PROJECT REPORT

On

CAB FARE PREDICTION

Submitted by
Neeta Bharti

edWisor

15-09-2019

TABLE OF CONTENTS

Chapter 1
Introduction

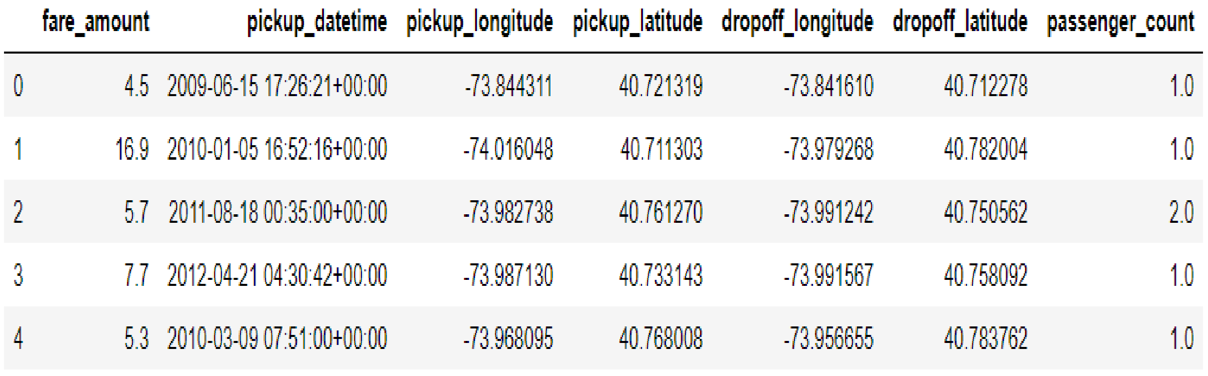
1.1 Problem Statement

The objective of this project is to predict Cab Fare amount. We have a cab rental start-up company. We have successfully run the pilot project and now want to launch our cab service across the country. We have collected the historical data from our pilot project and now have a requirement to apply analytics for fare prediction. We need to design a system that predicts the fare amount for a cab ride in the city.

1.2 Data

Our task is to build a model which will predict the cab fare amount depending on multiple factors/variables. Given below is a sample of the data set that we are using to predict the cab fare amount.

Table 1.1 Cab fare prediction sample data



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

As we can see in the table above we have the following 6 independent variables, using which we have to correctly predict the cab fare amount which is our target variable .

The details of data attributes in the dataset are as follows :

- 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-fare for the cab ride.

Chapter 2
Exploratory Data Analysis

Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis(EDA). Remember the quality of your inputs decide the quality of your output. So, once you have got your business hypothesis ready, it makes sense to spend lot of time and efforts here. During this project Data exploration, cleaning and preparation took up to 70% of my total project time.

Below are the steps involved to understand, clean and prepare the data for building the predictive model:

1. Variable Identification
2. Univariate Analysis
3. Bi-variate Analysis
4. Outlier analysis
5. Missing value analysis
6. Feature Engineering
7. Feature selection
8. Feature scaling

2.1 Variable Identification

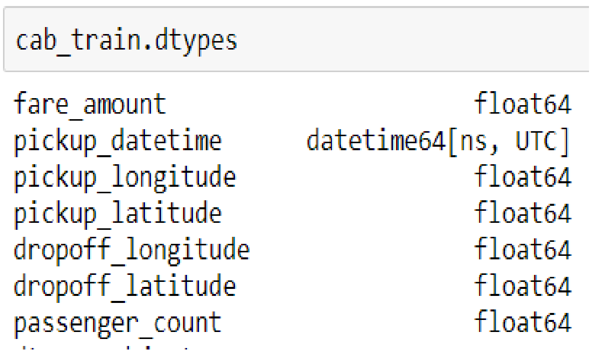
Under Variable Identification first, I identified the Predictor and Target variables. Next I identified the data types and category of the variables.

Table 2.1 Independent Variables

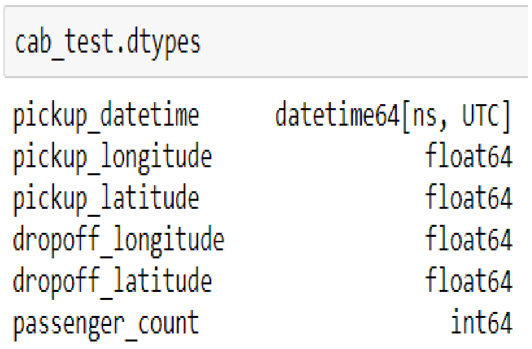
S.No	Variables
1	pickup_datetime
2	pickup_longitude
3	pickup_latitude
4	dropoff_longitude
5	dropoff_latitude
6	passenger_count

Target variable: fare_amount

Data type of the variables are as below:



```
cab_train.dtypes

fare_amount                          float64
pickup_datetime          datetime64[ns, UTC]
pickup_longitude                     float64
pickup_latitude                      float64
dropoff_longitude                    float64
dropoff_latitude                     float64
passenger_count                      float64
```



```
cab_test.dtypes

pickup_datetime    datetime64[ns, UTC]
pickup_longitude               float64
pickup_latitude                float64
dropoff_longitude              float64
dropoff_latitude               float64
passenger_count                  int64
```

2.2 Univariate Analysis

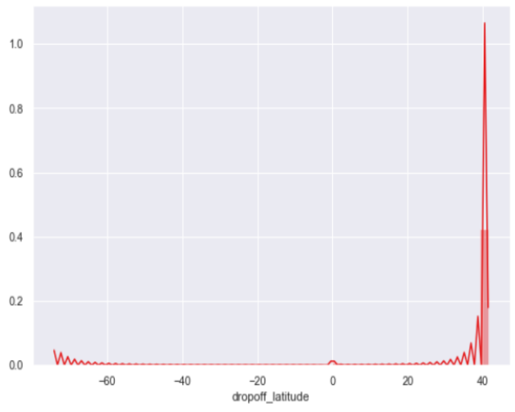
At this stage, we explore variables one by one. Method to perform univariate analysis will depend on whether the variable type is categorical or continuous. Let's look at these methods and statistical measures for categorical and continuous variables individually:

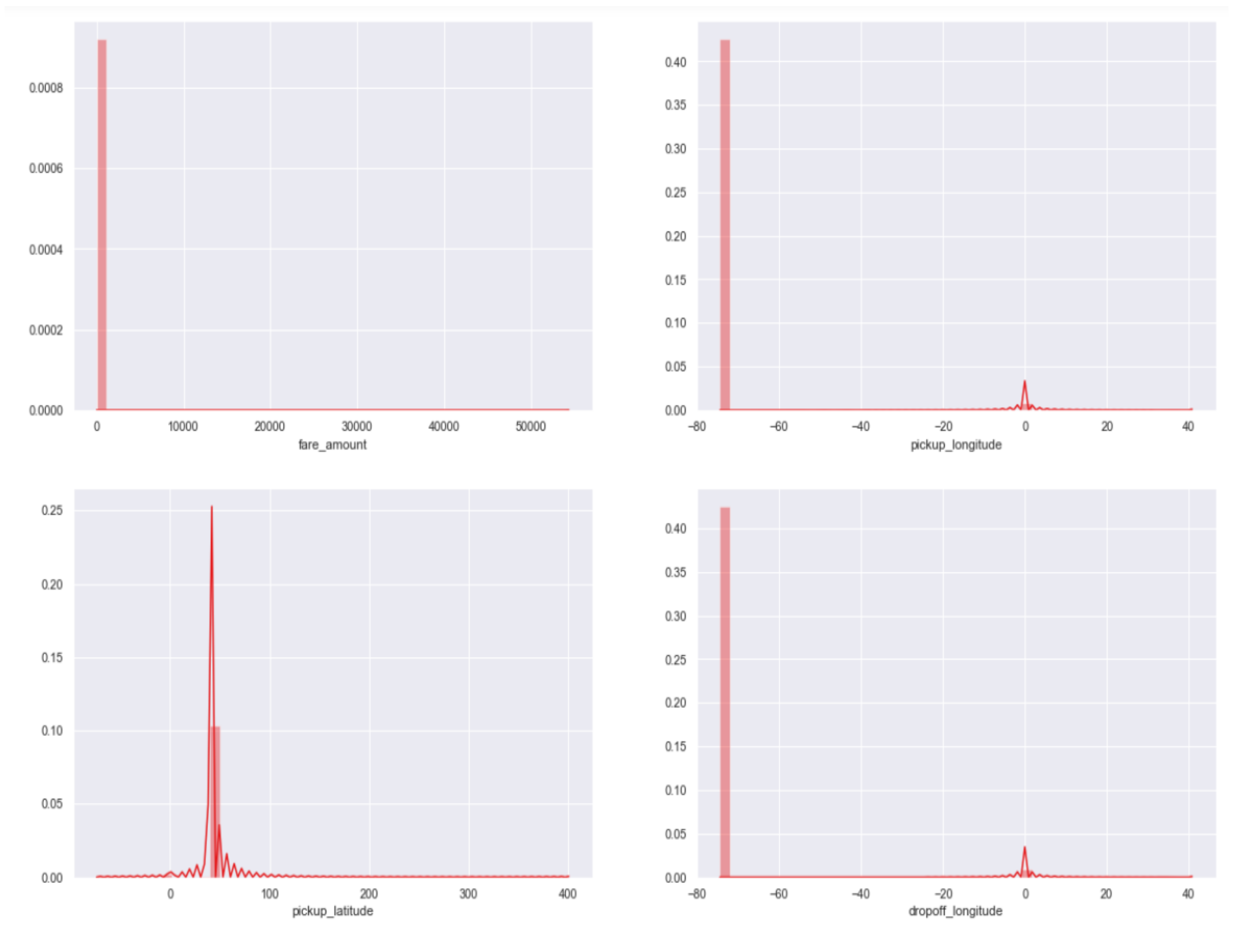
Continuous Variables: In case of continuous variables, we need to understand the central tendency and spread of the variable. These are measured using various statistical metrics visualization methods as shown below:

Categorical Variables: For categorical variables, we'll use frequency table to understand distribution of each category. We can also read as percentage of values under each category. It can be be measured using two metrics, Count and Count% against each category. Bar chart can be used as visualization.

Distribution Plot

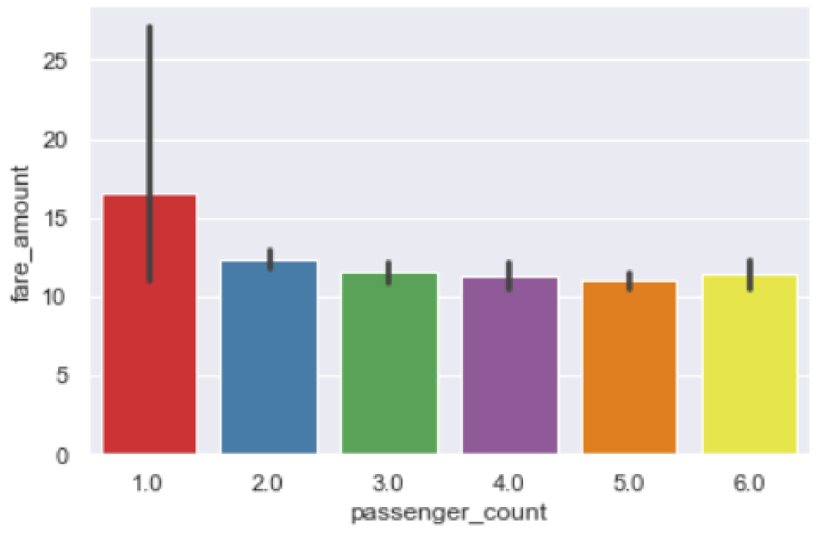
The distplot shows the distribution of a univariate set of observations. Some Histogram plots from seaborn library for each individual variable created using distplot() method.





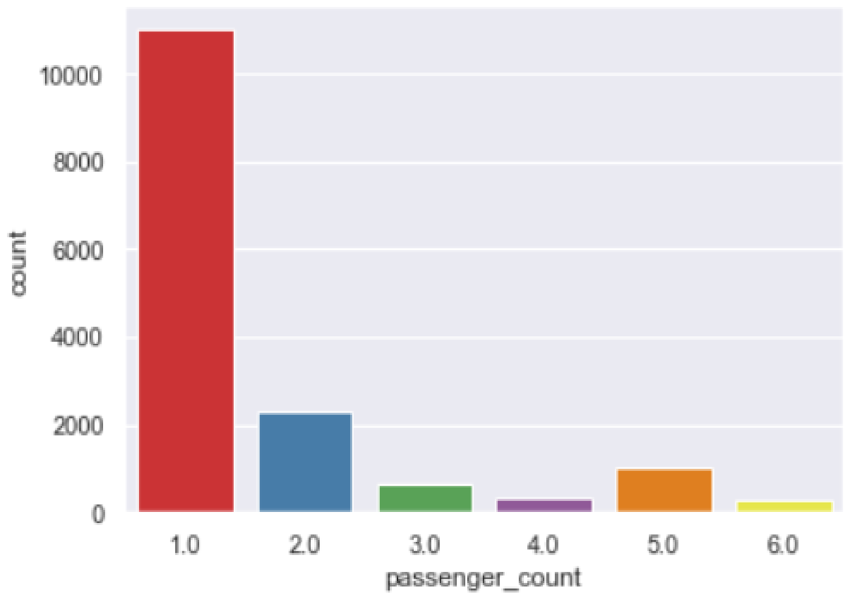
Bar Plot

These very similar plots allow us to get aggregate data off a categorical feature in our data. Barplot is a general plot that allows us to aggregate the categorical data based off some function, by default the mean:



Count plot

This is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why we only pass the x value:



2.3 Bi-variate Analysis

Bi-variate Analysis finds out the relationship between two variables. Here, we look for association and disassociation between variables at a pre-defined significance level. We can perform bi-variate analysis for any combination of categorical and continuous variables.
The combination can be: Categorical & Categorical, Categorical & Continuous and Continuous & Continuous. Different methods are used to tackle these combinations during analysis process. Let's understand the possible combinations in detail:

Continuous & Continuous**:** While doing bi-variate analysis between two continuous variables, we should look at scatter plot. It is a nifty way to find out the relationship between two variables. The pattern of scatter plot indicates the relationship between variables. The relationship can be linear or non-linear.

Scatter plot shows the relationship between two variable but does not indicates the strength of relationship amongst them. To find the strength of the relationship, we use Correlation. Correlation varies between -1 and +1.

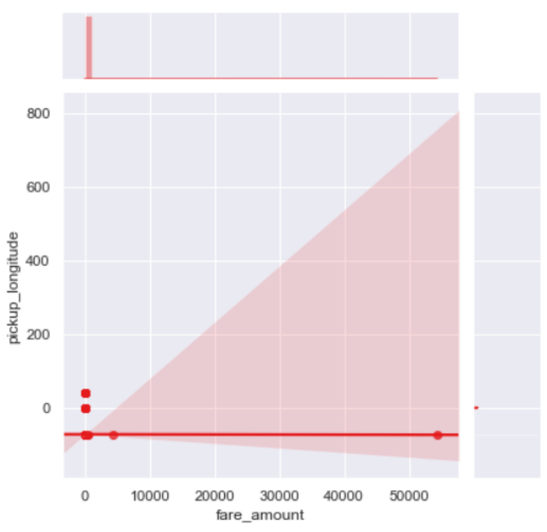
- -1: perfect negative linear correlation
- +1:perfect positive linear correlation and
- 0: No correlation

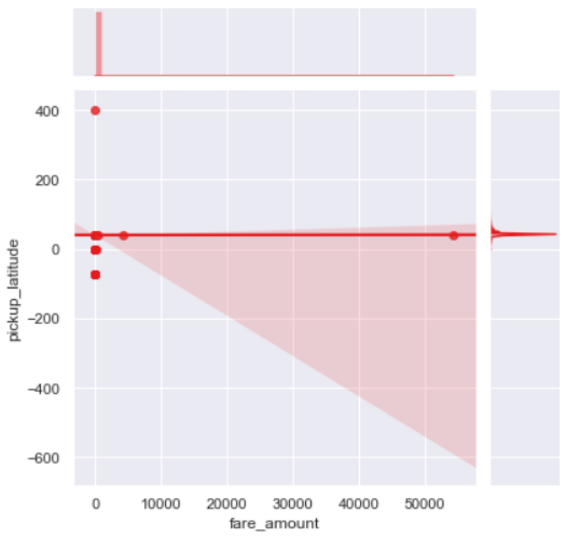
Joint Plot

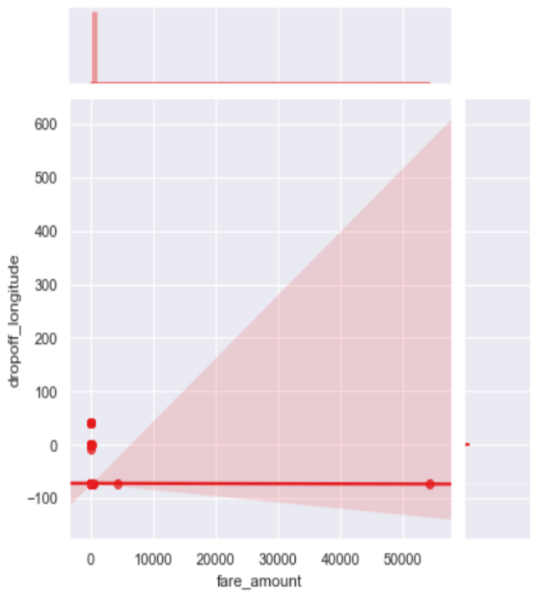
jointplot() allows us to basically match up two distplots for bivariate data. With our choice of what kind parameter to compare with:

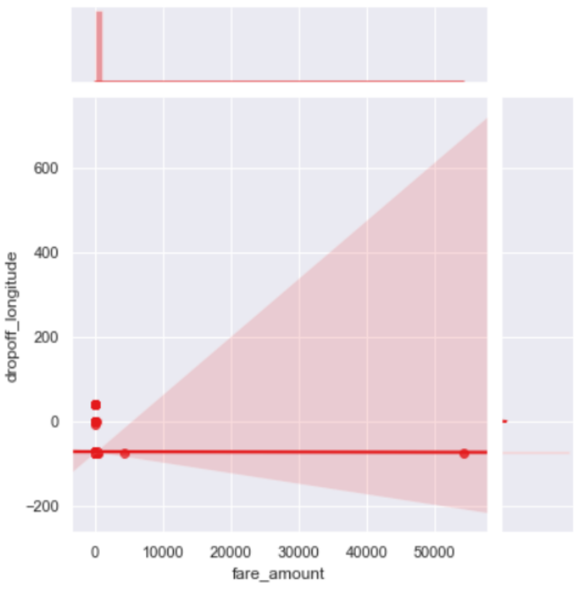
Some Jointplots:

- They are used for Bivariate Analysis.
- Here we have plotted Scatter plot with Regression line between 2 variables along with separate Bar plots of both variables.
- Plotted only for numerical/continuous variables.
- Target variable ‘fare_amount’ Vs each numerical variable.





