Submitted By:

Tejasvi Rastogi

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Description** | **Page No** |
| 1. | 1-Introduction | 3 |
| 2. | 1.1Problem Statement | 4 |
| 3. | 1.2 Data | 4-5 |
| 4. | 1.3 Exploratory Data Analysis | 5-6 |
| 4. | 2-Methodology | 7 |
| 5. | 2.1 Data Pre Processing | 8-10 |
| 6. | 2.1.1 Missing Value Analysis | 10-11 |
| 7. | 2.1.2 Feature Engineering | 11-13 |
| 8. | 2.1.3 Outlier Analysis | 13-15 |
| 9. | 2.1.4 Feature Selection | 15-17 |
| 10. | 2.1.5 Checking for VIFs | 17-18 |
| 11. | 2.1.6 Feature Scaling | 18-19 |
| 12. | 2.2 Business and Real Life Considerations | 19 |
| 13. | 3 Modeling | 20 |
| 14. | 3.1 Linear Regression Model | 21 |
| 15. | 3.2 Decision tree | 21-22 |
| 16. | 3.3 Random Forest | 22-23 |
| 17. | 3.4 XGBoost Model | 23 |
| 18. | 3.5 Hyper parameter tunning | 23-24 |
| 19. | 4. Conclusion | 25 |
| 20. | 4.1 Model Evaluation | 26 |
| 21. | 4.2 Model Selection | 27-28 |
| 22. | 5. R-Code | 29-48 |
| 23. | 6. Appendix and Insights from the project | 49-51 |
| 24. | 7. References | 52 |

**1.1 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.

**1.2 Data:**

Understanding of data is the very first and important step in the process of finding solution of any business problem. Here in our case our company has provided a data set with following features, we need to go through each and every variable of it to understand and for better functioning.

Size of Dataset Provided: - 16067 rows, 7 Columns (including dependent variable) Missing Values: Yes

Outliers Presented: Yes

Below mentioned is a list of all the variable names with their meanings:

|  |  |
| --- | --- |
| **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 |

Dataset Details:

2 separate data set for train and test data

**Train Data:**

Number of Attributes: 7

Number of Observations: 16067

Missing Values: Yes

**Train Data:**

Number of Attributes: 6

Number of Observations: 9914

Missing Values: No

**1.3 Exploratory Data Analysis**

**Variables Information:**

**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.

**passenger\_count** - an integer indicating the number of passengers in the cab ride.

**Fare\_amount**- a numeric value to be predicted in case of the test data sheet. This is already present in the train data set.

The target variable for our case is “Fare amount”. As it is given it is continuous in nature in the train data set, we shall consider our problem to be of **Regression Problem** for our test data set. Using the learning from the train set, we will predict the value of ‘fare’ amount in the test data set.

Our data looks like this:-

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| fare\_amount | pickup\_datetime | pickup\_longitude | pickup\_latitude | dropoff\_longitude | dropoff\_latitude | Passenger\_count |
| 4.5 | 2009-06-15 17:26:21 UTC | -73.844311 | 40.721319 | -73.84161 | 40.712278 | 1 |
| 16.9 | 2010-01-05 16:52:16 UTC | -74.016048 | 40.711303 | -73.979268 | 40.782004 | 1 |
| 5.7 | 2011-08-18 00:35:00 UTC | -73.982738 | 40.76127 | -73.991242 | 40.750562 | 2 |
| 7.7 | 2012-04-21 04:30:42 UTC | -73.98713 | 40.733143 | -73.991567 | 40.758092 | 1 |
| 5.3 | 2010-03-09 07:51:00 UTC | -73.968095 | 40.768008 | -73.956655 | 40.783762 | 1 |
| 12.1 | 2011-01-06 09:50:45 UTC | -74.000964 | 40.73163 | -73.972892 | 40.758233 | 1 |
| 7.5 | 2012-11-20 20:35:00 UTC | -73.980002 | 40.751662 | -73.973802 | 40.764842 | 1 |

Knowing the data-types of all the test variables:-

**fare\_amount object**

**pickup\_datetime object**

**pickup\_longitude float64**

**pickup\_latitude float64**

**dropoff\_longitude float64**

**dropoff\_latitude float64**

**passenger\_count float64**

**dtype: object**

Describe the shape of the data:-

df\_cab\_train.shape

(16067, 7)

**2.1. Pre-Processing:**

Data preprocessing is an important step in the [data mining](https://en.wikipedia.org/wiki/Data_mining) process. The phrase ["garbage in, garbage out"](https://en.wikipedia.org/wiki/GIGO) is particularly applicable to [data mining](https://en.wikipedia.org/wiki/Data_mining) and [machine learning](https://en.wikipedia.org/wiki/Machine_learning) projects. Data-gathering methods are often loosely controlled, resulting in [out-of-range](https://en.wikipedia.org/w/index.php?title=Range_error&action=edit&redlink=1) values (e.g., Income: −100), impossible data combinations (e.g., Sex: Male, Pregnant: Yes), [missing values](https://en.wikipedia.org/wiki/Missing_values), etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and [quality of data](https://en.wikipedia.org/wiki/Data_quality) is first and foremost before running an analysis. Often, data preprocessing is the most important phase of a [machine learning](https://en.wikipedia.org/wiki/Machine_learning) project, especially in [computational biology](https://en.wikipedia.org/wiki/Computational_biology). If there is much irrelevant and redundant information present or noisy and unreliable data, then [knowledge discovery](https://en.wikipedia.org/wiki/Knowledge_discovery) during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Data preprocessing includes [cleaning](https://en.wikipedia.org/wiki/Data_cleaning), [Instance selection](https://en.wikipedia.org/wiki/Instance_selection), [normalization](https://en.wikipedia.org/wiki/Data_normalization), [transformation](https://en.wikipedia.org/wiki/Data_transformation), [feature extraction](https://en.wikipedia.org/wiki/Feature_extraction) and [selection](https://en.wikipedia.org/wiki/Feature_selection), etc. The product of data preprocessing is the final [training set](https://en.wikipedia.org/wiki/Training_set). Data preprocessing involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

At many times the input data is (a) Noisy: containing errors or outliers (b) Inconsistent: containing discrepancies in codes or names.

When we are required to build a predictive model, we require to look and manipulate the data before we start modeling which includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots, all these steps is combined under one shed which is Exploratory Data Analysis, which includes following steps:

Data exploration and Cleaning

Missing values treatment

Outlier Analysis

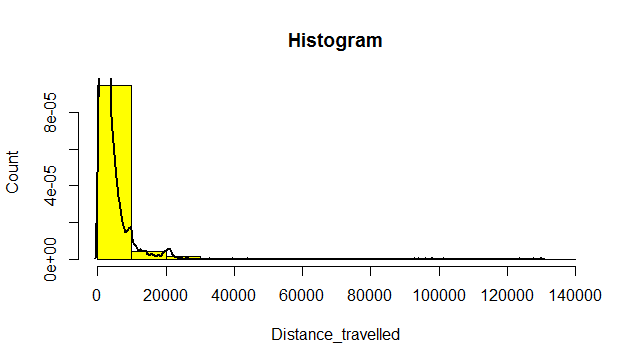
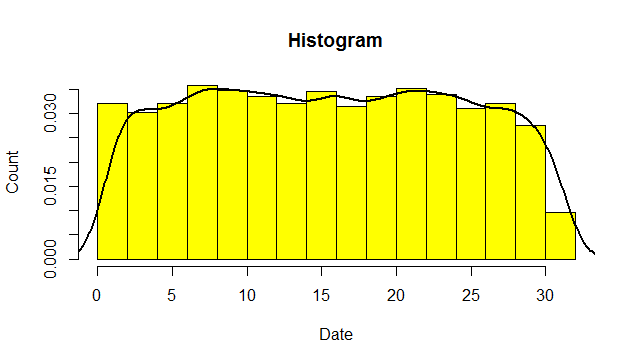
Feature Selection

