtry=2)
test_cab$fare_amount = predict(RF_model, test_cab)
#write.csv(test_cab,"final_test_R.csv",row.names = F)
```

Instruction to run the Above Written code:

For Python code:

1. Use Spider IDE, PyCharm IDE etc., any python software development (for <filename>.py extension).

2. Use Anaconda Navigator Data science ToolKit (Jupyter Notebook for <filename>.ipynb extention)

For R code:

1. Use R-studio (or)

2. Jupyter notebook with R-essentials packages

Just import the files in you Analytical / development environment and execute the code line by line for rechecking of error and for debugging purpose.

THE END