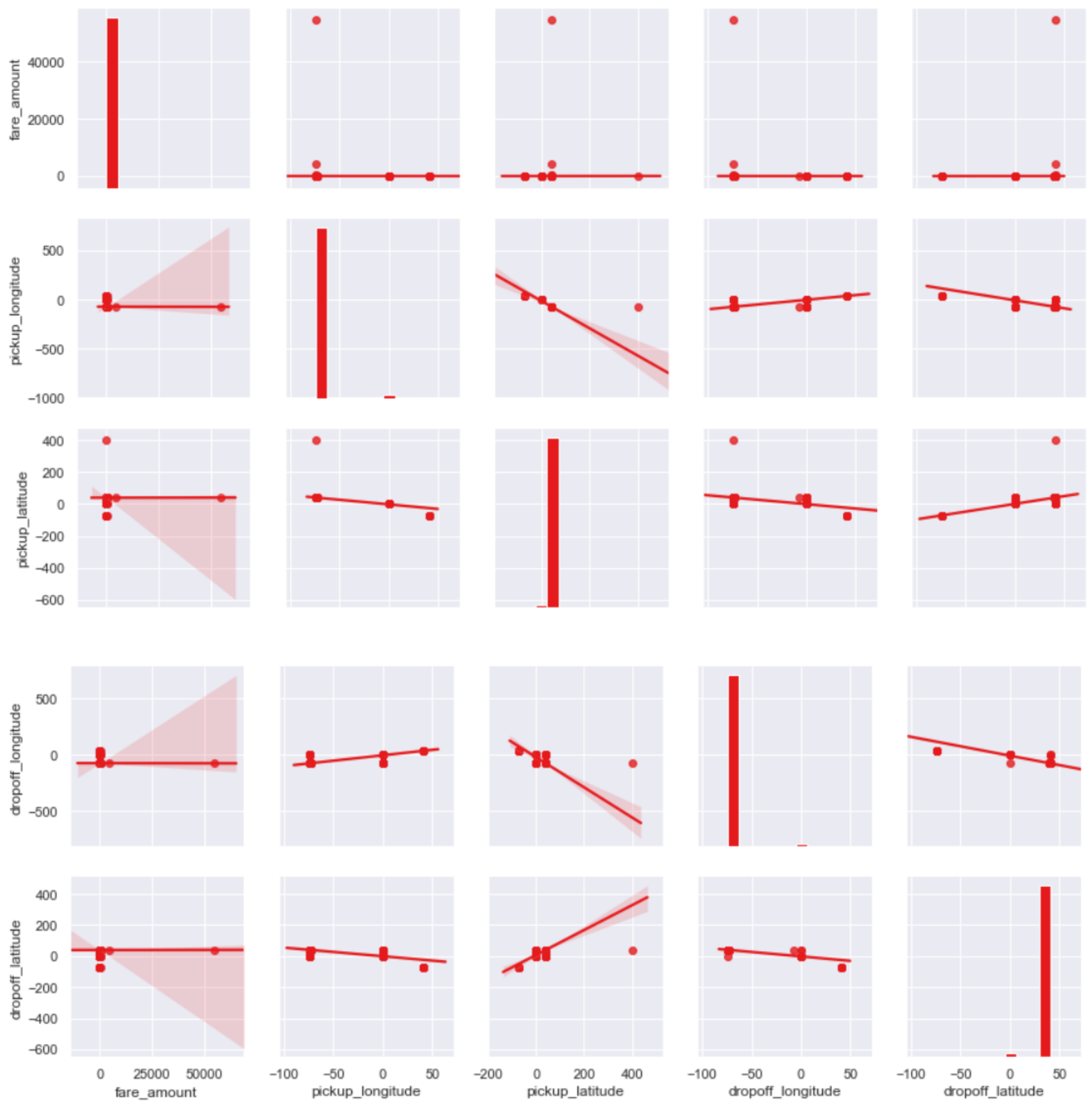




Pair Plot

Pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns).

Pairwise Plots for all Numerical variables:



2.4 Outlier Analysis

Outlier is commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

Methods to detect Outliers:

Most commonly used method to detect outliers is visualization. We use various visualization methods, like Box-plot, Histogram, Scatter Plot (above, we have used box plot and scatter plot for visualization).

Some analysts also use various thumb rules to detect outliers. Some of them are:

- Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR.

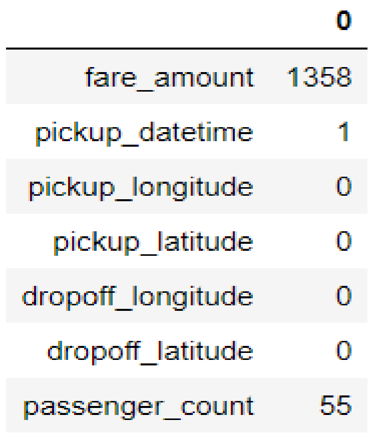
- Use capping methods. Any value which out of range of 5th and 95th percentile can be considered as outlier.
- Data points, three or more standard deviation away from mean are considered outlier.
- Outlier detection is merely a special case of the examination of data for influential data points and it also depends on the business understanding.

Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.
1. Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve So I have removed these fields.
2. Passenger_count variable (should be between 1 to 6 max)
I have removed the observations which are above 6 and below 1 and also the decimal values because a cab cannot hold these number of passengers.
3. Latitudes range from -90 to 90 and Longitudes range from -180 to 180.
I removed the observations which do not satisfy these ranges. I also removed the outliers like for pickup_lat>90 and values=0.
Finally, we loosed 16067-15662=405 observations because of irrelevant values.
Further I looked for outlier in the dataset by plotting Boxplots. There are outliers present in the data. I have removed these outliers. This is how I have done,

We can observe from the visualization that the variables like 'fare_amount' and 'passenger_count' have outliers. So I have removed the outliers as below :
Q1=first quartile
Q3=third quartile
IQR(Interquartile range)=Q3-Q1
A value is considered an outlier if it is less than 1.5 times the interquartile range below Q1 or more than 1.5 times the interquartile range above Q3.

1. I replaced them with Nan values or can say I created missing values.
2. Then I imputed those missing values with median method.

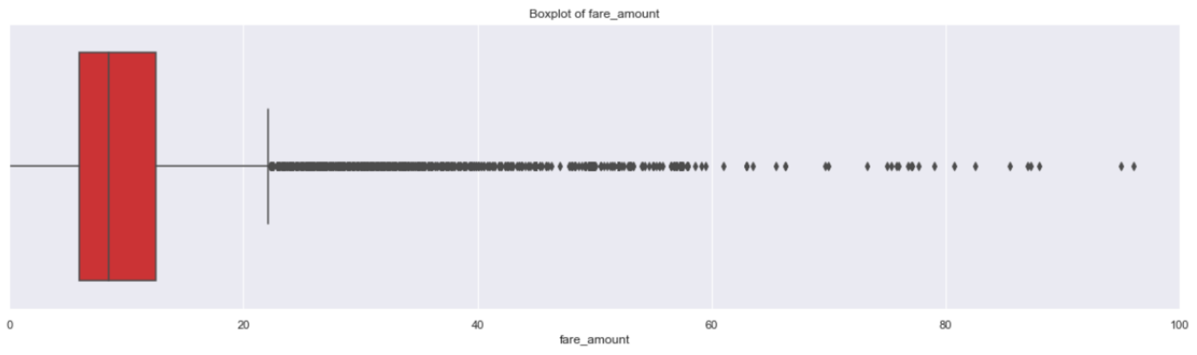
Code for outliers removal is given in Appendix.



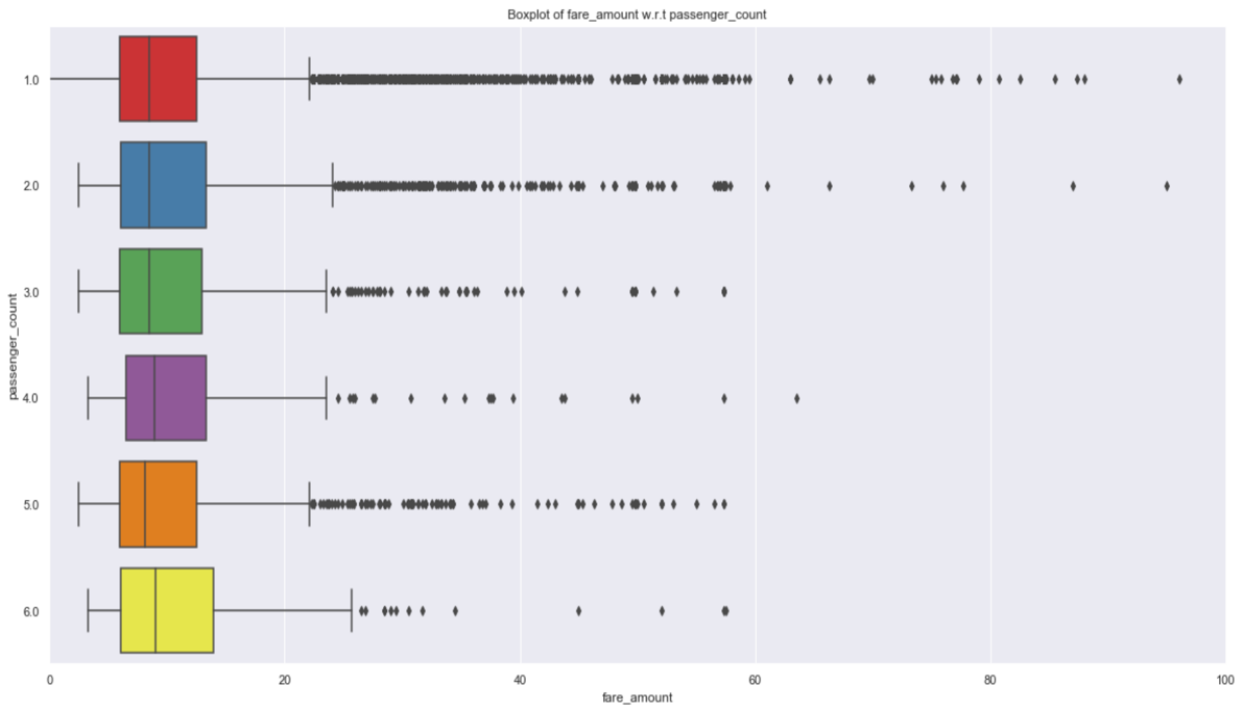
	0
fare_amount	1358
pickup_datetime	1
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
passenger_count	55

I have done Outlier Analysis only on Fare_amount and passenger_count just for now and I will do the outlier analysis for other variables after feature engineering latitudes and longitudes.

- Univariate Boxplots: Boxplots for target variable.



- Bivariate Boxplots: Boxplot for Numerical Variable Vs Categorical Variable.



Finally the outliers for fare_amount and passenger_count is removed.

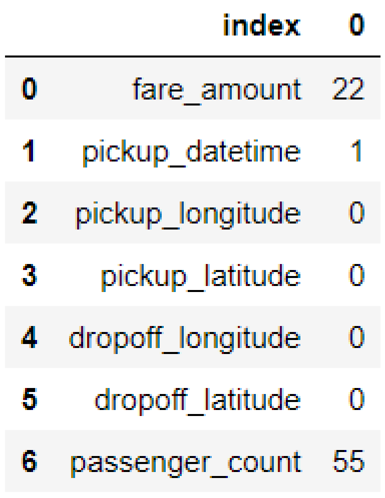
	0
fare_amount	0
pickup_datetime	0
pickup_longitude	0
pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
passenger_count	0

2.5 Missing Value Analysis

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behaviour and relationship with other variables correctly. It can lead to wrong prediction or classification.
Data has missing values because of the following reason:
Let's identify the reasons for occurrence of the missing values. They may occur at two stages:
1. Data Extraction: It is possible that there are problems with extraction process. In such cases, we should double-check for correct data with data guardians. Some hashing procedures can also be used to make sure data extraction is correct. Errors at data extraction stage are typically easy to find and can be corrected easily as well. 2. Data collection: These errors occur at time of data collection and are harder to correct. They can be categorized in four types:

- Missing completely at random: This is a case when the probability of missing variable is same for all observations. For example: respondents of data collection process decide that they will declare their earning after tossing a fair coin. If an head occurs, respondent declares his / her earnings & vice versa. Here each observation has equal chance of missing value.
- Missing at random: This is a case when variable is missing at random and missing ratio varies for different values / level of other input variables. For example: We are collecting data for age and female has higher missing value compare to male.
- Missing that depends on unobserved predictors: This is a case when the missing values are not random and are related to the unobserved input variable. For example: In a medical study, if a particular diagnostic causes discomfort, then there is higher chance of drop out from the study. This missing value is not at random unless we have included "discomfort" as an input variable for all patients.
- Missing that depends on the missing value itself: This is a case when the probability of missing value is directly correlated with missing value itself. For example: People with higher or lower income are likely to provide non-response to their earning.

Methods to treat Missing value:
Missing values can be easily treated using various methods like mean, median method, knn method to impute missing value.
Unfortunately, in this dataset I have found some missing values as below.



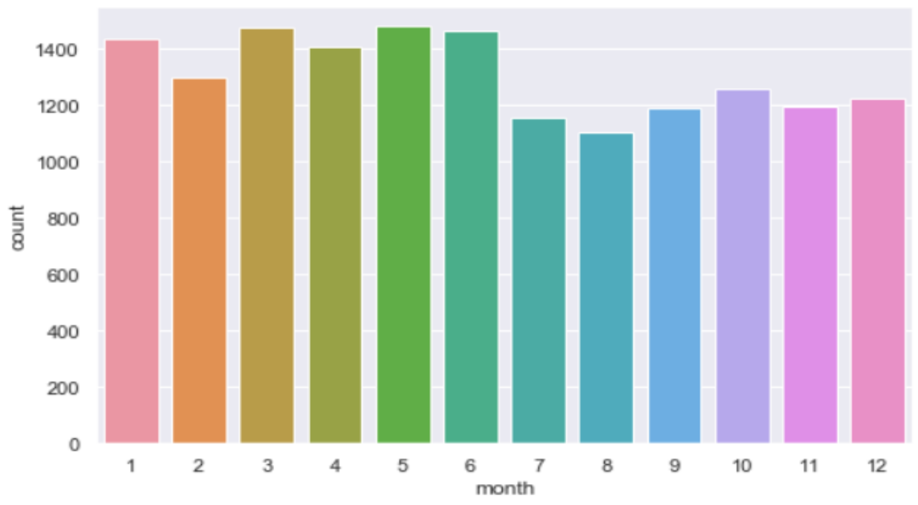
	index	0
0	fare_amount	22
1	pickup_datetime	1
2	pickup_longitude	0
3	pickup_latitude	0
4	dropoff_longitude	0
5	dropoff_latitude	0
6	passenger_count	55

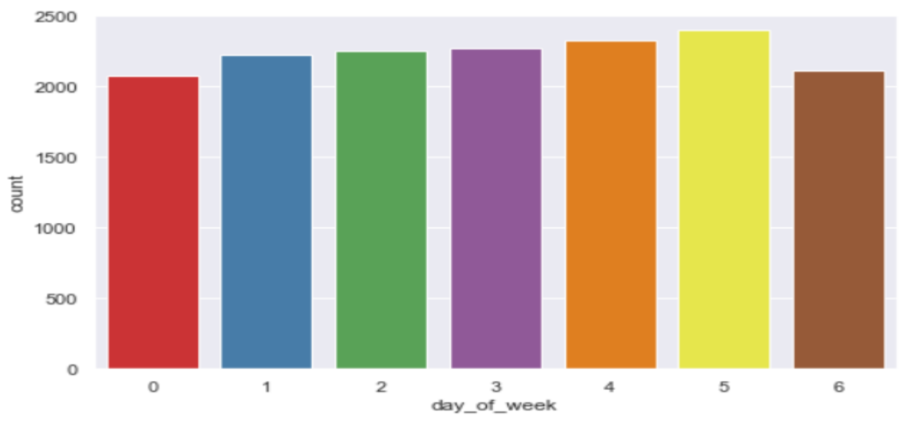
Therefore, I have done some missing value analysis. Before imputing I selected random row no-1000 and made it NA, so that I will compare original value with imputed value and choose best method which will impute value closer to actual value.
Median method is the best fit method here. So I imputed those missing value with median method and dropped one row that contain one missing column pickup_datetime.
Finally my dataset have no missing values.

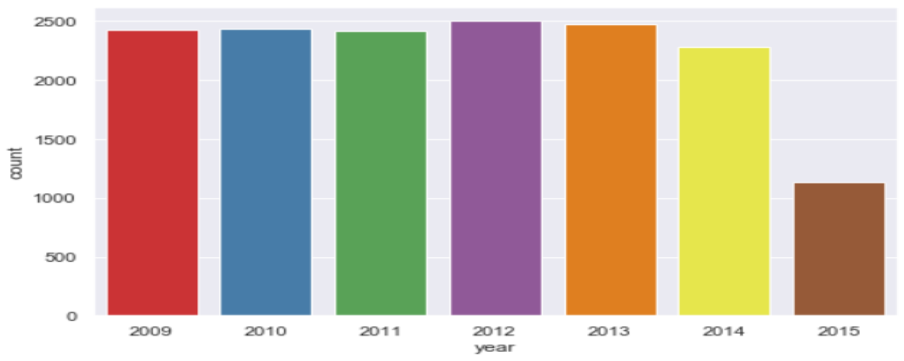
2.6 Feature Engineering

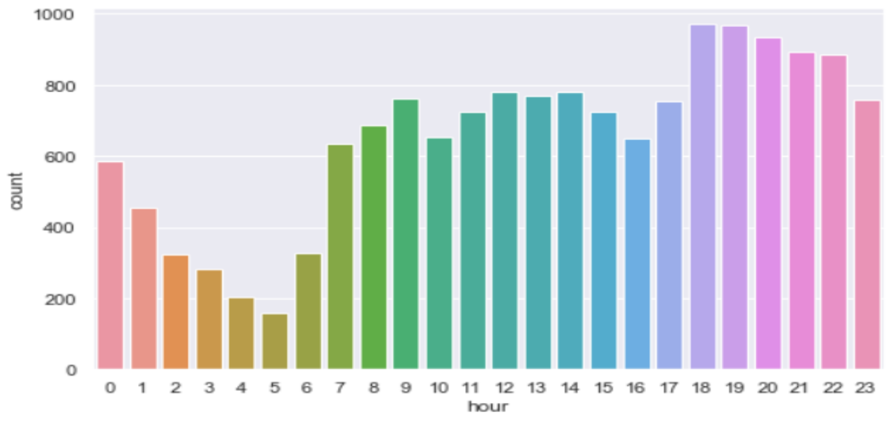
Feature engineering is the science (and art) of extracting more information from existing data. You are not adding any new data here, but you are actually making the data you already have more useful.

Feature / Variable creation is a process to generate a new variables / features based on existing variable(s). For example, say, we have date(dd-mm-yy) as an input variable in a data set. We can generate new variables like day, month, year, week, weekday that may have better relationship with target variable.
1. For 'pickup_datetime' variable: We will use this timestamp variable to create new variables. New features will be year, month, day_of_week, hour. 'year' will contain only years from pickup_datetime. For ex. 2009, 2010, 2011, etc. 'month' will contain only months from pickup_datetime. For ex. 1 for January, 2 for February, etc. 'day_of_week' will contain only week from pickup_datetime. For ex. 1 which is for Monday,2 for Tuesday,etc. 'hour' will contain only hours from pickup_datetime. For ex. 1, 2, 3, etc.