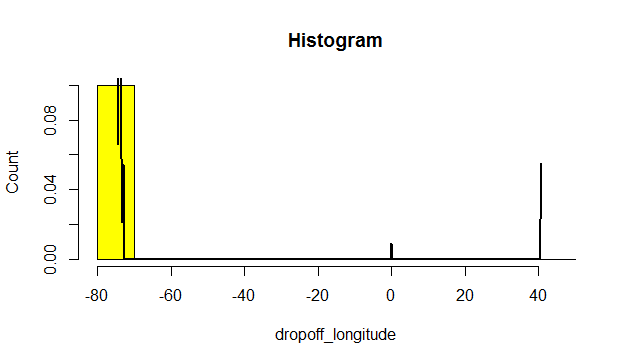
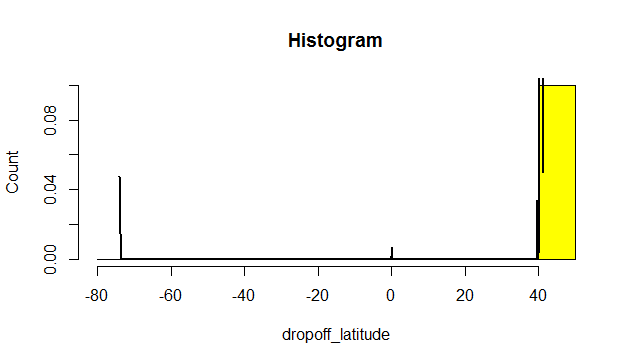
Features Scaling

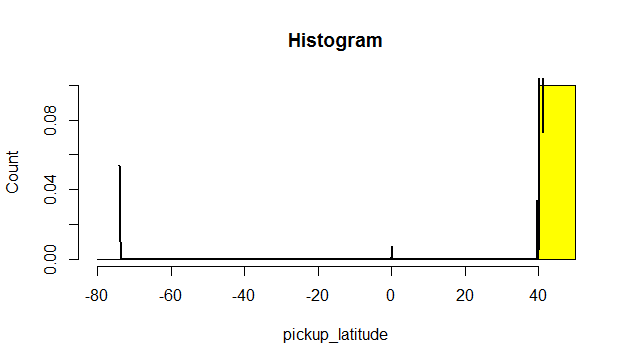
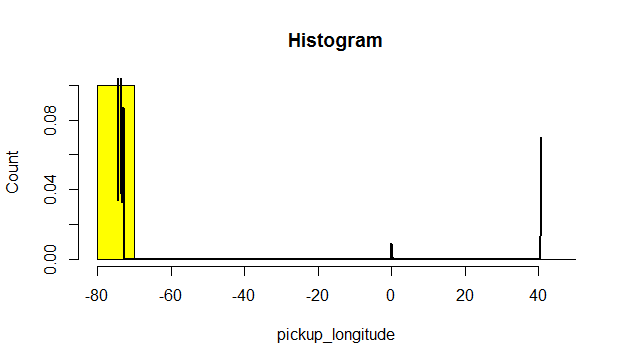
Skewness and Log transformation

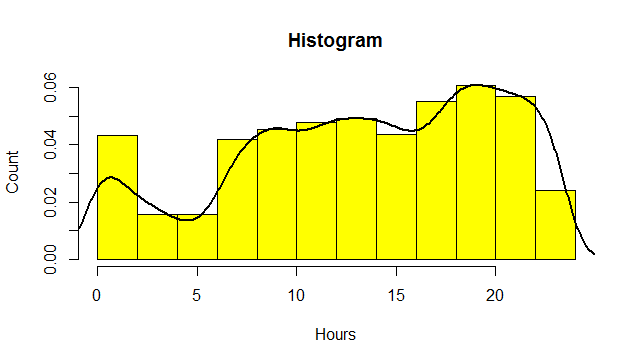
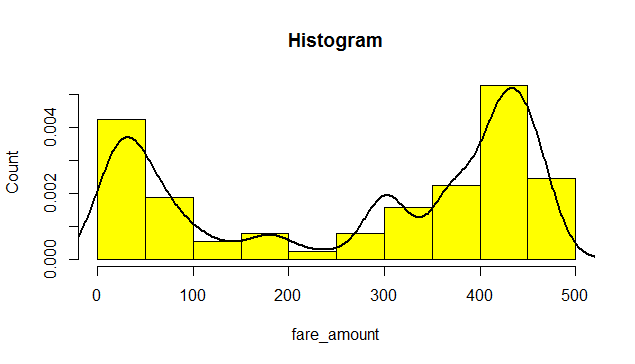
Visualization

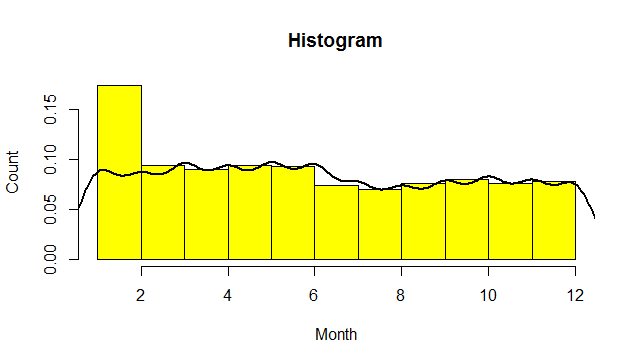
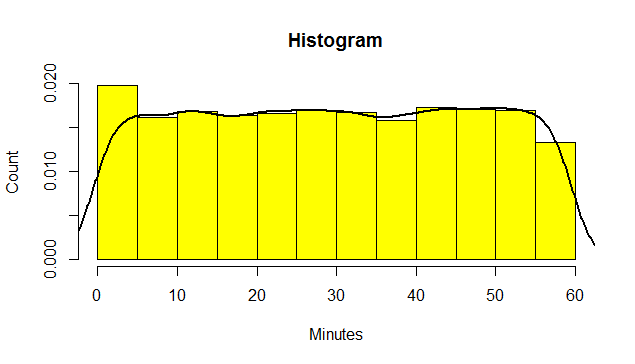
To first start for it we shall start with plotting histograms for the variables. This will help us to find whether our data is normally distributed or not. Regression problems require data to be normally distributed before feeding it to the model.

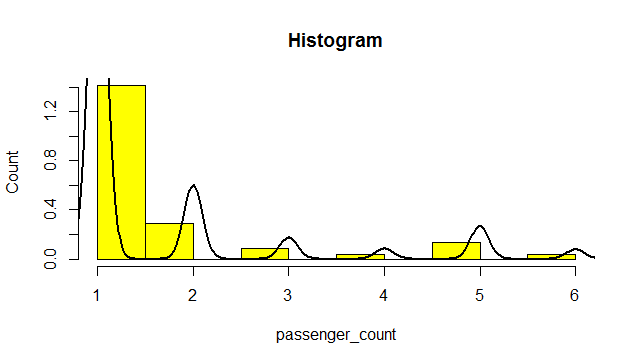
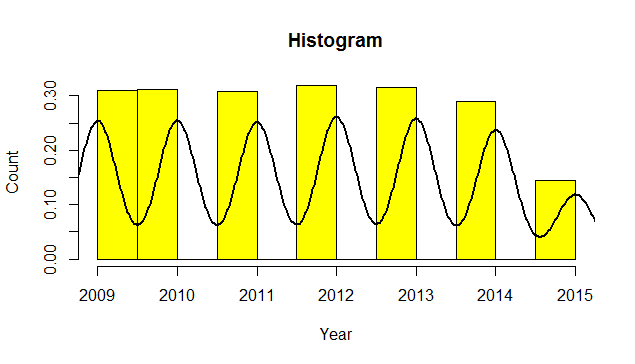












**Facts about Data exploration and Cleaning (Missing Values and Outliers)**

The very first step which comes with any data science project is data exploration and cleaning which includes following points as per this project:

* Separate the combined variables.
* As we know we have some negative values in fare amount so we have to remove those values.
* Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passengers counts more than 6 and less than 1.
* There are some outlier figures in the fare (like top 3 values) so we need to remove those.

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.

**2.2.1 MISSING VALUE ANALYSIS:**

The concept of missing values is important to understand in order to successfully [manage](http://www.statisticssolutions.com/academic-solutions/resources/dissertation-resources/data-entry-and-management/multiple-imputation-for-missing-data/) data.  If the missing values are not handled properly by the researcher, then he/she may end up drawing an inaccurate inference about the data.  Due to improper handling, the result obtained by the researcher will differ from ones where the missing values are present. Item non-response occurs when the respondent does not respond to certain questions due to stress, fatigue or lack of knowledge. The respondent may not respond because some questions are sensitive. This lack of answers would be considered missing values. Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

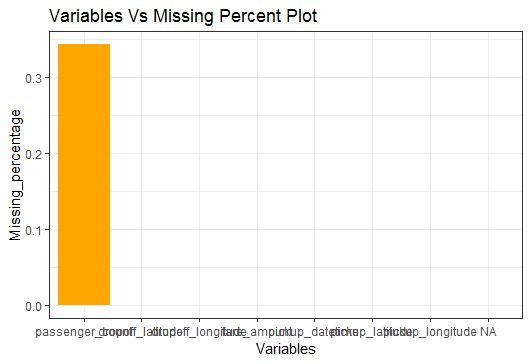
The missing data may reduce the precision of calculated statistics because there is less information than originally planned. Hence missing value analysis is necessary.