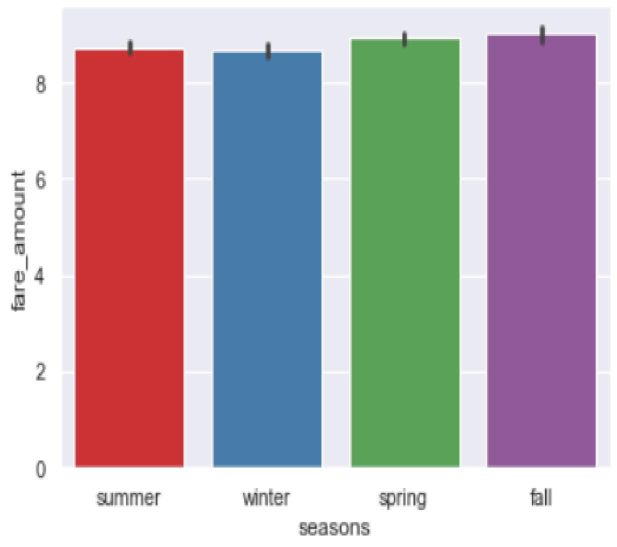


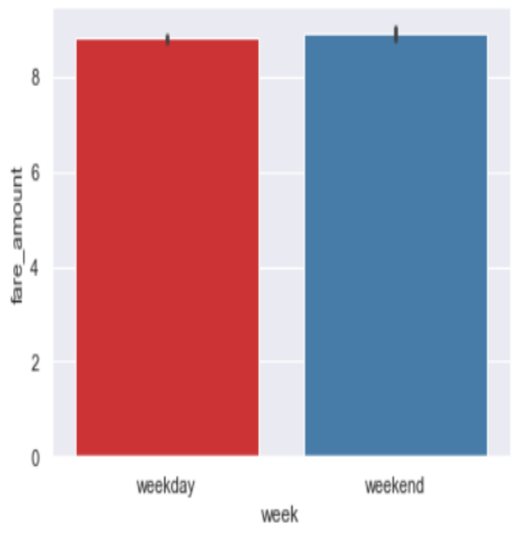


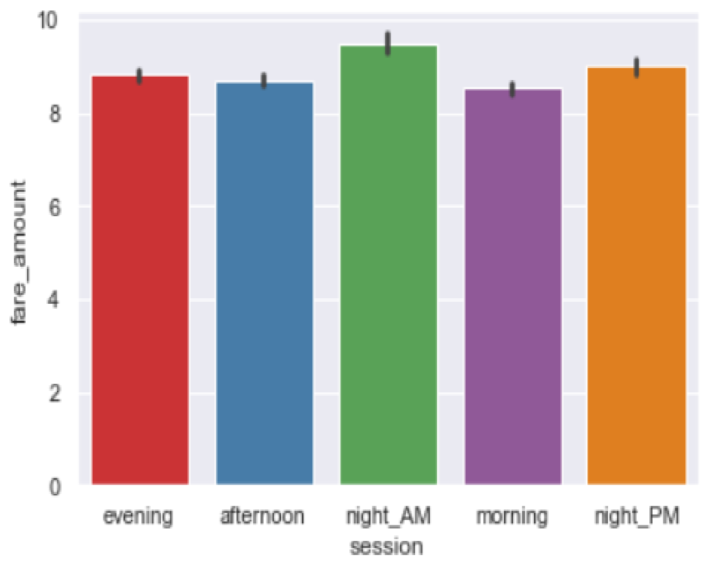




As we have now these new variables we will categorize them to new variables like Session from hour column, seasons from month column ,Week :weekday/weekend from day_of_week variable.
So, session variable which will contain categories—morning, afternoon, evening, night_PM, night_AM.
Seasons variable will contain categories—spring, summer, fall, winter.
Week will contain categories—weekday, weekend.
Let us see the barplot for these categorical variables.

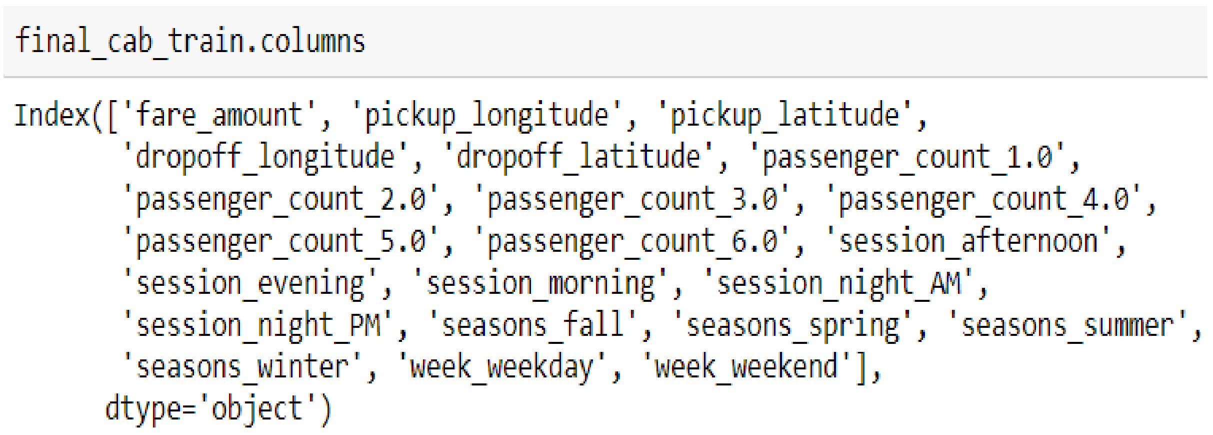






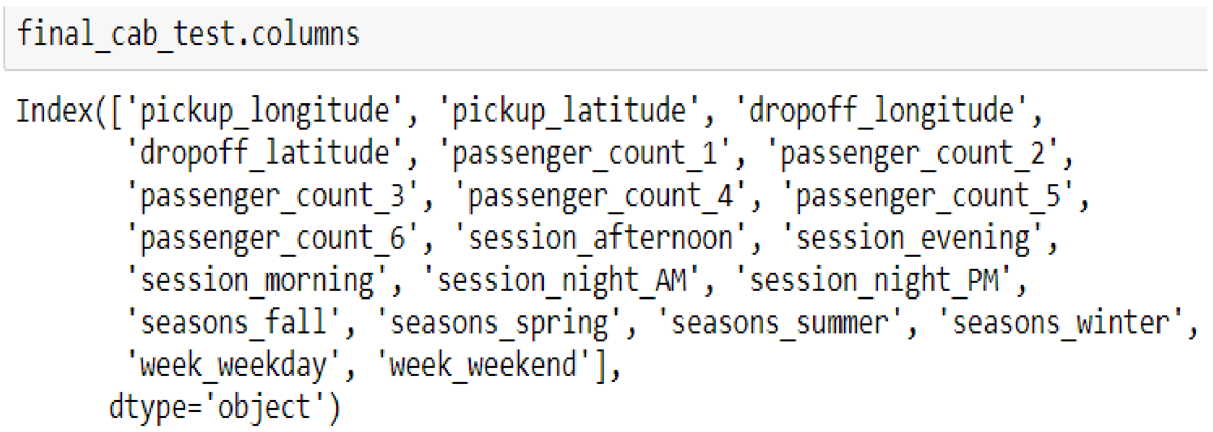
Conclusion
With the above barplot we can conclude that the fare_amount is highest at night session,and among season the fare_amount is highest during rainfall and avg fare_amount is almost same whether it's a week day or weekend.
Above conclusion is obvious if we think it logically.

2.**Creating dummy variables:** One of the most common application of dummy variable is to convert categorical variable into numerical variables. Dummy variables are also called Indicator Variables. It is useful to take categorical variable as a predictor in statistical models. Dummy variables are created for all the categorical variables. We will do one-hot-encoding on categorical variables like session, seasons, week variable, passenger_count.
After doing one-hot-encoding our train dataset has 22 columns. Below are the new columns:



```
final_cab_train.columns
```

```
Index(['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count_1.0',
       'passenger_count_2.0', 'passenger_count_3.0', 'passenger_count_4.0',
       'passenger_count_5.0', 'passenger_count_6.0', 'session_afternoon',
       'session_evening', 'session_morning', 'session_night_AM',
       'session_night_PM', 'seasons_fall', 'seasons_spring', 'seasons_summer',
       'seasons_winter', 'week_weekday', 'week_weekend'],
      dtype='object')
```



```
final_cab_test.columns
```

```
Index(['pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count_1', 'passenger_count_2',
       'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
       'passenger_count_6', 'session_afternoon', 'session_evening',
       'session_morning', 'session_night_AM', 'session_night_PM',
       'seasons_fall', 'seasons_spring', 'seasons_summer', 'seasons_winter',
       'week_weekday', 'week_weekend'],
      dtype='object')
```

3.Feature Engineering for latitude and longitude variable .As we have latitude and longitude data for pickup and dropoff, we will find the distance the cab travelled from pickup and dropoff location.
We will use both haversine and vincenty methods to calculate distance.

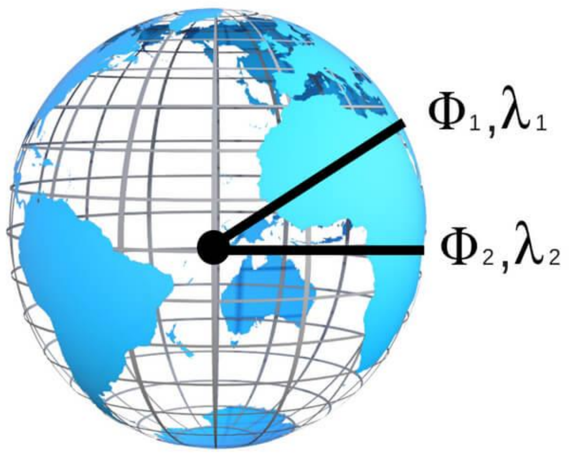
HAVERSINE FORMULA**:**

By using this formula we can easily compute the great-circle distance (The shortest distance between two points on the surface of a Sphere).

Haversine formula mostly used to calculate distances for spherical shape.

Formula:

haversine (d/r) = haversine ($\Phi 2 - \Phi 1$) + cos($\Phi 1$)cos($\Phi 2$)haversine($\lambda 2 - \lambda 1$)



Where d is the distance between two points with longitude and latitude (λ, Φ) and r is the radius of the earth.

VINCENTY FORMULA :

The formulae were developed by Thaddeus Vincenty (1975a) for calculating geodesic distances between a pair of latitude/longitude points on an ellipsoidal model of the Earth.

Unlike the Haversine method for calculating distance on a sphere, these formulae are an iterative method and assume the Earth is an ellipsoid.

This formula include a direct and an inverse method where:

1. Direct Method: It computes the location of a point that is a given distance and azimuth from another point
2. Inverse Method: It computes the geographical distance and azimuth between two given points.

For eg. In the Inverse Method outputs λ after assigning several constants including the length of the semi-major axis, length of the semi-minor axis, flattening, latitude coordinates, reduced latitudes, etc. The aim of the method is to minimise the value of the output λ (i.e. when the results converge to a desired degree of accuracy):

$$\sin\sigma = \sqrt{(\cos U_2 \sin\lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos\lambda)^2}$$

$$\cos\sigma = \sin U_1 \sin U_2 + \cos U_1 \cos U_2 \cos\lambda$$

$$\sigma = \arctan\frac{\sin\sigma}{\cos\sigma}$$

$$\sin\alpha = \frac{\cos U_1 \cos U_2 \sin\lambda}{\sin\sigma}$$

$$\cos^2\alpha = 1 - \sin^2\alpha$$

$$\cos(2\sigma_m) = \cos\sigma - \frac{2\sin U_1 sinU_2}{\cos^2\alpha}$$

$$C = \frac{f}{16}\cos^2\alpha\left[4 + f\left(4 - 3\cos^2\alpha\right)\right]$$

$$\lambda = L + (1 - C) f \sin\alpha\left\{\sigma + C\sin\sigma\left[\cos(2\sigma_m) + C\cos\sigma\left(-1 + 2\cos^2(2\sigma_m)\right)\right]\right\}$$

When the difference between the current value of λ and the value of λ from the previous iteration is less than the convergence tolerance then the final stage of the Inverse Method can be executed:

$$u^2 = \cos^2\alpha\frac{a^2 - b^2}{b^2}$$

$$A = 1 + \frac{u^2}{16384}\left\{4096 + u^2\left[-768 + u^2\left(320 - 175u^2\right)\right]\right\}$$

$$B = \frac{u^2}{1024}\left\{256 + u^2\left[-128 + u^2\left(74 - 47u^2\right)\right]\right\}$$

$$\Delta\sigma = B\sin\sigma\left\{\cos(2\sigma_m) + \frac{1}{4}B\left[\cos\sigma\left(-1 + 2\cos^2(2\sigma_m)\right) - \frac{1}{6}B\cos(2\sigma_m)\left(-3 + 4\sin^2\sigma\right)\left(-3 + 4\cos^2(2\sigma_m)\right)\right]\right\}$$

$$s = bA(\sigma - \Delta\sigma)$$

$$\alpha_1 = \arctan\left(\frac{\cos U_2 \sin\lambda}{\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos\lambda}\right)$$

$$\alpha_2 = \arctan\left(\frac{\cos U_1 \sin\lambda}{-\sin U_1 \cos U_2 + \cos U_1 \sin U_2 \cos\lambda}\right)$$

For vincenty, new variable name will be ‘geodesic’. As Vincenty is more accurate than haversine. Also, vincenty is prefered for short distances. Therefore, we will drop great_circle.

Below are the final train and test data columns after feature engineering:

```
final_cab_train.columns
```

```
Index(['fare_amount', 'passenger_count_1.0', 'passenger_count_2.0',
       'passenger_count_3.0', 'passenger_count_4.0', 'passenger_count_5.0',
       'passenger_count_6.0', 'session_afternoon', 'session_evening',
       'session_morning', 'session_night_AM', 'session_night_PM',
       'seasons_fall', 'seasons_spring', 'seasons_summer', 'seasons_winter',
       'week_weekday', 'week_weekend', 'geodesic'],
      dtype='object')
```

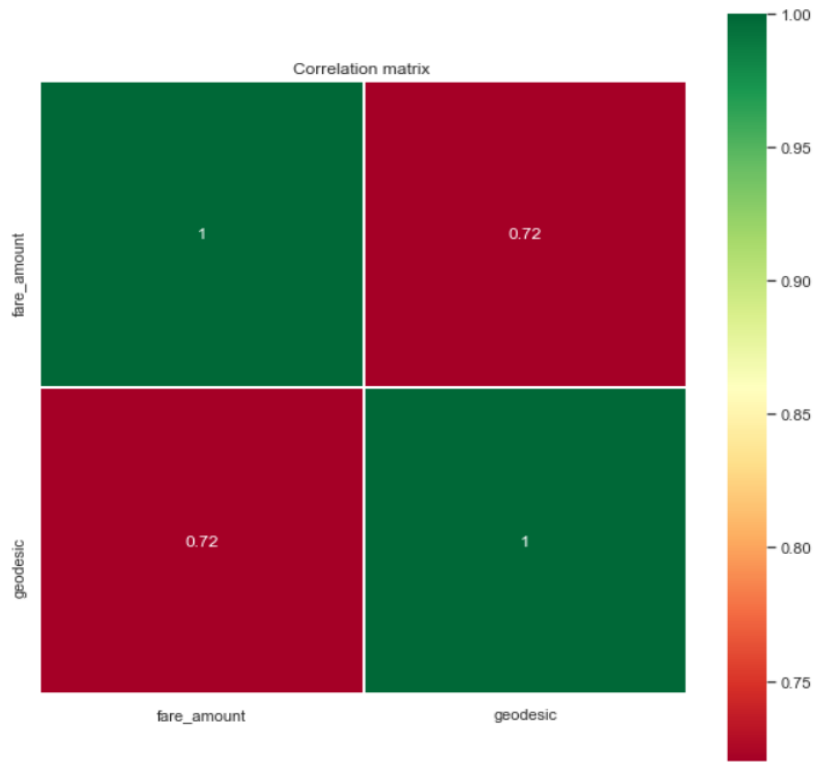
```
final_cab_test.columns
```

```
Index(['passenger_count_1', 'passenger_count_2', 'passenger_count_3',
       'passenger_count_4', 'passenger_count_5', 'passenger_count_6',
       'session_afternoon', 'session_evening', 'session_morning',
       'session_night_AM', 'session_night_PM', 'seasons_fall',
       'seasons_spring', 'seasons_summer', 'seasons_winter', 'week_weekday',
       'week_weekend', 'geodesic'],
      dtype='object')
```

2.7 Feature Selection

In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by some statistical techniques, like we look for features which will not be helpful in predicting the target variables. In this dataset we have to predict the fare_amount. Further below are some types of test involved for feature selection:

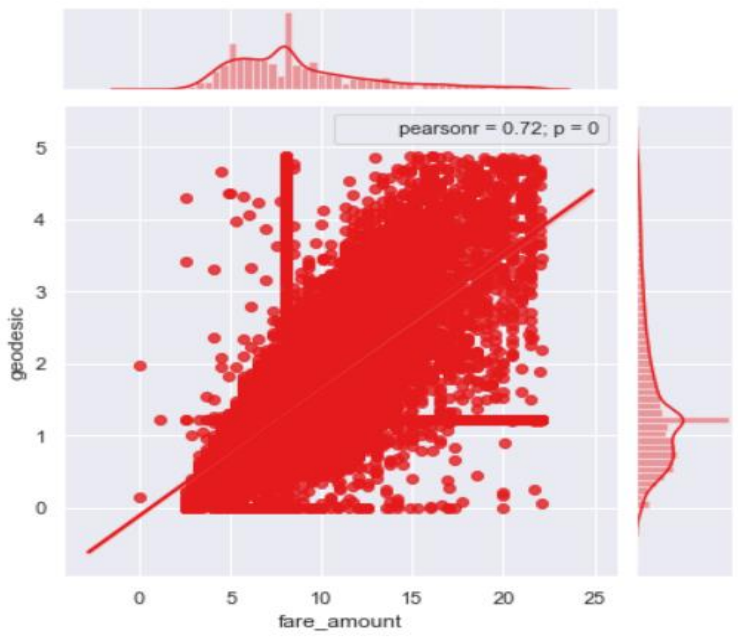
1. **Correlation analysis** – This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot. we can see that in correlation plot faded colour like skin colour indicates that 2 variables are highly correlated with each other. As the colour fades correlation values increases.



From above correlation plot we see that:

- 'fare_amount' and 'geodesic' are very highly correlated with each other.
- As fare_amount is the target variable and ‘geodesic’ is independent variable we will keep ‘geodesic’ because it will help to explain variation in fare_amount.

Jointplot between ‘geodesic’ and ‘fare_amount’:



2. **Chi-Square test of independence** – Unlike correlation analysis we will filter out only categorical variables and pass it to Chi-Square test.Chi-square test compares 2 categorical variables in a contingency table to see if they are related or not.

- Assumption for chi-square test: Dependency between Independent variable and dependent variable should be high and there should be no dependency among independent variables.
- Before proceeding to calculate chi-square statistic, we do the hypothesis testing: Null hypothesis: 2 variables are independent.
  Alternate hypothesis: 2 variables are not independent.
  The interpretation of chi-square test:

For theorical or excel sheet purpose: If chi-square statistics is greater than critical value then reject the null hypothesis saying that 2 variables are dependent and if it's less, then accept the null hypothesis saying that 2 variables are independent.
While programming: If p-value is less than 0.05 then we reject the null hypothesis saying that 2 variables are dependent and if p-value is greater than 0.05 then we accept the null hypothesis saying that 2 variables are independent.
Here we did the test between categorical independent variables pairwise.

- If p-value<0.05 then remove the variable,
- If p-value>0.05 then keep the variable.

After chi-square test we concluded that all the categorical variables are independent. So I have kept all the variables for my model.

3. **Multicollinearity**– In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when our model includes multiple factors that are correlated not just to our response variable, but also to each other.

- Multicollinearity increases the standard errors of the coefficients.
- Increased standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.
- In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant. Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.
- VIF is always greater or equal to 1. If VIF is 1 --- Not correlated to any of the variables. if VIF is between 1-5 --- Moderately correlated. if VIF is above 5 --- Highly correlated. If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.
- And if the VIF goes above 10, we can assume that the regression coefficients are poorly estimated due to multicollinearity.