Suppose the number of cases of missing values is extremely small; then, an expert researcher may drop or omit those values from the analysis.  In statistical language, if the number of the cases is less than 5% of the sample, then the researcher can drop them.

One the other hand if there is a larger number of missing values, i.e. 30% then it is better to drop those cases (rather than do imputation) and replace them.

Since the data which we have received consists of missing values, we need to first filter our data and find these missing values. After that we need to apply the best method which is computing the missing values and giving the nearest possible value.

In our case, we have more that 16000+ data records and the missing value percentage is very less i.e. 0.34%. Hence we have dropped those missing values.



**2.1.2Feature Engineering**

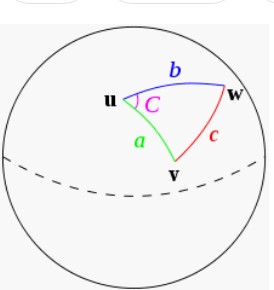
**Creating some new variables from the given variables.**

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, which relates the sides and angles of spherical triangles.



So our new extracted variables are:

* + fare\_amount
  + pickup\_datetime
  + pickup\_longitude
  + pickup\_latitude
  + dropoff\_longitude
  + dropoff\_latitude
  + passenger\_count
  + year
  + Month
  + Date
  + Day ofWeek
  + Hour
  + Minute
  + Distance

**Below are the names of Independent variables:**

**passenger\_count, year, Month, Date, Day of Week, Hour, distance**

Our Dependent variable is: **fare\_amount**

**Uniqueness inVariable**

We need to look at the unique number in the variables which help us to decide whether the variable is categorical or numeric. So, by using python script ‘nunique’ we tried to find out the unique values in each variable. We have also added the table below:

|  |  |
| --- | --- |
| **Variable Name** | **Unique Counts** |
| fare\_amount | 450 |
| passenger\_count | 7 |
| year | 7 |
| Month | 12 |
| Date | 31 |
| Day of Week | 7 |
| Hour | 24 |
| distance | 15424 |

**Dividing the variables into two categories basis their data types:**

Continuous variables - 'fare\_amount', 'distance'.

Categorical Variables - 'year', 'Month', 'Date', 'Day of Week', 'Hour', 'passenger\_count'

**2.1.3 Outlier Analysis:**

In [statistics](https://en.wikipedia.org/wiki/Statistics), an outlier is a [data point](https://en.wikipedia.org/wiki/Data_point) that differs significantly from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the [data set](https://en.wikipedia.org/wiki/Data_set). An outlier can cause serious problems in statistical analyses. Outliers can occur by chance in any distribution, but they often indicate either [measurement error](https://en.wikipedia.org/wiki/Measurement_error) or that the population has a [heavy-tailed distribution](https://en.wikipedia.org/wiki/Heavy-tailed_distribution). In the former case one wishes to discard them or use statistics that are [robust](https://en.wikipedia.org/wiki/Robust_statistics) to outliers, while in the latter case they indicate that the distribution has high [skewness](https://en.wikipedia.org/wiki/Skewness) and that one should be very cautious in using tools or intuitions that assume a [normal distribution](https://en.wikipedia.org/wiki/Normal_distribution). A frequent cause of outliers is a mixture of two distributions, which may be two distinct sub-populations, or may indicate 'correct trial' versus 'measurement error; this is modeled by a [mixture model](https://en.wikipedia.org/wiki/Mixture_model).

There exist data objects that do not comply with the general behaviour or model of the data. Such data objects, which are grossly different from or inconsistent with the remaining set of data, are called outliers. An outlier is an observation point that is distant from other observations.

We can make use of Boxplots to visualize the outliers.

For our case we have plotted Box plots for each of continuous variables. Below box plots have been helpful in deriving that there are many extreme values present in the variables. These extreme values may represent some anomaly or error in the data collection process.

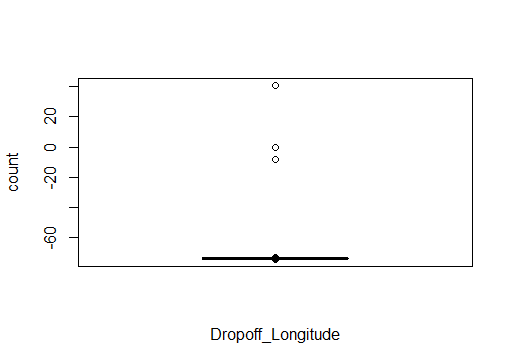
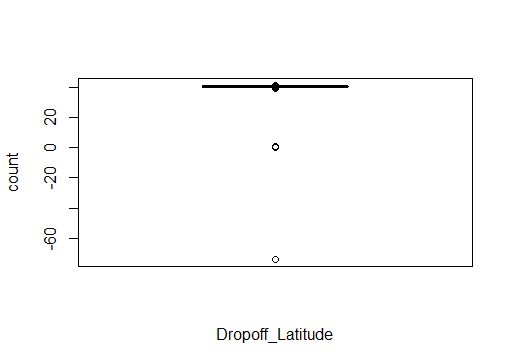
Outlier detection and treatment is always a tricky part especially when our dataset is small. The box plot method detects outlier if any value is present greater than (**Q3+(1.5\*IQR)**) or less than (**Q1–(1.5\*IQR)**)

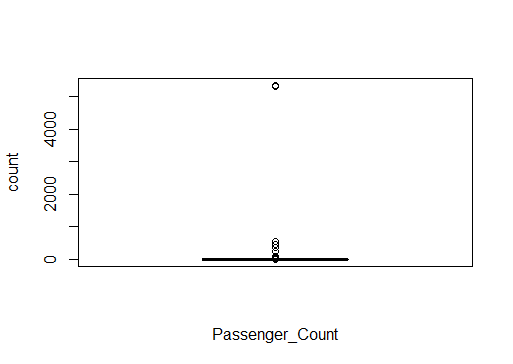
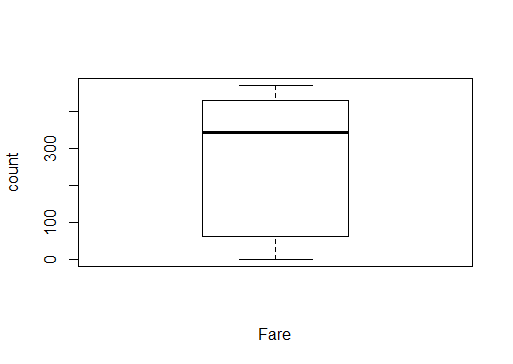
**Q1 >**25% of data are less than or equal to this value

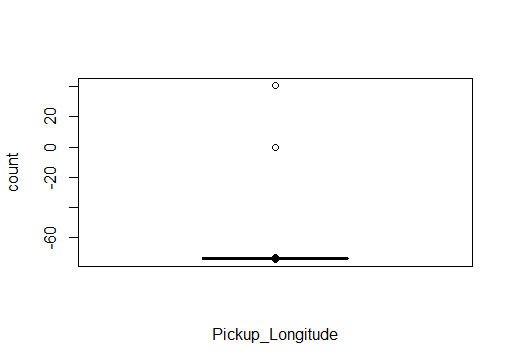
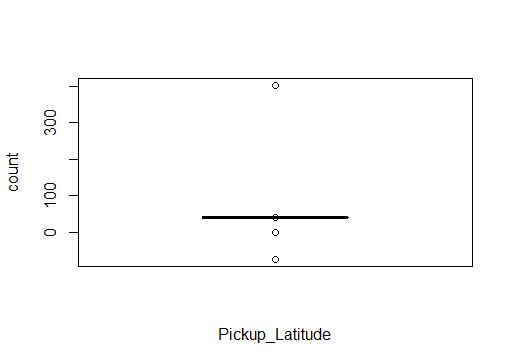
**Q2 or Median ->**50% of data are less than or equal to this value

**Q3 >**75% of data are less than or equal to this value

**IQR(Inter Quartile Range) =** Q3 – Q1







Although we have plotted box plots, but we cannot rely on them. None of the data manipulations and cleaning done by me is solely on basis of box plots or outliers. I have also considered some real life limitations and business constraints for data cleaning which I will be discussing post the data cleaning process.

**2.1.4 Features Selections:**

In [machine learning](https://en.wikipedia.org/wiki/Machine_learning) and [statistics](https://en.wikipedia.org/wiki/Statistics), **feature selection**, also known as **variable selection**, **attribute selection** or **variable subset selection**, is the process of selecting a subset of relevant [features](https://en.wikipedia.org/wiki/Feature_(machine_learning)) (variables, predictors) for use in model construction. Feature selection techniques are used for four reasons:

* Simplification of models to make them easier to interpret by researchers/users,
* shorter training times,
* to avoid the [curse of dimensionality](https://en.wikipedia.org/wiki/Curse_of_dimensionality),
* enhanced generalization by reducing [overfitting](https://en.wikipedia.org/wiki/Overfitting) (formally, reduction of [variance](https://en.wikipedia.org/wiki/Bias-variance_tradeoff))

The central premise when using a feature selection technique is that the data contains some features that are either *redundant* or *irrelevant*, and can thus be removed without incurring much loss of information. *Redundant* and *irrelevant* are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, I mean noise in data.

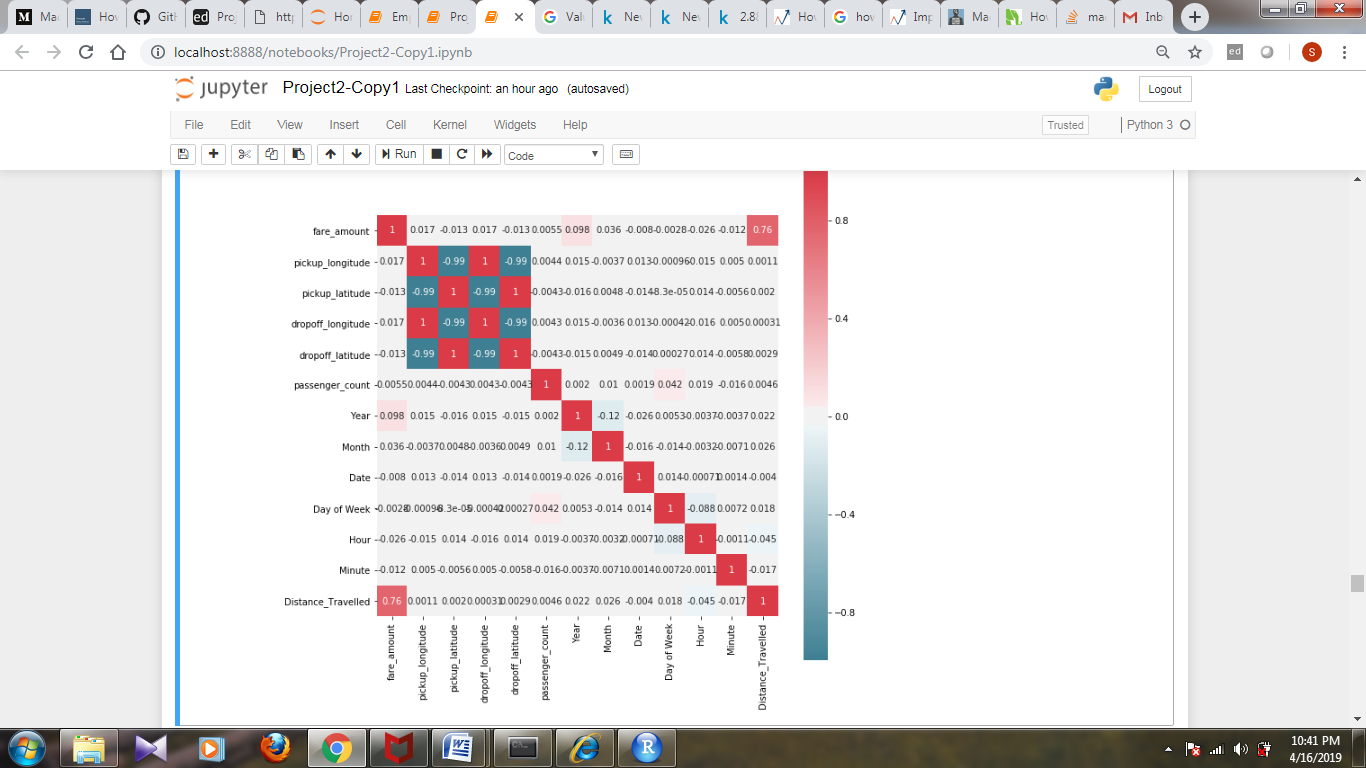
This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. I have myself witnessed feature subsets giving better results than complete set of feature for the same algorithm or – “Sometimes, less is better!”.

The process in which we select those features in our data that are most useful or most relevant for the problem being worked upon is a process called feature selection. Features in simple language are the variables present in the data set under consideration. Feature selection is also called variable selection or attributes selection. It mostly acts as a **filter, muting out features that aren’t useful** in addition to your existing features.

Feature selection is mainly done on the concept of multi collinearity. For 2 or more variables which are highly correlated to one another we can make use of only one variable and drop the rest of the variables which are just additionally adding to the unnecessary information in the data set.

For our project we have used Correlation Analysis (with help of Corrgram Visualization).

From correlation analysis we have found that no correlation exist between any of the independent variables.



**2.1.5 Checking for VIFs (Variation Inflation Factor):**

Checking VIF for our numerical variables:

vif(df\_cab\_train[,-1:-2]) Variables VIF

1 pickup\_longitude 5071.776751

2 pickup\_latitude 5952.353392

3 dropoff\_longitude 5233.333541

4 dropoff\_latitude 5511.930972

5 passenger\_count 1.002535

6 Hours 1.011518

7 Minutes 1.002144

8 Date 1.003489

9 Month 1.017528

10 Year 1.017314

11 Distance\_travelled 1.021678

3 variables from the 11 input variables have collinearity problem:

dropoff\_latitude, dropoff\_longitude , pickup\_latitude

After excluding the collinear variables, the linear correlation coefficients ranges between:

min correlation ( passenger\_count ~ pickup\_longitude ): 0.0001911228

max correlation ( Year ~ Month ): -0.1288029

---------- VIFs of the remained variables --------

Variables VIF

1 pickup\_longitude 1.000924

2 passenger\_count 1.001398

3 Hours 1.001249

4 Minutes 1.000803

5 Date 1.001891

6 Month 1.017719

7 Year 1.018737

8 Distance\_travelled 1.001331

**2.1.6 Feature Scaling:**

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Euclidean distance between two data points in their computations, this is a problem. Suppose if we have a data set of weight having values like 5kg and 5000gms. In this case the features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes. To suppress this effect, we need to bring all features to the same level of magnitudes.

There are majorly 2 methods followed for Feature Scaling process, namely Normalization and Standardization.

* Normalization rescales the data in the range of features to scale it in [0, 1] or [−1, 1].
* While Standardization transforms the data to have zero mean and a variance of 1.

For our project we have used Normalization method as the data was not uniformly distributed.

**2.2 Business & Real Life Considerations:**

As this is real time data provided to us we have to consider some real life situations we have gone ahead with below data removal on basis of some **domain knowledge**:

1. **Max Number of Passengers** – We have taken a maximum seating capacity of 0-6 in a cab.
2. **Coordinates** – We have removed the outliers or incorrect populated data of (0,0) location, which lies in the middle of the ocean. Also longitude is taken in the range of (-180,180) and latitude in the range of (-90, 90) which is the scientifically accepted value.
3. **Fare Amount**- We have removed only negative amount of fare. As for the rest cases, we were still not sure of the basic limitations the business might have setup. Like surcharges, waiting charges, cancellation charges etc. These details were not mentioned and can’t be assumed altogether. Hence we have continued with the current fare amounts we had.
4. **Distance Travelled**-We have made use of Haversine formula and the corresponding package to calculate the distance travelled in between the pickup and drop locations. We got few unacceptable values like distance value being in range of thousands, which should not be the case of cab business as regular intra-city travel.