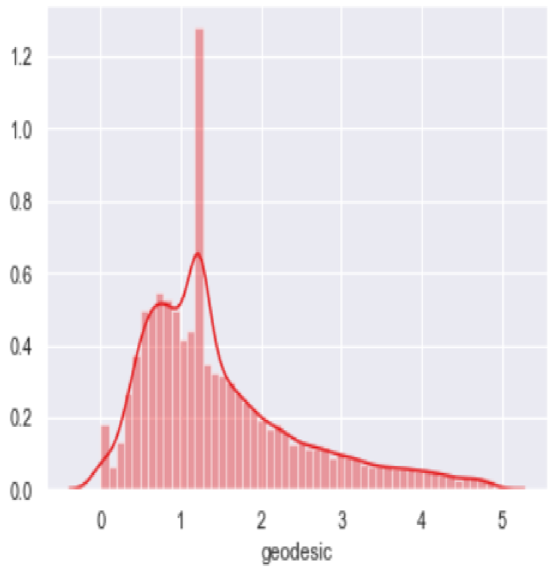
2.8 Feature Scaling

Data Scaling methods are used when we want our variables in data to scaled on common ground. It is performed only on continuous variables.

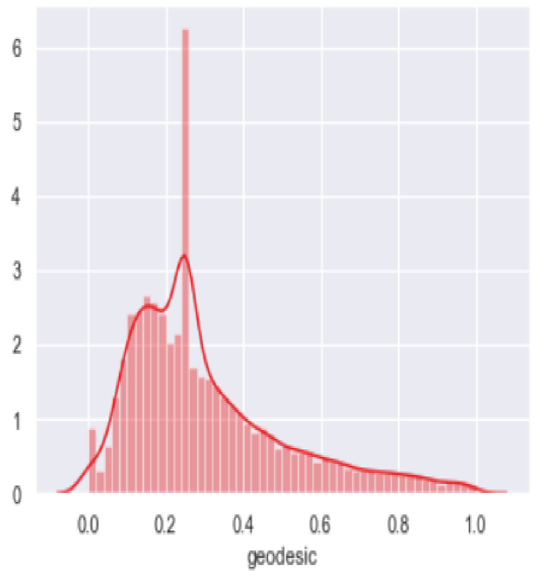
- **Normalization**: Normalization refer to the dividing of a vector by its length. normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalisation of data scales the data to a very small interval, where outliers can be loosed.

- **Standardization**: Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.
  Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric. Also, our independent numerical variable 'geodesic' is not distributed normally so we had chosen normalization over standardization.
  - We have checked variance for each column in dataset before Normalisation - High variance will affect the accuracy of the model. So, we want to normalise that variance. Graphs based on which standardization was chosen: Note: It is performed only on Continuous variables.
  distplot() for 'geodesic' feature

before normalization



after normalization



2.9 Conclusion

After Exploratory data analysis we can conclude below points:

1. We have to predict cab fare amount with the help of given historical data having independent variables like pickup latitude, pickup longitude, dropoff latitude, dropoff longitude, passenger count and pickup_datetime.
2. In univariate analysis we saw the distribution plot of all the variables. And also concluded that in most of the ride the passenger count is 1.
3. In bivariate analysis we saw the relationship between all the independent variables with the target variable.
4. In outlier analysis first we observed the outliers on the basis of general understanding and removed such outliers. Then we observed the outliers with the help of boxplot and further replaced those outliers values with NA and then imputed it with median method.
5. In missing value analysis we imputed the values with median.
6. In feature engineering we created some new variables like month, year, hour, weekday with the help of variable pickup datetime. And further created variables like session ,season, week with the help of hour, month , weekday respectively. Next we created a new variable to get the distance travelled with the help of latitude and longitude data. At last we created dummy variables with the help of one hot encoding method to convert categorical variables to numerical variables.
7. In feature selection we concluded with the help of correlation plot that the only numerical data geodesic is highly correlated with the target variable. We did chi square test to see the relationship between categorical data and concluded that all the categorical data are important.
8. In feature scaling we used normalization to normalize the numerical variable geodesic.
9. After data pre-processing we will go for modelling process.

Chapter 3
Modelling

3.1Training and Testing Sets

There is one final step of data preparation: splitting data into training and testing sets. During training, we let the model 'see' the answers, in this case the actual bike rent count, so it can learn how to predict the bike rent count from the features. We expect there to be some relationship between all the features and the target value, and the model's job is to learn this relationship during training. Then, when it comes time to evaluate the model, we ask it to make predictions on a testing set where it only has access to the features (not the answers) Because we do have the actual answers for the test set, we can compare these predictions to the true value to judge how accurate the model is. Generally, when training a model, we randomly split the data into training and testing sets to get a representation of all data points .I am setting the random state to 101 which means the results will be the same each time I run the split for reproducible results.
For this model I have divided the dataset into train and test part using random sampling. Where train contains 70% data of data set and test contains 30% data and contains 19 variables where 'fare_amount' variable is the target variable.

3.2 Model Selection

Our problem statement wants us to predict the fare_amount. This is a Regression problem.
So, we are going to build regression models on training data and predict it on test data. Model having less error rate and more accuracy will be our final model.
In this project I have built models using 6 Regression Algorithms:

I. Linear Regression
II. Ridge Regression
III. Lasso Regression
IV. Decision Tree
V. Random Forest
VI. Xgboost Regression

I. Linear Regression

It is used to estimate real values (ex- bike rental counts.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation
Y= a*X + b.
In this equation:

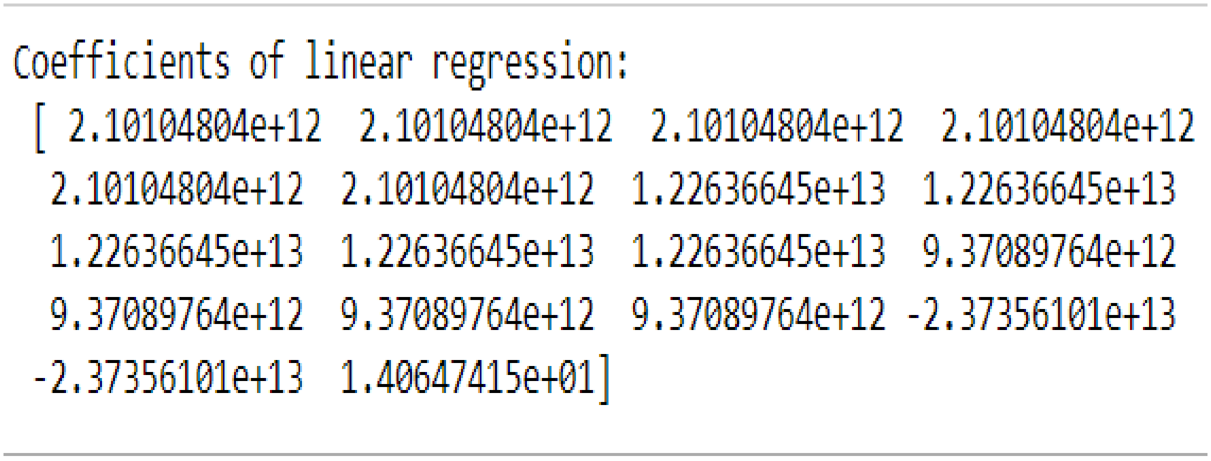
- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.
Linear Regression is mainly of two types:
Simple Linear Regression and Multiple Linear Regression.

Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent variables.
In my modelling I have used multiple linear regression as number of independent variables are more than one.

Code for Model testing and prediction is given in Appendix.

Below are the coefficients obtained in linear regression model.



```
Coefficients of linear regression:
 [ 2.10104804e+12  2.10104804e+12  2.10104804e+12  2.10104804e+12
  2.10104804e+12  2.10104804e+12  1.22636645e+13  1.22636645e+13
  1.22636645e+13  1.22636645e+13  1.22636645e+13  9.37089764e+12
  9.37089764e+12  9.37089764e+12  9.37089764e+12 -2.37356101e+13
 -2.37356101e+13  1.40647415e+01]
```

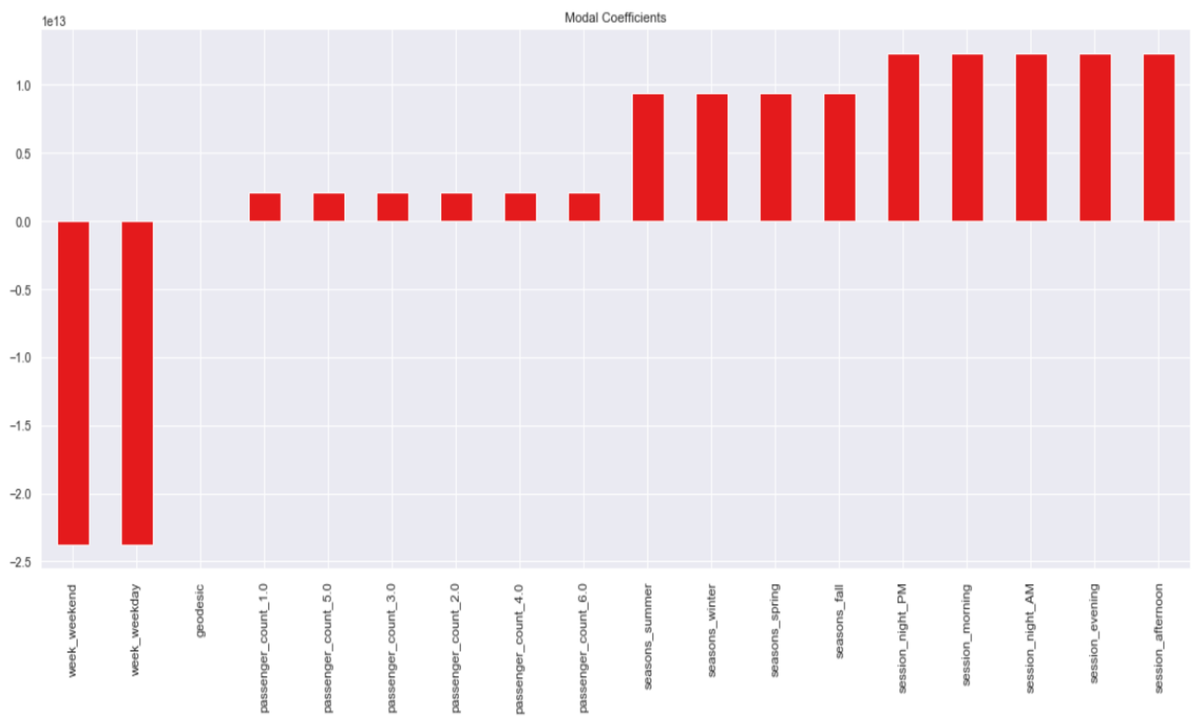


Fig: Visualization of coefficients of multiple linear regression model

We can see that coefficients of some variables are much higher as compared to rest of the coefficients. Therefore the total fare_amount of the cab would be more driven by these higher coefficients features.

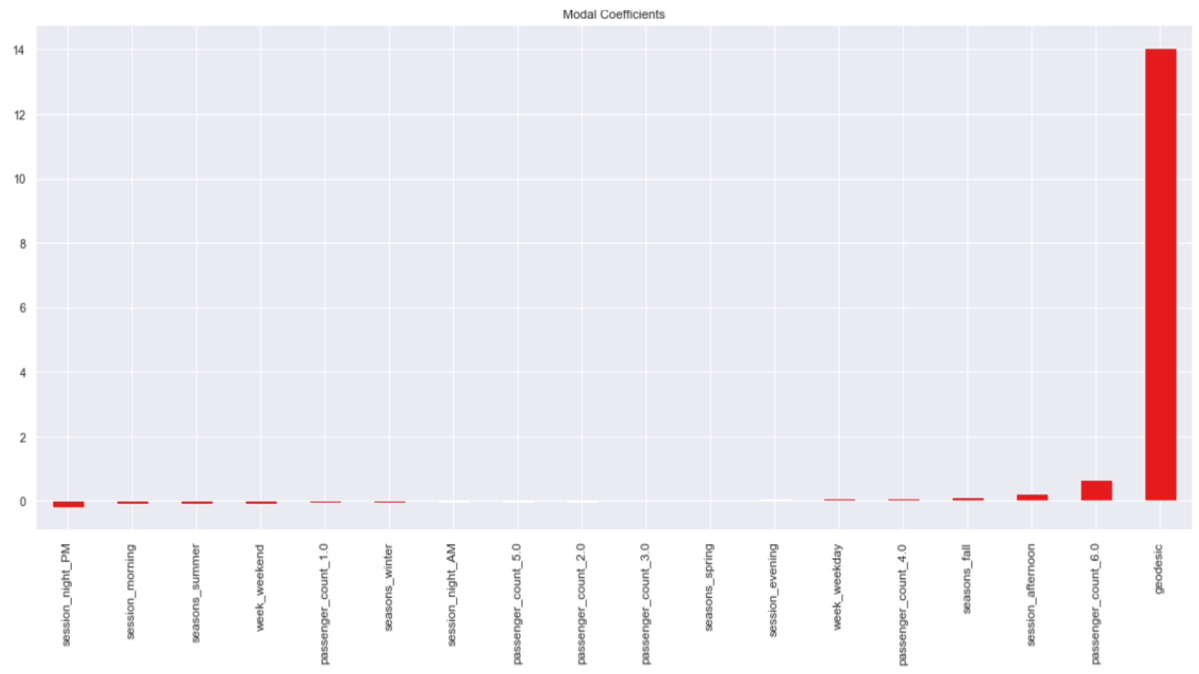
Hence, we will try to reduce the magnitude of coefficients in our model. For this purpose, we have different types of regression techniques which uses regularization to overcome this problem. So let us try them.

II. Ridge Regression

Let us first implement it on our problem and check our results that whether it performs better than our linear regression model.
Code for Model testing and prediction is given in Appendix.
So, we can see that there is a slight improvement in our model because the value of the R-Square has been increased. Note that value of alpha, which is hyperparameter of Ridge, which means that they are not automatically learned by the model instead they have to be set manually.

The coefficients obtained in our ridge model is as below:

```
Coefficients of ridge:
 [-5.85185269e-02 -1.26572567e-02  1.53407728e-02  9.61249971e-02
 -2.35487692e-02  6.75135875e-01  2.16478549e-01  4.39246415e-02
 -9.42572792e-02 -3.86570559e-02 -1.86472551e-01  1.29613641e-01
  2.37430401e-02 -9.38441877e-02 -5.82643483e-02  9.14204250e-02
 -9.14204250e-02  1.40470050e+01]
```



We can see from the above plot that we reduced the magnitude of coefficients in our model.

So, now we have an idea how to implement it but let us take a look at the mathematics side also. Till now our idea was to basically minimize the cost function, such that values predicted are much closer to the desired result.

Now take a look back again at the cost function for ridge regression.

$$\min\left(||Y - X(\theta)||_2^2 + \lambda||\theta||_2^2\right)$$

Here if we notice, we come across an extra term, which is known as the penalty term. λ given here, is actually denoted by alpha parameter in the ridge function. So by changing the values of alpha, we are basically controlling the penalty term. Higher the values of alpha, bigger is the penalty and therefore the magnitude of coefficients are reduced.

Important Points:

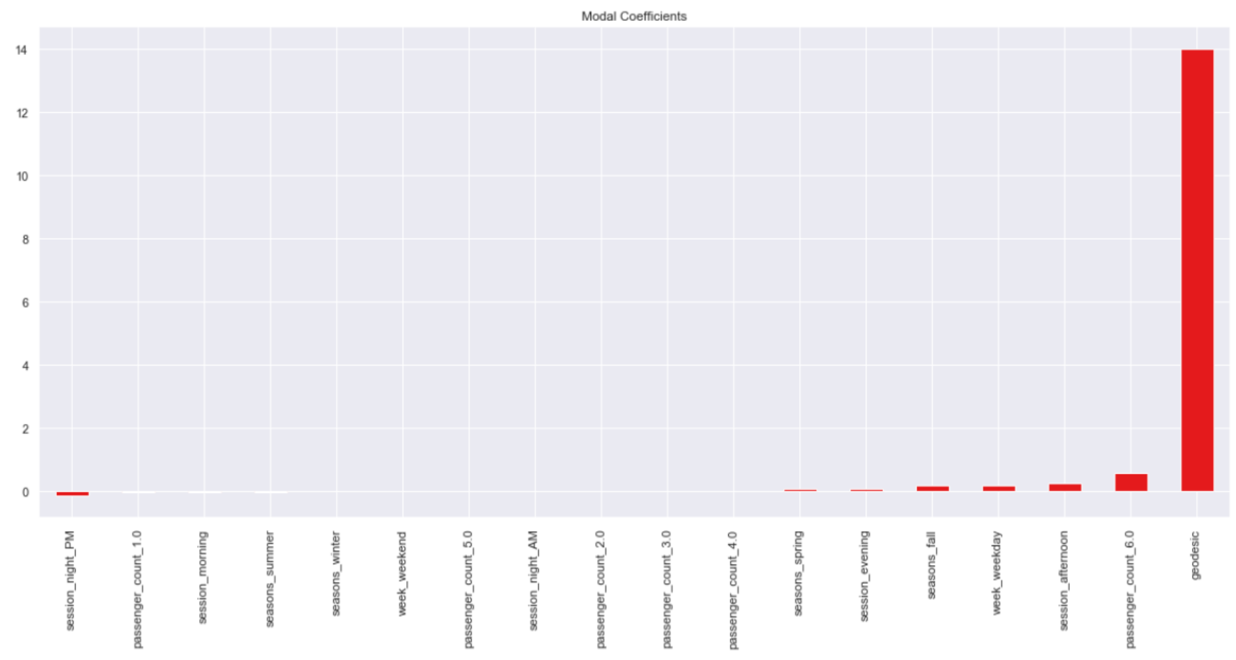
- It shrinks the parameters, therefore it is mostly used to prevent multicollinearity.
- It reduces the model complexity by coefficient shrinkage.

III. Lasso Regression

LASSO (Least Absolute Shrinkage Selector Operator), is quite similar to ridge.
Code for Model testing and prediction is given in Appendix.
As we can see that, both the mse and the value of R-square for our model has been improved. Therefore, lasso model is predicting better than both linear and ridge.

Below are the coefficients obtained in lasso model.

```
Coefficients of lasso:
 [-5.20173068e-02 -0.00000000e+00  0.00000000e+00  4.21259179e-04
 -0.00000000e+00  5.74264128e-01  2.47938216e-01  7.60923502e-02
 -4.59388280e-02  0.00000000e+00 -1.32812689e-01  1.68478268e-01
  6.46989567e-02 -3.67612757e-02 -2.80516316e-03  1.69203715e-01
 -6.43917383e-17  1.40093132e+01]
```



Lasso selects the only some feature while reduces the coefficients of others to zero. This property is known as feature selection and which is absent in case of ridge.

Mathematics behind lasso regression is quiet similar to that of ridge only difference being instead of adding squares of theta, we will add absolute value of theta.

$$\min\left(\left\|Y - X\theta\right\|_2^2 + \lambda\|\theta\|_1\right)$$

Here too, λ is the hypermeter, whose value is equal to the alpha in the Lasso function.

Important Points:

- It is generally used when we have more number of features, because it automatically does feature selection.

IV. Decision Tree

Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

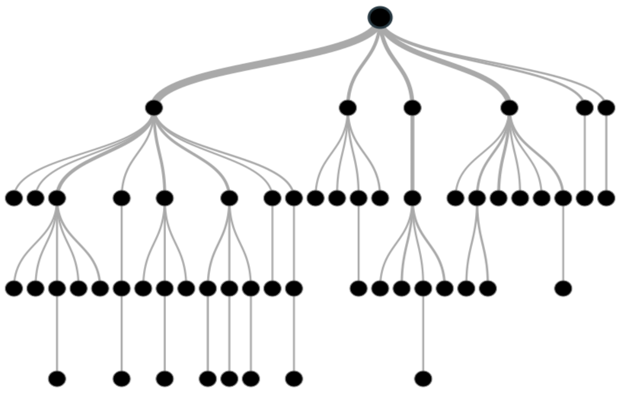
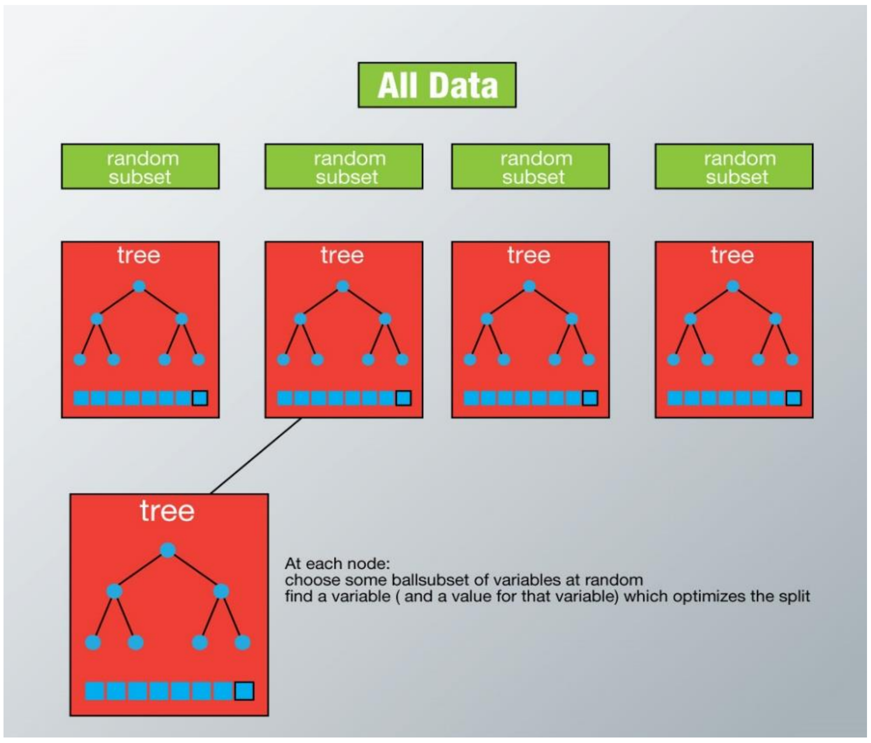


Fig: Tree base model

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

V. Random Forest

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some ballsubset of variables at random
find a variable (and a value for that variable) which optimizes the split

VI. Xgboost Regression

XGBoost is an ensemble learning method. Sometimes, it may not be sufficient to rely upon the results of just one machine learning model. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.

The models that form the ensemble, also known as base learners, could be either from the same learning algorithm or different learning algorithms**.** Bagging and boosting are two widely used ensemble learners**.**

Bagging

While decision trees are one of the most easily interpretable models, they exhibit highly variable behaviour. Consider a single training dataset that we randomly split into two parts. Now, let’s use each part to train a decision tree in order to obtain two models.

When we fit both these models, they would yield different results. Decision trees are said to be associated with high variance due to this behaviour. Bagging or boosting aggregation helps to reduce the variance in any learner. Several decision trees which are generated in parallel, form the base learners of bagging technique. Data sampled with replacement is fed to these learners for training. The final prediction is the averaged output from all the learners.

Boosting

In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these weak learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance.

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm.

The XGBoost Advantage:

I've always admired the boosting capabilities that this algorithm infuses in a predictive model. When I explored more about its performance and science behind its high accuracy, I discovered many advantages:

1. Regularization:

Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting. In fact, XGBoost is also known as a 'regularized boosting' technique.

2. Parallel Processing:

XGBoost implements parallel processing and is blazingly faster as compared to GBM.

3. High Flexibility:

XGBoost allows users to define custom optimization objectives and evaluation criteria.This adds a whole new dimension to the model and there is no limit to what we can do.

4. Handling Missing Values

XGBoost has an in-built routine to handle missing values. The user is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.

5. Tree Pruning:

A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XGBoost on the other hand make splits upto the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.Another advantage is that sometimes a split of negative loss say

-2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper and it will see a combined effect of +8 of the split and keep both.

6. Built-in Cross-Validation

XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.This is unlike GBM where we have to run a grid-search and only a limited values can be tested.

7. Continue on Existing Model

User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications. GBM implementation of sklearn also has this feature so they are even on this point.

3.3 Hyperparameter Optimization

- To find the optimal hyperparameter we have used sklearn. model_selection.GridSearchCV. and sklearn.model_selection.RandomizedSearchCV
- GridSearchCV tries all the parameters that we provide it and then returns the best suited parameter for data.
- We gave parameter dictionary to GridSearchCV which contains keys which are parameter names and values are the values of parameters which we want to try for.

Below are best hyperparameter we found for different models:
I. Multiple Linear Regression:
Tuned Decision linear reg Parameters: {'copy_X': True, 'fit_intercept': False}
Best score is 0.5199961309129493

II. Ridge Regression:
Tuned Decision ridge Parameters: {'alpha': 0.0009540954763499944, 'max_iter': 500, 'normal ize': True}
Best score is 0.5202123253724414

III. Lasso Regression:
Tuned Decision lasso Parameters: {'alpha': 0.0020235896477251557, 'max_iter': 500, 'normalize': False}
Best score is 0.5202477706005241

IV. Decision Tree Regression:
Tuned Decision Tree Parameters: {'max_depth': 6, 'min_samples_split': 14}
Best score is 0.549080164010327

V. Random Forest Regression:
Tuned Random Forest Parameters: {'n_estimators': 400, 'min_samples_split': 4, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 11, 'bootstrap': False}
Best score is 0.5459675059361133

VI. Xgboost regression:
Tuned Xgboost Parameters:{Tuned Xgboost Parameters: {'subsample': 0.9000000000000001, 'reg_alpha': 0.00014563484775012445, 'n_estimators': 200, 'max_depth': 3, 'learning_rate': 0.6500000000000001, 'colsample_bytree': 0.1, 'colsample_bynode': 0.30000000000000004, 'colsample_bylevel': 0.7000000000000001}
Best score is 0.5573181145905641

3.4 Conclusion

In modelling process we can conclude below points:

1. Since this is a regression problem we have done the modelling with the help of algorithm like linear regression, decision tree and random forest.
2. In linear regression the total fare_amount of the cab was more driven by the higher coefficients features. Hence, we tried to reduce the magnitude of coefficients in our model with the help of ridge and lasso regression model.
3. Further we went for tree based modelling to get more accurate results.
4. In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Hence to improve the accuracy we went for XGBoost method for modelling.
5. Further we found the best hyperparameter(tuned) for all the models.

Chapter 4
Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using Predictive performance as the criteria to compare and evaluate models.
Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

4.1 Error Metrics

Regression Metrics to evaluate our Model.
In this section will review 4 of the most common metrics for evaluating predictions on regression machine learning problems:

1. Mean Absolute Percentage Error (MAPE)
2. Mean Squared Error (MSE)
3. R^2 (R-square)
4. Adjusted R square

4.1.1 Mean Absolute Percentage Error (MAPE)

The mean absolute percentage error (MAPE) is a statistical measure of how accurate a forecast system is. It measures this accuracy as a percentage, and can be calculated as the average absolute percent error for each time period minus actual values divided by actual values. Where At is the actual value and Ft is the forecast value, this is given by:

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

4.1.2 Mean Squared Error (MSE)

It is perhaps the most simple and common metric for regression evaluation, but also probably the least useful. It is defined by the equation

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i is the actual expected output and $\hat{y}_i$ is the model's prediction.

MSE basically measures average squared error of our predictions. For each point, it calculates square difference between the predictions and the target and then average those values.
The higher this value, the worse the model is. It is never negative, since we're squaring the individual prediction-wise errors before summing them, but would be zero for a perfect model

4.1.3 R^2 (R-square)

The R^2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature, this measure is called the coefficient of determination

It determines how much of the total variation in Y (dependent variable) is explained by the variation in X (independent variable). Mathematically, it can be written as:

$$R - Square = 1 - \frac{\sum(Y_{actual} - Y_{predicted})\hat{}2}{\sum(Y_{actual} - Y_{mean})\hat{}2}$$

In general the value of R-square is always between 0 and 1, where 0 means that the model does not model explain any variability in the target variable (Y) and 1 meaning it explains full variability in the target variable.

But actually the maximum value of R^2 is 1 but minimum can be minus infinity.
For example, consider a really crappy model predicting highly negative value for all the observations even though y_actual is positive. In this case, R^2 will be less than 0. This is a highly unlikely scenario but the possibility still exists.

4.1.4 Adjusted R square

The only drawback of R^2 is that if new predictors (X) are added to our model, R^2 only increases or remains constant but it never decreases. We can not judge that by increasing complexity of our model, are we making it more accurate?

That is why, we use "Adjusted R-Square".

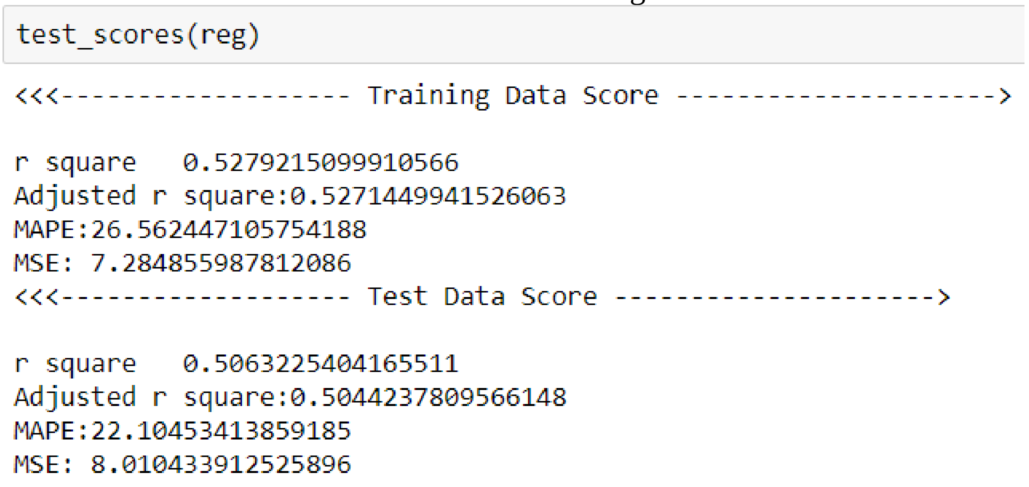
The Adjusted R-Square is the modified form of R-Square that has been adjusted for the number of predictors in the model. It incorporates model's degree of freedom. The adjusted R-Square only increases if the new term improves the model accuracy.

$$R^2 \text{ adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where
R^2 = Sample R square
p = Number of predictors
N = total sample size
Error metrics calculated for our model is as below:

Below are the error metrics obtained with my dataset for model evaluation:

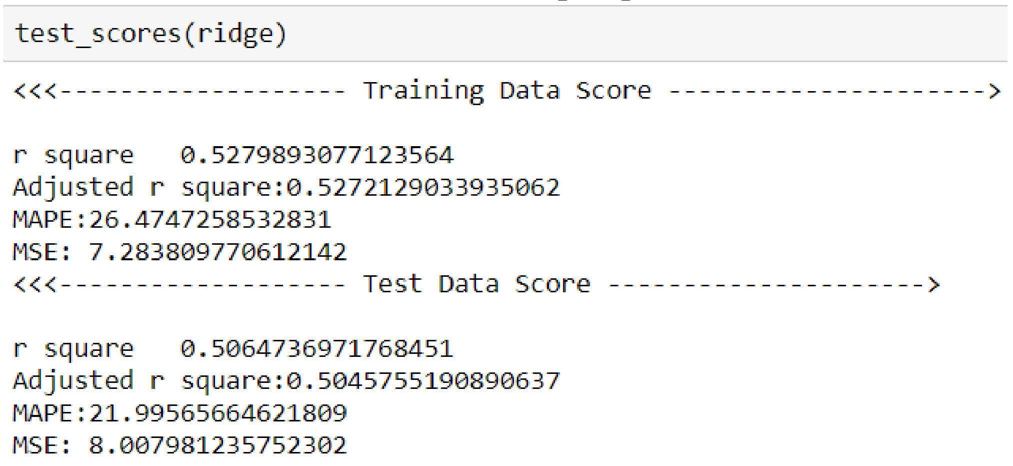
Error metrics for Linear regression



```
test_scores(reg)
```

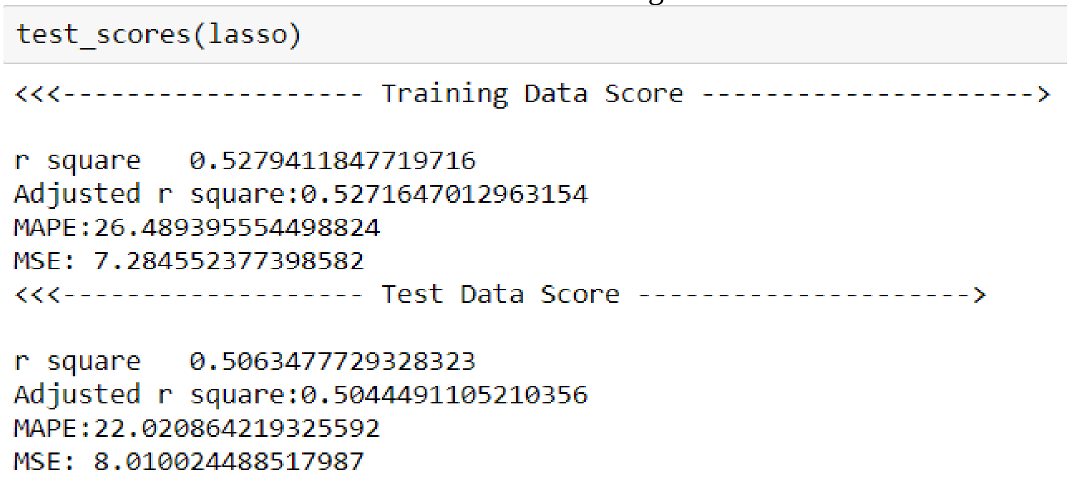
```
<<<------------------- Training Data Score --------------------->

r square   0.5279215099910566
Adjusted r square:0.5271449941526063
MAPE:26.562447105754188
MSE: 7.284855987812086
<<<------------------- Test Data Score --------------------->

r square   0.5063225404165511
Adjusted r square:0.5044237809566148
MAPE:22.10453413859185
MSE: 8.010433912525896
```

Error metrics for Ridge regression



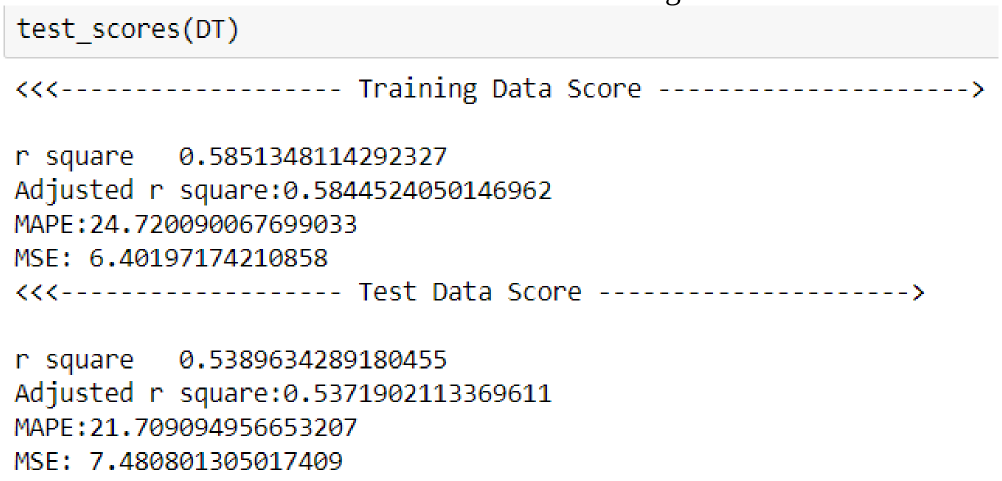
```
test_scores(ridge)
```

```
<<<------------------- Training Data Score --------------------->

r square   0.5279893077123564
Adjusted r square:0.5272129033935062
MAPE:26.4747258532831
MSE: 7.283809770612142
<<<------------------- Test Data Score --------------------->

r square   0.5064736971768451
Adjusted r square:0.5045755190890637
MAPE:21.99565664621809
MSE: 8.007981235752302
```


Error metrics for Lasso regression

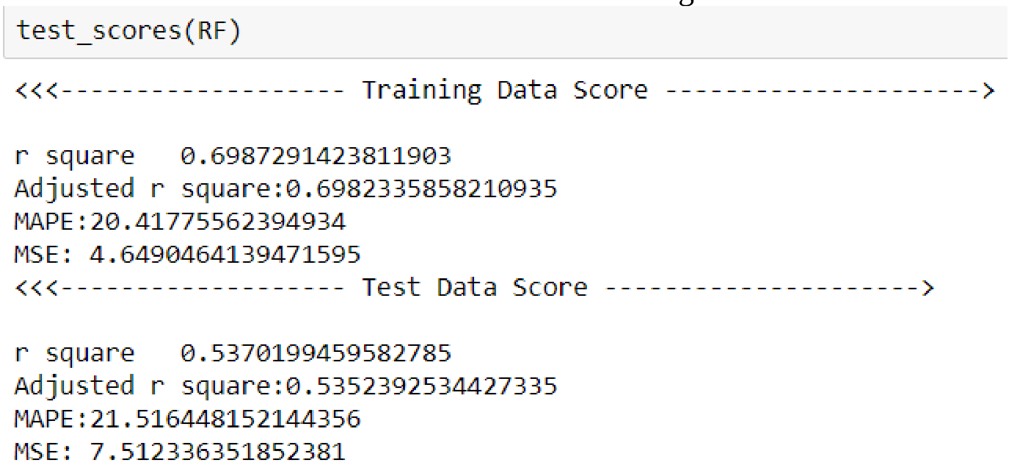
```
test_scores(lasso)
```

```
<<<------------------- Training Data Score --------------------->

r square   0.5279411847719716
Adjusted r square:0.5271647012963154
MAPE:26.489395554498824
MSE: 7.284552377398582
<<<------------------- Test Data Score --------------------->

r square   0.5063477729328323
Adjusted r square:0.5044491105210356
MAPE:22.020864219325592
MSE: 8.010024488517987
```


Error metrics for Decision Tree regression

```
test_scores(DT)
```

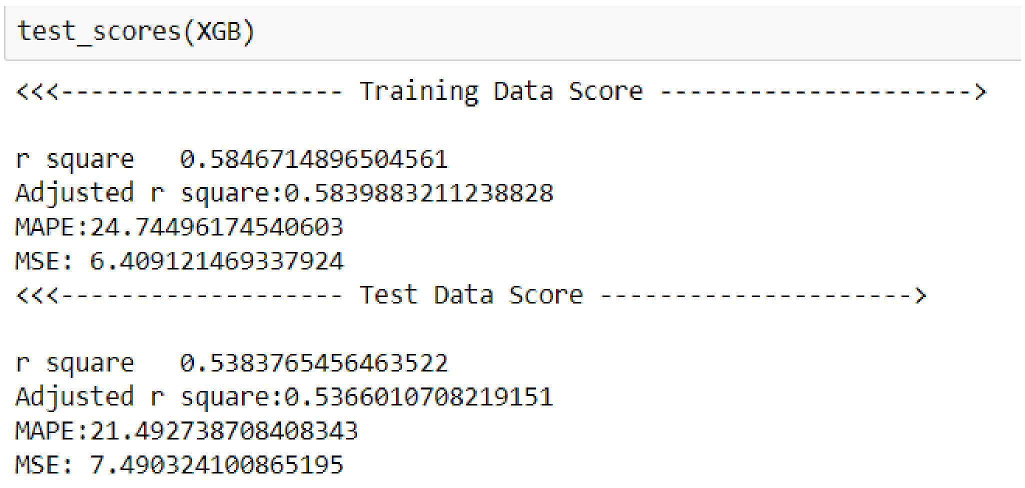
```
<<<------------------- Training Data Score --------------------->

r square   0.5851348114292327
Adjusted r square:0.5844524050146962
MAPE:24.720090067699033
MSE: 6.40197174210858
<<<------------------- Test Data Score --------------------->

r square   0.5389634289180455
Adjusted r square:0.5371902113369611
MAPE:21.709094956653207
MSE: 7.480801305017409
```


Error metrics for Random Forest regression

```
test_scores(RF)
```

```
<<<------------------- Training Data Score --------------------->

r square   0.6987291423811903
Adjusted r square:0.6982335858210935
MAPE:20.41775562394934
MSE: 4.6490464139471595
<<<------------------- Test Data Score --------------------->

r square   0.5370199459582785
Adjusted r square:0.5352392534427335
MAPE:21.516448152144356
MSE: 7.512336351852381
```

Error metrics for XG Boost



```
test_scores(XGB)
```

```
<<<------------------- Training Data Score --------------------->

r square   0.5846714896504561
Adjusted r square:0.5839883211238828
MAPE:24.74496174540603
MSE: 6.409121469337924
<<<------------------- Test Data Score --------------------->

r square   0.5383765456463522
Adjusted r square:0.5366010708219151
MAPE:21.492738708408343
MSE: 7.490324100865195
```

4.2 Conclusion

In Model evaluation we concluded below points:

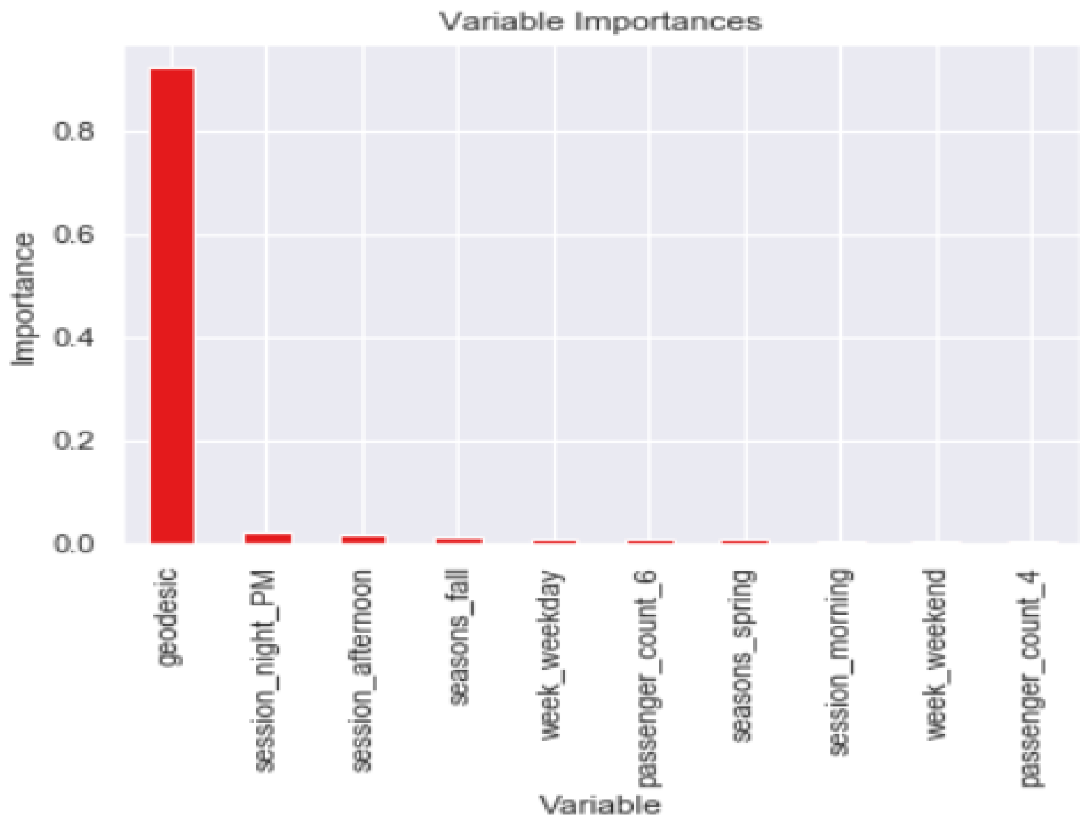
1. Since this is a regression problem we went for regression metrics like MAPE, MSE, r-square and adjusted r-square.
2. After evaluating all the error metrics we concluded that the error metrics of XGBoost model is the best fit as XGBoost model has minimum error (MAPE,MAE) and high accuracy(r-square, adjusted r-square) compared to other models.

Chapter 5
Final model

- I have created a model on entire training dataset.
- Then saved the model for later use.

I have trained a Xgboost model on entire training dataset and used that model to predict on test data. Also, I have saved model for later use.
Training Data Score for our final XGBoost model is:

r square 0.5746852674350544
Adjusted r square:0.574195837363058
MAPE:23.543908531465544
MSE: 6.664698031743432



Chapter 6
Model Deployment

Before running and deploying the code we need to create a virtual environment.

General understanding of Project environments

- The best practice to perform any project is to create a virtual environment for each project.
- The main reason behind creation of virtual environment is –” A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated python virtual environments for them.”
- To better understand above example, consider cricket stadiums. Each stadium has different conditions and environments and players have to adapt to that condition in order to execute their game. Same applies to python codes.
- Dependencies in above statement could be anything, starting from software versions to packages required to execute the project. Each project varies with dependencies i.e. package version or list of packages required.

Model deployment pre-requisites

- Create a pickle file of your final model in your notebook and save it to your project folder.
- Use flask to create an API which will access this pickle file.
- Download and install Postman application tool to integrate your flask API.
- Postman application will give you a feel of how end user will be seeing the outcome of your model.
- Main objective of any machine learning model is to solve business problem and solution should be deployed so that end-non-technical users can use the final ML solution.

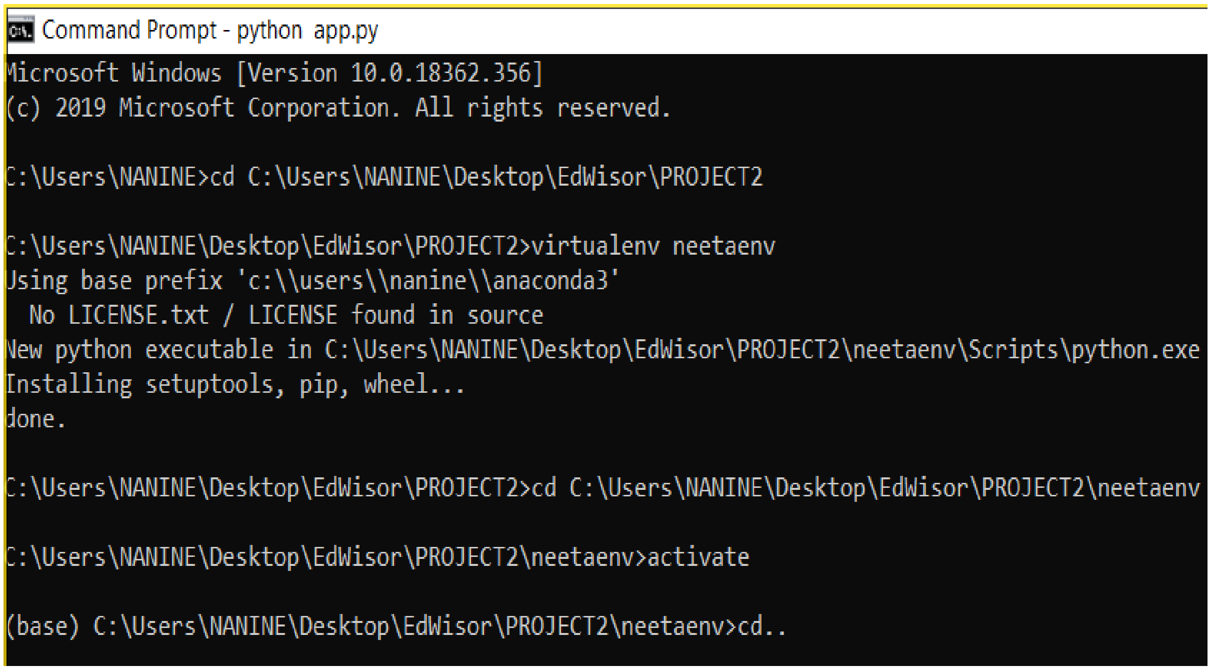
Flask API code- It’s a simple python code

```
from flask import Flask, request
import pandas as pd
import json
import pickle

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def apicall():
    """API Call
    Pandas dataframe (sent as a payload) from API Call
    """
    test_json = request.get_json()
    print(test_json)
```

```
  #import pdb; pdb.set_trace()
  test = pd.DataFrame(test_json,index=[0])
  result = model.predict(test)
  print(result)
  return json.dumps(result.tolist())

if __name__ == '__main__':
  filename = 'cab_fare_xgboost_model.pkl'
  model = pickle.load(open(filename, 'rb'))
  app.run()
```

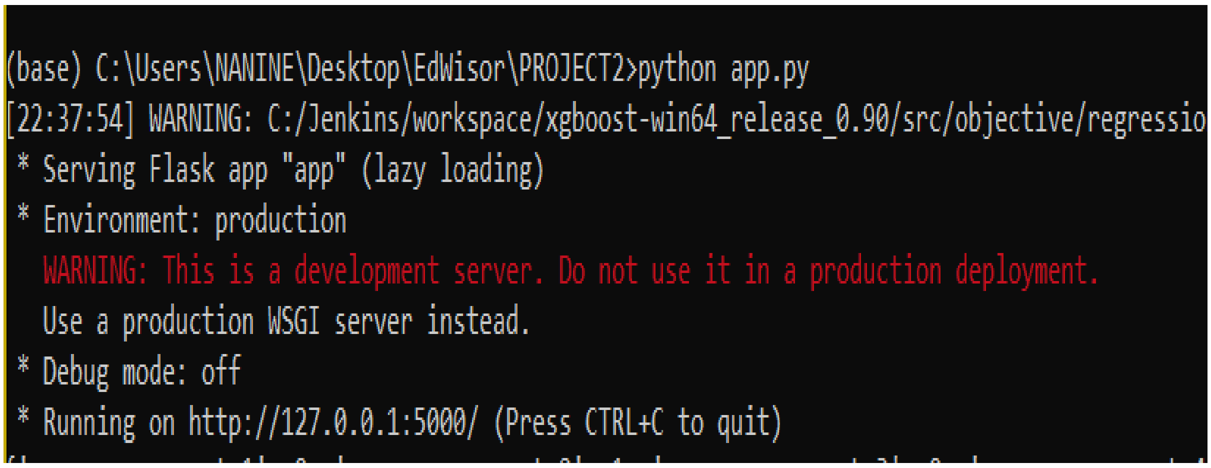
Steps to deploy the model using FLASK, pickle file and Postman application

- Use terminal/cmd prompt. Create and activate virtual environment.
- Run your flask_file_name.py in your virtual environment using the code python flask_file.py
- Previous command will execute the model stored in pickle file and return a generic URL.
- Below is the screenshot of command prompt where I have created the virtual environment and generated a URL.



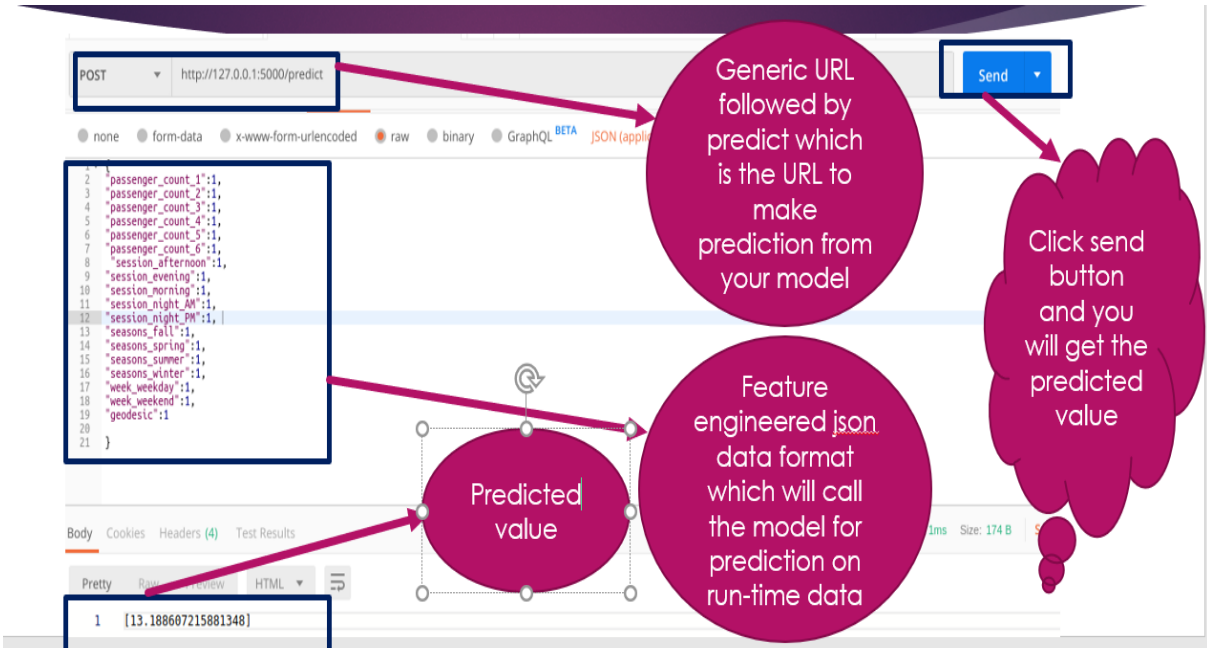
Command Prompt - python app.py

```
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

:\Users\NANINE>cd C:\Users\NANINE\Desktop\EdWisor\PROJECT2

:\Users\NANINE\Desktop\EdWisor\PROJECT2>virtualenv neetaenv
Jsing base prefix 'c:\\users\\nanine\\anaconda3'
  No LICENSE.txt / LICENSE found in source
New python executable in C:\Users\NANINE\Desktop\EdWisor\PROJECT2\neetaenv\Scripts\python.exe
Installing setuptools, pip, wheel...
done.

:\Users\NANINE\Desktop\EdWisor\PROJECT2>cd C:\Users\NANINE\Desktop\EdWisor\PROJECT2\neetaenv

:\Users\NANINE\Desktop\EdWisor\PROJECT2\neetaenv>activate

(base) C:\Users\NANINE\Desktop\EdWisor\PROJECT2\neetaenv>cd..
```



```
(base) C:\Users\NANINE\Desktop\EdWisor\PROJECT2>python app.py
[22:37:54] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regressio
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Copy that URL and go to POSTMAN application.
- Select POST as method in POSTMAN application provide your generic URL appended with /predict. /predict is your application URL present in flask python file to get prediction out of your model.
- Don't forget to send the data in json format because you have done the feature engineering and your model has no longer understanding of original data. Your model was trained with feature engineering data.
- Below is the screenshot of postman application where I have posted the URL and obtained the predicted value.



```
POST    http://127.0.0.1:5000/predict
```

```
{
"passenger_count_1":1,
"passenger_count_2":1,
"passenger_count_3":1,
"passenger_count_4":1,
"passenger_count_5":1,
"passenger_count_6":1,
 "session_afternoon":1,
"session_evening":1,
"session_morning":1,
"session_night_AM":1,
"session_night_PM":1,
"seasons_fall":1,
"seasons_spring":1,
"seasons_summer":1,
"seasons_winter":1,
"week_weekday":1,
"week_weekend":1,
"geodesic":1

}
```

```
1    [13.188607215881348]
```

Send
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL BETA
JSON (appli
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
HTML
1ms
Size: 174 B
Generic URL followed by predict which is the URL to make prediction from your model
Click send button and you will get the predicted value
Predicted value
Feature engineered json data format which will call the model for prediction on run-time data

Appendix A- Python code

```
#LOAD LIBRARIES
# loading the required libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
from geopy.distance import geodesic
from geopy.distance import great_circle
from scipy.stats import chi2_contingency
import statsmodels.api as sm
from statsmodels.formula.api import ols
from patsy import dmatrices
%matplotlib inline
#libraries used for modelling and evaluation part
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
#pip install xgboost
from xgboost import XGBRegressor
import xgboost as xgb

#set working directory
os.chdir("C:/Users/NANINE/Desktop/EdWisor/PROJECT2")

#read the dataset
cab_train = pd.read_csv('train_cab.csv',dtype={'fare_amount':np.float64},na_values={'fare_a
mount':'430-'})
cab_test = pd.read_csv('test.csv')
data=[cab_train,cab_test]
for i in data:
i['pickup_datetime']  = pd.to_datetime(i['pickup_datetime'],errors='coerce')
```

```
cab_train.head(5)

# EXPLORATORY DATA ANALYSIS(EDA)
#1) variable identification
cab_train.info()
cab_train.dtypes
cab_train.head()
cab_train.shape
cab_test.head(5)
cab_test.info()
cab_test.dtypes
cab_test.describe()
cab_train.describe()
#we will convert passenger_count into a categorical variable because passenger_count is not
a continuous variable.
#passenger_count cannot take continuous values. and also they are limited in number if its a
cab.
cat_var=['passenger_count']
num_var=['fare_amount','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_lati
tude']

#2) univariate analysis
sns.set(style='darkgrid',palette='Set1')
plt.figure(figsize=(20,20))
plt.subplot(321)
sns.distplot(cab_train['fare_amount'],bins=50)
plt.subplot(322)
sns.distplot(cab_train['pickup_longitude'],bins=50)
plt.subplot(323)
sns.distplot(cab_train['pickup_latitude'],bins=50)
plt.subplot(324)
sns.distplot(cab_train['dropoff_longitude'],bins=50)
plt.subplot(325)
sns.distplot(cab_train['dropoff_latitude'],bins=50)
plt.show()

#3) bivariate analysis
sns.jointplot(x='fare_amount',y='pickup_longitude',data=cab_train,kind = 'reg')
plt.show()
sns.jointplot(x='fare_amount',y='pickup_latitude',data=cab_train,kind = 'reg')
plt.show()
sns.jointplot(x='fare_amount',y='dropoff_longitude',data=cab_train,kind = 'reg')
plt.show()
sns.jointplot(x='fare_amount',y='dropoff_longitude',data=cab_train,kind = 'reg')
plt.show()
sns.pairplot(data=cab_train[num_var],kind='reg',dropna=True)
plt.show()
```

# 4) outlier analysis
#Removing values which are not within desired range(outlier) depending upon basic underst anding of dataset.
#1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -v e ,So we will remove these fields.

```
sum(cab_train['fare_amount']<0)
cab_train[cab_train['fare_amount']<0]
cab_train = cab_train.drop(cab_train[cab_train['fare_amount']<0].index, axis=0)
```

#2.Passenger_count variable (should be between 1 to 6 max)

```
sum(cab_train['passenger_count']>6)
sum(cab_train['passenger_count']<1)
cab_train['passenger_count'].unique()
cab_test['passenger_count'].unique()
array([1, 2, 3, 4, 5, 6], dtype=int64)
```

#We will remove 79 observation which are above 6 ,below 1 and decimal values because a ca b cannot hold these number of
#passengers.

```
cab_train = cab_train.drop(cab_train[cab_train['passenger_count']>6].index, axis=0)
cab_train = cab_train.drop(cab_train[cab_train['passenger_count']<1].index, axis=0)
cab_train = cab_train.drop(cab_train[cab_train['passenger_count']==1.3].index, axis=0)
cab_train['passenger_count'].unique()
array([ 1.,  2.,  3., nan,  6.,  5.,  4.])
sum(cab_train['passenger_count']>6)
0
sum(cab_train['passenger_count']<1)
0
```

#3.Latitudes range from -90 to 90.Longitudes range from -180 to 180. Removing which does not satisfy these ranges

```
print('pickup_longitude above 180={}'.format(sum(cab_train['pickup_longitude']>180)))
print('pickup_longitude below -180={}'.format(sum(cab_train['pickup_longitude']<-180)))
print('pickup_latitude above 90={}'.format(sum(cab_train['pickup_latitude']>90)))
print('pickup_latitude below -90={}'.format(sum(cab_train['pickup_latitude']<-90)))
print('dropoff_longitude above 180={}'.format(sum(cab_train['dropoff_longitude']>180)))
print('dropoff_longitude below -180={}'.format(sum(cab_train['dropoff_longitude']<-180)))
print('dropoff_latitude below -90={}'.format(sum(cab_train['dropoff_latitude']<-90)))
print('dropoff_latitude above 90={}'.format(sum(cab_train['dropoff_latitude']>90)))
pickup_longitude above 180=0
pickup_longitude below -180=0
pickup_latitude above 90=1
pickup_latitude below -90=0
dropoff_longitude above 180=0
dropoff_longitude below -180=0
dropoff_latitude below -90=0
dropoff_latitude above 90=0
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(cab_train[i]==0)))
```

```
pickup_longitude equal to 0=311
pickup_latitude equal to 0=311
dropoff_longitude equal to 0=312
dropoff_latitude equal to 0=310
#remove the outliers like for pickup_lat>90 and values=0
cab_train = cab_train.drop(cab_train[cab_train['pickup_latitude']>90].index, axis=0)
for i in ['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']:
    cab_train = cab_train.drop(cab_train[cab_train[i]==0].index, axis=0)
cab_train.shape
 (15662, 7)
#So, we lossed 16067-15662=405 observations because of irrelevant values.
#Create dataframe with number of missing value
missing_val = pd.DataFrame(cab_train.isnull().sum())
#Reset index
missing_val = missing_val.reset_index()
missing_val

cab_train['fare_amount'] = cab_train['fare_amount'].fillna(cab_train['fare_amount'].median())
#before missing value analysis lets do outlier analysis
#Outlier Analysis using Boxplot #We Will do Outlier Analysis only on Fare_amount just for n
ow and we will do outlier analysis
#after feature engineering laitudes and longitudes.
cab_train.describe()
#Univariate Boxplots: Boxplots for  target variable.
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=cab_train['fare_amount'],data=cab_train,orient='h')
plt.title('Boxplot of fare_amount')
plt.show()

#Bivariate Boxplots: Boxplot for Numerical Variable Vs Categorical Variable.
plt.figure(figsize=(20,10))
plt.xlim(0,100)
sns.boxplot(x=cab_train['fare_amount'],y=cab_train['passenger_count'],data=cab_train,orient
='h')
plt.title('Boxplot of fare_amount w.r.t passenger_count')
# plt.savefig('Boxplot of fare_amount w.r.t passenger_count.png')
plt.show()

pd.DataFrame(cab_train.isnull().sum())
def outlier_treatment(col):
    ''' calculating outlier indices and replacing them with NA  '''
    #Extract quartiles
    q75, q25 = np.percentile(cab_train[col], [75 ,25])
    print(q75,q25)
    #Calculate IQR
    iqr = q75 - q25
```

```
#Calculate inner and outer fence
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)
print(minimum,maximum)
#Replace with NA
cab_train.loc[cab_train[col] < minimum,col] = np.nan
cab_train.loc[cab_train[col] > maximum,col] = np.nan
outlier_treatment('fare_amount')
cab_train['fare_amount'].describe()
pd.DataFrame(cab_train.isnull().sum())
#check for the best method to fill mising values (mean,median,mode)
# Choosing a random values to replace it as NA
a=cab_train['fare_amount'].loc[1000]
print('fare_amount at loc-1000:{}'.format(a))
# Replacing 1 value with NA
cab_train['fare_amount'].loc[1000] = np.nan
print('Value after replacing with nan:{}'.format(cab_train['fare_amount'].loc[1000]))
# Impute with mean
print('Value if imputed with mean:{}'.format(cab_train['fare_amount'].fillna(cab_train['fare_a
mount'].mean()).loc[1000]))
# Impute with median
print('Value if imputed with median:{}'.format(cab_train['fare_amount'].fillna(cab_train['fare
_amount'].median()).loc[1000]))
fare_amount at loc-1000:7.0
Value after replacing with nan:nan
Value if imputed with mean:8.912721806614034
Value if imputed with median:8.0
#we can see median imputation is best here so lets impute the missing value with median
cab_train['fare_amount'] = cab_train['fare_amount'].fillna(cab_train['fare_amount'].median())
cab_train.describe()
cab_train['passenger_count'] = cab_train['passenger_count'].fillna(cab_train['passenger_count
'].median())
cab_train['passenger_count'].unique()
array([1., 2., 3., 6., 5., 4.])
pd.DataFrame(cab_train.isnull().sum())
cab_train.head()
cab_train=cab_train.dropna()
pd.DataFrame(cab_train.isna().sum())
cab_train['passenger_count'] = cab_train['passenger_count'].apply(np.int64)
cab_train.head()
cab_test.head()
#finally we have no missing values
#Feature Engineering
# 1.Feature Engineering for timestamp variable we will derive new features from pickup_dat
etime variable
#new features will be year,month,day_of_week,hour
data = [cab_train,cab_test]
```

```
for i in data:
    i["year"] = i["pickup_datetime"].apply(lambda row: row.year)
    i["month"] = i["pickup_datetime"].apply(lambda row: row.month)
    i["day_of_week"] = i["pickup_datetime"].apply(lambda row: row.dayofweek)
    i["hour"] = i["pickup_datetime"].apply(lambda row: row.hour)
plt.figure(figsize=(8,5))
sns.countplot(cab_train['year'])
plt.figure(figsize=(8,5))
sns.countplot(cab_train['month'])
plt.figure(figsize=(8,5))
sns.countplot(cab_train['day_of_week'])
plt.figure(figsize=(8,5))
sns.countplot(cab_train['hour'])
sns.barplot(x='day_of_week',y='fare_amount',data=cab_train)
sns.barplot(x='year',y='fare_amount',data=cab_train)
sns.barplot(x='month',y='fare_amount',data=cab_train)
sns.barplot(x='hour',y='fare_amount',data=cab_train)

#2 Now we will use month,day_of_week,hour to derive new features like sessions in a day,sea
sons in a year,week:weekend/weekday
def f(x):
    ''' for sessions in a day using hour column '''
    if (x >=5) and (x <= 11):
        return 'morning'
    elif (x >=12) and (x <=16 ):
        return 'afternoon'
    elif (x >= 17) and (x <= 20):
        return'evening'
    elif (x >=21) and (x <= 23) :
        return 'night_PM'
    elif (x >=0) and (x <=4):
        return'night_AM'
def g(x):
    ''' for seasons in a year using month column'''
    if (x >=3) and (x <= 5):
        return 'spring'
    elif (x >=6) and (x <=8 ):
        return 'summer'
    elif (x >= 9) and (x <= 11):
        return'fall'
    elif (x >=12)|(x <= 2) :
        return 'winter'
def h(x):
    ''' for week:weekday/weekend in a day_of_week column '''
    if (x >=0) and (x <= 4):
        return 'weekday'
    elif (x >=5) and (x <=6 ):
        return 'weekend'
cab_train['session'] = cab_train['hour'].apply(f)
cab_test['session'] = cab_test['hour'].apply(f)
cab_train['seasons'] = cab_train['month'].apply(g)
cab_test['seasons'] = cab_test['month'].apply(g)
cab_train['week'] =cab_train['day_of_week'].apply(h)
cab_test['week'] = cab_test['day_of_week'].apply(h)
cab_train.head()
cab_test.head()
cab_train=cab_train.drop(['pickup_datetime','year', 'month', 'day_of_week','hour'],axis=1)
cab_test=cab_test.drop(['pickup_datetime','year', 'month','day_of_week','hour'],axis=1)
cab_train.columns
cab_test.columns
sns.barplot(x='session',y='fare_amount',data=cab_train)
sns.barplot(x='seasons',y='fare_amount',data=cab_train)
sns.barplot(x='week',y='fare_amount',data=cab_train)

#dummy variables are created for all the categorical variables
cat_feats=['passenger_count','session','seasons','week']
final_cab_train = pd.get_dummies(cab_train,columns=cat_feats)
final_cab_train.head()
final_cab_train.columns
final_cab_test = pd.get_dummies(cab_test,columns=cat_feats)
final_cab_test.head()
final_cab_test.columns

#3.Feature Engineering for latitude and longitude variable As we have latitude and longitude
data for pickup and dropoff, we
#will find the distance the cab travelled from pickup and dropoff location.
#pip install geopy
# Calculate distance the cab travelled from pickup and dropoff location using great_circle fr
om geopy library
data = [final_cab_train, final_cab_test]
for i in data:
   i['great_circle']=i.apply(lambda x: great_circle((x['pickup_latitude'],x['pickup_longitude'])
, (x['dropoff_latitude'],   x['dropoff_longitude'])).miles, axis=1) i['geodesic']=i.apply(lambda
x: geodesic((x['pickup_latitude'],x['pickup_longitude']), (x['dropoff_latitude'],   x['dropoff_lo
ngitude'])).miles, axis=1)
final_cab_train.head()
final_cab_train.columns
final_cab_test.columns
pd.DataFrame(final_cab_train.isna().sum())
#We will remove the variables which were used to feature engineer new variables
final_cab_train=final_cab_train.drop(['pickup_longitude', 'pickup_latitude',
       'dropoff_longitude','dropoff_latitude','great_circle'],axis=1)
final_cab_test=final_cab_test.drop(['pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude','great_circle' ],axis=1)
```

```
final_cab_train.head()
final_cab_train.columns
final_cab_test.columns
#outlier analysis for geodesic
plt.figure(figsize=(20,5))
sns.boxplot(x=final_cab_train['geodesic'],data=final_cab_train,orient='h')
plt.title('Boxplot of geodesic ')
plt.show()
plt.figure(figsize=(20,5))
plt.xlim(0,100)
sns.boxplot(x=final_cab_train['geodesic'],data=final_cab_train,orient='h')
plt.title('Boxplot of geodesic ')
plt.show()
plt.figure(figsize=(20,5))
sns.boxplot(x=final_cab_test['geodesic'],data=final_cab_test,orient='h')
plt.title('Boxplot of geodesic ')
plt.show()
def outlier_treatment(col):
    ''' calculating outlier indices and replacing them with NA  '''
    #Extract quartiles
    q75, q25 = np.percentile(final_cab_train[col], [75 ,25])
    print(q75,q25)
    #Calculate IQR
    iqr = q75 - q25
    #Calculate inner and outer fence
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)
    print(minimum,maximum)
     #Replace with NA
    final_cab_train.loc[final_cab_train[col] < minimum,col] = np.nan
    final_cab_train.loc[final_cab_train[col] > maximum,col] = np.nan
outlier_treatment('geodesic')
2.4253596312061476 0.7815138509168493
-1.6842548195170979 4.891128301640094
pd.DataFrame(final_cab_train.isnull().sum())
final_cab_train['geodesic'] = final_cab_train['geodesic'].fillna(final_cab_train['geodesic'].med
ian())
pd.DataFrame(final_cab_train.isnull().sum())
#1.Correlation Analysis
final_cab_train.dtypes
cat_var=['passenger_count_1', 'passenger_count_2','passenger_count_3', 'passenger_count_4',
'passenger_count_5','passenger_count_6', 'session_afternoon', 'session_evening','session_mor
ning', 'session_night_AM', 'session_night_PM','seasons_fall', 'seasons_spring', 'seasons_sum
mer', 'seasons_winter','week_weekday', 'week_weekend']
num_var=['fare_amount','geodesic']
# heatmap using correlation matrix
plt.figure(figsize=(10,10))
```

```
sns.heatmap(final_cab_train[num_var].corr(), square=True, cmap='RdYlGn',linewidths=0.5,l
inecolor='w',annot=True)
plt.title('Correlation matrix ')
plt.show()
sns.jointplot(x='fare_amount',y='geodesic',data=final_cab_train,kind = 'reg').annotate(stats.pe
arsonr)
plt.show()
final_cab_train.columns
final_cab_train.dtypes
#loop for chi square values
for i in cat_var:
    for j in cat_var:
        if(i != j):
            chi2, p, dof, ex = chi2_contingency(pd.crosstab(final_cab_train[i],final_cab_train[j]))
            if(p < 0.05):
                print(i,"and",j,"are dependent on each other with",p,'----Remove')
            else:
                print(i,"and",j,"are independent on each other with",p,'----Keep')
final_cab_train.columns
Index(['fare_amount', 'passenger_count_1', 'passenger_count_2',
       'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
       'passenger_count_6', 'session_afternoon', 'session_evening',
       'session_morning', 'session_night_AM', 'session_night_PM',
       'seasons_fall', 'seasons_spring', 'seasons_summer', 'seasons_winter',
       'week_weekday', 'week_weekend', 'geodesic'],
      dtype='object')
#Feature Scaling with normalization
sns.distplot(final_cab_train['geodesic'],bins=50)
#Normalization
final_cab_train['geodesic'] = (final_cab_train['geodesic'] - min(final_cab_train['geodesic']))/(
max(final_cab_train['geodesic']) - min(final_cab_train['geodesic']))
final_cab_test['geodesic'] = (final_cab_test['geodesic'] - min(final_cab_test['geodesic']))/(max
(final_cab_test['geodesic']) - min(final_cab_test['geodesic']))
final_cab_train['geodesic'].var()
0.04151214214402438
sns.distplot(final_cab_train['geodesic'],bins=50)

#MODELLING PART
#Splitting train into train and validation subsets
#X_train y_train--are train subset X_test y_test--are validation subset
from sklearn.model_selection import train_test_split
X = final_cab_train.drop('fare_amount',axis=1)
y = final_cab_train['fare_amount']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
def scores(y, y_):
    print('r square  ', metrics.r2_score(y, y_))
    print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(y, y_))*(len(y)-1)/(len(y)-X_trai
n.shape[1]-1)))
    print('MAPE:{}'.format(np.mean(np.ma.masked_invalid(np.abs((y - y_) / y))*100)))
    print('MSE:', metrics.mean_squared_error(y, y_))
def test_scores(model):
    print('<<<------------------- Training Data Score --------------------->')
    print()
    #Predicting result on Training data
    y_pred = model.predict(X_train)
    scores(y_train,y_pred)
    print('<<<------------------- Test Data Score --------------------->')
    print()
    # Evaluating on Test Set
    y_pred = model.predict(X_test)
    scores(y_test,y_pred)
# Hyperparameter Optimization
# Setup the parameters and distributions to sample from:
param_dist = {'copy_X':[True, False],
          'fit_intercept':[True,False]}
# Instantiate a Linear reg classifier: reg
reg = LinearRegression()

# Instantiate the gridSearchCV object: reg_cv
reg_cv = GridSearchCV(reg, param_dist, cv=5,scoring='r2')
# Fit it to the data
reg_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Decision linear reg Parameters: {}".format(reg_cv.best_params_))
print("Best score is {}".format(reg_cv.best_score_))
Tuned Decision linear reg Parameters: {'copy_X': True, 'fit_intercept': False}
Best score is 0.5199961309129493
reg = LinearRegression(copy_X= True , fit_intercept= False)
#training the model
reg.fit(X_train,y_train)
#predicting on test
pred_LR = reg.predict(X_test)
# The coefficients
print('Coefficients of linear regression: \n', reg.coef_)
Coefficients of linear regression:
 [ 2.10104804e+12  2.10104804e+12  2.10104804e+12  2.10104804e+12
  2.10104804e+12  2.10104804e+12  1.22636645e+13  1.22636645e+13
  1.22636645e+13  1.22636645e+13  1.22636645e+13  9.37089764e+12
  9.37089764e+12  9.37089764e+12  9.37089764e+12 -2.37356101e+13
 -2.37356101e+13  1.40647415e+01]
#checking the magnitude of coefficients graphically
predictors = X_train.columns
coef = pd.Series(reg.coef_,predictors).sort_values()
```

```
coef.plot(kind='bar', title='Modal Coefficients', figsize= (20,8))
from sklearn.model_selection import cross_val_score
# Compute 5-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(reg,X,y,cv=5,scoring='neg_mean_squared_error')
# Print the 5-fold cross-validation scores
print(cv_scores)
print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
[-7.21501423 -7.26636985 -7.23730696 -7.8003281  -8.04261163]
Average 5-Fold CV Score: -7.512326153320194
#lets reduce the magnitude of coefficients that is high in our model #REGULARISATION
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'alpha':np.logspace(-4, 0, 50),'normalize':[True,False],'max_iter':range(500,500
0,500)}
# Instantiate a Decision ridge classifier: ridge
ridge = Ridge()
# Instantiate the gridSearchCV object: ridge_cv
ridge_cv = GridSearchCV(ridge, param_dist, cv=5,scoring='r2')
# Fit it to the data
ridge_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Decision ridge Parameters: {}".format(ridge_cv.best_params_))
print("Best score is {}".format(ridge_cv.best_score_))
Tuned Decision ridge Parameters: {'alpha': 0.0009540954763499944, 'max_iter': 500, 'norma
lize': True}
Best score is 0.5202123253724414
#Training Ridge Regression Model
ridge = Ridge(alpha=0.0009540954763499944,max_iter= 500, normalize=True)
ridge.fit(X_train,y_train)
pred_ridge = ridge.predict(X_test)
# The coefficients
print('Coefficients of ridge: \n', ridge.coef_)
Coefficients of ridge:
predictors = X_train.columns
coef = pd.Series(ridge.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients',figsize=(20,8))
<matplotlib.axes._subplots.AxesSubplot at 0x270181c8c50>
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'alpha':np.logspace(-4, 0, 50),
        'normalize':[True,False],
          'max_iter':range(500,5000,500)}
# Instantiate a Decision lasso classifier: lasso
lasso = Lasso()
# Instantiate the gridSearchCV object: lasso_cv
lasso_cv = GridSearchCV(lasso, param_dist, cv=5,scoring='r2')

# Fit it to the data
lasso_cv.fit(X, y)
```

```
# Print the tuned parameters and score
print("Tuned Decision lasso Parameters: {}".format(lasso_cv.best_params_))
print("Best score is {}".format(lasso_cv.best_score_))
Tuned Decision lasso Parameters: {'alpha': 0.0020235896477251557, 'max_iter': 500, 'normal
ize': False}
Best score is 0.5202477706005241
#Training Lasso Regression Model
lasso = Lasso(alpha=0.0020235896477251557, max_iter= 500, normalize= False)
lasso.fit(X_train,y_train)
pred_LASSO = lasso.predict(X_test)
# The coefficients
print('Coefficients of lasso: \n', lasso.coef_)
#checking the magnitude of coefficients
predictors = X_train.columns
coef = pd.Series(lasso.coef_,predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients',figsize=(20,8))
#scatter plot for actual value and predicted value
plt.scatter(y_test,pred_LR)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
Text(0, 0.5, 'Predicted Y')
plt.scatter(y_test,pred_ridge)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
Text(0, 0.5, 'Predicted Y')
plt.scatter(y_test,pred_LASSO)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
Text(0, 0.5, 'Predicted Y')
sns.distplot((y_test-pred_LR),bins=50);
#Axis labels and title
plt.title('Residual plot for linear reg');
sns.distplot((y_test-pred_ridge),bins=50);
#Axis labels and title
plt.title('Residual plot for ridge');
sns.distplot((y_test-pred_LASSO),bins=50);
#Axis labels and title
plt.title('Residual plot for lasso');

#4th algorithm (Decision Tree Regression)
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'max_depth': range(2,16,2),  'min_samples_split': range(2,16,2)}
# Instantiate a Decision Tree classifier: tree
DT = DecisionTreeRegressor()
# Instantiate the gridSearchCV object: tree_cv
DT_cv = GridSearchCV(DT, param_dist, cv=5)
```

```
# Fit it to the data
DT_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(DT_cv.best_params_))
print("Best score is {}".format(DT_cv.best_score_))
Tuned Decision Tree Parameters: {'max_depth': 6, 'min_samples_split': 14}
Best score is 0.549080164010327
DT = DecisionTreeRegressor(max_depth= 6, min_samples_split=14)
DT.fit(X_train,y_train)
pred_DT = DT.predict(X_test)
print(DT.feature_importances_) #use inbuilt class feature_importances of tree based classifie
rs
#plot graph of feature importances for better visualization
feat_importances = pd.Series(DT.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='bar')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');
plt.show()
[1.54804554e-04 5.50224623e-04 1.76694650e-03 0.00000000e+00
 0.00000000e+00 1.48082994e-03 5.45135529e-03 8.73066819e-04
 1.75677061e-03 6.96131347e-03 2.99723188e-03 4.88907779e-04
 0.00000000e+00 0.00000000e+00 0.00000000e+00 1.77005560e-03
 7.50300376e-04 9.74998193e-01]
# Create the random grid
random_grid = {'n_estimators': range(100,500,100),
               'max_depth': range(5,20,1),
               'min_samples_leaf':range(2,5,1),
              'max_features':['auto','sqrt','log2'],
              'bootstrap': [True, False],
              'min_samples_split': range(2,5,1)}
# Instantiate a Decision Forest classifier: Forest
RF = RandomForestRegressor()
# Instantiate the gridSearchCV object: Forest_cv
RF_cv = RandomizedSearchCV(RF, random_grid, cv=5)
# Fit it to the data
RF_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Random Forest Parameters: {}".format(RF_cv.best_params_))
print("Best score is {}".format(RF_cv.best_score_))
Tuned Random Forest Parameters: {'n_estimators': 400, 'min_samples_split': 2, 'min_samples
_leaf': 4, 'max_features': 'auto', 'max_depth': 7, 'bootstrap': True}
Best score is 0.5629638447670416
RF = RandomForestRegressor(n_estimators=400,min_samples_split=4,min_samples_leaf=2,
max_depth=11,bootstrap=True)
RF.fit(X_train,y_train)
pred_RF = RF.predict(X_test)
```

```
pred_RF
array([14.33813728,  5.66816693,  8.82054115, ...,  7.5575661 ,
        6.3413322 , 11.8057228 ])
print(RF.feature_importances_) #use inbuilt class feature_importances of tree based classifie
rs
#plot graph of feature importances for better visualization
feat_importances = pd.Series(RF.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='bar')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');
plt.show()
[0.00729491 0.00590004 0.00445749 0.00207096 0.00449791 0.00302026
 0.00912825 0.00526199 0.00572262 0.00895506 0.00653032 0.00851441
 0.00674426 0.00579288 0.00759316 0.00629327 0.00619711 0.89602509]
from sklearn.model_selection import cross_val_score
# Compute 5-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(RF,X,y,cv=5,scoring='neg_mean_squared_error')
# Print the 5-fold cross-validation scores
print(cv_scores)
print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
[-6.92263757 -6.70173777 -6.92052252 -7.32980623 -7.26668458]
Average 5-Fold CV Score: -7.028277733426552
#Improving accuracy using XGBOOST Improve Accuracy
data_dmatrix = xgb.DMatrix(data=X,label=y)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test)
dtrain,dtest,data_dmatrix
# Create the random grid
para = {'n_estimators': range(100,500,100),
              'max_depth': range(3,10,1),
        'reg_alpha':np.logspace(-4, 0, 50),
        'subsample': np.arange(0.1,1,0.2),
        'colsample_bytree': np.arange(0.1,1,0.2),
        'colsample_bylevel': np.arange(0.1,1,0.2),
        'colsample_bynode': np.arange(0.1,1,0.2),
       'learning_rate': np.arange(.05, 1, .05)}
# Instantiate a Decision Forest classifier: Forest
XGB = XGBRegressor()
# Instantiate the gridSearchCV object: Forest_cv
xgb_cv = RandomizedSearchCV(XGB, para, cv=5)
# Fit it to the data
xgb_cv.fit(X, y)
# Print the tuned parameters and score
print("Tuned Xgboost Parameters: {}".format(xgb_cv.best_params_))
print("Best score is {}".format(xgb_cv.best_score_))
```

```
XGB = XGBRegressor(subsample=0.9000000000000001,reg_alpha=0.00014563484775012
445,n_estimators=200,max_depth=3,learning_rate=0.6500000000000001, XGB.fit(X_train,y
_train)
pred_XGB = XGB.predict(X_test)
[21:02:20] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/reg
ression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
print(XGB.feature_importances_) #use inbuilt class feature_importances of tree based classif
iers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(XGB.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='bar')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');
plt.show()
#Finalize the best model.Create the model on entire training dataset and save the model for l
ater use
def score(y, y_):
   print('r square  ', metrics.r2_score(y, y_))
   print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(y, y_))*(len(y)-1)/(len(y)-X_trai
n.shape[1]-1)))
   print('MAPE:{}'.format(np.mean(np.ma.masked_invalid(np.abs((y - y_) / y))*100)))
   print('MSE:', metrics.mean_squared_error(y, y_))
def scores(model):
   print('<<<------------------- Training Data Score --------------------->')
   print()
   #Predicting result on Training data
   y_pred = model.predict(X)
   score(y,y_pred)
final_cab_test.columns
Index(['passenger_count_1', 'passenger_count_2', 'passenger_count_3',
       'passenger_count_4', 'passenger_count_5', 'passenger_count_6',
       'session_afternoon', 'session_evening', 'session_morning',
       'session_night_AM', 'session_night_PM', 'seasons_fall',
       'seasons_spring', 'seasons_summer', 'seasons_winter', 'week_weekday',
       'week_weekend', 'geodesic'],
      dtype='object')
final_cab_train.columns
Index(['fare_amount', 'passenger_count_1', 'passenger_count_2',
       'passenger_count_3', 'passenger_count_4', 'passenger_count_5',
       'passenger_count_6', 'session_afternoon', 'session_evening',
       'session_morning', 'session_night_AM', 'session_night_PM',
       'seasons_fall', 'seasons_spring', 'seasons_summer', 'seasons_winter',
       'week_weekday', 'week_weekend', 'geodesic'],
      dtype='object')
final_cab_test.shape
 (9914, 18)
final_cab_train.shape
```

```
 (15661, 19)
final_cab_test.count()
cab=pd.read_csv('test.csv')
test_pickup_datetime=cab['pickup_datetime']
# Instantiate a xgb regressor: xgb
XGB = XGBRegressor(subsample=0.9000000000000001,reg_alpha=0.00014563484775012
445,n_estimators=200,max_depth=3,learning_rate=0.6500000000000001,colsample_bytree=
0.1,colsample_bynode=0.30000000000000004,colsample_bylevel=0.7000000000000001)
# Fit the regressor to the data
XGB.fit(X,y)
scores(XGB)
print(XGB.feature_importances_) #use inbuilt class feature_importances of tree based classif
iers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(XGB.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='bar')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable Importances');
plt.show()
# Predictions
pred = XGB.predict(final_cab_test)
#save the model
results_wrt_date = pd.DataFrame({"pickup_datetime":test_pickup_datetime,"fare_amount" :
pred})
results_wrt_date.to_csv("predictions_xgboost.csv",index=False)
results_wrt_date
#for model deployment
from sklearn.externals import joblib
# Save the model as a pickle in a file
joblib.dump(XGB, 'cab_fare_xgboost_model.pkl')
```

Appendix B- (R code)

```
rm(list = ls())
setwd("C:/Users/NANINE/Desktop/EdWisor/PROJECT2")
# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
      "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats')
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)

# loading datasets
cab_train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
cab_test = read.csv("test.csv")
cab_test_pickup_datetime = cab_test["pickup_datetime"]
# Structure of data
str(cab_train)
str(cab_test)
summary(cab_train)
summary(cab_test)
head(cab_train,5)
head(cab_test,5)

# Exploratory Data Analysis
# Changing the data types of variables
cab_train$fare_amount = as.numeric(as.character(cab_train$fare_amount))
cab_train$passenger_count=round(cab_train$passenger_count)

#OUTLIER ANALYSIS
# Removing values which are not within desired range(outlier) depending upon basic
understanding of dataset.

# 1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -
ve and also cannot be 0. So we will remove these fields.
cab_train[which(cab_train$fare_amount < 1 ),]
nrow(cab_train[which(cab_train$fare_amount < 1 ),])
cab_train = cab_train[-which(cab_train$fare_amount < 1 ),]

#2.Passenger_count variable
for (i in seq(4,11,by=1))
  {  print(paste('passenger_count above ' ,i,nrow(cab_train[which(cab_train$passenger_count
> i ),])))}
# so 20 observations of passenger_count is consistenly above from 6,7,8,9,10
passenger_counts, let's check them.
```

```
cab_train[which(cab_train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
cab_train[which(cab_train$passenger_count <1 ),]
nrow(cab_train[which(cab_train$passenger_count <1 ),])
nrow(cab_train[which(cab_train$passenger_count >6 ),])
# We will remove these 58 observations and 20 observation which are above 6 value because
a cab cannot hold these number of passengers.
cab_train = cab_train[-which(cab_train$passenger_count < 1 ),]
cab_train = cab_train[-which(cab_train$passenger_count > 6),]
# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does
not satisfy these ranges
print(paste('pickup_longitude above 180=',nrow(cab_train[which(cab_train$pickup_longitude
>180 ),])))
print(paste('pickup_longitude above -
180=',nrow(cab_train[which(cab_train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(cab_train[which(cab_train$pickup_latitude >
90 ),])))
print(paste('pickup_latitude above -90=',nrow(cab_train[which(cab_train$pickup_latitude < -
90 ),])))
print(paste('dropoff_longitude above
180=',nrow(cab_train[which(cab_train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -
180=',nrow(cab_train[which(cab_train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(cab_train[which(cab_train$dropoff_latitude <
-90 ),])))
print(paste('dropoff_latitude above 90=',nrow(cab_train[which(cab_train$dropoff_latitude >
90 ),])))
# There's only one outlier which is in variable pickup_latitude.So we will remove it with nan.
# Also we will see if there are any values equal to 0.
nrow(cab_train[which(cab_train$pickup_longitude == 0 ),])
nrow(cab_train[which(cab_train$pickup_latitude == 0 ),])
nrow(cab_train[which(cab_train$dropoff_longitude == 0 ),])
nrow(cab_train[which(cab_train$pickup_latitude == 0 ),])
# there are values which are equal to 0. we will remove them.
cab_train = cab_train[-which(cab_train$pickup_latitude > 90),]
cab_train = cab_train[-which(cab_train$pickup_longitude == 0),]
cab_train = cab_train[-which(cab_train$dropoff_longitude == 0),]

#MISSING VALUE ANALYSIS
missing_val = data.frame(apply(cab_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$Missing_percentage/nrow(cab_train)) *100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val
unique(cab_train$passenger_count)
unique(cab_test$passenger_count)
cab_train[,'passenger_count'] = factor(cab_train[,'passenger_count'], labels=(1:6))
```

```
cab_test[,'passenger_count'] = factor(cab_test[,'passenger_count'], labels=(1:6))


# 1.For Passenger_count:
cab_train$passenger_count[1000]
cab_train$passenger_count[1000] = NA
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]}
# Mode Method
getmode(cab_train$passenger_count)
# We can't use mode method because data will be more biased towards passenger_count=1

# 2.For fare_amount:
cab_train$fare_amount[1000]
cab_train$fare_amount[1000]= NA

# Mean Method
mean(cab_train$fare_amount, na.rm = T)
#Median Method
median(cab_train$fare_amount, na.rm = T)
sum(is.na(cab_train))
str(cab_train)
summary(cab_train)
summary(cab_test)

#Outlier Analysis
# We Will do Outlier Analysis only on Fare_amount just for now and we will do outlier
analysis after feature engineering laitudes and longitudes.
# Boxplot for fare_amount
pl1 = ggplot(cab_train,aes(x = factor(passenger_count),y = fare_amount))
pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1,
notch=FALSE)+ylim(0,100)

# Replace all outliers with NA and impute
vals = cab_train[,"fare_amount"] %in% boxplot.stats(cab_train[,"fare_amount"])$out
cab_train[which(vals),"fare_amount"] = NA
#lets check the NA's
sum(is.na(cab_train$fare_amount))

cab_train$fare_amount[is.na(cab_train$fare_amount)] = median(cab_train$fare_amount,
na.rm = T)
sum(is.na(cab_train$fare_amount))
sum(is.na(cab_train$passenger_count))
cab_train$passenger_count[is.na(cab_train$passenger_count)] = 2
sum(is.na(cab_train$passenger_count))
sum(is.na(cab_train))
sum(is.na(cab_test))
#feature engineering
# 2.Calculate the distance travelled using longitude and latitude
```

```
deg_to_rad = function(deg){
  (deg * pi) / 180}
haversine = function(long1,lat1,long2,lat2){
  #long1rad = deg_to_rad(long1)
  phi1 = deg_to_rad(lat1)
  #long2rad = deg_to_rad(long2)
  phi2 = deg_to_rad(lat2)
  delphi = deg_to_rad(lat2 - lat1)
  dellamda = deg_to_rad(long2 - long1)
  a = sin(delphi/2) * sin(delphi/2) + cos(phi1) * cos(phi2) *
    sin(dellamda/2) * sin(dellamda/2)
  c = 2 * atan2(sqrt(a),sqrt(1-a))
  R = 6371e3
  R * c / 1000 #1000 is used to convert to meters}
# Using haversine formula to calculate distance fr both train and test
cab_train$dist =
haversine(cab_train$pickup_longitude,cab_train$pickup_latitude,cab_train$dropoff_longitud
e,cab_train$dropoff_latitude)
cab_test$dist =
haversine(cab_test$pickup_longitude,cab_test$pickup_latitude,cab_test$dropoff_longitude,c
ab_test$dropoff_latitude)
View(cab_train)
# We will remove the variables which were used to feature engineer new variables
cab_train = subset(cab_train,select = -
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
cab_test = subset(cab_test,select = -
c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
View(cab_train)
View(cab_test)
sum(is.na(cab_train$pickup_datetime))

# 1.Feature Engineering for timestamp variable
# we will derive new features from pickup_datetime variable
# new features will be year,month,day_of_week,hour
#Convert pickup_datetime from factor to date time
cab_train$pickup_date = as.Date(as.character(cab_train$pickup_datetime))
cab_train$pickup_weekday = as.factor(format(cab_train$pickup_date,"%u"))# Monday = 1
cab_train$pickup_mnth = as.factor(format(cab_train$pickup_date,"%m"))
cab_train$pickup_yr = as.factor(format(cab_train$pickup_date,"%Y"))
pickup_time = strptime(cab_train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
cab_train$pickup_hour = as.factor(format(pickup_time,"%H"))
cab_train = subset(cab_train,select = -c(pickup_datetime,pickup_date,pickup_yr))
View(cab_train)

#Add same features to test set
cab_test$pickup_date = as.Date(as.character(cab_test$pickup_datetime))
cab_test$pickup_weekday = as.factor(format(cab_test$pickup_date,"%u"))# Monday = 1
cab_test$pickup_mnth = as.factor(format(cab_test$pickup_date,"%m"))
cab_test$pickup_yr = as.factor(format(cab_test$pickup_date,"%Y"))
pickup_time = strptime(cab_test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
```

```
cab_test$pickup_hour = as.factor(format(pickup_time,"%H"))
cab_test = subset(cab_test,select = -c(pickup_datetime,pickup_date,pickup_yr))
View(cab_test)
str(cab_test)
str(cab_train)
#Feature selection
numeric_index = sapply(cab_train,is.numeric) #selecting only numeric
numeric_data = cab_train[,numeric_index]
cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
corrgram(cab_train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")

# Feature Scaling
 #Normalisation
 cab_train[,'dist'] = (cab_train[,'dist'] - min(cab_train[,'dist']))/
    (max(cab_train[,'dist'] - min(cab_train[,'dist'])))
 print('dist')
View(cab_train)
cab_test[,'dist'] = (cab_test[,'dist'] - min(cab_test[,'dist']))/
   (max(cab_test[,'dist'] - min(cab_test[,'dist'])))
View(cab_test)
sum(is.na(cab_train))
cab_train = na.omit(cab_train)
sum(is.na(cab_train))
sum(is.na(cab_test))

#converting multilevel categorical variable into binary dummy variable
cnames= c("pickup_mnth","pickup_weekday","passenger_count","pickup_hour")
cab_train_new=cab_train[,cnames]
fare_amount=data.frame(cab_train$fare_amount)
names(fare_amount)[1]="fare_amount"
cab_train_new <- fastDummies::dummy_cols(cab_train_new)
cab_train_new= subset(cab_train_new,select = -
c(pickup_mnth,pickup_weekday,passenger_count,pickup_hour))
d3 = cbind(cab_train_new,cab_train)
d3= subset(d3,select = -
c(pickup_mnth,pickup_weekday,passenger_count,pickup_hour,fare_amount))
cab_train_new=cbind(d3,fare_amount)
View(cab_train_new)
str(cab_train_new)
cnames= c("pickup_mnth","pickup_weekday","passenger_count","pickup_hour")
cab_test_new=cab_test[,cnames]
cab_test_new <- fastDummies::dummy_cols(cab_test_new)
cab_test_new= subset(cab_test_new,select = -
c(pickup_mnth,pickup_weekday,passenger_count,pickup_hour))
cab_test_new = cbind(cab_test_new,cab_test)
cab_test_new= subset(cab_test_new,select = -
c(pickup_mnth,pickup_weekday,passenger_count,pickup_hour))
View(cab_test_new)
str(cab_test_new)
```

```
#divide data into test and train
train_index = sample(1:nrow(cab_train_new), 0.7 * nrow(cab_train_new))
train = cab_train_new[train_index,]
test = cab_train_new[-train_index,]
str(train)

#Linear regression model making
lm_model = lm(fare_amount ~., data = train)
pred_LR = predict(lm_model,test[,-51])

#Decision tree regression
fit = rpart(fare_amount ~ ., data = train, method = "anova")
pred_DT = predict(fit, test[,-51])

#Random Forest Model
RF_model = randomForest(fare_amount ~ ., train, importance = TRUE, ntree = 200)
pred_RF = predict(RF_model, test[,-51])
plot(RF_model)
#evaluating MApe value
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y))*100}
MAPE(test[,51],  pred_LR)
MAPE(test[,51],  pred_DT)
MAPE(test[,51],  pred_RF)
str(cab_test_new)
str(cab_train_new)

#final model
#Random Forest Model
RF_model_out = randomForest(fare_amount ~ ., cab_train_new, importance = TRUE, ntree
= 200)
# Saving the trained model
saveRDS(RF_model_out, "./final_RF_model_using_R.rds")
# loading the saved model
final_model <- readRDS("./final_RF_model_using_R.rds")
print(final_model)
# Lets now predict on test dataset
pred_RF_out= predict(RF_model_out,cab_test_new)
RF_pred = data.frame(cab_test_pickup_datetime,"predictions" = pred_RF_out)

# Now lets write(save) the predicted fare_amount in disk as .csv format
write.csv(RF_pred,"RF_predictions_R.csv",row.names = FALSE)
```

References

- https://learning.edwisor.com/
- https://www.analyticsvidhya.com/
- https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
- https://www.researchgate.net/publication/324706525_Taxi_Fare_Rate_Classification_Using_Deep_Networks
- https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/