**3 Modeling:**

After all the pre-processing steps are done, we will be using some data prediction models on our processed data to predict the target variable. We have built following models on our data set:

**3.1 Linear Regression:**

Linear regression is a basic and commonly used type of predictive analysis. Linear regression is used to describe relationship among variables. Linear Regression may be used for predicting one variable or it may predict multiple variables. These cases are known as Simple Linear Regression and Multiple Linear Regression respectively.

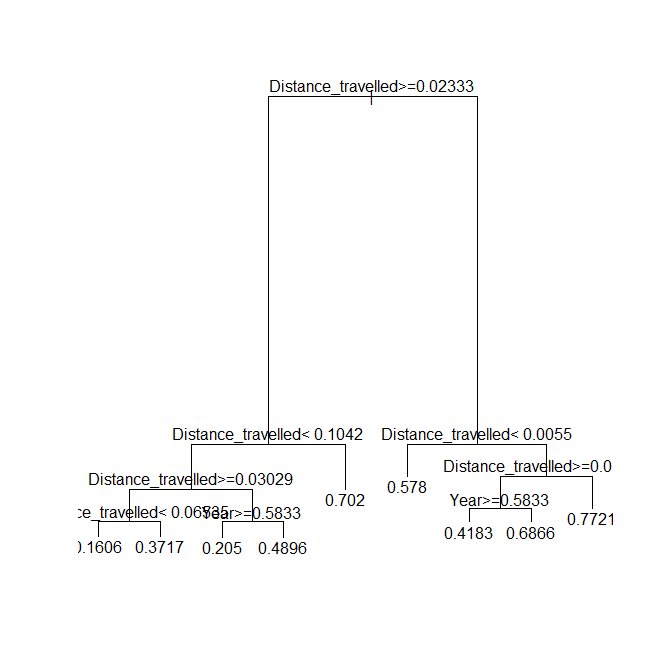
The one simple case is where a dependent variable may be related to independent or explanatory variable. The following equation can be used for depicting linear relationship:

y = b0+b1x

**3.2 Decision Tree**

A **decision tree** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. A decision tree is a flowchart-like structure in which each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression). Decision Tree algorithms are referred to as **CART** **(Classification and Regression Trees)**. 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.

**Graphical representation of Decision Tree:**



**3.3 Random Forest**

Random forests or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks that operates by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set). The first algorithm for random decision forests was created by [Tin Kam Ho](https://en.wikipedia.org/wiki/Tin_Kam_Ho)using the [random subspace method](https://en.wikipedia.org/wiki/Random_subspace_method), which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg. An extension of the algorithm was developed by [Leo Breiman](https://en.wikipedia.org/wiki/Leo_Breiman) and Adele Cutler, who registered "Random Forests" as a [trademark](https://en.wikipedia.org/wiki/Trademark) (as of 2019, owned by [Minitab, Inc.](https://en.wikipedia.org/wiki/Minitab)). The extension combines Breiman's "[bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating)" idea and random selection of features, introduced first by Ho and later independently by Amit and [Geman](https://en.wikipedia.org/wiki/Donald_Geman) in order to construct a collection of decision trees with controlled variance. Random forests or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks, that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of [over fitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set).

Random forests are a popular ensemble method that can be used to build [predictive models](https://www.datascience.com/resources/white-papers/executives-guide-to-predictive-data-modeling) for both classification and regression problems. In the case of a random forest, the model creates an entire forest of random uncorrelated decision trees to arrive at the best possible answer. The idea behind Random Forest is to build ‘n’ number of trees to have more accuracy in dataset. It is called random forest as we are building n no. of trees randomly.

**3.4 XGBoost Model:**

Its name stands for **eXtreme Gradient Boosting**, it was developed by Tianqi Chen and now is part of a wider collection of open-source libraries developed by the Distributed Machine Learning Community (DMLC). XGBoost is a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed. Specifically, it was engineered to exploit every bit of memory and hardware resources for tree boosting algorithms.

The implementation of XGBoost offers several advanced features for model tuning, computing environments and algorithm enhancement. It is capable of performing the three main forms of gradient boosting (Gradient Boosting (GB), Stochastic GB and Regularized GB) and it is robust enough to support fine tuning and addition of regularization parameters. According to Tianqi Chen, the latter is what makes it superior and different to other libraries.

**3.5 Hyperparameter Tuning**

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 hyperparameters 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 hyperparameter because this is set by the data scientist.

Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter 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

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

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

**3.1 Conclusion**

After the development of our models, we are going to evaluate our models, select the best model for our dataset and try to predict the fares for the test data

**3.2 Model Evaluation**

For evaluating our model, we shall take help of various error metrics, to support our model selection criteria.

Since, in our project the target variable prediction was that of a continuous variable type, we have made use of the following metrics:

**Root Mean Square Error (RMSE):**

Root Mean Square Error (RMSE) is the standard deviation of the [residuals](https://www.statisticshowto.datasciencecentral.com/residual/) (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are.

**R-Squared:**

R-squared 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 determinations for multiple regressions.

RMSE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our model.

Mean Absolute percentage error (MAPE) measures the deviation from the actual data in terms of percentage and consider both the negative and positive errors which cancels out each other. Hence we use RMSE which is more accurate and by squaring the errors they become positive and does not cancels each other and stays in existence till the end of commutation thus adding more accuracy to the result.

**3.3 Model Selection:**

On the basis of the RMSE and R- Squared value for the 3 models developed, we can decide which model to select for our project:

Lower values of **RMSE** and higher value of **R-Squared Value** indicate better fit.

**Decision Tree Model:**

|  |  |  |
| --- | --- | --- |
|  | **RMSE Value** | **R Squared Value** |
| **R Program** | 0.2842566 | 0.4061057 |
| **Python Program** | 0.011668971661228942 | 0.6931500145566145 |

**Linear Regression:**

|  |  |  |
| --- | --- | --- |
|  | **RMSE Value** | **R Squared Value** |
| **R Program** | 0.35931131 | 0.05158535 |
| **Python Program** | 0.010903823903314923 | 0.732071696110435 |

**Random Forest:**

|  |  |  |
| --- | --- | --- |
|  | **RMSE Value** | **R Squared Value** |
| **R Program** | 0.2710569 | 0.4604025 |
| **Python Program** | 0.01213402203821154 | 0.6682044969839842 |

**XGBoost:**

|  |  |  |
| --- | --- | --- |
|  | **RMSE Value** | **R Squared Value** |
| **R Program** | 0.275750 | 0.440784 |
| **Python Program** | NA | NA |

Hence for our project, I have selected XGBoost Model for R, since it has the lowest RMSE. A question here is, even Random forest has close values of RMSE to XGBoost, but we did not go for Random Forest because, we observed a considerable difference in the RMSE values for Train and Test Data i.e. (referring a snippet of R code from Random Forest Model Section)

># For training data

>print(postResample(pred = pred\_train\_RF, obs = train[,1]))

RMSE Rsquared MAE

0.12402476 0.92113869 0.08824874

># For testing data

>print(postResample(pred = pred\_test\_RF, obs = test[,1]))

RMSE Rsquared MAE

0.2710569 0.4604025 0.1936760

There was a considerable difference in the RMSE values. This might be the result of over or under fitting on data for Random Forest.

**Hence we shall go for XGBoost in R to predict the cab fare for the test data**

As for Python – We can see that Linear Regression Model performed the best and had the lowest and highest value for RMSE and MAE.

(I could not implement XGBoost Model in Python due to some installation issues covered in the miscellaneous section.)

**Hence we shall go for Linear Regression in Python to predict the cab fare for the test data**

R-Code

# Clearing the environment

rm(list=ls(all=T))

# Setting working directory

setwd("G:/Data Analytics/Cab-Fare-Project")

getwd()

#Load Libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

#install.packages(x)

lapply(x, require, character.only = TRUE)

install.packages("xlsx")

library(xlsx)

library(ggplot2)

## Reading the data

df\_cab\_train= read.csv('train\_cab.csv')

df\_cab\_test=read.csv('test.csv')

#----------------------------------------------------Exploratory Data Analysis------------------------------------------------------

# Shape of the data

dim(df\_cab\_train)

# Viewing data

View(df\_cab\_train)

# Structure of the data

str(df\_cab\_train)

# Variable namesof the data

colnames(df\_cab\_train)

#summary of the data set

summary(df\_cab\_train)

#

class(df\_cab\_train)

# Shape of the data

dim(df\_cab\_test)

# Viewing data

View(df\_cab\_test)

# Structure of the data

str(df\_cab\_test)

# Variable namesof the data

colnames(df\_cab\_test)

#summary of the data set

summary(df\_cab\_test)

#

class(df\_cab\_test)

#---------------------------------Analysis----------------------

#During the loading of the data it was found that fare amount was considered as a favtor value.

#Comparing the same inital load with python as the corresponding values, it was found that the Null values in R were replaced as 1 in the fare\_amount section

#Hence that is incorrect data being populated and we have to drop those values

#since fare\_amount was of type factor, converting it to numeric

df\_cab\_train$fare\_amount=as.numeric(df\_cab\_train$fare\_amount)

df\_cab\_train= subset(df\_cab\_train, fare\_amount != 1)

train\_col= colnames(df\_cab\_train)

#------------------------------------Missing Values Analysis---------------------------------------------------#

#converting factor type pickup time to a character first

df\_cab\_test$pickup\_datetime=as.character(df\_cab\_test$pickup\_datetime)

#Creating dataframe with missing values present in each variable

missing\_val = data.frame(apply(df\_cab\_train,2,function(x){sum(is.na(x))}))

missing\_val$Columns = row.names(missing\_val)

names(missing\_val)[1] = "Missing\_percentage"

#Calculating percentage missing value

missing\_val$Missing\_percentage = (missing\_val$Missing\_percentage/nrow(df\_cab\_train)) \* 100

missing\_val

# Sorting missing\_val in Descending order

missing\_val = missing\_val[order(-missing\_val$Missing\_percentage),]

row.names(missing\_val) = NULL

# Reordering columns

missing\_val = missing\_val[,c(2,1)]

# Saving output result into csv file

write.csv(missing\_val, "Missing\_perc\_R.csv", row.names = F)

#Since missing values are present only in passenger count with a 0.34%, hence removing those values from our data set

df\_cab\_train=na.omit(df\_cab\_train)

df\_cab\_test=na.omit(df\_cab\_test)

ggplot(data = missing\_val[1:18,], aes(x=reorder(Columns, -Missing\_percentage),y = Missing\_percentage))+

geom\_bar(stat = "identity",fill = "orange")+xlab("Variables")+

ggtitle("Variables Vs Missing Percent Plot") + theme\_bw()

#-------------------------------Outliers and Plots--------------------------

# Boxplot for continuous variables

boxplot(df\_cab\_train$fare\_amount, xlab="Fare",ylab="count")

boxplot(df\_cab\_train$pickup\_longitude, xlab="Pickup\_Longitude",ylab="count")

boxplot(df\_cab\_train$pickup\_latitude, xlab="Pickup\_Latitude",ylab="count")

boxplot(df\_cab\_train$dropoff\_latitude, xlab="Dropoff\_Latitude",ylab="count")

boxplot(df\_cab\_train$dropoff\_longitude, xlab="Dropoff\_Longitude",ylab="count")

boxplot(df\_cab\_train$passenger\_count, xlab="Passenger\_Count",ylab="count")

#-------------------------------Refining the data----------------------------

#converting factor type pickup time to a character first

df\_cab\_train$pickup\_datetime=as.character(df\_cab\_train$pickup\_datetime)

#converting inot datetime

df\_cab\_train$pickup\_datetime<- as.POSIXct(df\_cab\_train$pickup\_datetime,format = "%Y-%m-%d %H:%M:%S",tz="")

#Extracting date and time values

df\_cab\_train$Hours <- as.numeric(format(as.POSIXct(strptime(df\_cab\_train$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%H"))

df\_cab\_train$Minutes <- as.numeric(format(as.POSIXct(strptime(df\_cab\_train$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%M"))

df\_cab\_train$Date <- as.numeric(format(as.POSIXct(strptime(df\_cab\_train$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%d"))

df\_cab\_train$Month <- as.numeric(format(as.POSIXct(strptime(df\_cab\_train$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%m"))

df\_cab\_train$Year <- as.numeric(format(as.POSIXct(strptime(df\_cab\_train$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%Y"))

df\_cab\_train=na.omit(df\_cab\_train)

#converting into datetime

df\_cab\_test$pickup\_datetime<- as.POSIXct(df\_cab\_test$pickup\_datetime,format = "%Y-%m-%d %H:%M:%S",tz="")

#Extracting date and time values

df\_cab\_test$Hours <- as.numeric(format(as.POSIXct(strptime(df\_cab\_test$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%H"))

df\_cab\_test$Minutes <- as.numeric(format(as.POSIXct(strptime(df\_cab\_test$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%M"))

df\_cab\_test$Date <- as.numeric(format(as.POSIXct(strptime(df\_cab\_test$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%d"))

df\_cab\_test$Month <- as.numeric(format(as.POSIXct(strptime(df\_cab\_test$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%m"))

df\_cab\_test$Year <- as.numeric(format(as.POSIXct(strptime(df\_cab\_test$pickup\_datetime,"%Y-%m-%d %H:%M:%S",tz="")) ,format = "%Y"))

df\_cab\_test=na.omit(df\_cab\_test)

#--------------------------------------------------------------------------

#Fare amount should be a positive value, passenger to be less than 7

df\_cab\_train= subset(df\_cab\_train, passenger\_count <7 & passenger\_count >0)

df\_cab\_train= subset(df\_cab\_train, passenger\_count!=0.12)

df\_cab\_test=subset(df\_cab\_test, passenger\_count < 7 & passenger\_count > 0)

df\_cab\_train= subset(df\_cab\_train, fare\_amount >0)

install.packages('geosphere')

library(geosphere)

df\_cab\_train= subset(df\_cab\_train, (pickup\_longitude <180 & pickup\_longitude > -180))

df\_cab\_train= subset(df\_cab\_train, (pickup\_latitude<90 & pickup\_latitude > -90))

df\_cab\_train= subset(df\_cab\_train, (dropoff\_latitude<90 & dropoff\_latitude > -90))

df\_cab\_train= subset(df\_cab\_train, (dropoff\_longitude <180 & dropoff\_longitude > -180))

df\_cab\_test= subset(df\_cab\_test, (pickup\_longitude <180 & pickup\_longitude > -180))

df\_cab\_test= subset(df\_cab\_test, (pickup\_latitude<90 & pickup\_latitude > -90))

df\_cab\_test= subset(df\_cab\_test, (dropoff\_latitude<90 & dropoff\_latitude > -90))

df\_cab\_test= subset(df\_cab\_test, (dropoff\_longitude <180 & dropoff\_longitude > -180))

#Using Haversine Function of Geosphere package

#distHaversine(c(-73.84431,40.72132), c(-73.84161,40.71228), r=6378137)

df\_cab\_train$Distance\_travelled <-distHaversine(cbind(df\_cab\_train$pickup\_longitude, df\_cab\_train$pickup\_latitude),cbind(df\_cab\_train$dropoff\_longitude,df\_cab\_train$dropoff\_latitude))

df\_cab\_test$Distance\_travelled <-distHaversine(cbind(df\_cab\_test$pickup\_longitude, df\_cab\_test$pickup\_latitude),cbind(df\_cab\_test$dropoff\_longitude,df\_cab\_test$dropoff\_latitude))

#Converting in KM

df\_cab\_train$Distance\_travelled<-(df\_cab\_train$Distance\_travelled)/1000

df\_cab\_test$Distance\_travelled<-(df\_cab\_test$Distance\_travelled)/1000

#Analysis over distance

#BoxPlot

boxplot(df\_cab\_train$Distance\_travelled)

summary(df\_cab\_train$Distance\_travelled)

#Min. 1st Qu. Median Mean 3rd Qu. Max.

#0.000 1.216 2.128 15.082 3.856 8677.252

df\_cab\_train[order(-df\_cab\_train$Distance\_travelled),]

#After googling we got to know that coordinate (0.00000,0.00000) lies in the middle of the ocean.

#Hence removing these values

df\_cab\_train= subset(df\_cab\_train, (pickup\_longitude != 0.00000 & pickup\_longitude != 0.00000))

df\_cab\_train= subset(df\_cab\_train, (dropoff\_longitude != 0.00000 & dropoff\_longitude != 0.00000))

df\_cab\_test= subset(df\_cab\_test, (pickup\_longitude != 0.00000 | pickup\_longitude != 0.00000))

df\_cab\_test= subset(df\_cab\_test, (dropoff\_longitude != 0.00000 | dropoff\_longitude != 0.00000))

#BoxPlot

boxplot(df\_cab\_train$Distance\_travelled)

df\_cab\_train[order(df\_cab\_train$Distance\_travelled),]

#After again plotting the values of boxplot, and ordering the data set in reverse order on basis of distance

df\_cab\_train= subset(df\_cab\_train, Distance\_travelled < 4452067)

#considering a business case that a passenger can travel at max 50 km by cab

df\_cab\_train=subset(df\_cab\_train,Distance\_travelled < 51)

df\_cab\_train

#----------------------------Visualizations------------------------------------

df\_columns=colnames(df\_cab\_train)

df\_columns=c('fare\_amount','pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude',

'passenger\_count','Hours','Minutes','Date','Month','Year','Distance\_travelled')

for (i in df\_columns){

hist(df\_cab\_train[,i], col="yellow", prob=TRUE, main="Histogram",xlab=i, ylab="Count")

lines(density(df\_cab\_train[,i]), lwd=2)

}

#BoxPlot

boxplot(df\_cab\_test$Distance\_travelled)

df\_cab\_test[order(-df\_cab\_test$Distance\_travelled),]

#-------------------------Normalising data--------------------------

for(i in df\_columns)

{

print(i)

df\_cab\_train[,i] = (df\_cab\_train[,i] - min(df\_cab\_train[,i]))/(max(df\_cab\_train[,i])-min(df\_cab\_train[,i]))

}

#Taking some business scenarios to draw some inference

h <- hist(df\_cab\_train$passenger\_count)

#Maximum number of passengers were single

plot(df\_cab\_train$fare\_amount, df\_cab\_train$Distance\_travelled, main = "DistanceVsFare",

xlab = "Fare", ylab = "Distance",

pch = 19, frame = FALSE)

#------------------------------------------Model Development--------------------------------------------#

#Divide data into train and test using stratified sampling method

set.seed(123)

train.index = sample(1:nrow(df\_cab\_train), 0.8 \* nrow(df\_cab\_train))

train = df\_cab\_train[ train.index,]

test = df\_cab\_train[-train.index,]

train =train[,-2]

test=test[,-2]

##Decision tree for classification

#Develop Model on training data

library(rpart)

df\_dtree=rpart(fare\_amount~., data = train, method = "anova")

dev.new()

plot(df\_dtree)

text(df\_dtree)

summary(df\_dtree)

printcp(df\_dtree)

#df\_cab\_train[,1]

#write rules into disk

write(capture.output(summary(df\_dtree)), "Rules.txt")

#Lets predict for test data

pred\_test\_DT = predict(df\_dtree,test[,-1])

#Lets predict for train data

pred\_train\_DT = predict(df\_dtree,train[,-1])

install.packages("caret")

library(caret)

# For training data

print(postResample(pred = pred\_train\_DT, obs = train[,1]))

# RMSE Rsquared MAE

#0.2868771 0.3944942 0.2146257

# For testing data

print(postResample(pred = pred\_test\_DT, obs = test[,1]))

# RMSE Rsquared MAE

#0.2908909 0.3784100 0.2172989

#-----------------Linear Regression-----------------------

#check multicollearity

install.packages("usdm")

library(usdm)

vif(df\_cab\_train[,-1:-2])

vifcor(df\_cab\_train[,-1:-2], th = 0.9)

#run regression model

LR\_model = lm( fare\_amount ~., data = train)

#Summary of the model

summary(LR\_model)

#Lets predict for test data

pred\_test\_LR = predict(LR\_model,test[,-1])

#Lets predict for train data

pred\_train\_LR = predict(LR\_model,train[,-1])

#Predict

predictions\_LR = predict(LR\_model, test[,2:12])

# For training data

print(postResample(pred = pred\_train\_LR, obs = train[,1]))

# RMSE Rsquared MAE

#0.35559168 0.06968557 0.31445119

# For testing data

print(postResample(pred = pred\_test\_LR, obs = test[,1]))

# RMSE Rsquared MAE

#0.35704205 0.06346288 0.31768677

#------------------RANDOM FOREST------------------

install.packages("randomForest")

library(randomForest)

#RUN RANDOM FOREST

RF\_model = randomForest(fare\_amount ~., train, importance = TRUE, ntree = 300)

#Summary of the model

summary(RF\_model)

#Lets predict for test data

pred\_test\_RF = predict(RF\_model,test[,-1])

#Lets predict for train data

pred\_train\_RF = predict(RF\_model,train[,-1])

#Predict

predictions\_RF = predict(RF\_model, test[,2:12])

# For training data

print(postResample(pred = pred\_train\_RF, obs = train[,1]))

# RMSE Rsquared MAE

#0.12397933 0.92167556 0.08846157

# For testing data

print(postResample(pred = pred\_test\_RF, obs = test[,1]))

# RMSE Rsquared MAE

#0.2720930 0.4578688 0.1965762

#----------------------------XGBOOST----------------------

install.packages('devtools')

install.packages('xgboost')

install.packages('gbm')

#https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/

library(xgboost)

library(readr)

library(stringr)

library(gbm)

fit\_XGB = gbm(fare\_amount~., data = train, n.trees = 500, interaction.depth = 2)

#Lets predict for training data

pred\_XGB\_train = predict(fit\_XGB, train, n.trees = 500)

#Lets predict for testing data

pred\_XGB\_test = predict(fit\_XGB,test, n.trees = 500)

# For training data

print(postResample(pred = pred\_XGB\_train, obs = train$fare\_amount))

#RMSE Rsquared MAE

#0.2658812 0.4811690 0.1949224

# For testing data

print(postResample(pred = pred\_XGB\_test, obs = test$fare\_amount))

#RMSE Rsquared MAE

#0.2754497 0.4425815 0.2013333

#---------------------------------------Predicting using XGBoost--------------

#Lets predict for test data, adding new column for Predicted fare

df\_cab\_test$Predicted\_Fare = predict(fit\_XGB,df\_cab\_test, n.trees = 500)

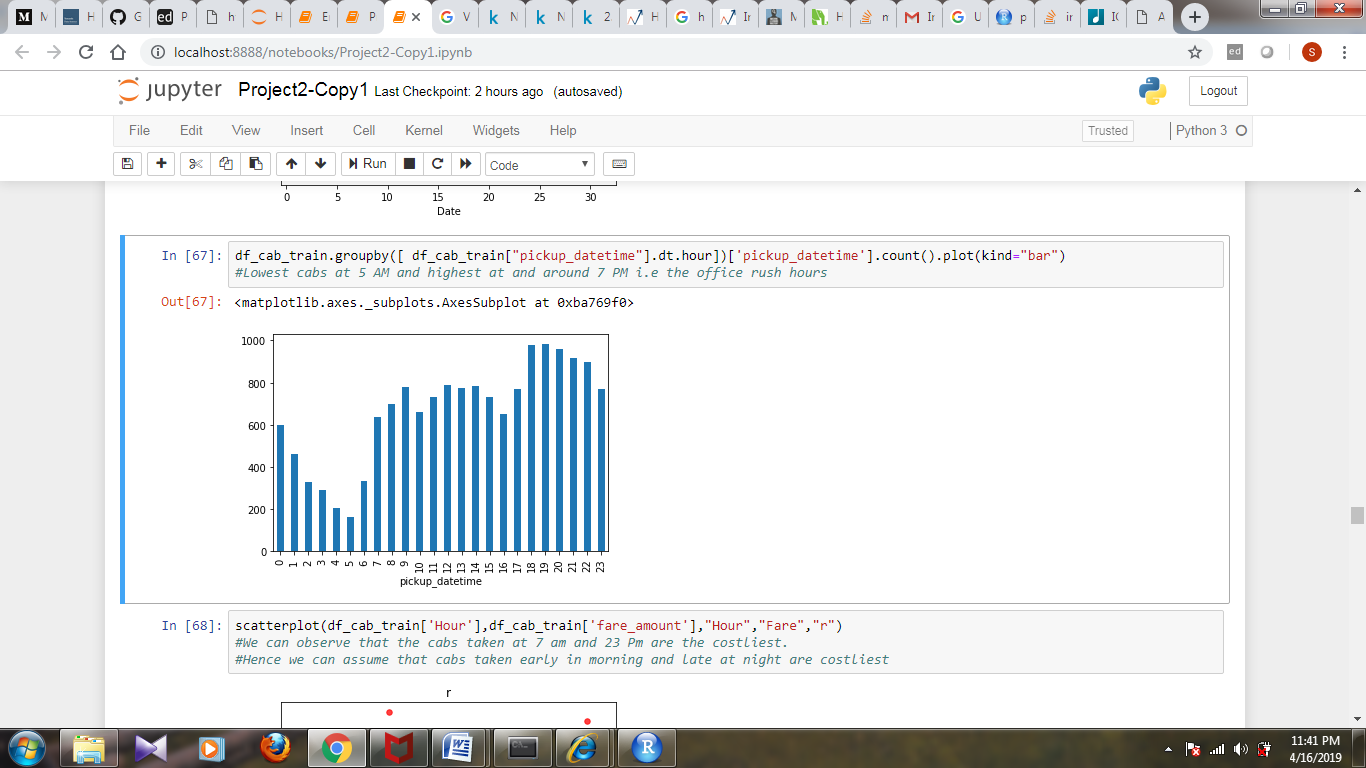
df\_cab\_test

**Miscellaneous Figures**

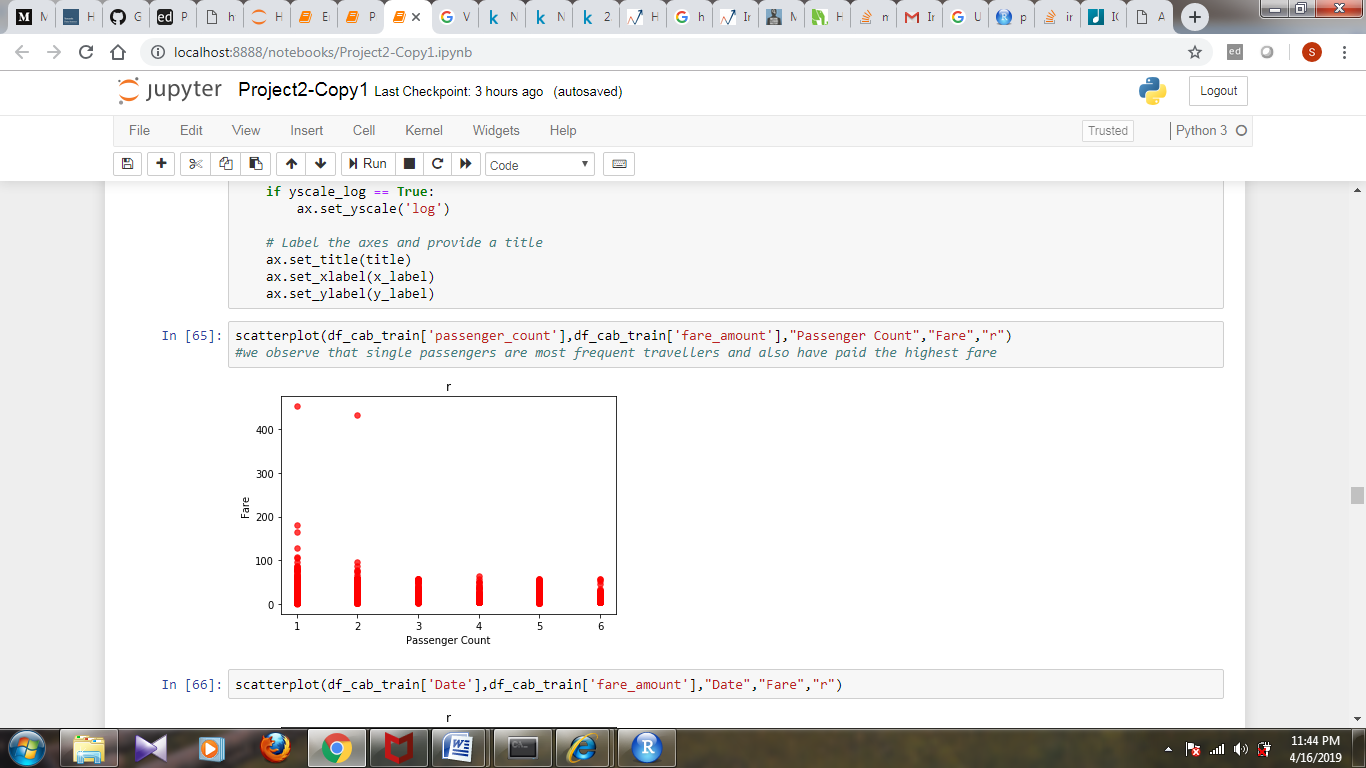
**Plots to predict some insights on the basis of data**

1. Relationship between Hour and number of rides:

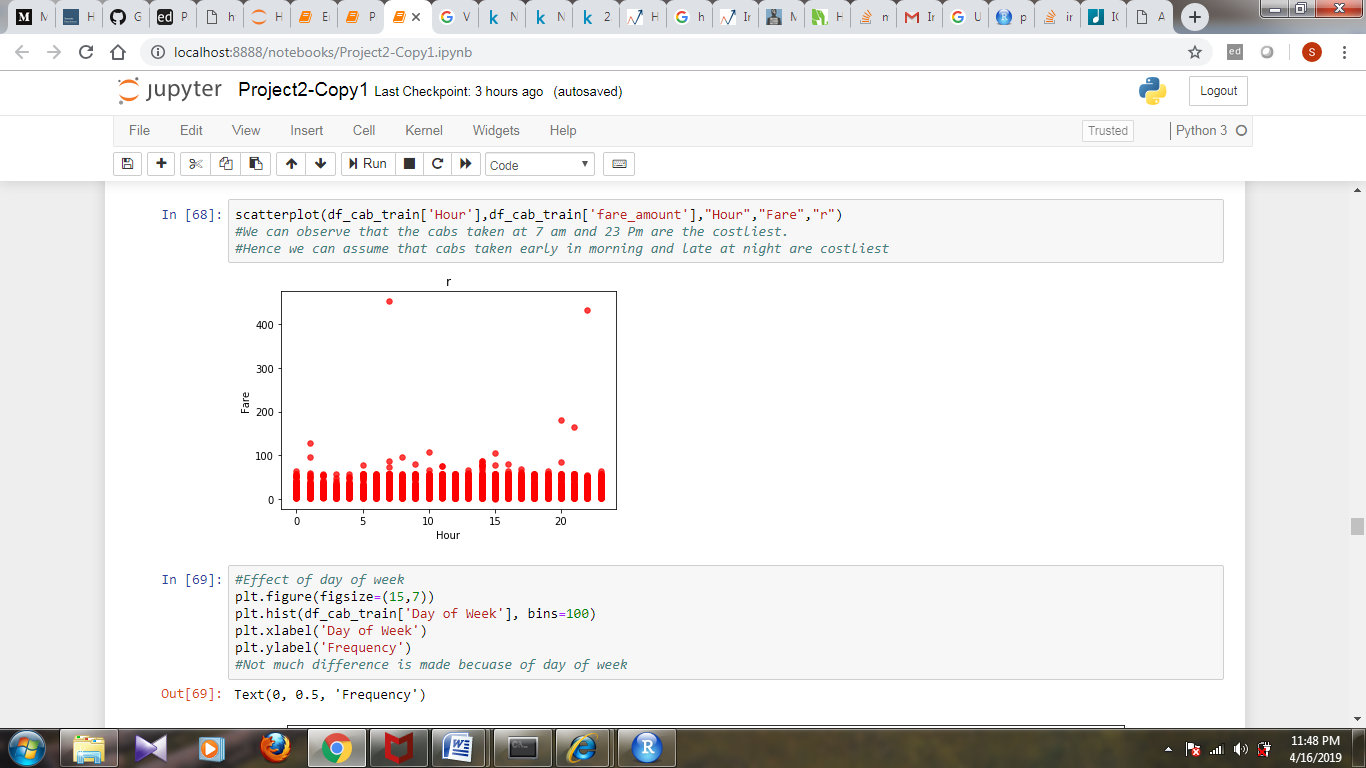
This depicts that maximum rides are taken at 6-8 PM and minimum at 5 AM.



1. Maximum number of passengers are single :



1. Fare is usually higher at some specific hours:



1. Not much relation is observed on the basis of day of week:

