**Project Name:**

**Cab Fare Prediction**

**Prepared by:**

**Kriti Upadhyay**

**Submission Date: 27.12.2019**

**CONTENT TABLE:**

**PART 1: IMPLEMENTATION BY R**

|  |  |  |
| --- | --- | --- |
| **S. no.** | **Topic** | **Page No.** |
| 1 | Introduction | 2 |
| 2 | Data Loading | 5 |
| 3 | Exploratory Analysis | 5 |
| 4 | Missing Value Analysis | 8 |
| 5 | Outlier Value Analysis | 11 |
| 6 | Feature Engineering | 14 |
| 7 | Feature Selection | 17 |
| 9 | Feature Scaling | 20 |
| 8 | Multicollinearity | 23 |
| 10 | Model Development | 25 |
| 11 | Model Selection | 30 |
| 12 | Conclusion/Result | 30 |
|  |  |  |
|  |  |  |
| **S. no.** | **Topic** | **Page No.** |
| 1 | Introduction |  |
| 2 | Data Loading | 33 |
| 3 | Exploratory Analysis | 37 |
| 4 | Missing Value Analysis | 40 |
| 5 | Outlier Value Analysis | 44 |
| 6 | Feature Engineering | 48 |
| 7 | Feature Selection | 59 |
| 8 | Multicollinearity | 63 |
| 9 | Feature Scaling | 65 |
| 10 | Model Development | 67 |
| 11 | Model Selection | 71 |
| 12 | Conclusion/Result | 71 |

**Introduction**

**Problem Statement:**

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

**Data Set provided:**

1) train\_cab.zip

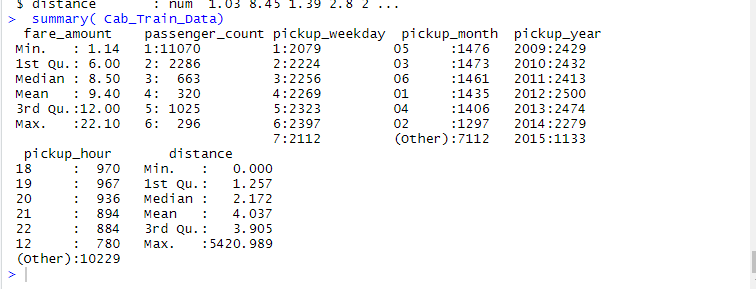
2) test.zip

**Data set contains below number of attributes:**

* pickup\_datetime - timestamp value indicating when the cab ride started.
* pickup\_longitude - float for longitude coordinate of where the cab ride started.
* pickup\_latitude - float for latitude coordinate of where the cab ride started.
* dropoff\_longitude - float for longitude coordinate of where the cab ride ended.
* dropoff\_latitude - float for latitude coordinate of where the cab ride ended.
* passenger\_count - an integer indicating the number of passengers in the cab

ride.

**Train Dataset Summary:**



**Fig 1: Summary of train dataset**

**Workflow of the Project:**

1. Prepare Problem (Load libraries, Load dataset)
2. Summarize Data (Descriptive statistics, Data visualizations)
3. Prepare Data (Data Cleaning, Feature Selection, Data Transforms)
4. Evaluate Algorithms (Split-out validation dataset, Testing and evaluating metric, Checking Algorithms, Comparing Algorithms)
5. Improve Accuracy (Algorithm Tuning, Ensembles)
6. Selecting Model (Predictions on validation dataset, create standalone model on entire training dataset, Predicting cab fare amount for test dataset.

**PART 1: IMPLEMENTATION BY R**

**Step-1:** **Data Loading**

Data Set is taken from the link provided and then loaded into the R environment for performing Data pre-processing techniques which are the necessary steps in the data science to organize data into proper format before feeding it to the model, because every Model accepts the data in a specific format form only.

**Following commands used to perform above task:**

#Loading datasets

Cab\_Train\_Data = read.csv("train\_cab.csv", header = T, na.strings = c(" ", "", "NA"))

Cab\_Test\_Data = read.csv("test.csv")

test\_pickup\_datetime = Cab\_Test\_Data["pickup\_datetime"]

#Viewing Structure of both train and test data

str( Cab\_Train\_Data)

str( Cab\_Test\_Data)

**Step-2 :** **Exploratory Data Analysis**

Exploratory Data Analysis is a process of analyzing data in detail and Removing values which are not within desired range depending upon basic understanding of dataset. In our case of dataset, we explored data upon following understanding that are as follows:

1. Fare\_amount variable has a negative value, which doesn't make sense because price amount cannot be negative and also cannot be 0. So, we will remove the observations having negative fare amount.
2. 20 observations of passenger\_count variable is consistently above from 6,7,8,9,10,11 passenger\_counts, we need to remove these fields as cab contain maximum number of passenger 6 and also Removing 58 observations having passenger\_count = 0
3. Latitudes range should be from -90 to 90 and Longitudes range should be from -180 to 180.Removing which does not satisfy these ranges

**Following codes used to perform above task:**

**###############EXPLORATORY DATA ANALYSIS ###################**

#Changing the data types of variables

Cab\_Train\_Data$fare\_amount= as.numeric(as.character(Cab\_Train\_Data$fare\_amount))

Cab\_Train\_Data$passenger\_count=round(Cab\_Train\_Data$passenger\_count)

#Removing values which are not within desired range depending upon basic understanding of dataset.

#1. Fare amount has a negative value, which doesn't make sense. A price amount cannot be negative and also cannot be 0. So, we will remove these fields.

#Rows with negative Fare amount

Cab\_Train\_Data[which(Cab\_Train\_Data$fare\_amount < 1 ),]

#Count of rows having negative fare amount

nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$fare\_amount < 1 ),])

#Removing rows from data containing negative fare amount

Cab\_Train\_Data= Cab\_Train\_Data[-which(Cab\_Train\_Data$fare\_amount < 1 ),]

#2. Passenger\_count variable

for (i in seq(4,11,by=1))

{

print(paste('passenger\_count above ' ,i,nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$passenger\_count > i ),])))

}

#So 20 observations of passenger\_count is consistenly above from 6,7,8,9,10,11 passenger\_counts, checking them.

Cab\_Train\_Data[which(Cab\_Train\_Data$passenger\_count > 6 ),]

#We need to see if there are any passenger\_count = 0

Cab\_Train\_Data[which(Cab\_Train\_Data$passenger\_count <1 ),]

#Getting number of observation having passenger\_count==0

nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$passenger\_count <1 ),])

#Removing 58 observations having passenger\_count==0 and 20 observation which are above 6 because a cab cannot hold these number of passengers.

Cab\_Train\_Data = Cab\_Train\_Data[-which(Cab\_Train\_Data$passenger\_count < 1 ),]

Cab\_Train\_Data = Cab\_Train\_Data[-which(Cab\_Train\_Data$passenger\_count > 6),]

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

#Getting rows of Latitudes range from -90 to 90. Longitudes range from -180 to 180.

print(paste('pickup\_longitude above 180=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_longitude >180 ),])))

print(paste('pickup\_longitude above -180=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_longitude < -180 ),])))

print(paste('pickup\_latitude above 90=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_latitude > 90 ),])))

print(paste('pickup\_latitude above -90=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_latitude < -90 ),])))

print(paste('dropoff\_longitude above 180=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$dropoff\_longitude > 180 ),])))

print(paste('dropoff\_longitude above -180=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$dropoff\_longitude < -180 ),])))

print(paste('dropoff\_latitude above -90=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$dropoff\_latitude < -90 ),])))

print(paste('dropoff\_latitude above 90=',nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$dropoff\_latitude > 90 ),])))

#Removing pickup\_latitude above 90

Cab\_Train\_Data= Cab\_Train\_Data[-which(Cab\_Train\_Data$pickup\_latitude > 90),]

#Getting rows having longitude and latitude values which are equal to 0

nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_longitude == 0 ),])

nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_latitude == 0 ),])

nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$dropoff\_longitude == 0 ),])

nrow(Cab\_Train\_Data[which(Cab\_Train\_Data$pickup\_latitude == 0 ),])

#Removing longitude and latitude values which are equal to 0

Cab\_Train\_Data = Cab\_Train\_Data[-which(Cab\_Train\_Data$pickup\_longitude == 0),]

Cab\_Train\_Data = Cab\_Train\_Data[-which(Cab\_Train\_Data$dropoff\_longitude == 0),]

**Step-3: Missing Value Analysis**

After loading the data, the first step performed is missing value analysis where we compute the missing values or information present in the data set. Here the missing value analysis is computed by using the method of KNN Imputation, because it computes the missing values by using Euclidean Distance formula and this method gives accurate results mostly than compared to mean median method.

**Following codes used to perform above task:**

**############### MISSING VALUE ANALYSIS ###################**

#Missing value detection in each variable of the dataset by using apply function that uses Sum(is.na()) function as an argument and will return total count of the missing values present in each variable of the dataset.

missing\_val = data.frame(apply(Cab\_Train\_Data,2,function(x){sum(is.na(x))}))

missing\_val

#Converting row names into column and renaming variable name

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

row.names(missing\_val) = NULL

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

#Calculating Missing\_percentage

missing\_val$Missing\_percentage = (missing\_val$Missing\_percentage/nrow(Cab\_Train\_Data)) \* 100

#Sorting variable

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

#Rearranging columns

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

missing\_val

**#Getting unqiue passenger count for train and test data**

unique(Cab\_Train\_Data$passenger\_count)

unique(Cab\_Test\_Data$passenger\_count)

#Converting datatype of passenger\_count variable as factor for both test and train data

Cab\_Train\_Data[,'passenger\_count'] = factor(Cab\_Train\_Data[,'passenger\_count'], labels=(1:6))

Cab\_Test\_Data[,'passenger\_count'] = factor(Cab\_Test\_Data[,'passenger\_count'], labels=(1:6))

#Applying different methods for computing missing values

#Mode Method

#Cab\_Train\_Data$passenger\_count[1000]

#Cab\_Train\_Data$passenger\_count[1000] = NA

#getmode = function(v)

#{

#uniqv = unique(v)

#uniqv[which.max(tabulate(match(v, uniqv)))]

#}

#getmode(Cab\_Train\_Data$passenger\_count)

#For Passenger\_count variable:

#Actual value = 1

#Mode = 1

#We can't use mode method because data is more biased towards passenger\_count=1

# Mean Method

#Cab\_Train\_Data$fare\_amount[1000]

#Cab\_Train\_Data$fare\_amount[1000]= NA

#mean(Cab\_Train\_Data$fare\_amount, na.rm = T)

# Median Method

#Cab\_Train\_Data$fare\_amount[1000]

#Cab\_Train\_Data$fare\_amount[1000]= NA

#median(train$fare\_amount, na.rm = T)

# kNN Imputation

#Cab\_Train\_Data$fare\_amount[1000]

#Cab\_Train\_Data$fare\_amount[1000]=NA

#Cab\_Train\_Data = knnImputation(Cab\_Train\_Data, k = 181)

# For fare\_amount variable:

# Actual value = 18.1,

# Mean = 15.117,

# Median = 8.5,

# KNN = 18.28

#Since the missing values computed by different methods but only KNN imputation method would be good choice as the value computed by it is more closer to the actual values.

#Computing missing values using KNN imputation method

Cab\_Train\_Data = knnImputation(Cab\_Train\_Data, k = 181)

#Check for missing values

sum(is.na(Cab\_Train\_Data))

#Getting structure and summary of the train data

str(Cab\_Train\_Data)

summary(Cab\_Train\_Data)

**Step-3: Outliers Detection & Removal**

**Outliers Detection:**

Once, the missing values are computed, the next step is to detect and remove outliers present in the data set. These outliers are nothing but the extreme values present in the data set.

In this project, missing values are detected using Box-plot method because it gives the graphical representation of the presence of the outliers in the data set and completely distinguish the outliers present in the lower and upper fence of the box plot. Using replace outlier by NA method in our dataset as deleting outliers can cause information loss which might be important for the computation of the model.

**Following codes used to perform above task:**

**################## OUTLIER ANALYSIS ###################**

#Selecting only numeric variables

Numeric\_index= sapply(Cab\_Train\_Data,is.numeric)

Numeric\_data=Cab\_Train\_Data[,Numeric\_index]

#Variables names containing numeric data

Cnames=colnames(Numeric\_data)

Cnames

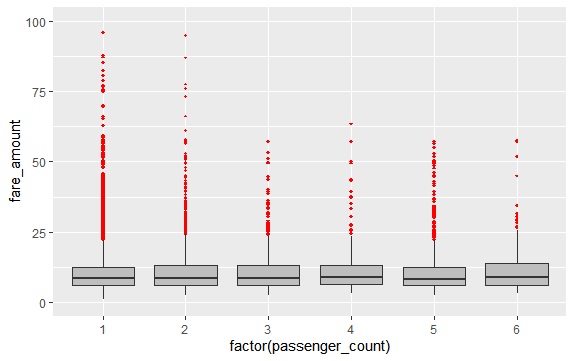
#Performing Outlier Analysis only on Fare\_amount because other variables will be required to perform feature engineering.

# Boxplot for fare\_amount

pl1 = ggplot(Cab\_Train\_Data,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)

Detection of the outliers present in the fare\_amount with respect to the passenger\_count variable using Box plots are shown below:



**Fig 2: fare\_amount variable with outliers**

**Outlier Removal:**

Outlier replaced by NA method: This method is chosen because we cannot afford deletion of the outliers as sometimes outliers contains important information. After replacing outliers with NA, the NA values are further computed by the method KNN imputation. Replacing outliers with NA present in the dataset as this is the best method to deal with the NA as taught in lecture.

**Following codes used to perform above task:**

# Replace all outliers with NA and impute

vals=Cab\_Train\_Data[,"fare\_amount"]%in% boxplot.stats(Cab\_Train\_Data[,"fare\_amount"])$out

Cab\_Train\_Data[which(vals),"fare\_amount"] = NA

#lets check the NA's

sum(is.na(Cab\_Train\_Data$fare\_amount))

#Imputing with KNN

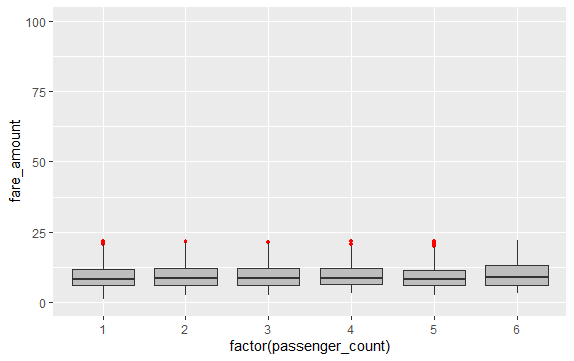
Cab\_Train\_Data = knnImputation(Cab\_Train\_Data,k=3)

#Checking the missing values

sum(is.na(Cab\_Train\_Data$fare\_amount))

str(Cab\_Train\_Data)

Boxplot for outliers present in the fare\_amount with respect to the passenger\_count variable after outlier removal are shown below:



**Fig 3: fare\_amount variable without outliers**

**STEP 4: Feature Engineering**

Below features are derived from the existing variable pickup\_datetime for efficient understanding of data and for good modelling purpose too:

1. pickup\_date

2. pickup\_weekday

3. pickup\_month

4. pickup\_year

5. pickup\_hour

6. distance

**Following codes used to perform above task:**

**################## FEATURE ENGINEERING ###################**

#1. Feature Engineering for timestamp variable

#we will derive new features from pickup\_datetime variable and new features will be year,month,day\_of\_week,hour

#Convert pickup\_datetime from factor to date time

Cab\_Train\_Data$pickup\_date = as.Date(as.character(Cab\_Train\_Data$pickup\_datetime))

Cab\_Train\_Data$pickup\_weekday = as.factor(format(Cab\_Train\_Data$pickup\_date,"%u"))

Cab\_Train\_Data$pickup\_month = as.factor(format(Cab\_Train\_Data$pickup\_date,"%m"))

Cab\_Train\_Data$pickup\_year = as.factor(format(Cab\_Train\_Data$pickup\_date,"%Y"))

pickup\_time = strptime(Cab\_Train\_Data$pickup\_datetime,"%Y-%m-%d %H:%M:%S")

Cab\_Train\_Data$pickup\_hour = as.factor(format(pickup\_time,"%H"))

#Adding similar features to test dataset

Cab\_Test\_Data$pickup\_date = as.Date(as.character(Cab\_Test\_Data$pickup\_datetime))

Cab\_Test\_Data$pickup\_weekday = as.factor(format(Cab\_Test\_Data$pickup\_date,"%u"))

Cab\_Test\_Data$pickup\_month = as.factor(format( Cab\_Test\_Data$pickup\_date,"%m"))

Cab\_Test\_Data$pickup\_year = as.factor(format( Cab\_Test\_Data$pickup\_date,"%Y"))

pickup\_time = strptime( Cab\_Test\_Data$pickup\_datetime,"%Y-%m-%d %H:%M:%S")

Cab\_Test\_Data$pickup\_hour = as.factor(format(pickup\_time,"%H"))

#Check for NA values

missing\_val = data.frame(apply(Cab\_Train\_Data,2,function(x){sum(is.na(x))}))

missing\_val

#One NA was present in variable pickup\_datetime which created NA in above new feature engineered variables.

#Removing that 1 row of NA's

Cab\_Train\_Data= na.omit(Cab\_Train\_Data)

#Displaying Cab\_Train\_Data

Cab\_Train\_Data

#Removing the variables which were used to engineer new variables from both train and test dataset

Cab\_Train\_Data = subset(Cab\_Train\_Data,select = -c(pickup\_datetime,pickup\_date))

Cab\_Test\_Data= subset( Cab\_Test\_Data,select = -c(pickup\_datetime,pickup\_date))

#2. Calculate the distance travelled using longitude and latitude

#function to convert degree into radian

deg\_to\_rad = function(deg)

{

(deg \* pi) / 180

}

#Using haversine formula to calculate distance for both train and test data

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

#1000 is used to convert to meters

R \* c / 1000

}

Cab\_Train\_Data$distance = haversine( Cab\_Train\_Data$pickup\_longitude, Cab\_Train\_Data$pickup\_latitude, Cab\_Train\_Data$dropoff\_longitude, Cab\_Train\_Data$dropoff\_latitude)

Cab\_Test\_Data$distance = haversine(Cab\_Test\_Data$pickup\_longitude,Cab\_Test\_Data$pickup\_latitude,Cab\_Test\_Data$dropoff\_longitude,Cab\_Test\_Data$dropoff\_latitude)

#Removing the variables which were used to engineer new variables from both train and test dataset

Cab\_Train\_Data = subset( Cab\_Train\_Data,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude))

Cab\_Test\_Data = subset(Cab\_Test\_Data,select = -c(pickup\_longitude,pickup\_latitude,dropoff\_longitude,dropoff\_latitude))

#Getting summary and structure of the train data

str( Cab\_Train\_Data)

summary( Cab\_Train\_Data)

**STEP-5:** **Feature selection**

This stage involves the process of reducing variables on the basis of correlation present in the variables of the dataset. Correlation plot is being used to analyze the correlation among the variables and reduce the dimensionality on the basis of correlation between variables. 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.

**Following codes used to perform above task:**

**################## FEATURE SELECTION ###################**

**################## CORRELATION PLOT ####################**

#Selecting only numeric variables

Numeric\_index= sapply(Cab\_Train\_Data,is.numeric)

Numeric\_data=Cab\_Train\_Data[,Numeric\_index]

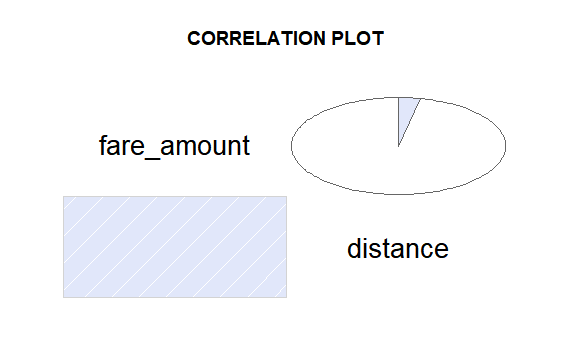
#Variables names containing numeric data

Cnames=colnames(Numeric\_data)

Cnames

#Correlation plot

corrgram(Cab\_Train\_Data[,Cnames],order=F,upper.panel=panel.pie,text.panel=panel.txt,main="CORRELATION PLOT")



**Fig 4: Correlation plot**

From above correlation plot we see that:

• 'fare\_amount' and 'distance' are very highly correlated with each other.

• As fare\_amount is the target variable and ‘distance’ is independent variable we will keep ‘distance’ because it will help to explain variation in fare\_amount.

**2. Analysis of Variance (Anova) Test –** It is carried out to compare between each group in a categorical variable. ANOVA only lets us know the means for different groups are same or not. It doesn’t help us identify which mean is different.

Hypothesis testing:

- **Null Hypothesis**: mean of all categories in a variable are same.

- **Alternate Hypothesis**: mean of at least one category in a variable is different.

If p-value is less than 0.05 then we reject the null hypothesis and if p-value is greater than 0.05 then we accept the null hypothesis.

**Following codes used to perform above task:**

**################## ANOVA TEST ###################**

#ANOVA for categorical variables with target numeric variable

Anova\_results = aov(fare\_amount ~ passenger\_count + pickup\_hour + pickup\_weekday + pickup\_month + pickup\_year,data = Cab\_Train\_Data)

#Summary of anova result

summary(Anova\_results)

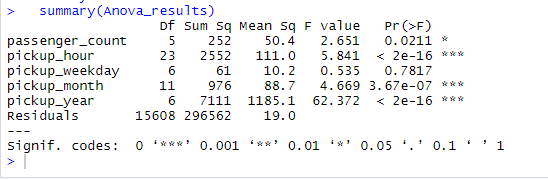
#pickup\_weekday has p value greater than 0.05, so rejecting this variable

Cab\_Train\_Data = subset(Cab\_Train\_Data,select=-pickup\_weekday)

#Also remove that variable from from test dataset

Cab\_Test\_Data = subset(Cab\_Test\_Data,select=-pickup\_weekday)

Below is the Anova analysis table for each categorical variable:



**Fig 5: ANOVA test on Variables**

pickup\_weekday variable has p value greater than 0.05, so rejecting this variable

**STEP-6:** **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. Normalization 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 ‘distance’ is not distributed normally so we had chosen normalization over standardization. High variance will affect the accuracy of the model. So, we want to normalise that variance. It is performed only on Continuous variables.

**Following codes used to perform above task:**

**################## FEATURE SCALING ###################**

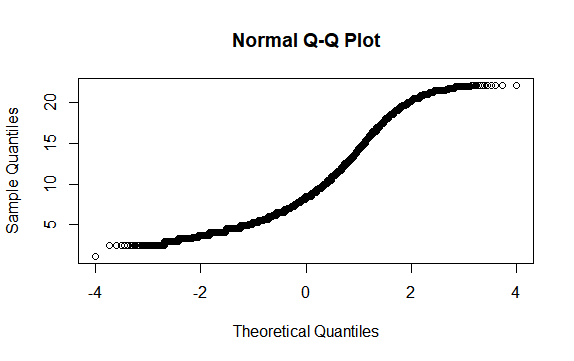
#Normality check

qqnorm(Cab\_Train\_Data$fare\_amount)

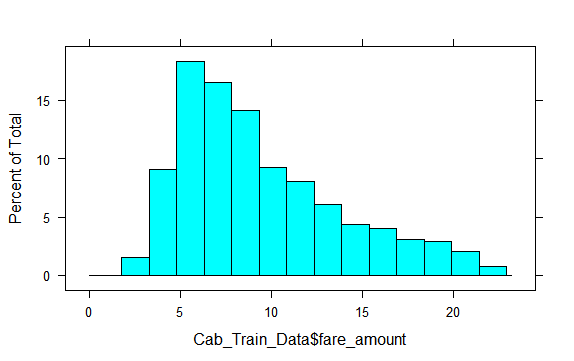
histogram(Cab\_Train\_Data$fare\_amount)

qqnorm(Cab\_Train\_Data$distance)

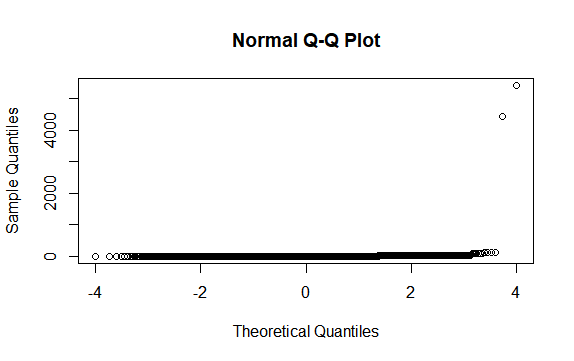
histogram(Cab\_Train\_Data$distance)



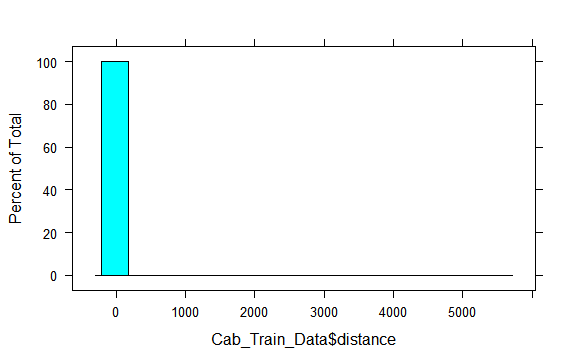
**Fig 6: Normality check on fare\_amount variable**



**Fig 7: Normality check on fare\_amount variable using histogram**



**Fig 8: Normality check on distance variable**



**Fig 9: Normality check on distance variable using histogram**

On observing plots, it is clear that the data of the distance variable is not uniformly distributed. Hence in this applying normalization for proper scaling of the distance variable.

#Normalisation (distance variable is not uniformly distributed)

Cab\_Train\_Data[,'distance'] = (Cab\_Train\_Data[,'distance'] - min(Cab\_Train\_Data[,'distance']))/

(max(Cab\_Train\_Data[,'distance'] - min(Cab\_Train\_Data[,'distance'])))

**STEP 7:** **Multicollinearity**

In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other. Multicollinearity increases the standard errors of the coefficients. It Increases 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, you can assume that the regression coefficients are poorly estimated due to multicollinearity.**

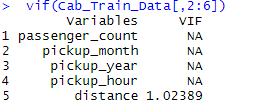
**Following codes used to perform above task:**

#check for multicollinearity

#vif(Cab\_Train\_Data[,2:6])

#vifcor(Cab\_Train\_Data[,-1], th = 0.9)

Below is the table for VIF analysis for each independent variable:



**Fig 10: VIF analysis**

**STEP 8: Splitting train and Validation Dataset**

We have used **createDataPartition()** method to divide whole Dataset into train and validation dataset. 20% is in validation dataset and 80% is in training data. We will test the performance of model on validation dataset. The model which performs best will be chosen to perform on test dataset provided along with original train dataset.

* X\_train y\_train--are train subset.
* X\_test y\_test--are validation subset.

**Following codes used to perform above task:**

**################## SPLITTING DATA ###################**

set.seed(1000)

#Splitting 80% of Cab-Train\_Data in train\_data and 20% in Validation Dataset(test\_data)

train\_index = createDataPartition(Cab\_Train\_Data$fare\_amount,p=0.80,list = FALSE)

train\_data = Cab\_Train\_Data[train\_index,]

test\_data = Cab\_Train\_Data[-train\_index,]

**STEP 9: Model Development**

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. In this project I have built models using four Algorithms:

* Linear Regression
* Decision Tree
* Random Forest
* Xgboost Regression

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific error metrics like –

Regression metrics for our Models:

* MAE (Mean Absolute Error)
* MAPE (Mean Absolute Percentage Error)
* MSE (Mean square Error)
* RMSE (Root Mean Square Error)

**################## LINEAR REGRESSION ###################**

#Running Regression model

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

#Summary of the model

summary(lm\_model)

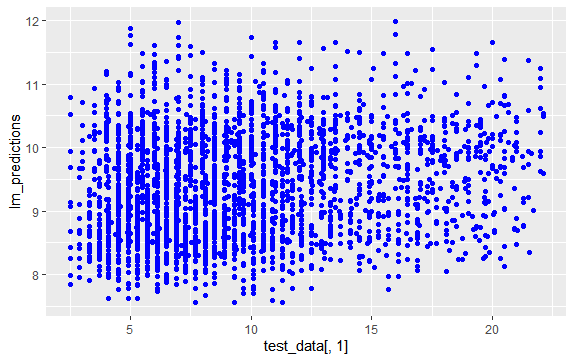
str(train\_data)

#Predicting test\_data using predict() method

lm\_predictions = predict(lm\_model,test\_data[,2:6])

#plotting regression model on the basis of test\_data

qplot(x = test\_data[,1], y = lm\_predictions, data = test\_data, color = I("blue"), geom = "point")



**Fig 11: Linear Regression predicted plot**

#Evaluation of the linear regression model on the basis of test\_data

regr.eval(test\_data[,1],lm\_predictions)

**# mae mse rmse mape**

**# 3.5120358 19.0085569 4.3598804 0.4544157**

**#Error rate =0.45**

**#Accuracy=55%**

**################## DECISION TREE ###################**

#Running Decision tree model

Dt\_model = rpart(fare\_amount ~ ., data = train\_data, method = "anova")

#Summary of the model

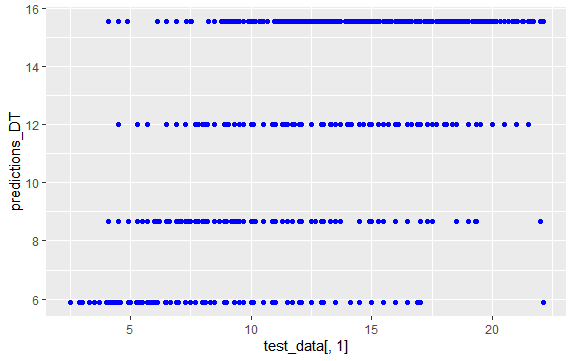
summary(Dt\_model)

#Predicting test cases using predict() method

predictions\_DT = predict(Dt\_model, test\_data[,2:6])

#plotting Decision tree model on the basis predicted test\_data

qplot(x = test\_data[,1], y = predictions\_DT, data = test\_data, color = I("blue"), geom= "point")



**Fig 12: Decision tree predicted plot**

#Evaluation of Decision tree model on the basis of test\_data

regr.eval(test\_data[,1],predictions\_DT)

**# mae mse rmse mape**

**# 1.9674997 7.0442171 2.6540944 0.2317727**

**#Error rate =0.23**

**#Accuracy=77%**

**################## RANDOM FOREST ###################**

#Running Random Forest model

rf\_model = randomForest(fare\_amount ~.,data=train\_data)

#Summary of the model

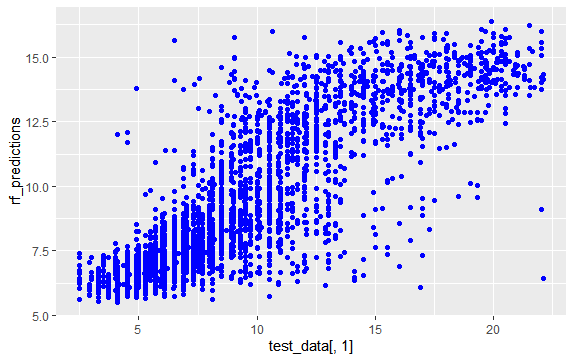
summary(rf\_model)

#Predicting test cases using predict() method

rf\_predictions = predict(rf\_model,test\_data[,2:6])

#plotting Random Forest model on the basis predicted test\_data

qplot(x = test\_data[,1], y = rf\_predictions, data = test\_data, color = I("blue"), geom = "point")



**Fig 13: Random Forest predicted plot**

#Evaluation of Random Forest model on the basis of test\_data

regr.eval(test\_data[,1],rf\_predictions)

**# mae mse rmse mape**

**# 1.8838831 6.2805142 2.5060954 0.2312715**

**#Error rate =0.23**

**#Accuracy=77%**

**################## XGBOOST MODEL ###################**

#Running xgboost model

train\_data\_matrix = as.matrix(sapply(train\_data[-1],as.numeric))

test\_data\_matrix = as.matrix(sapply(test\_data[-1],as.numeric))

xgboost\_model =xgboost(data = train\_data\_matrix,label = train\_data$fare\_amount,nrounds = 15,verbose = FALSE)

#Summary of the model

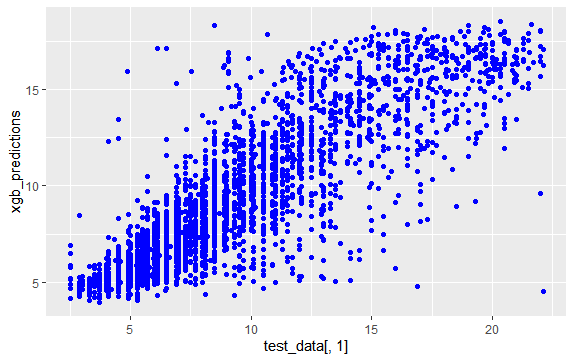
summary(xgboost\_model)

#Predicting test cases using predict() method

xgb\_predictions = predict(xgboost\_model,test\_data\_matrix)

#plotting xgboost model on the basis predicted test\_data

qplot(x = test\_data[,1], y = xgb\_predictions, data = test\_data, color = I("blue"), geom = "point")



**Fig 14: XGBOOST model predicted plot**

#Evaluation of Random Forest model on the basis of test\_data

regr.eval(test\_data[,1],xgb\_predictions)

**# mae mse rmse mape**

**# 1.6206727 5.2563892 2.2926817 0.1835422**

**#Error rate =0.18**

**#Accuracy=82%**

**STEP 10: Model Selection**

After developing model, we will evaluate the performance of the model by considering generated Regression metrics for our Models in development stage:

* MAE (Mean Absolute Error)
* MAPE (Mean Absolute Percentage Error)
* MSE (Mean square Error)
* RMSE (Root Mean Square Error)

The lesser the error rate better will be the performance and the accuracy of the model. On comparing these regression metrics of different models applied on the dataset we will select that model who has low error rate and better accuracy compared to other models for cab fare prediction.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Models | MAE | MAPE | MSE | RMSE | Error Rate | Accuracy |
| Linear Regression | 3.51 | 0.45 | 19.00 | 4.35 | 0.45 | 55% |
| Decision Tree | 1.96 | 0.23 | 7.04 | 2.65 | 0.23 | 77% |
| Random Forest | 1.88 | 0.23 | 6.28 | 2.50 | 0.23 | 77% |
| Xgboost model | 1.62 | 0.18 | 5.2 | 2.2 | 0.18 | 82% |

On Comparing different regression metrics Xgboost model found to be better model than other models and have good performance and accuracy as well as low error rate.So, selecting XGBOOST model for cab fare prediction .

**Conclusion/Result:**

**################## CAB FARE PREDICTION ###################**

#Training model on whole training Dataset and saving model using xgboost model has it has more accuracy when compared to other models

train\_data\_matrix1 = as.matrix(sapply(Cab\_Train\_Data[-1],as.numeric))

test\_data\_matrix1 = as.matrix(sapply(Cab\_Test\_Data,as.numeric))

#Running xgboost model on enire train data

xgboost\_model1 = xgboost(data = train\_data\_matrix1,label = Cab\_Train\_Data$fare\_amount,nrounds = 15,verbose = FALSE)

#Saving the trained model

saveRDS(xgboost\_model1, "./Trained\_Xgboost\_model\_using\_R.rds")

#loading the saved model

Final\_Trained\_model= readRDS("./Trained\_Xgboost\_model\_using\_R.rds")

print(Final\_Trained\_model)

#Predicting fare\_amount on test dataset

xgb = predict(Final\_Trained\_model,test\_data\_matrix1)

xgb\_pred = data.frame(test\_pickup\_datetime,"predictions" = xgb)

#Writing the predicted fare\_amount in disk in .csv format

write.csv(xgb\_pred,"Cab\_Fare\_Prediction\_By\_R.csv",row.names = FALSE)

**Note: The predicted fare\_amount for the test data is present in the file "Cab\_Fare\_Prediction\_By\_R.csv" which is attached with the report.**

**PART 2: IMPLEMENTATION BY PYTHON**

**Step-1:** **Data Loading**

Data Set is taken from the link provided and then loaded into the python environment for performing Data pre-processing techniques which are the necessary steps in the data science to organize data into proper format before feeding it to the model, because every Model accepts the data in a specific format form only.

**Following commands used to perform above task:**

# loading the required libraries

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import scipy.stats as stats

from geopy.distance import geodesic

from scipy.stats import chi2\_contingency

import statsmodels.api as sm

from statsmodels.formula.api import ols

from patsy import dmatrices

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

import xgboost as xgb

# Importing dataset

Cab\_Train\_Data = pd.read\_csv('train\_cab.csv',dtype={'fare\_amount':np.float64},na\_values={'fare\_amount':'430-'})

Cab\_Test\_Data = pd.read\_csv('test.csv')

#Summary of the train dataset

Cab\_Train\_Data.describe()

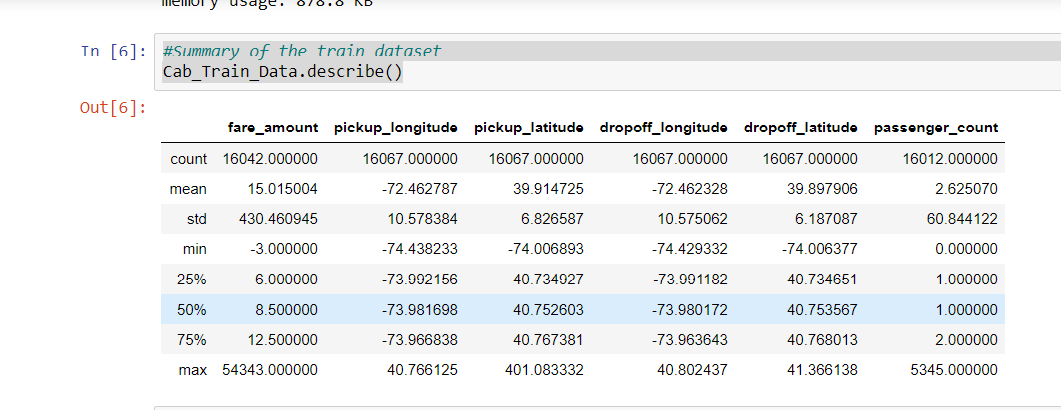


Fig 1: Summary of train dataset

#Summary of the test dataset

Cab\_Test\_Data.describe()

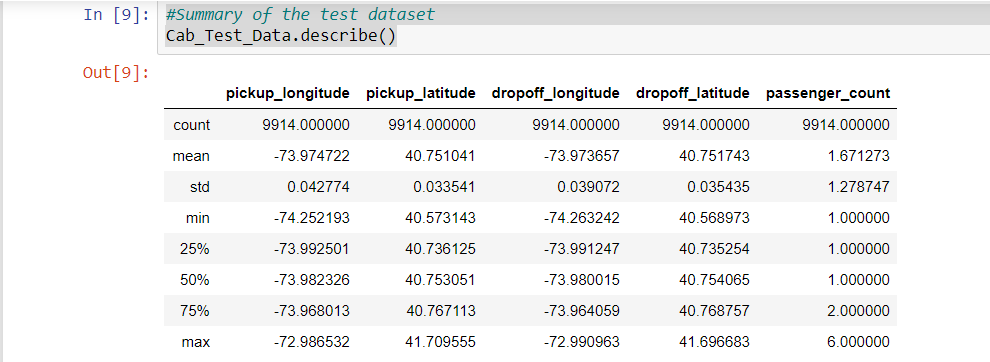


Fig 2: Summary of test dataset

**####################### DATA VISUALIZATION ###############**

**# setting up the sns for plots**

**sns.set(style='darkgrid',palette='Set1')**

**# Plotting histogram for variables of the train dataset**

**plt.figure(figsize=(20,20))**

**plt.subplot(321)**

**\_ = sns.distplot(Cab\_Train\_Data['fare\_amount'],bins=50)**

**plt.subplot(322)**

**\_ = sns.distplot(Cab\_Train\_Data['pickup\_longitude'],bins=50)**

**plt.subplot(323)**

**\_ = sns.distplot(Cab\_Train\_Data['pickup\_latitude'],bins=50)**

**plt.subplot(324)**

**\_ = sns.distplot(Cab\_Train\_Data['dropoff\_longitude'],bins=50)**

**plt.subplot(325)**

**\_ = sns.distplot(Cab\_Train\_Data['dropoff\_latitude'],bins=50)**

**#Saving Plots**

**plt.savefig('histogrambypython.png')**

**#Displaying plots**

**plt.show()**

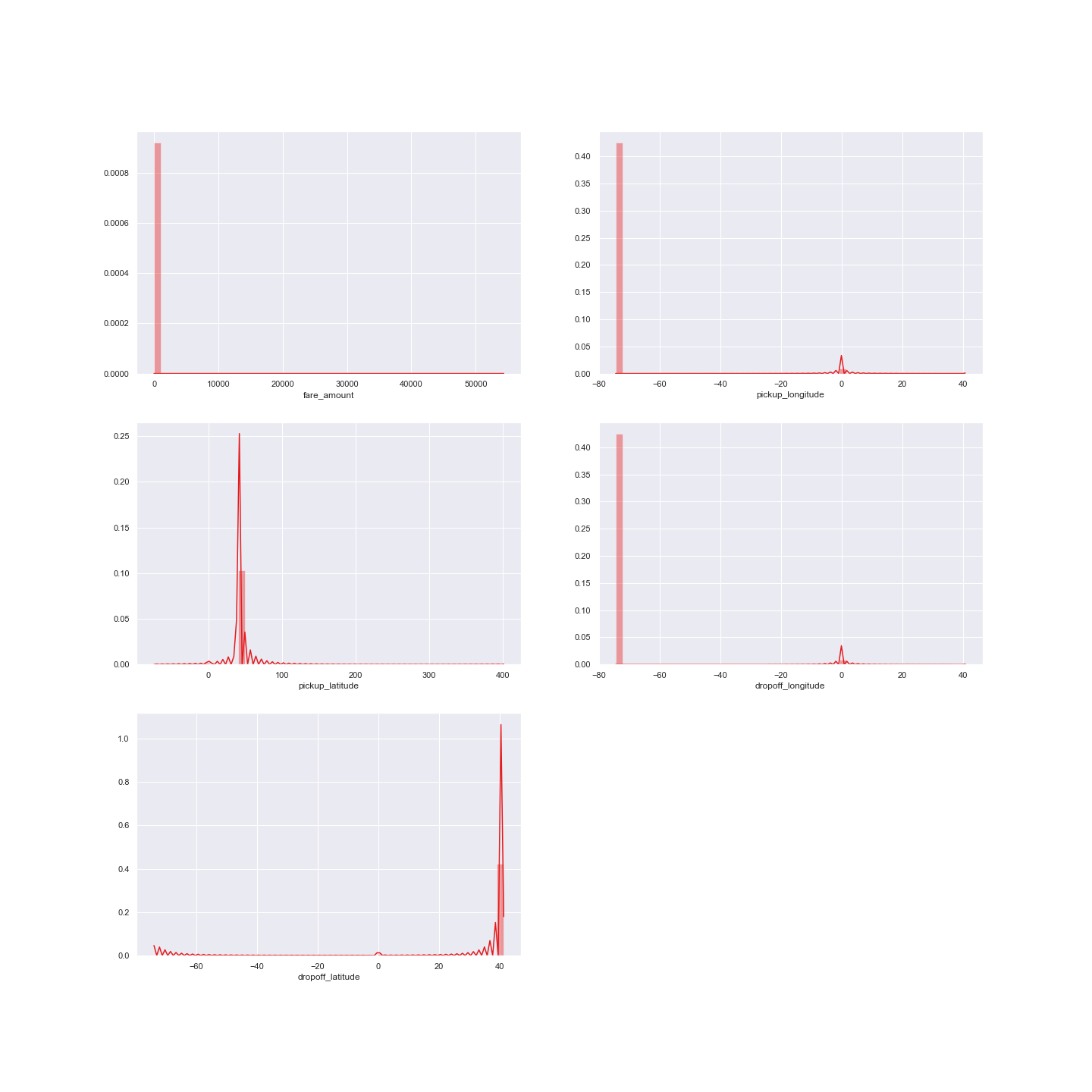


Fig 3: Data visualization

**Step-2:** **Exploratory Data Analysis**

Exploratory Data Analysis is a process of analyzing data in detail and Removing values which are not within desired range depending upon basic understanding of dataset. In our case of dataset, we explored data upon following understanding that are as follows:

1. Fare\_amount variable has a negative value, which doesn't make sense because price amount cannot be negative and also cannot be 0. So, we will remove the observations having negative fare amount.
2. 20 observations of passenger\_count variable is consistently above from 6,7,8,9,10,11 passenger\_counts, we need to remove these fields as cab contain maximum number of passenger 6 and also Removing 58 observations having passenger\_count = 0
3. Latitudes range should be from -90 to 90 and Longitudes range should be from -180 to 180.Removing which does not satisfy these ranges

**Following codes used to perform above task:**

**###############EXPLORATORY DATA ANALYSIS ###################**

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

#1. Fare amount has a negative value, which do not make sense. A price amount cannot be negative and also cannot be 0. So, we will remove these fields.

# Finding total number of observations have fare amount negative

sum(Cab\_Train\_Data['fare\_amount']<1)

#Getting records having fare amount in negatives

Cab\_Train\_Data[Cab\_Train\_Data['fare\_amount']<1]

#Removing records having fare amount in negatives

Cab\_Train\_Data= Cab\_Train\_Data.drop(Cab\_Train\_Data[Cab\_Train\_Data['fare\_amount']<1].index, axis=0)

Cab\_Train\_Data.loc[Cab\_Train\_Data['fare\_amount'] < 1,'fare\_amount'] = np.nan

#Checking for negative value in fare amount variable

sum(Cab\_Train\_Data['fare\_amount']<1)

#2. Passenger\_count variable needs to convert into a categorical variable because passenger\_count is not a continuous variable.

# passenger\_count cannot take continuous values also they are limited in number if it’s a cab that is maximum number of passengers can sit in a cab is 6.

#Fetching records having passenger count greater than 4

for i in range(4,11):

print('passenger\_count above' +str(i)+'={}'.format(sum(Cab\_Train\_Data['passenger\_count']>i)))

**Output:**

passenger\_count above4=1367

passenger\_count above5=322

passenger\_count above6=20

passenger\_count above7=20

passenger\_count above8=20

passenger\_count above9=20

passenger\_count above10=20

#20 observations of passenger\_count is consistenly above from 6,7,8,9,10 passenger\_counts, let's check them.

Cab\_Train\_Data[Cab\_Train\_Data['passenger\_count']>6]

#Checking for any passenger\_count<1

Cab\_Train\_Data[Cab\_Train\_Data['passenger\_count']<1]

#Finding total number of records having passenger\_count<1

len(Cab\_Train\_Data[Cab\_Train\_Data['passenger\_count']<1])

#passenger\_count variable conains value equal to 0 but test data does not contain passenger\_count=0. So, we will remove those 0 values.

#passenger\_count variable conains value equal to 0 but test data does not contain passenger\_count=0. So, we will remove those 0 values.

#Also dropping 20 observations which are above 6 because a cab cannot hold number of passengers greater than 6 .

Cab\_Train\_Data = Cab\_Train\_Data.drop(Cab\_Train\_Data[Cab\_Train\_Data['passenger\_count']>6].index, axis=0)

Cab\_Train\_Data = Cab\_Train\_Data.drop(Cab\_Train\_Data[Cab\_Train\_Data['passenger\_count']<1].index, axis=0)

#Check for passenger\_count>6

sum(Cab\_Train\_Data['passenger\_count']>6)

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

#Getting data for checking ranges

print('pickup\_longitude above 180={}'.format(sum(Cab\_Train\_Data['pickup\_longitude']>180)))

print('pickup\_longitude below -180={}'.format(sum(Cab\_Train\_Data['pickup\_longitude']<-180)))

print('pickup\_latitude above 90={}'.format(sum(Cab\_Train\_Data['pickup\_latitude']>90)))

print('pickup\_latitude below -90={}'.format(sum(Cab\_Train\_Data['pickup\_latitude']<-90)))

print('dropoff\_longitude above 180={}'.format(sum(Cab\_Train\_Data['dropoff\_longitude']>180)))

print('dropoff\_longitude below -180={}'.format(sum(Cab\_Train\_Data['dropoff\_longitude']<-180)))

print('dropoff\_latitude below -90={}'.format(sum(Cab\_Train\_Data['dropoff\_latitude']<-90)))

print('dropoff\_latitude above 90={}'.format(sum(Cab\_Train\_Data['dropoff\_latitude']>90)))

**Output:**

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

#Only one outlier which is in variable pickup\_latitude.So we will remove it with nan.

#Checking for any values equal to 0.

for i in ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:

print(i,'equal to 0={}'.format(sum(Cab\_Train\_Data[i]==0)))

#Removing values =0 with NA

Cab\_Train\_Data= Cab\_Train\_Data.drop(Cab\_Train\_Data[Cab\_Train\_Data['pickup\_latitude']>90].index, axis=0)

for i in ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:

Cab\_Train\_Data = Cab\_Train\_Data.drop(Cab\_Train\_Data[Cab\_Train\_Data[i]==0].index, axis=0)

for i in ['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']:

Cab\_Train\_Data.loc[Cab\_Train\_Data[i]==0,i] = np.nan

Cab\_Train\_Data.loc[Cab\_Train\_Data['pickup\_latitude']>90,'pickup\_latitude'] = np.nan

**Step-3: Missing Value Analysis**

After loading the data, the first step performed is missing value analysis where we compute the missing values or information present in the data set. Here the missing value analysis is computed by using the mean method, because it computes the missing values closest to the actual value when compared to the other methods like mean and mode .Since KNN imputation method is not working in my python environment because it is no supported by my version of python also tried a lot to import it many times, So used median method.

In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks. Some missing values are in form of NA. missing values left behind after outlier analysis; missing values can be in any form. Unfortunately, in this dataset we have found some missing values. Therefore, we will do some missing value analysis. Before imputed we selected random row no-1000 and made it NA, so that we will compare original value with imputed value and choose best method which will impute value closer to actual value.

**Following codes used to perform above task:**

**############### MISSING VALUE ANALYSIS ###################**

#Create dataframe with missing percentage

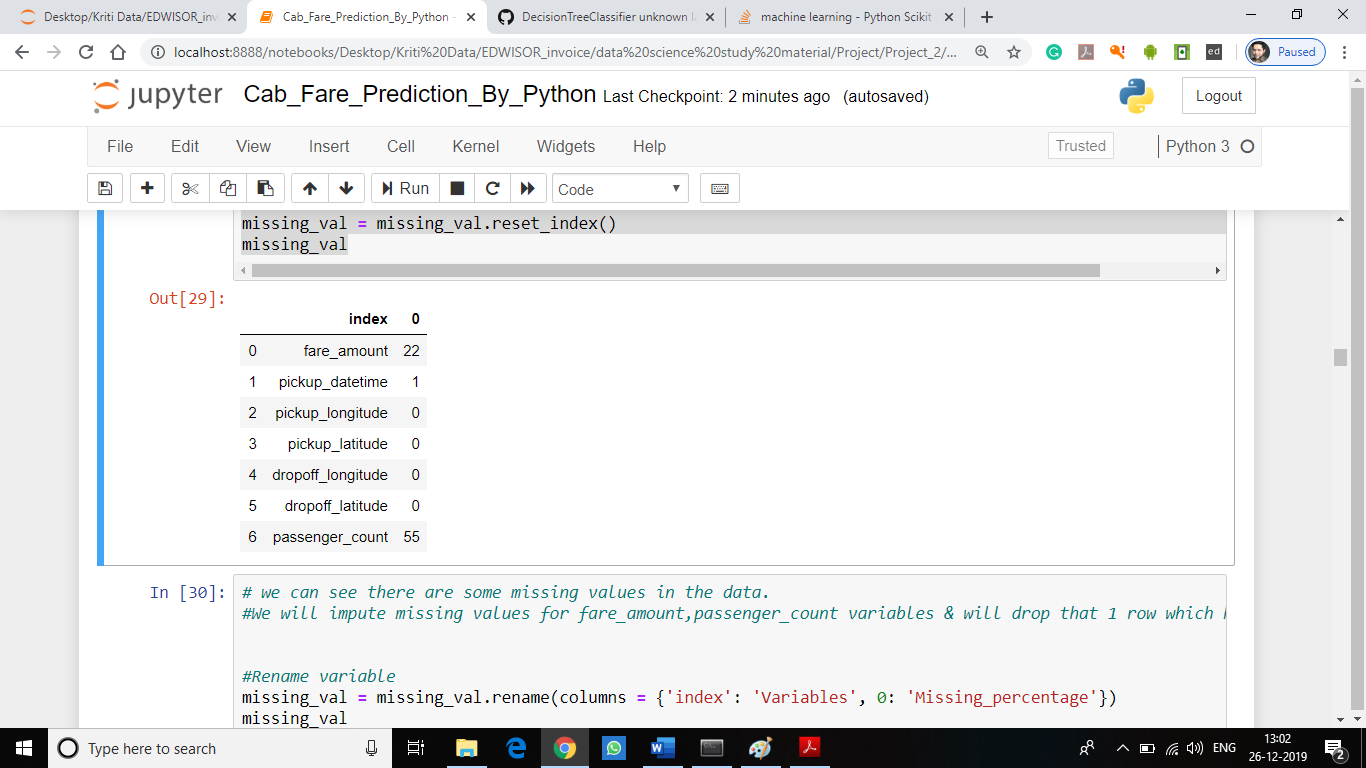
missing\_val = pd.DataFrame(Cab\_Train\_Data.isnull().sum())

#Reset index

missing\_val = missing\_val.reset\_index()

missing\_val

**Output:**



# we can see there are some missing values in the data.

#We will impute missing values for fare\_amount,passenger\_count variables & will drop that 1 row which has missing value in pickup\_datetime.

#Rename variable

missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_percentage'})

missing\_val

#Calculate percentage

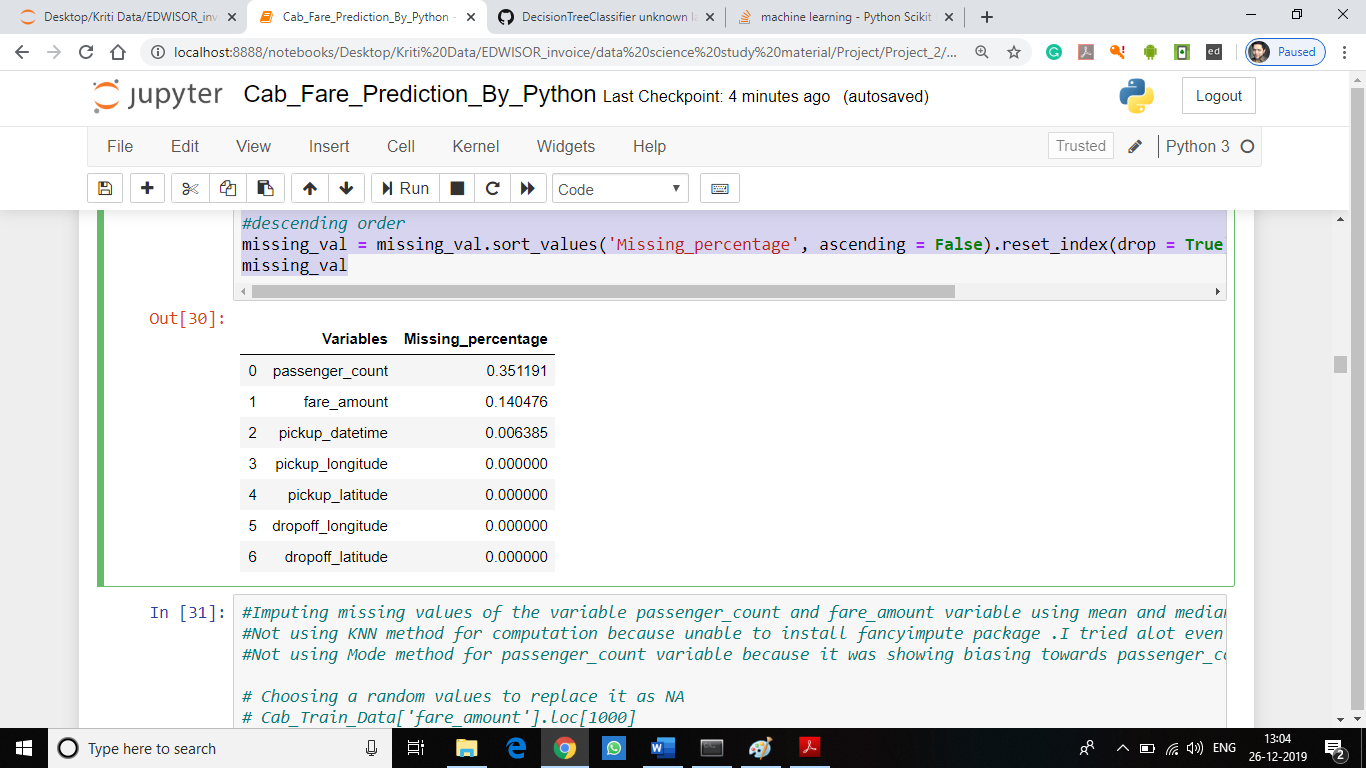
missing\_val['Missing\_percentage'] = (missing\_val['Missing\_percentage']/len(Cab\_Train\_Data))\*100

#descending order

missing\_val = missing\_val.sort\_values('Missing\_percentage', ascending = False).reset\_index(drop = True)

missing\_val

**Output:**



#Imputing missing values of the variable passenger\_count and fare\_amount variable using mean and median method

#Not using KNN method for computation because unable to install fancyimpute package. I tried a lot even perform many times of installing and uninstalling of the python

#Not using Mode method for passenger\_count variable because it was showing biasing towards passenger\_count =1

# Choosing a random value to replace it as NA

# Cab\_Train\_Data['fare\_amount'].loc[1000]

# Replacing 7.0 with NA

#Cab\_Train\_Data['fare\_amount'].loc[1000] = np.nan

#Cab\_Train\_Data['fare\_amount'].loc[1000]

#Imputing by mean method

#print('Value imputed by mean:{}'.format(Cab\_Train\_Data['fare\_amount'].fillna(Cab\_Train\_Data['fare\_amount'].mean()).loc[1000]))

# Imputing by median method

#print('Value imputed by median:{}'.format(Cab\_Train\_Data['fare\_amount'].fillna(Cab\_Train\_Data['fare\_amount'].median()).loc[1000]))

**#Missing value computed by different methods:**

**# Actual value = 7.0**

**# Mean = 15.117**

**# Median = 8.5**

#we will separate pickup\_datetime into a different dataframe and then merge with train in feature engineering step.

pickup\_datetime=pd.DataFrame(Cab\_Train\_Data['pickup\_datetime'])

#dropping pickup\_datetime variable from train dataset

Cab\_Train\_Data=Cab\_Train\_Data.drop('pickup\_datetime',axis=1)

#Since value computed by median method is closer to the actual value when compared to mean method so will be using median method to compute missing values present in the dataset.

Cab\_Train\_Data["fare\_amount"]=Cab\_Train\_Data["fare\_amount"].fillna(Cab\_Train\_Data["fare\_amount"].median())

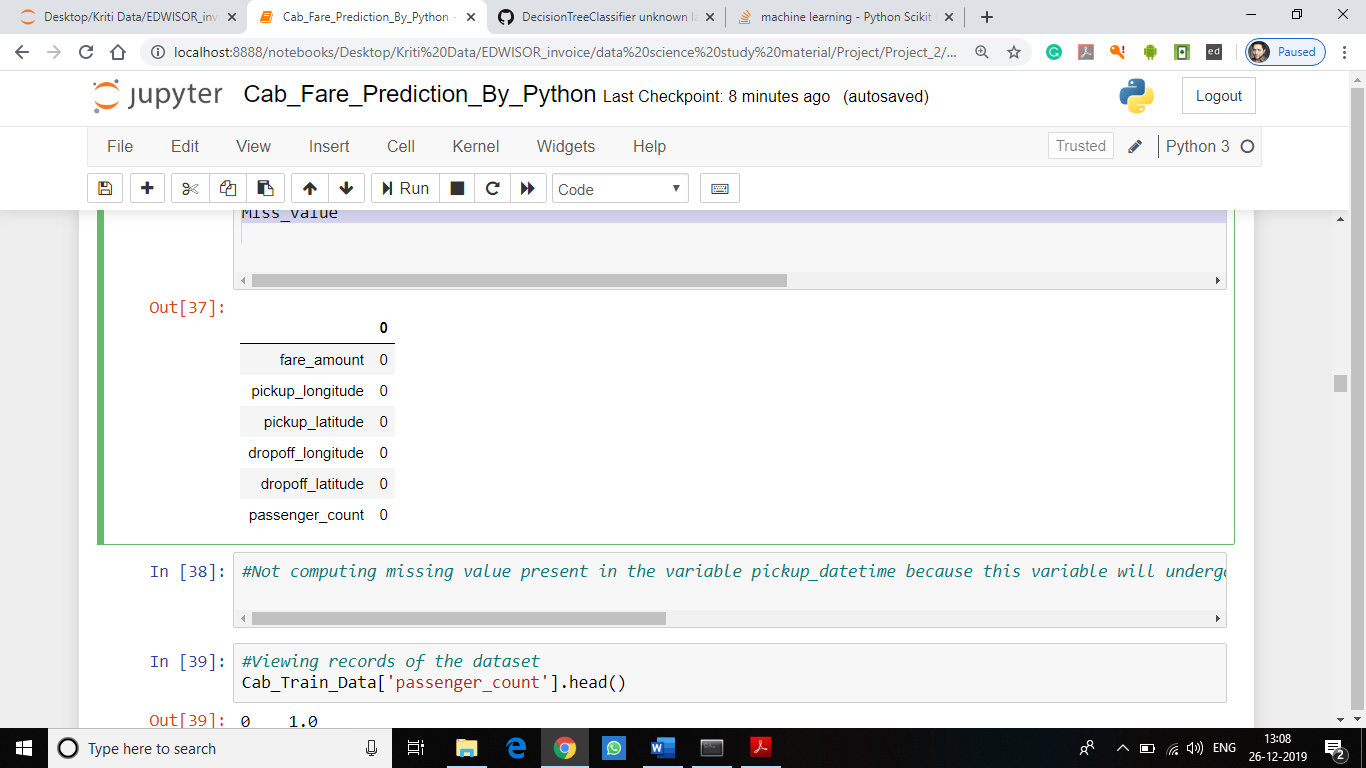
Cab\_Train\_Data["passenger\_count"]=Cab\_Train\_Data["passenger\_count"].fillna(Cab\_Train\_Data["passenger\_count"].median())

#Checking the number of missing values in the variables after computing missing value.

Miss\_value=pd.DataFrame(Cab\_Train\_Data.isnull().sum())

Miss\_value

**Output:**



**Step-3: Outliers Detection & Removal**

**Outliers Detection:**

Once, the missing values are computed, the next step is to detect and remove outliers present in the data set. These outliers are nothing but the extreme values present in the data set.

In this project, missing values are detected using Box-plot method because it gives the graphical representation of the presence of the outliers in the data set and completely distinguish the outliers present in the lower and upper fence of the box plot. Using replace outlier by NA method in out dataset as deleting outliers can cause information loss which might be important for the computation of the model.

**Following codes used to perform above task:**

**################## OUTLIER ANALYSIS ###################**

#Outlier anaysis on fare\_amount variable

plt.figure(figsize=(20,5))

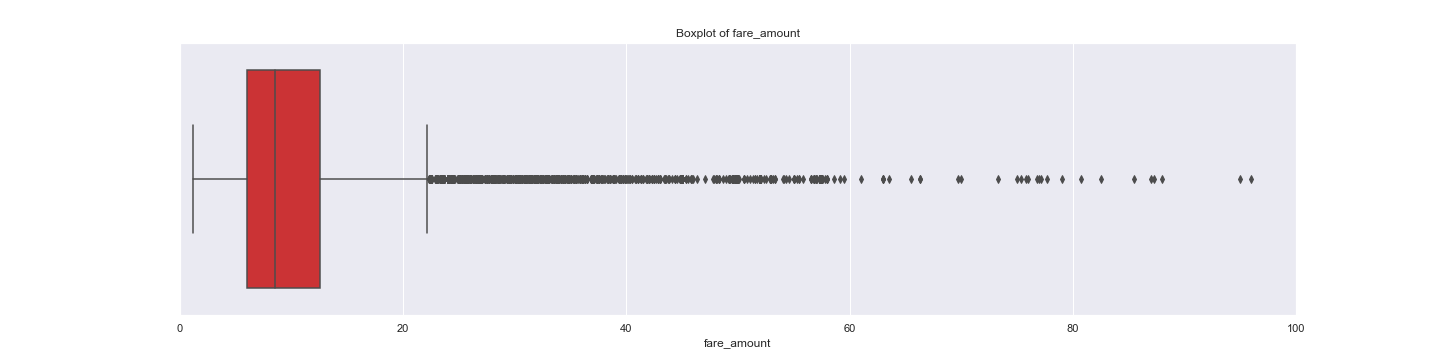
plt.xlim(0,100)

sns.boxplot(x=Cab\_Train\_Data['fare\_amount'],data=Cab\_Train\_Data,orient='h')

plt.title('Boxplot of fare\_amount')

plt.savefig('bp\_of\_fare\_amount\_python.png')

plt.show()



**Fig 4: Boxplot for fare\_amount variable**

#Bivariate Boxplots (Boxplot for Numerical Variable Vs Categorical Variable)

plt.figure(figsize=(20,10))

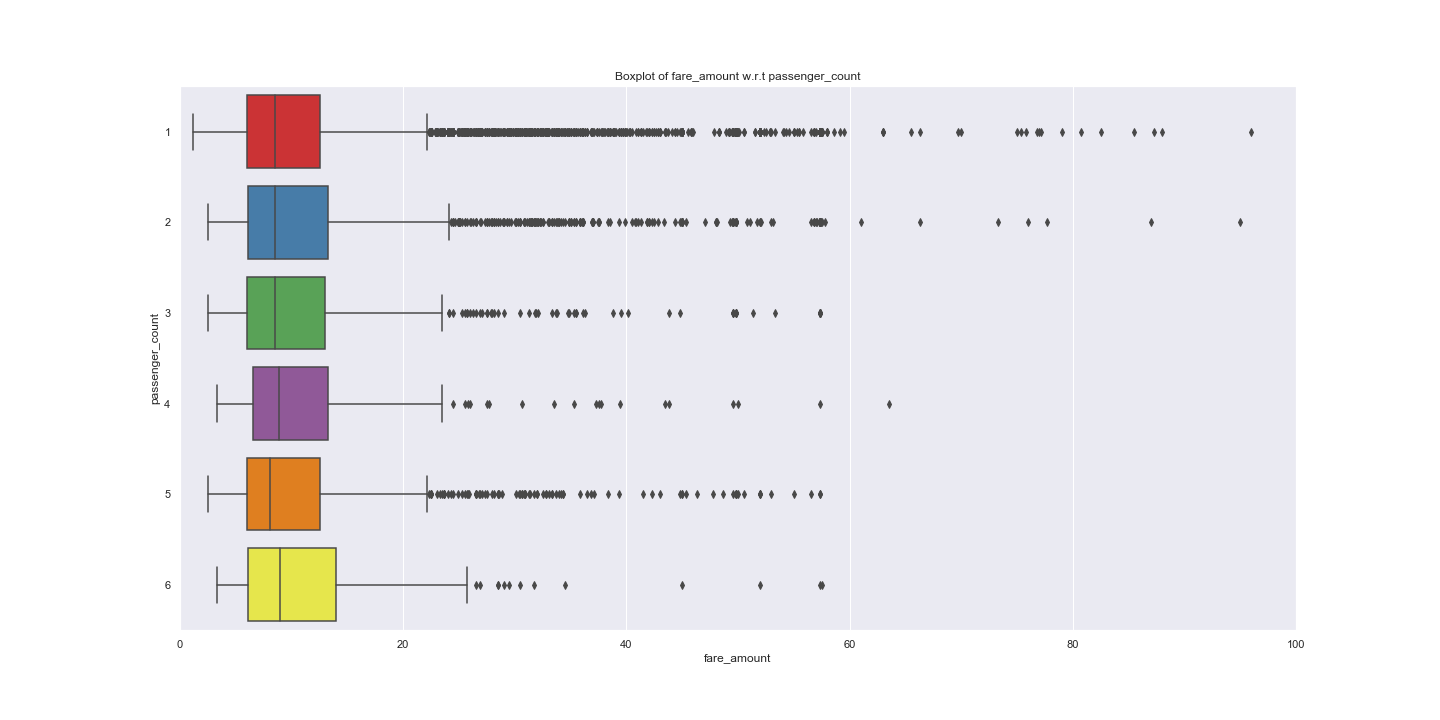
plt.xlim(0,100)

\_ = sns.boxplot(x=Cab\_Train\_Data['fare\_amount'],y=Cab\_Train\_Data['passenger\_count'],data=Cab\_Train\_Data,orient='h')

plt.title('Boxplot of fare\_amount w.r.t passenger\_count')

plt.savefig('Boxplot\_of\_fare\_amount\_w.r.t\_passenger\_count\_python.png')

plt.show()



**Fig 5: Boxplot of fare\_amount variable with respect to passenger\_count variable**

**Outlier Removal:**

Outlier replaced by NA method: This method is chosen because we cannot afford deletion of the outliers as sometimes outliers contains important information. After replacing outliers with NA, the NA values are further computed by the median method. Replacing outliers with NA present in the dataset as this is the best method to deal with the NA as taught in lecture.

**Following codes used to perform above task:**

#Calculating outlier and replacing them with NA

def outlier\_treatment(col):

#Extract quartiles

q75, q25 = np.percentile(Cab\_Train\_Data[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\_Data.loc[Cab\_Train\_Data[col] < minimum,col] = np.nan

Cab\_Train\_Data.loc[Cab\_Train\_Data[col] > maximum,col] = np.nan

#Outlier analysis

outlier\_treatment('fare\_amount')

outlier\_treatment('pickup\_longitude')

outlier\_treatment('pickup\_latitude')

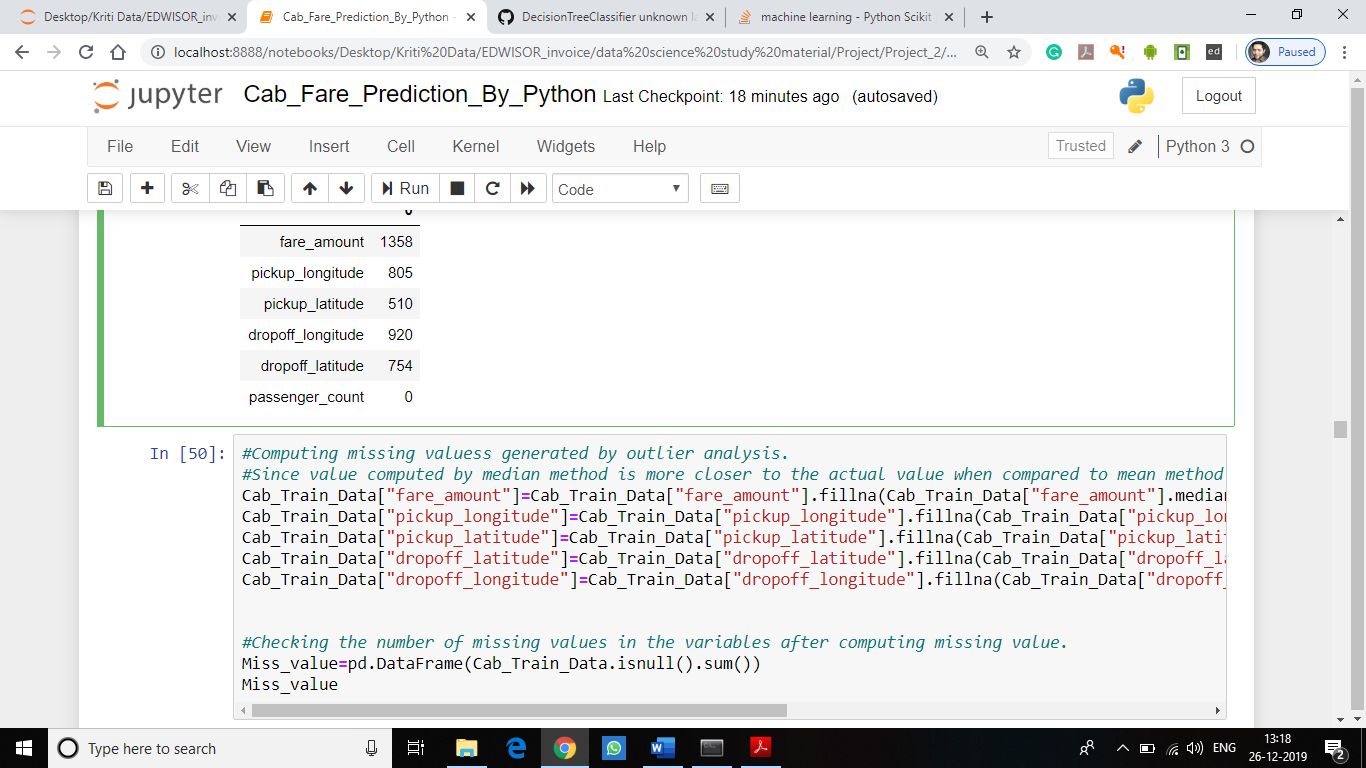
outlier\_treatment('dropoff\_longitude')

outlier\_treatment('dropoff\_latitude')

#Check for null values generated by outliers

pd.DataFrame(Cab\_Train\_Data.isnull().sum())

**Output:**



#Computing missing values generated by outlier analysis.

#Since value computed by median method is closer to the actual value when compared to mean method so will be using median method to compute missing values present in the dataset.

Cab\_Train\_Data["fare\_amount"]=Cab\_Train\_Data["fare\_amount"].fillna(Cab\_Train\_Data["fare\_amount"].median())

Cab\_Train\_Data["pickup\_longitude"]=Cab\_Train\_Data["pickup\_longitude"].fillna(Cab\_Train\_Data["pickup\_longitude"].median())

Cab\_Train\_Data["pickup\_latitude"]=Cab\_Train\_Data["pickup\_latitude"].fillna(Cab\_Train\_Data["pickup\_latitude"].median())

**Cab\_Train\_Data["dropoff\_latitude"]=Cab\_Train\_Data["dropoff\_latitude"].fillna(Cab\_Train\_Data["dropoff\_latitude"].median())**

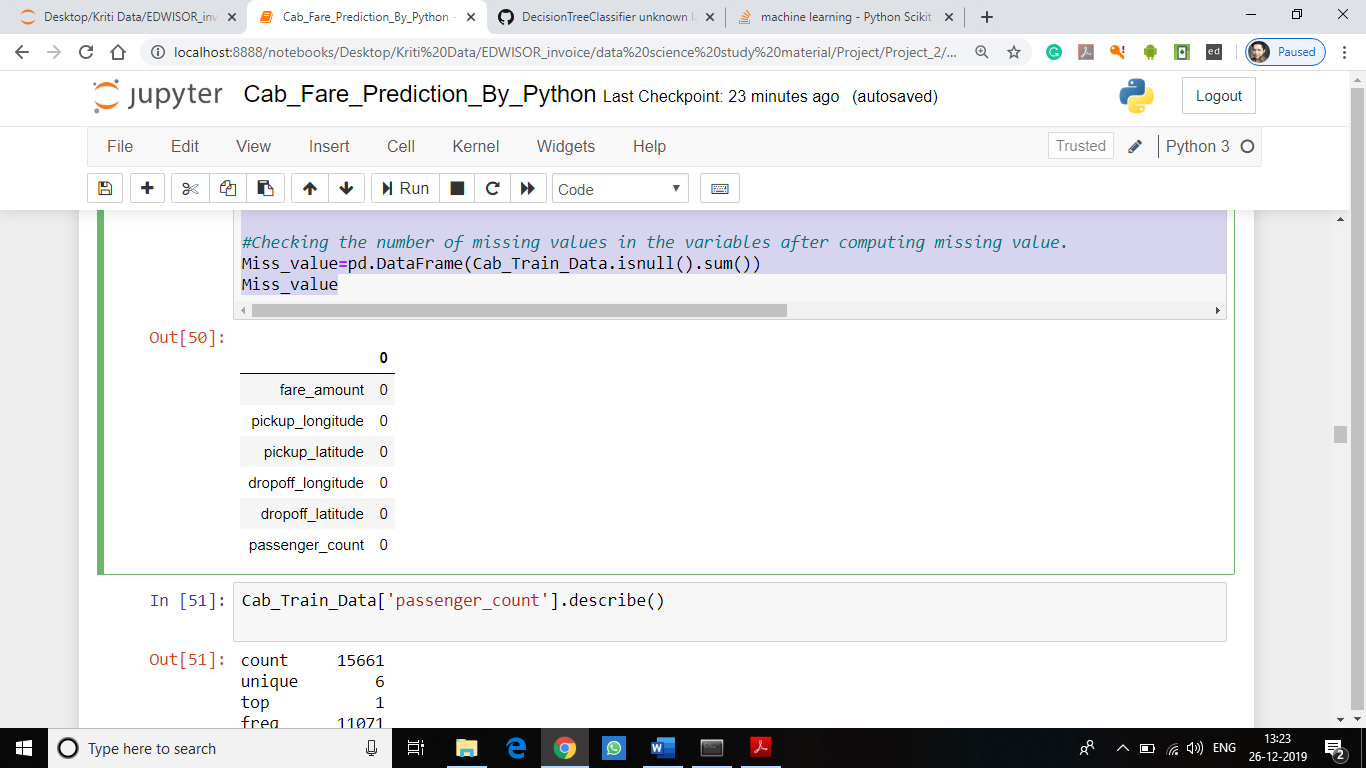
**Cab\_Train\_Data["dropoff\_longitude"]=Cab\_Train\_Data["dropoff\_longitude"].fillna(Cab\_Train\_Data["dropoff\_longitude"].median())**

#Checking the number of missing values in the variables after computing missing value.

Miss\_value=pd.DataFrame(Cab\_Train\_Data.isnull().sum())

Miss\_value

**Output:**



**STEP 4: Feature Engineering**

**1. ‘pickup\_datetime’ variable:**

Feature Engineering is used to drive new features from existing features. 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.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.

**2. ‘passenger\_count’ variable:**

As passenger\_count is a categorical variable we will one-hot-encode it.

**3**. **‘Latitudes’ and ‘Longitudes’ variables:**

As we have latitude and longitude data for pickup and dropoff, we will find the distance the cab travelled from pickup and dropoff location using geodesic method from geopy library.

**Following codes used to perform above task:**

**################## FEATURE ENGINEERING ###################**

#Deriving new features from pickup\_datetime variables like year,month,day\_of\_week,hour

#Joining two Dataframes pickup\_datetime and train

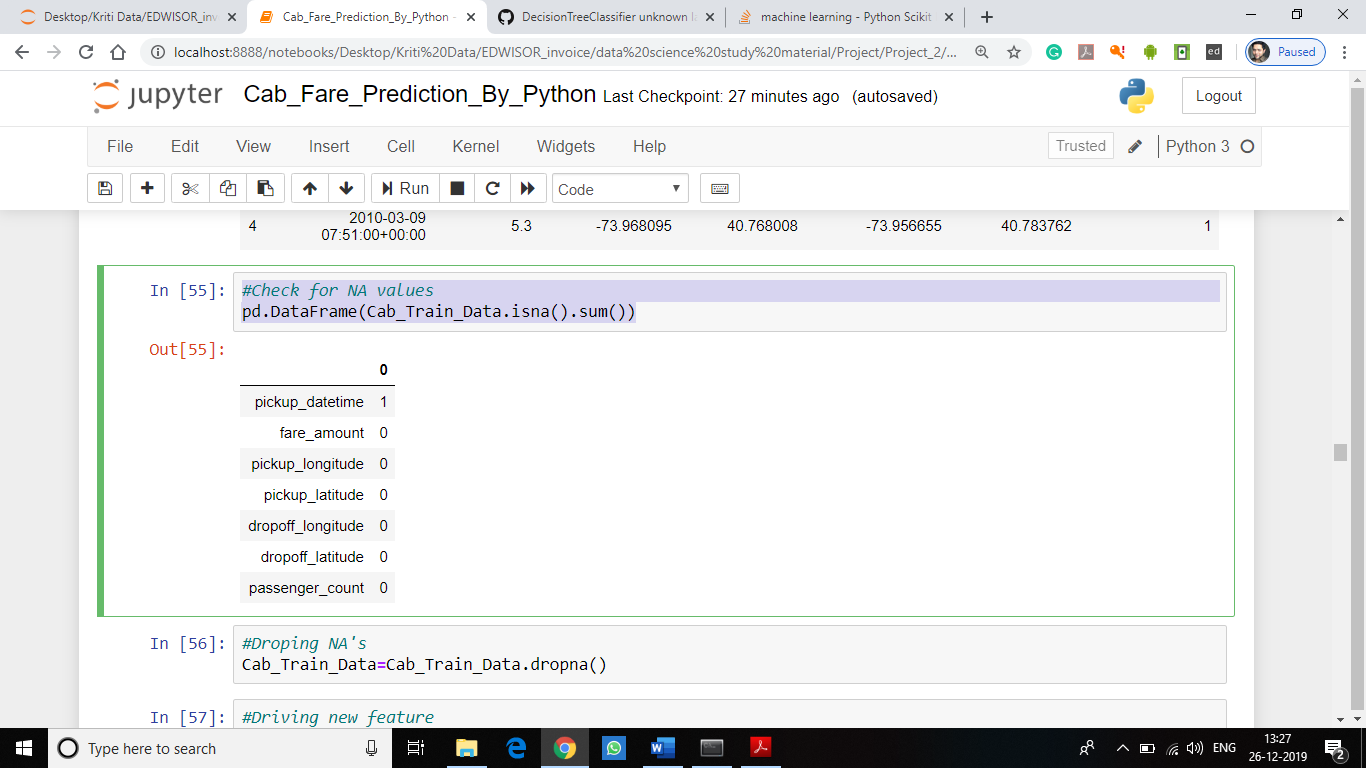
Cab\_Train\_Data = pd.merge(pickup\_datetime,Cab\_Train\_Data,right\_index=True,left\_index=True)

Cab\_Train\_Data.head()

#Check for NA values

pd.DataFrame(Cab\_Train\_Data.isna().sum())

**Ouptut:**



#Droping NA's

Cab\_Train\_Data=Cab\_Train\_Data.dropna()

#Driving new feature

data=[Cab\_Train\_Data,Cab\_Test\_Data]

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)

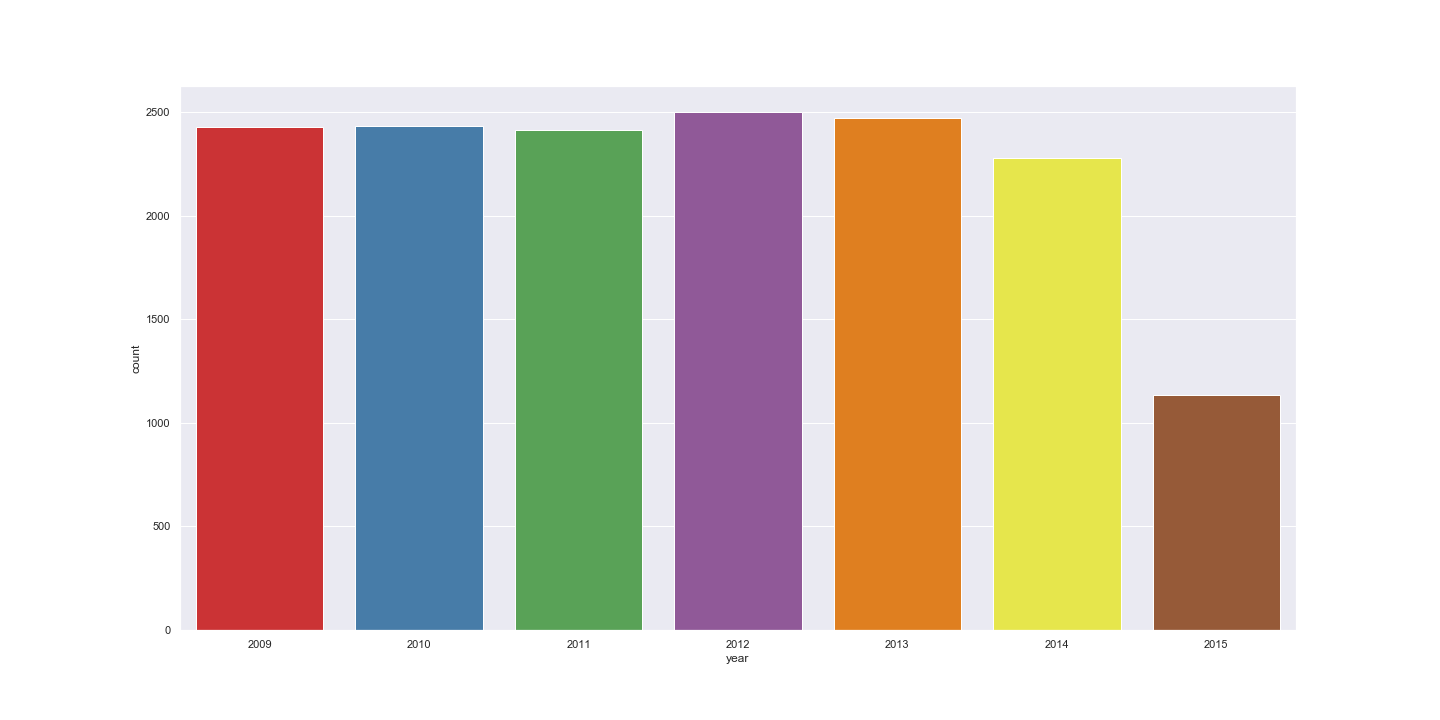
i["hour"] = i["pickup\_datetime"].apply(lambda row: row.hour)

#Plotting new features

plt.figure(figsize=(20,10))

sns.countplot(Cab\_Train\_Data['year'])

plt.savefig('year\_python.png')

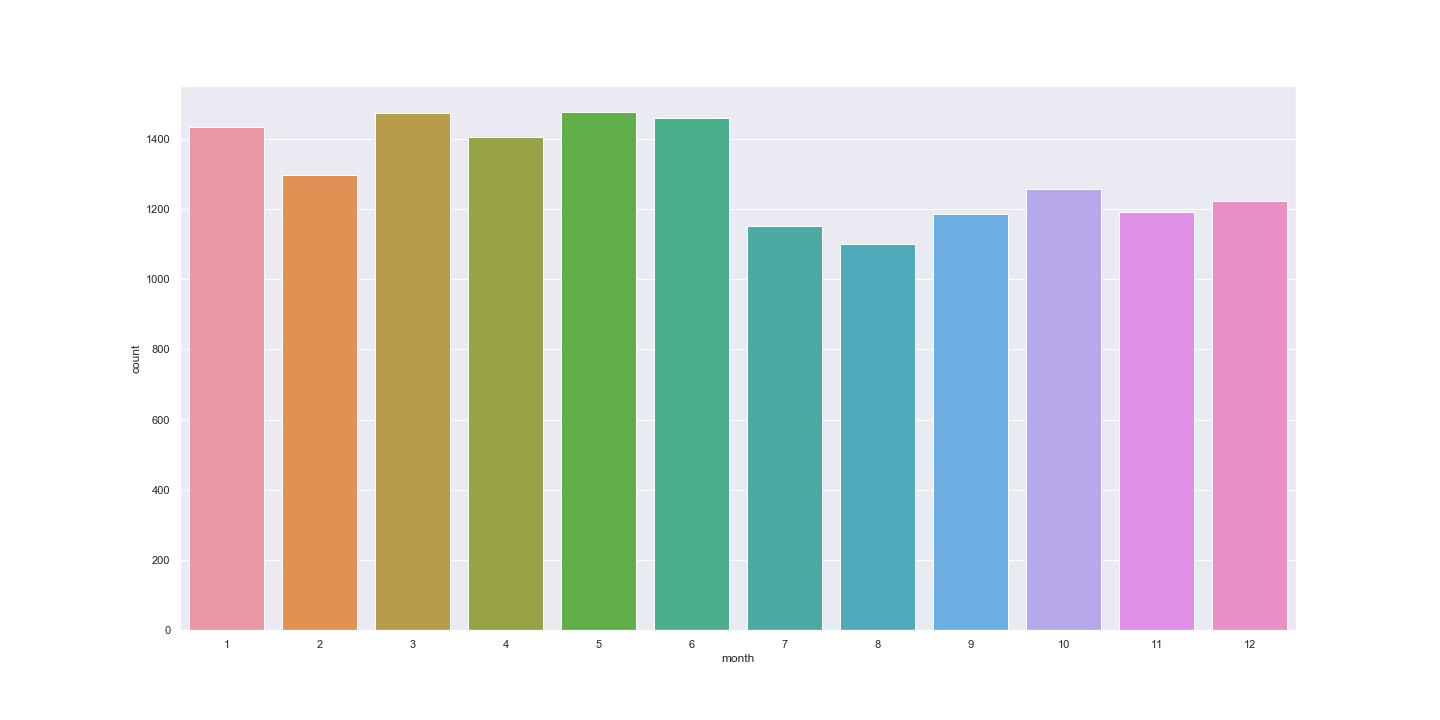


**Fig 6: Plot for year variable**

plt.figure(figsize=(20,10))

sns.countplot(Cab\_Train\_Data['month'])

plt.savefig('month\_python.png')

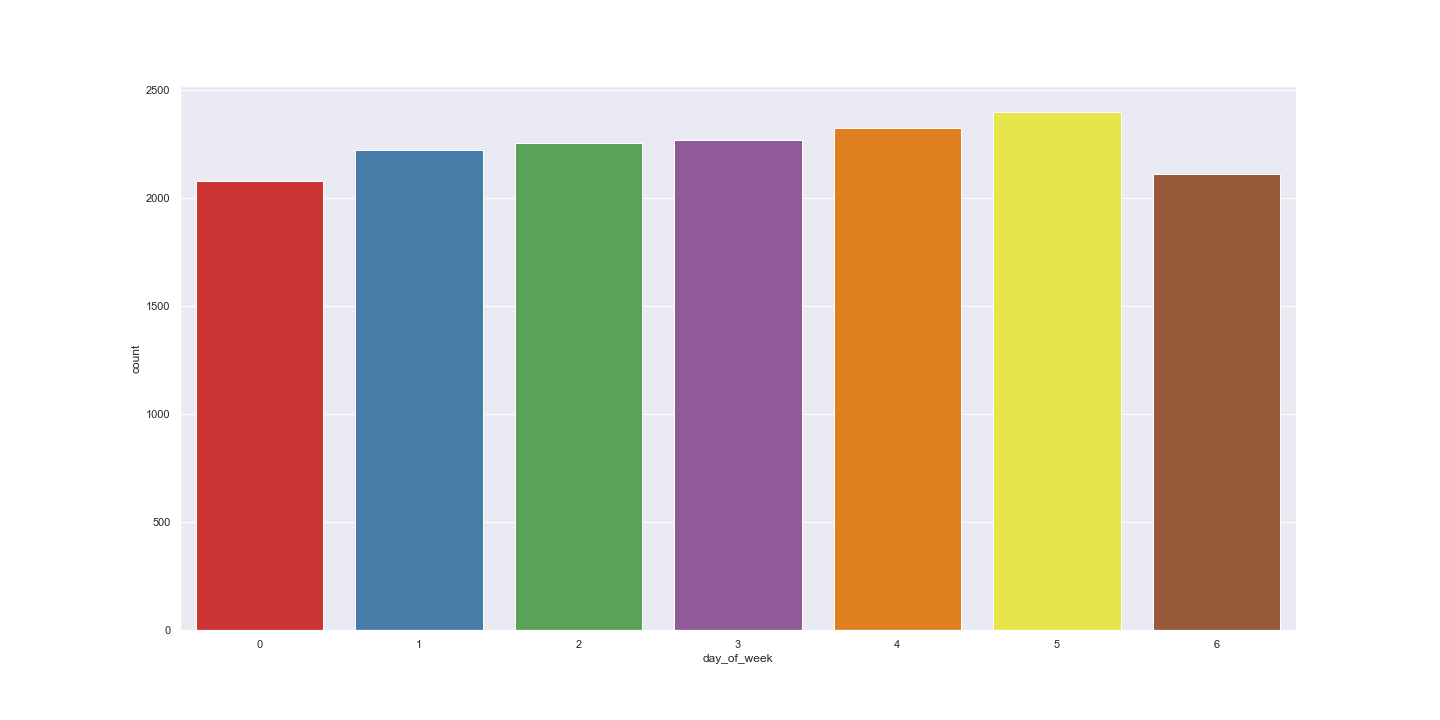


**Fig 7: Plot for month variable**

plt.figure(figsize=(20,10))

sns.countplot(Cab\_Train\_Data['day\_of\_week'])

plt.savefig('day\_of\_week\_python.png')

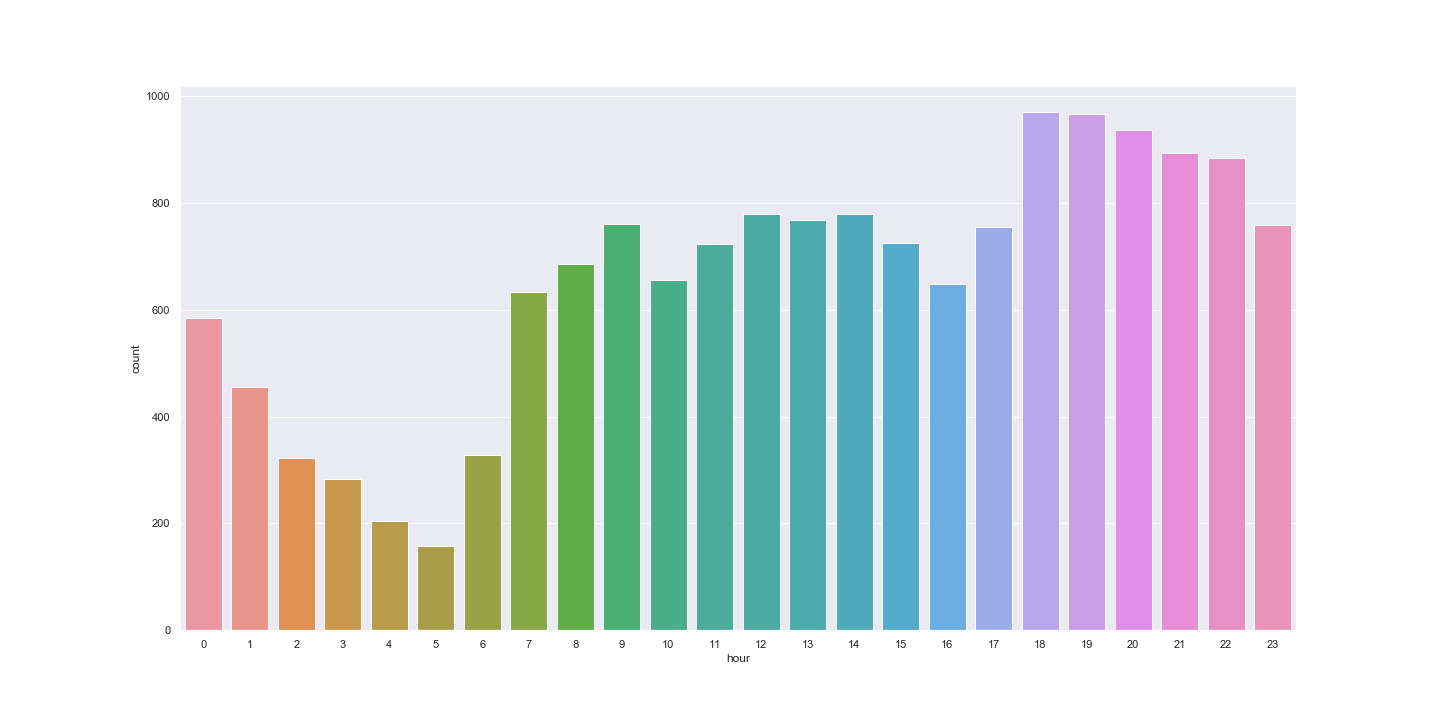


**Fig 8: Plot for month variable**

plt.figure(figsize=(20,10))

sns.countplot(Cab\_Train\_Data['hour'])

plt.savefig('hour\_python.png')



**Fig 9: Plot for month variable**

#Using month,day\_of\_week,hour for deriving new features like sessions in a day,seasons in a year,week:weekend/weekday

#function for sessions in a day using hour variable

def session(x):

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'

#function for seasons in a year using month variable

def season(x):

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'

#function for weekday/weekend in a day\_of\_week variable

def day(x):

if (x >=0) and (x <= 4):

return 'weekday'

elif (x >=5) and (x <=6 ):

return 'weekend'

#Using session function for deriving session variable from hour

Cab\_Train\_Data['session'] = Cab\_Train\_Data['hour'].apply(session)

Cab\_Test\_Data['session'] = Cab\_Test\_Data['hour'].apply(session)

#Using seasons function for deriving season variable from month

Cab\_Train\_Data['seasons'] =Cab\_Train\_Data['month'].apply(season)

Cab\_Test\_Data['seasons'] = Cab\_Test\_Data['month'].apply(season)

#Using day function for deriving day variable from day\_of\_week

Cab\_Train\_Data['week'] = Cab\_Train\_Data['day\_of\_week'].apply(day)

Cab\_Test\_Data['week'] = Cab\_Test\_Data['day\_of\_week'].apply(day)

#2. Feature Engineering for passenger\_count variable

#Models in scikit learn require numerical input,if dataset contains categorical variables then we have to encode them,using one hot encoding technique for passenger\_count variable.

temp = pd.get\_dummies(Cab\_Train\_Data['passenger\_count'], prefix = 'passenger\_count')

Cab\_Train\_Data =Cab\_Train\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Test\_Data['passenger\_count'], prefix = 'passenger\_count')

Cab\_Test\_Data = Cab\_Test\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Train\_Data['seasons'], prefix = 'season')

Cab\_Train\_Data = Cab\_Train\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Test\_Data['seasons'], prefix = 'season')

Cab\_Test\_Data = Cab\_Test\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Train\_Data['week'], prefix = 'week')

Cab\_Train\_Data = Cab\_Train\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Test\_Data['week'], prefix = 'week')

Cab\_Test\_Data = Cab\_Test\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Train\_Data['session'], prefix = 'sessions')

Cab\_Train\_Data = Cab\_Train\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Test\_Data['session'], prefix = 'sessions')

Cab\_Test\_Data = Cab\_Test\_Data.join(temp)

temp = pd.get\_dummies(Cab\_Train\_Data['year'], prefix = 'year')

Cab\_Train\_Data = Cab\_Train\_Data.join(temp)

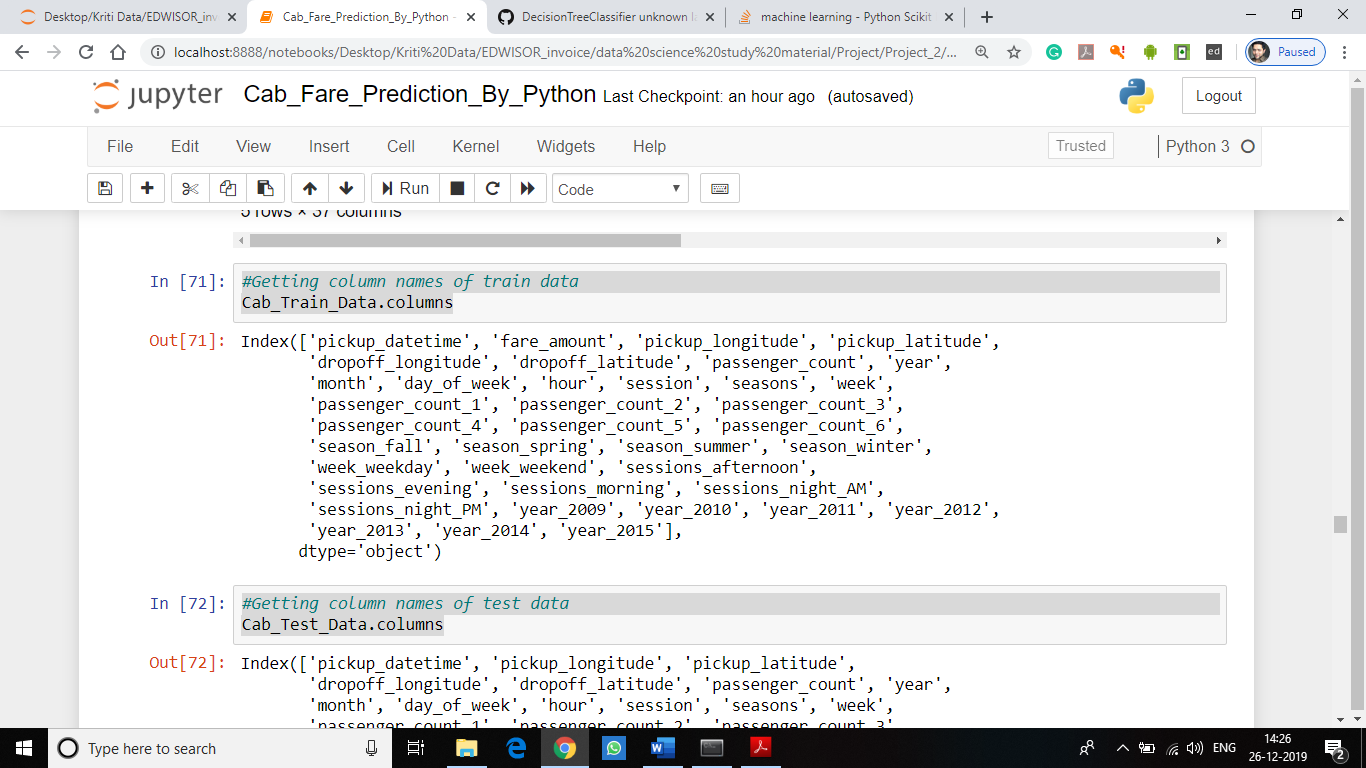
temp = pd.get\_dummies(Cab\_Test\_Data['year'], prefix = 'year')

Cab\_Test\_Data = Cab\_Test\_Data.join(temp)

#Getting column names of train data

Cab\_Train\_Data.columns

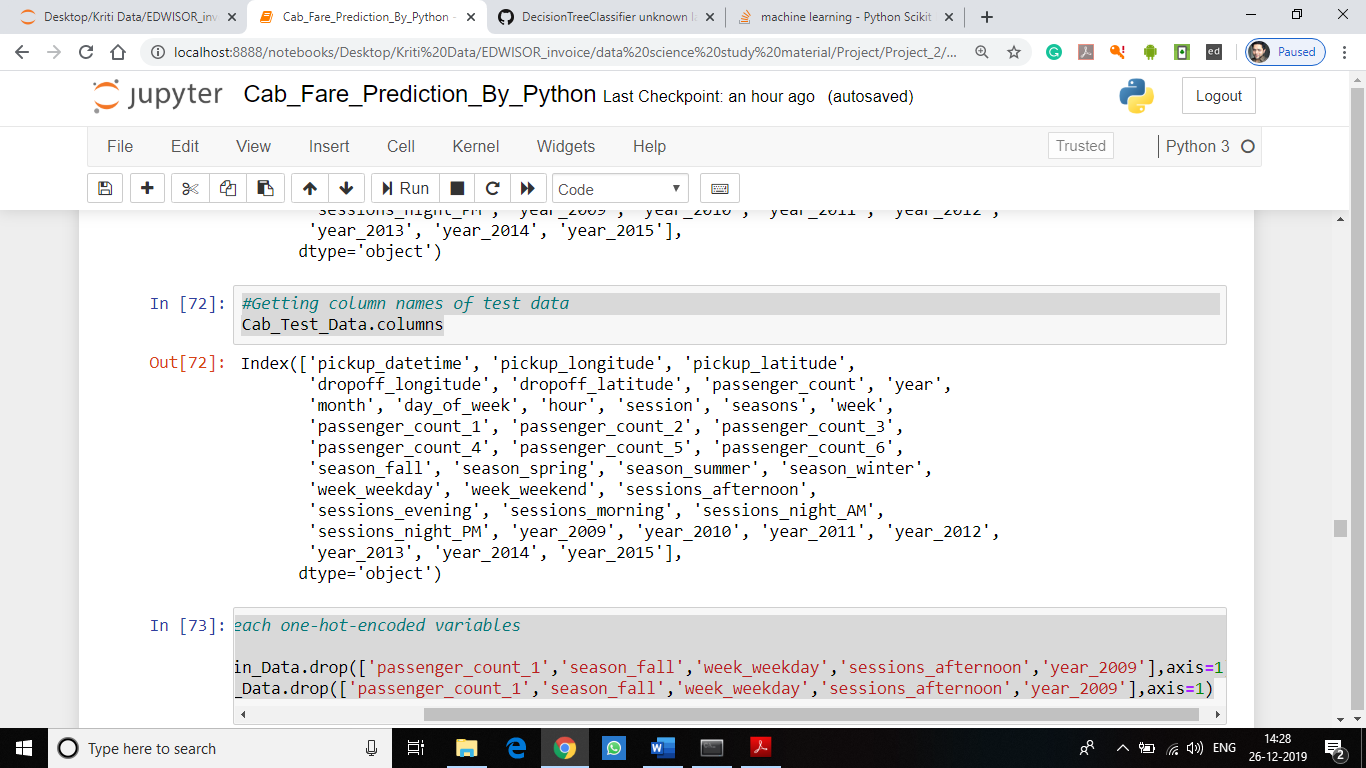
**Output:**



#Getting column names of test data

Cab\_Test\_Data.columns

**Output:**



#drop one column from each one-hot-encoded variables

Cab\_Train\_Data=Cab\_Train\_Data.drop(['passenger\_count\_1','season\_fall','week\_weekday','sessions\_afternoon','year\_2009'],axis=1)

Cab\_Test\_Data=Cab\_Test\_Data.drop(['passenger\_count\_1','season\_fall','week\_weekday','sessions\_afternoon','year\_2009'],axis=1)

#3. Feature Engineering for latitude and longitude variable

#finding the distance, the cab travelled from pickup and dropoff longitudes and latitudes.

# Calculate distance the cab travelled from pickup and dropoff location using geodesic from geopy library

data=[Cab\_Train\_Data,Cab\_Test\_Data]

for i in data:

i['distance']=i.apply(lambda x: geodesic((x['pickup\_latitude'],x['pickup\_longitude']), (x['dropoff\_latitude'], x['dropoff\_longitude'])).miles, axis=1)

#Removing variables which were used to feature engineer new variables

Cab\_Train\_Data=Cab\_Train\_Data.drop(['pickup\_datetime','pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons', 'week'],axis=1)

Cab\_Test\_Data=Cab\_Test\_Data.drop(['pickup\_datetime','pickup\_longitude', 'pickup\_latitude',

'dropoff\_longitude', 'dropoff\_latitude', 'passenger\_count', 'year',

'month', 'day\_of\_week', 'hour', 'session', 'seasons', 'week'],axis=1)

#Check for outliers in distance

plt.figure(figsize=(20,5))

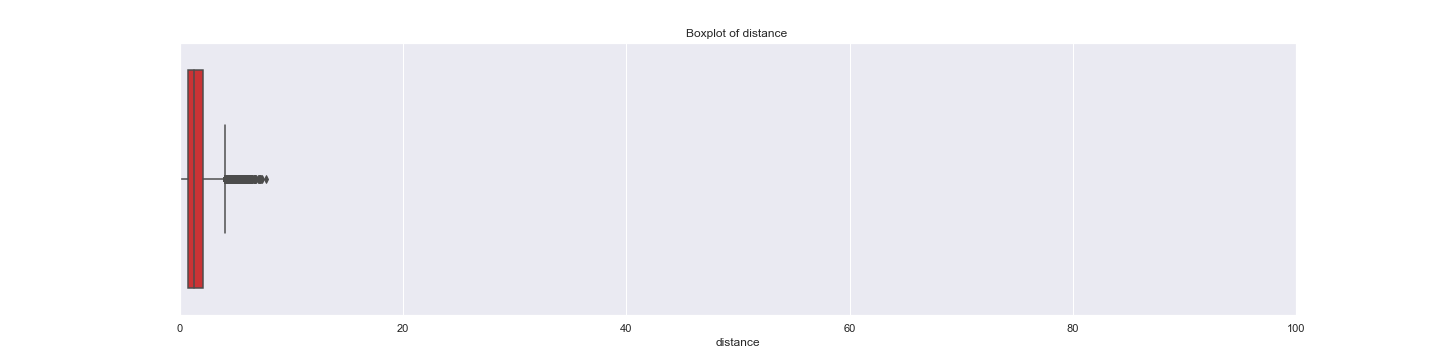
plt.xlim(0,100)

sns.boxplot(x=Cab\_Train\_Data['distance'],data=Cab\_Train\_Data,orient='h')

plt.title('Boxplot of distance ')

plt.savefig('bp\_distance\_python.png')

plt.show()



**Fig 10: BoxPlot for distance variable**

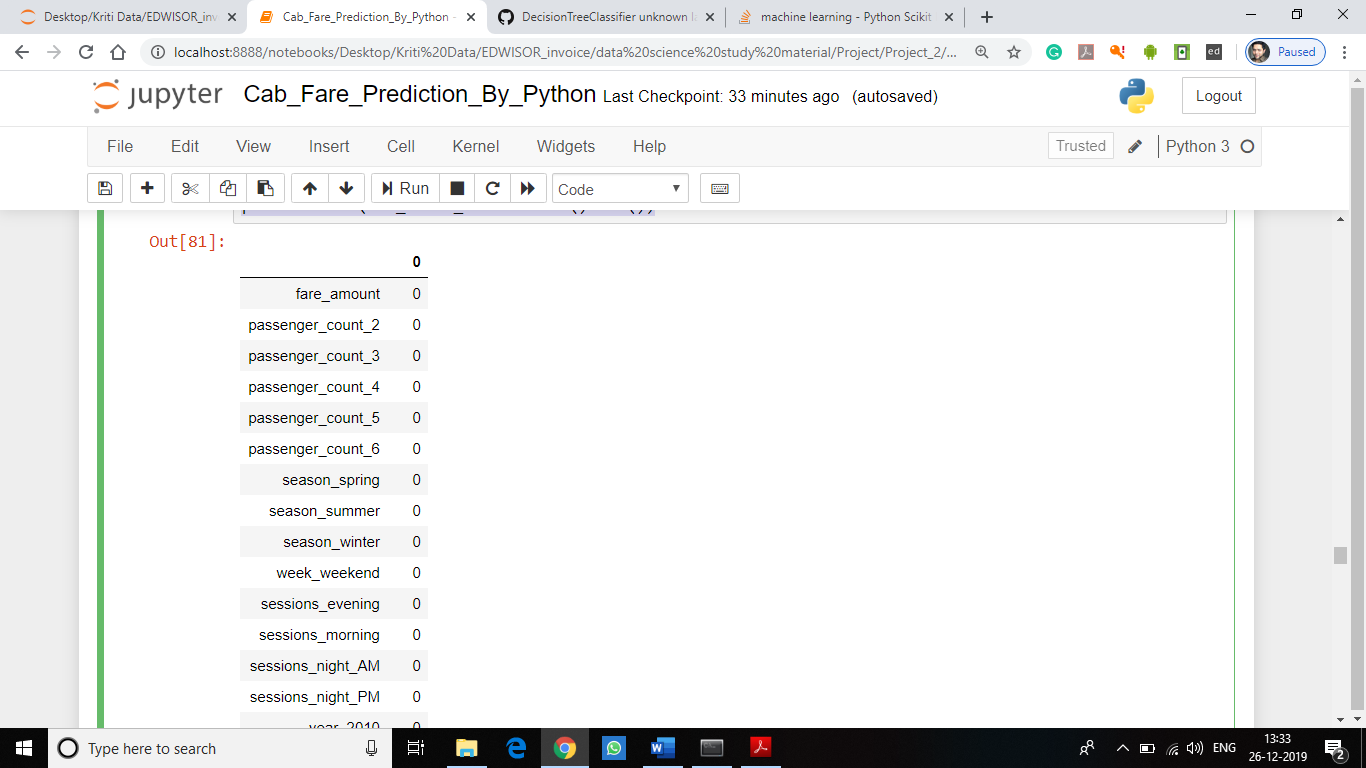
#Calling outlier function and replacing outliers with NA

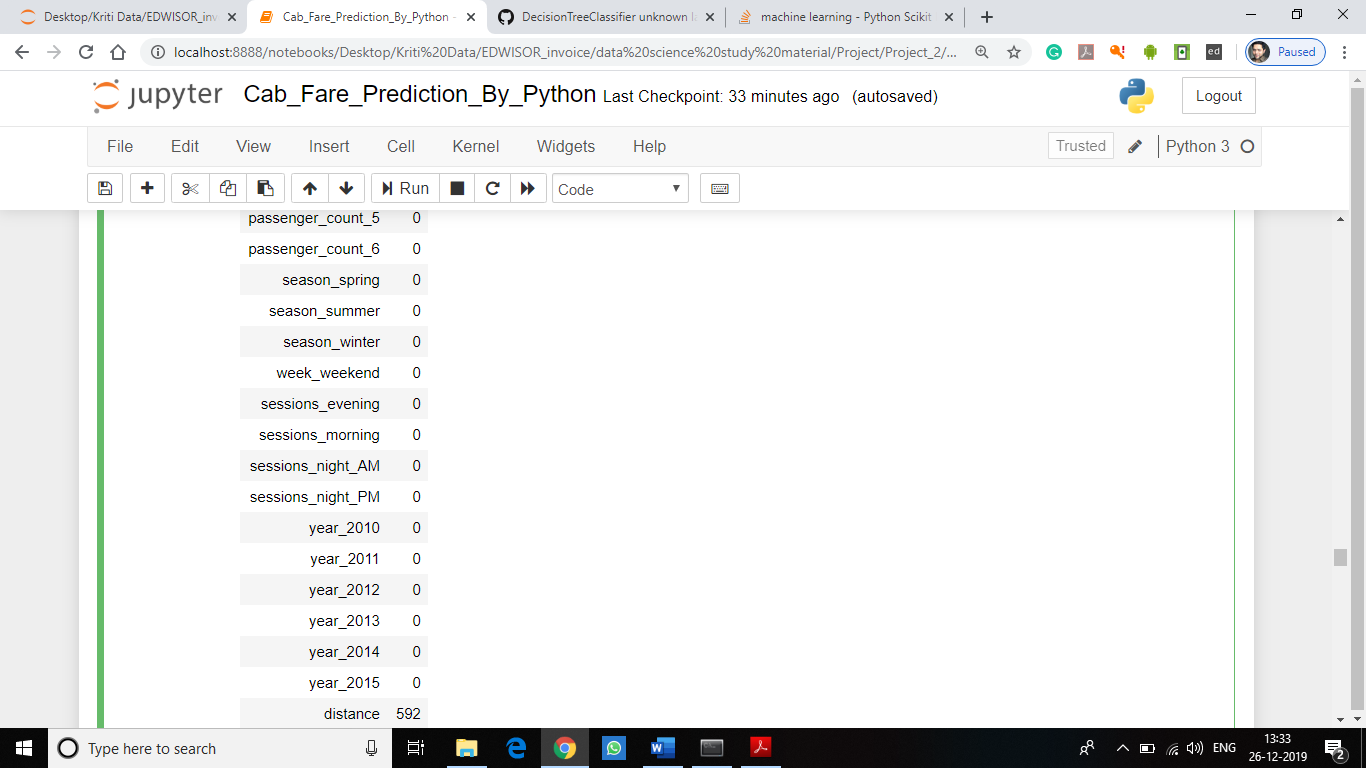
outlier\_treatment('distance')

#Check for NA values

pd.DataFrame(Cab\_Train\_Data.isnull().sum())

**Output:**





#Computing missing values generated by outlier analysis.

#Since value computed by median method is closer to the actual value when compared to mean method so will be using median method to compute missing values present in the dataset.

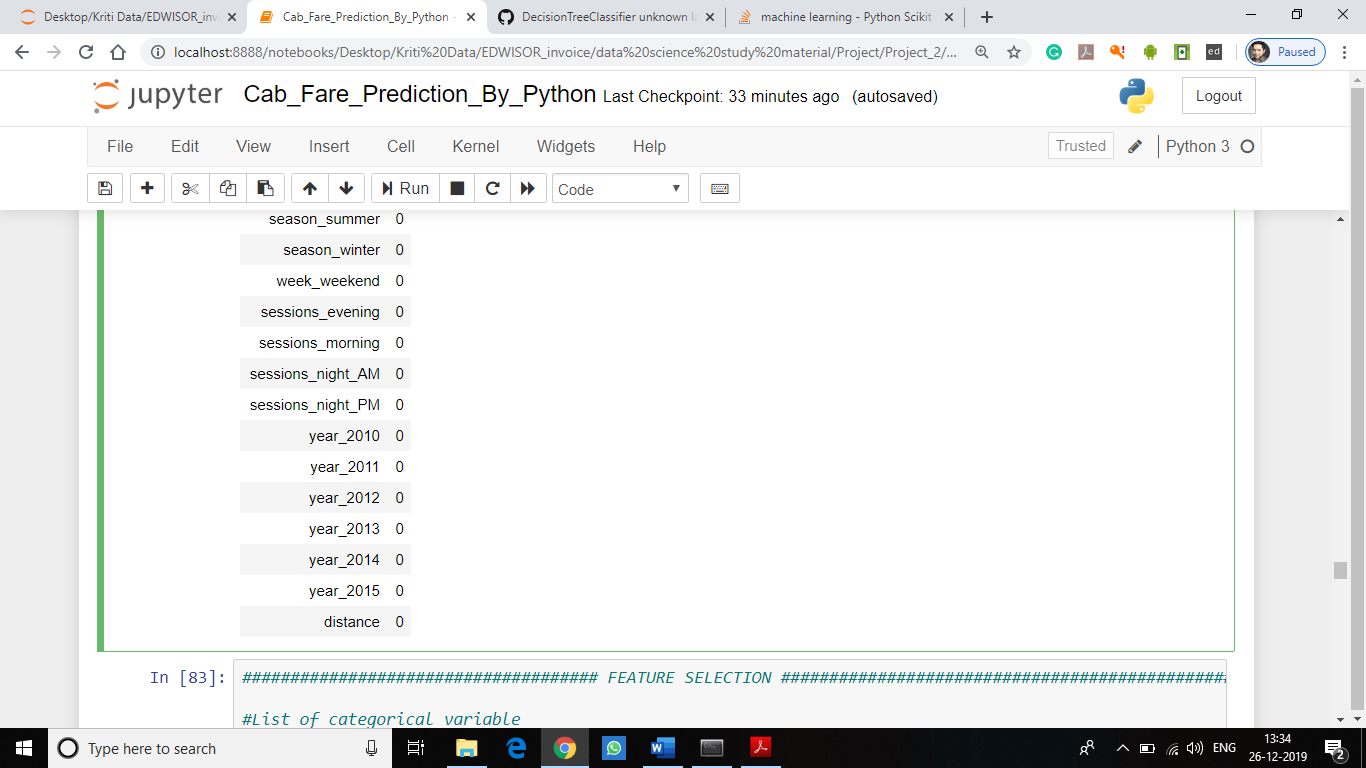
Cab\_Train\_Data["distance"]=Cab\_Train\_Data["distance"].fillna(Cab\_Train\_Data["distance"].median())

#Checking the number of missing values in the variables after computing missing value.

Miss\_value=pd.DataFrame(Cab\_Train\_Data.isnull().sum())

Miss\_value

**Output:**



**STEP-5:** **Feature selection**

This stage involves the process of reducing variables on the basis of correlation present in the variables of the dataset. Correlation plot is being used to analyze the correlation among the variables and reduce the dimensionality on the basis of correlation between variables. 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.

**Following codes used to perform above task:**

**################## FEATURE SELECTION ###################**

**################## CORRELATION PLOT ####################**

#Plotting correlation graph for the variables to look the correlation among variables.

# heatmap using correlation matrix

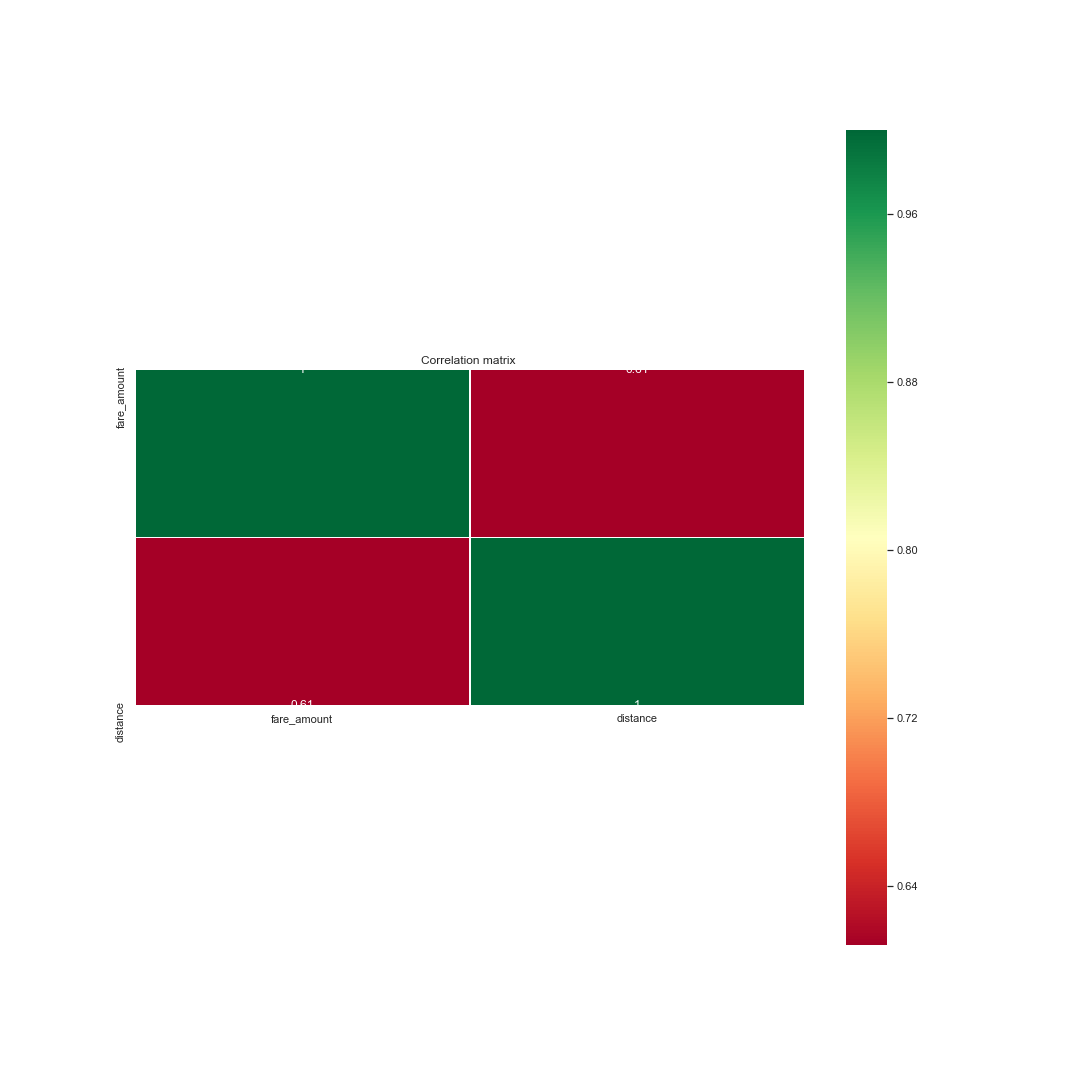
plt.figure(figsize=(15,15))

\_ = sns.heatmap(Cab\_Train\_Data[num\_var].corr(), square=True, cmap='RdYlGn',linewidths=0.5,linecolor='w',annot=True)

plt.title('Correlation matrix ')

plt.savefig('correlation.png')

plt.show()



**Fig 11: Correlation plot**

From above correlation plot we see that:

• 'fare\_amount' and 'distance' are very highly correlated with each other.

• As fare\_amount is the target variable and ‘distance’ is independent variable we will keep ‘distance’ because it will help to explain variation in fare\_amount.

**2. Analysis of Variance (Anova) Test –** It is carried out to compare between each group in a categorical variable. ANOVA only lets us know the means for different groups are same or not. It doesn’t help us identify which mean is different.

Hypothesis testing:

- **Null Hypothesis**: mean of all categories in a variable are same.

- **Alternate Hypothesis**: mean of at least one category in a variable is different.

If p-value is less than 0.05 then we reject the null hypothesis and if p-value is greater than 0.05 then we accept the null hypothesis.

**Following codes used to perform above task:**

**################## ANOVA TEST ###################**

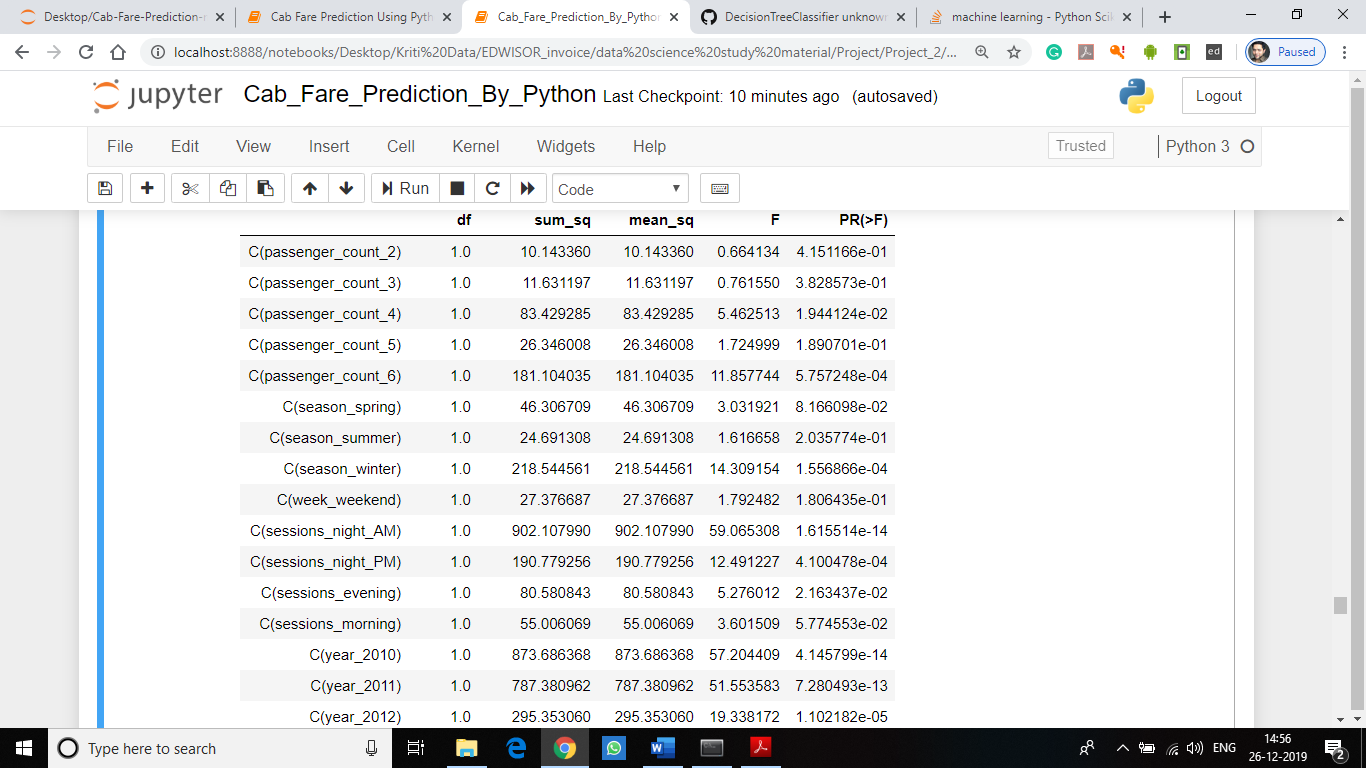
#ANOVA Test for variance analysis

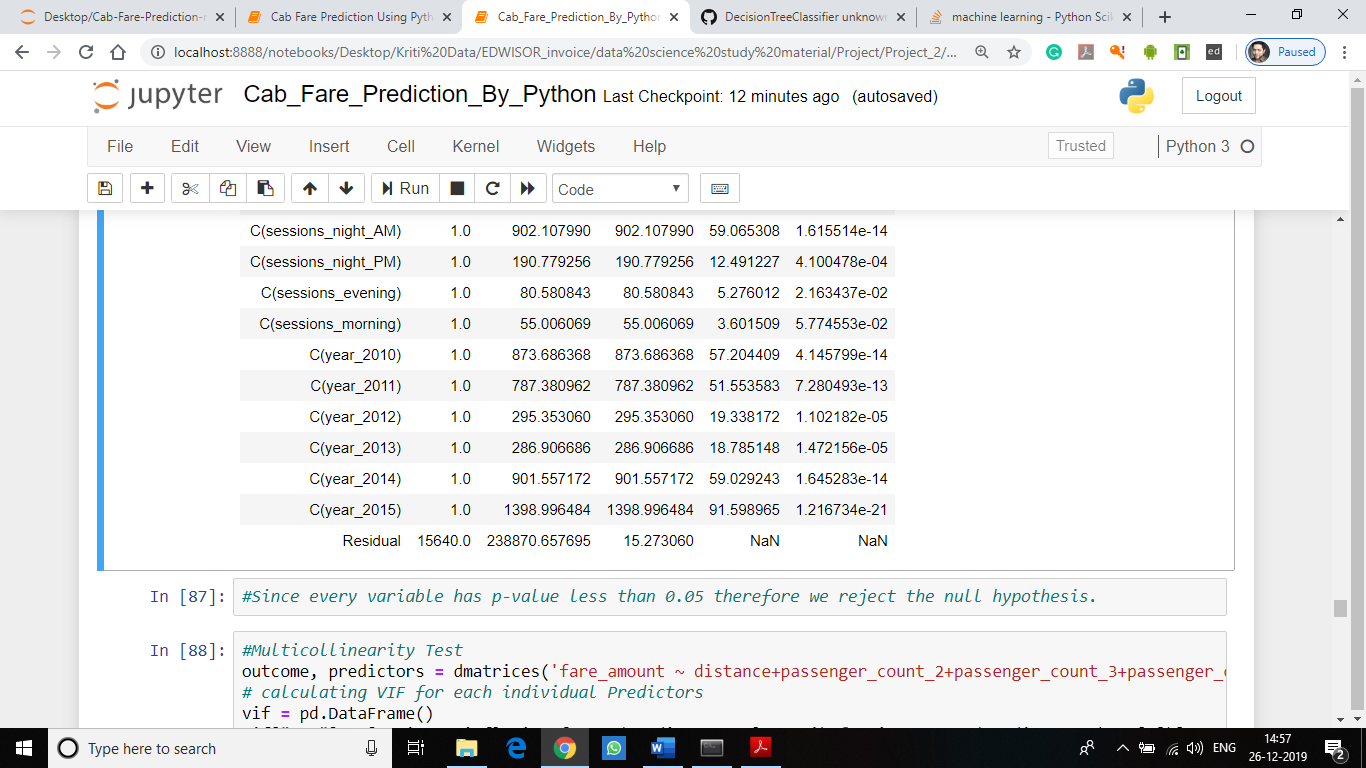
model = ols('fare\_amount ~ C(passenger\_count\_2)+C(passenger\_count\_3)+C(passenger\_count\_4)+C(passenger\_count\_5)+C(passenger\_count\_6)+C(season\_spring)+C(season\_summer)+C(season\_winter)+C(week\_weekend)+C(sessions\_night\_AM)+C(sessions\_night\_PM)+C(sessions\_evening)+C(sessions\_morning)+C(year\_2010)+C(year\_2011)+C(year\_2012)+C(year\_2013)+C(year\_2014)+C(year\_2015)',data=Cab\_Train\_Data).fit()

aov\_table = sm.stats.anova\_lm(model)

aov\_table

Below is the Anova analysis table for each categorical variable:





**Fig 12: ANOVA test on Variables**

Since every variable has p-value less than 0.05 therefore we reject the null hypothesis.

**STEP 6:** **Multicollinearity**

In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other. Multicollinearity increases the standard errors of the coefficients. It Increases 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, you can assume that the regression coefficients are poorly estimated due to multicollinearity.**

**Following codes used to perform above task:**

#Multicollinearity Test

outcome, predictors = dmatrices('fare\_amount ~ distance+passenger\_count\_2+passenger\_count\_3+passenger\_count\_4+passenger\_count\_5+passenger\_count\_6+season\_spring+season\_summer+season\_winter+week\_weekend+sessions\_night\_AM+sessions\_night\_PM+sessions\_evening+sessions\_morning+year\_2010+year\_2011+year\_2012+year\_2013+year\_2014+year\_2015',Cab\_Train\_Data, return\_type='dataframe')

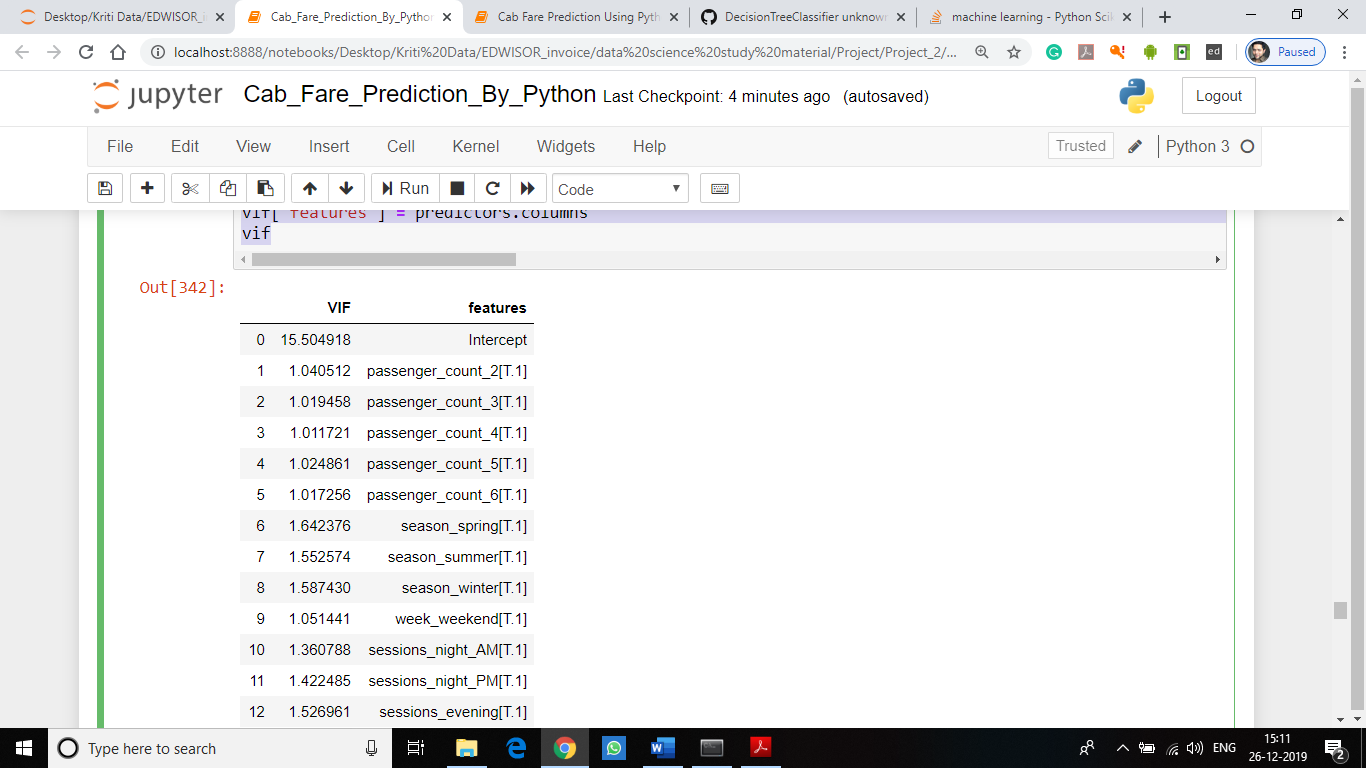
# calculating VIF for each individual Predictors

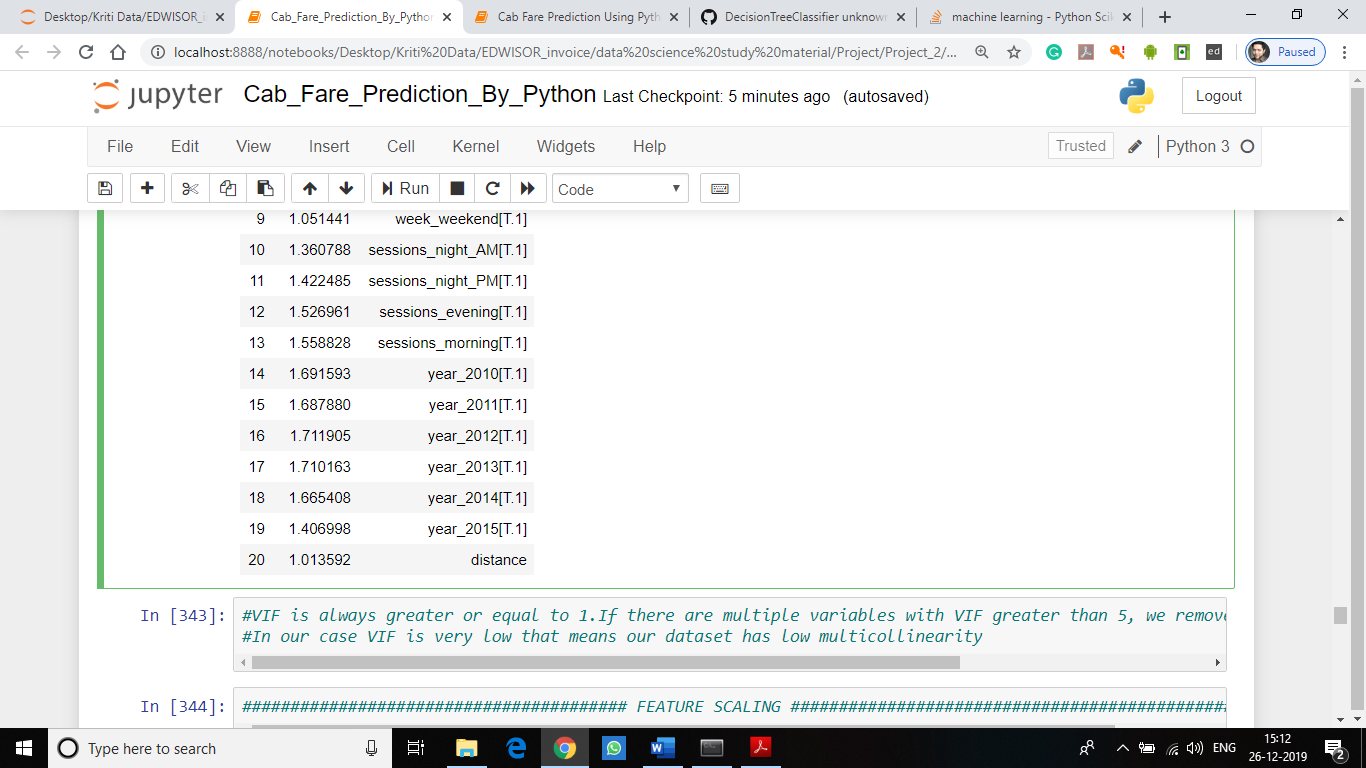
vif = pd.DataFrame()

vif["VIF"] = [variance\_inflation\_factor(predictors.values, i) for i in range(predictors.shape[1])]

vif["features"] = predictors.columns

vif





**Fig 13: VIF analysis**

#VIF is always greater or equal to 1. If there are multiple variables with VIF greater than 5, we remove the variable with the highest VIF.

#In our case VIF is very low that means our dataset has low multicollinearity

**STEP-7** : **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. Normalization 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 ‘distance’ is not distributed normally so we had chosen normalization over standardization. High variance will affect the accuracy of the model. So, we want to normalise that variance. It is performed only on Continuous variables.

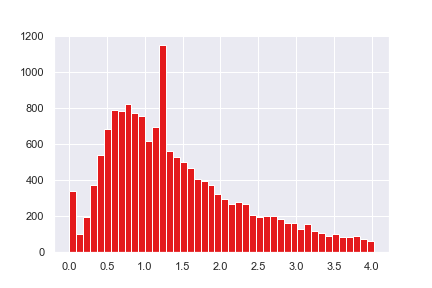
**Following codes used to perform above task:**

**################## FEATURE SCALING ###################**

#Normality check

plt.hist(Cab\_Train\_Data["distance"],bins="auto")

plt.savefig('Normality\_check\_Python.png')



**Fig 14: Normality check on distance variable**

On observing plots, it is clear that the data of the distance variable is not uniformly distributed. Hence in this applying normalization for proper scaling of the distance variable.

#Since data is not normally distributed so applying normalization method to normalize datasets

Cab\_Train\_Data['distance'] = (Cab\_Train\_Data['distance'] - min(Cab\_Train\_Data['distance']))/(max(Cab\_Train\_Data['distance']) - min(Cab\_Train\_Data['distance']))

Cab\_Test\_Data['distance'] = (Cab\_Test\_Data['distance'] - min(Cab\_Test\_Data['distance']))/(max(Cab\_Test\_Data['distance']) - min(Cab\_Test\_Data['distance']))

**STEP 8: Splitting train and Validation Dataset**

We have used **train\_test\_split()** method to divide whole Dataset into train and validation dataset. 20% is in validation dataset and 80% is in training data. We will test the performance of model on validation dataset. The model which performs best will be chosen to perform on test dataset provided along with original train dataset.

* X\_train Y\_train--are train subset.
* X\_test Y\_test--are validation subset.

**OR**

train,test = train\_test\_split(Cab\_Train\_Data, test\_size = 0.2)

**Following codes used to perform above task:**

**################## SPLITTING DATA ###################**

#Splitting train into train and validation subsets

train,test = train\_test\_split(Cab\_Train\_Data, test\_size = 0.2)

**OR**

#Splitting train into train and validation subsets

X = Cab\_Train\_Data.drop('fare\_amount',axis=1).values

Y = Cab\_Train\_Data['fare\_amount'].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.25, random\_state=42)

**STEP 9: Model Development**

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. In this project I have built models using four Algorithms:

* Linear Regression
* Decision Tree
* Random Forest
* Xgboost Regression

We will evaluate performance on validation dataset which was generated using Sampling. We will deal with specific error metrics like –

Regression metrics for our Models:

* MAPE(Mean Absolute Percentage Error)

**################## LINEAR REGRESSION ###################**

#Running regression model

model\_LR = LinearRegression().fit(X\_train,Y\_train)

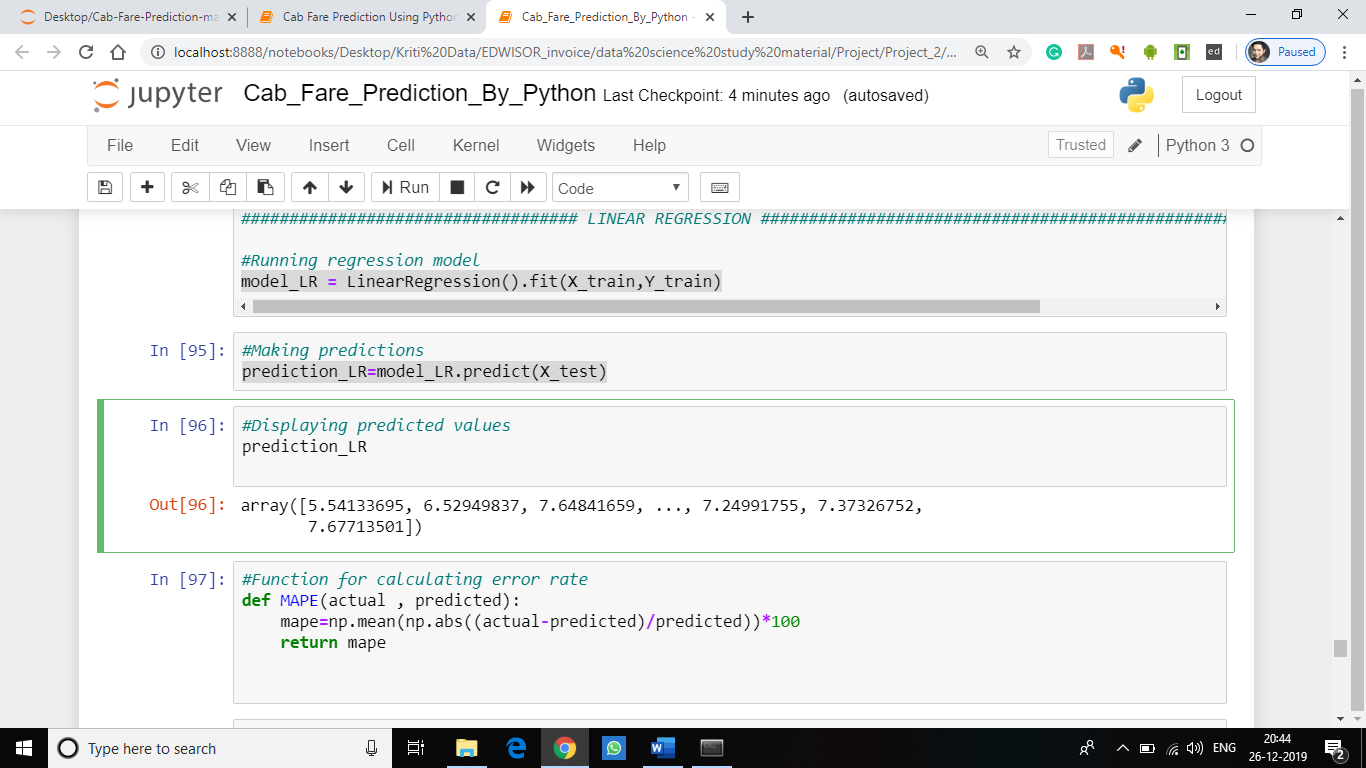
#Making predictions

prediction\_LR=model\_LR.predict(X\_test)

#Displaying predicted values

prediction\_LR

**Output:**



#Function for calculating error rate

def MAPE(actual , predicted):

mape=np.mean(np.abs((actual-predicted)/predicted))\*100

return mape

#Calculate MAPE

MAPE(Y\_test,prediction\_LR)

**Output:**

**25.119894528549107**

**#Linear Regression model**

**#Error rate = 25.1**

**#Accuracy = 74.9%**

**################## DECISION TREE ###################**

#Running regression DT model

model\_DT=DecisionTreeRegressor(max\_depth=10).fit(X\_train,Y\_train)

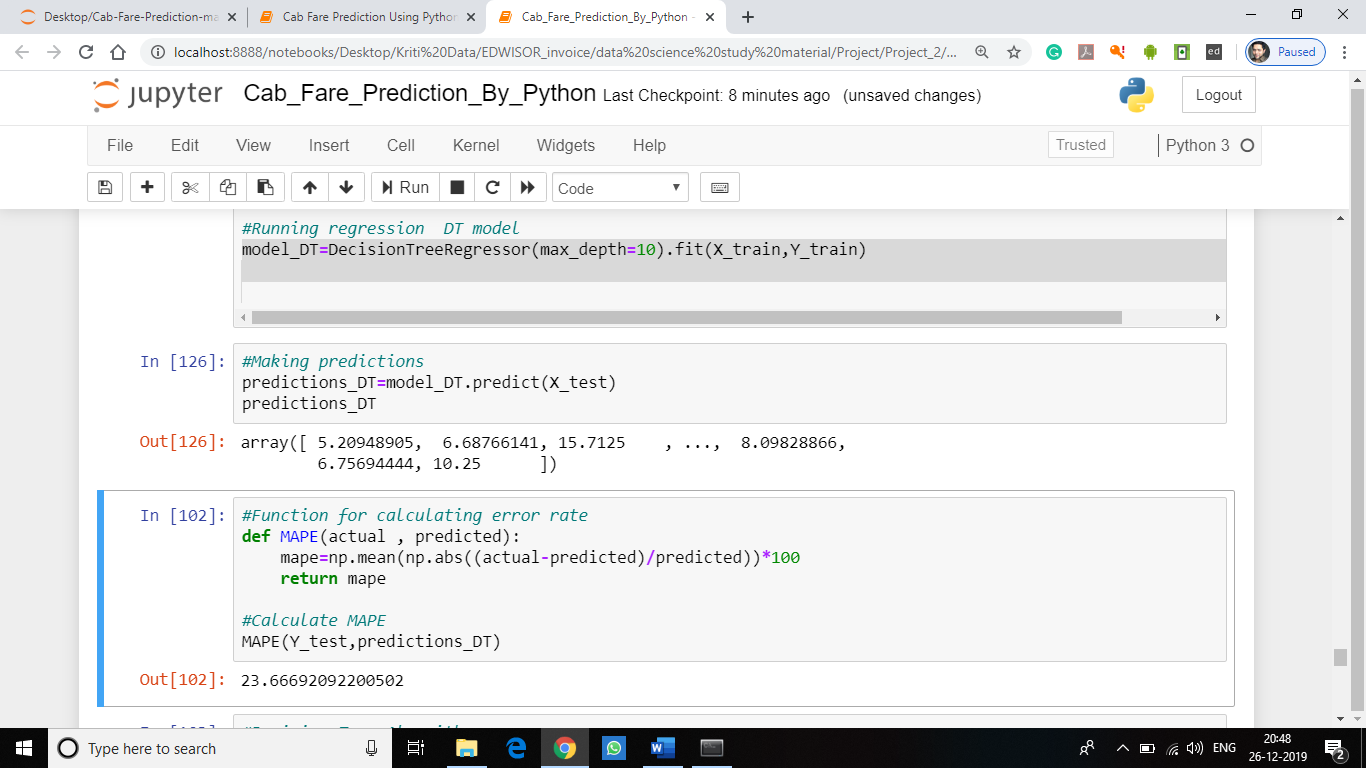
#Making predictions

predictions\_DT=model\_DT.predict(X\_test)

#Displaying predicted values

prediction\_DT

**Output:**



#Function for calculating error rate

def MAPE(actual , predicted):

mape=np.mean(np.abs((actual-predicted)/predicted))\*100

return mape

#Calculate MAPE

MAPE(Y\_test,predictions\_DT)

**Output:**

**23.66692092200502**

**#Decision Tree Algorithm**

**#Error rate = 23.6**

**#Accuracy = 76.4%**

**################## RANDOM FOREST ###################**

#Running Random Forest model

model\_RF = RandomForestRegressor(n\_estimators=600,max\_features=5).fit(X\_train,Y\_train)

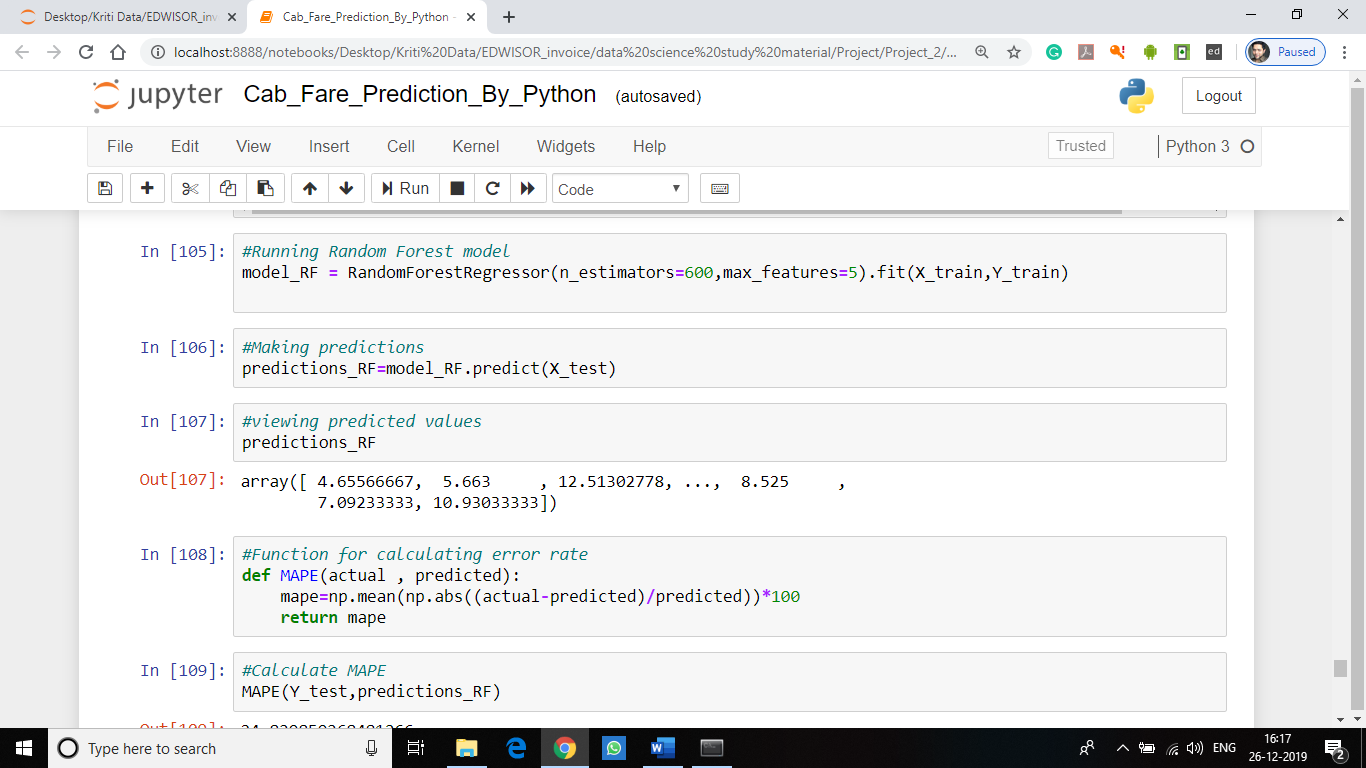
#Making predictions

predictions\_RF=model\_RF.predict(X\_test)

#viewing predicted values

predictions\_RF

**Output:**



#Function for calculating error rate

def MAPE(actual , predicted):

mape=np.mean(np.abs((actual-predicted)/predicted))\*100

return mape

#Calculate MAPE

MAPE(Y\_test,predictions\_RF)

**Output:**

**24.06036226**

**#Random Forest model**

**#Error rate= 24.8**

**#Accuracy= 75.2%**

**################## XGBOOST MODEL ###################**

#Running XGboost algorithm

Xgb = XGBRegressor()

Xgb.fit(X\_train,Y\_train)

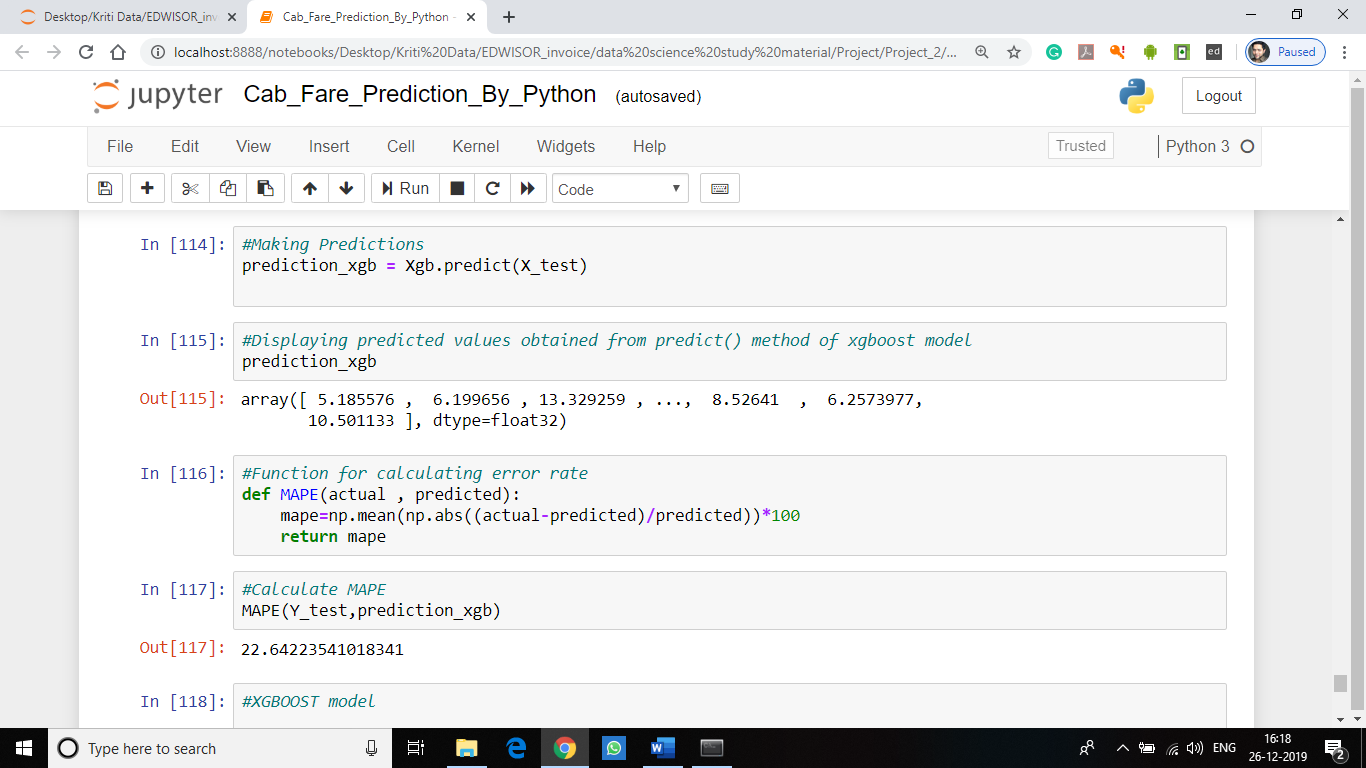
#Making Predictions

prediction\_xgb = Xgb.predict(X\_test)

#Displaying predicted values obtained from predict() method of xgboost model

prediction\_xgb

**Output:**



#Function for calculating error rate

def MAPE(actual , predicted):

mape=np.mean(np.abs((actual-predicted)/predicted))\*100

return mape

#Calculate MAPE

MAPE(Y\_test,prediction\_xgb)

**Output:**

**22.64223541018341**

**#XGBOOST model**

**#Error rate = 22.6**

**#Accuracy = 77.4%**

**STEP 10: Model Selection**

After developing model, we will evaluate the performance of the model by considering generated Regression metrics for our Models in development stage:

* MAPE (Mean Absolute Percentage Error)

The lesser the error rate better will be the performance and the accuracy of the model. On comparing these regression metrics of different models applied on the dataset we will select that model who has low error rate and better accuracy compared to other models for cab fare prediction.

|  |  |  |  |
| --- | --- | --- | --- |
| Models | MAPE | Error Rate | Accuracy |
| Linear Regression | 25.1 | 25.1 | 74.9% |
| Decision Tree | 23.6 | 23.6 | 76.4% |
| Random Forest | 24.7 | 24.7 | 75.3% |
| Xgboost model | 22.6 | 22.6 | 77.4% |

On Comparing different regression metrics Xgboost model found to be better model than other models and have good performance and accuracy as well as low error rate. So, selecting XGBOOST model for cab fare prediction.

**Conclusion/Result:**

**################### CAB FARE PREDICTION #####################**

#Selecting XGBOOST algorithm than other algorithms as it has comparatively low error rate and better accuracy when compared with other models

#Using XGBOOST model for cab fare prediction

#loading test dataset

Test\_data=pd.read\_csv('test.csv')

test\_pickup\_datetime=Test\_data['pickup\_datetime']

#predicting fare\_amount from test dataset

Fare\_prediction =Xgb.predict(Cab\_Test\_Data.values)

#Viewing predicted fare\_amount

Fare\_prediction

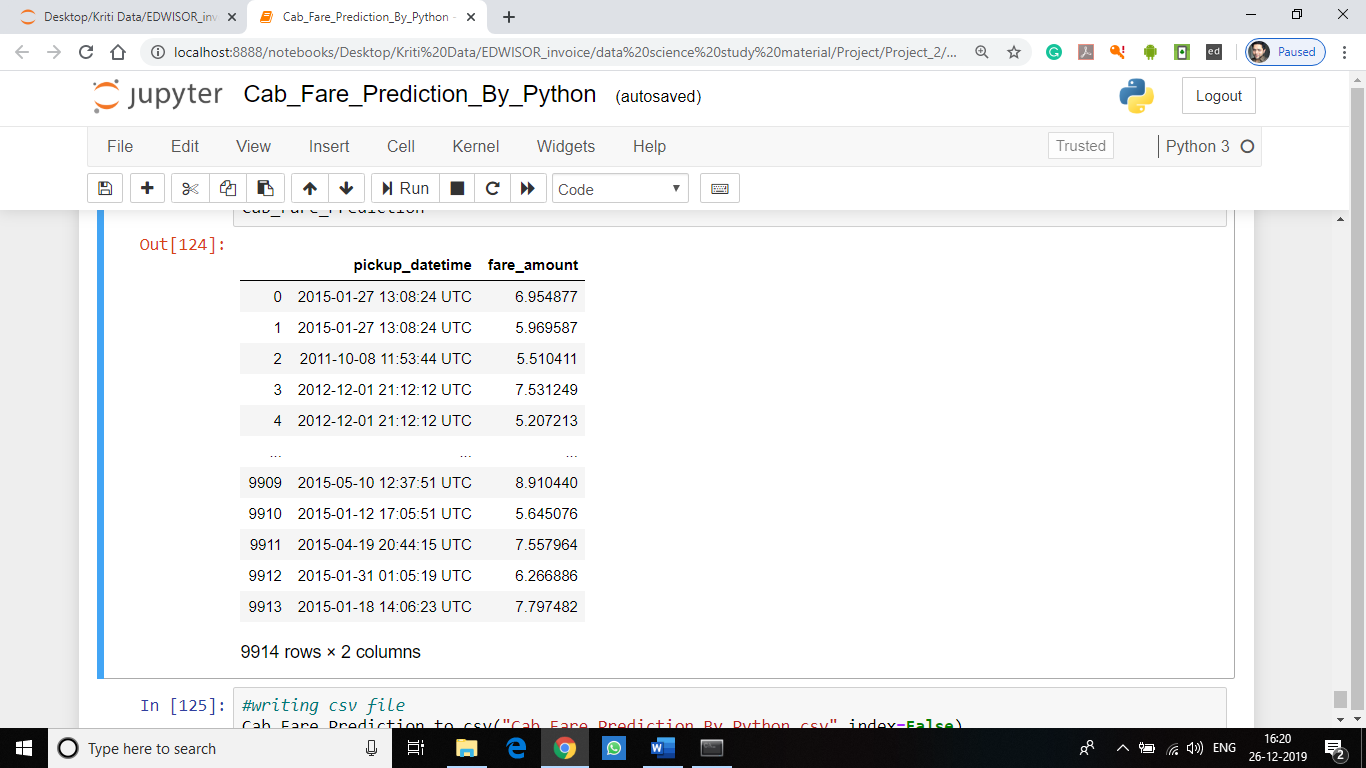
#Prediction of fare\_amount with respect to the test data

Cab\_Fare\_Prediction = pd.DataFrame({"pickup\_datetime":test\_pickup\_datetime,"fare\_amount" : Fare\_prediction})

#Displaying predicted fare\_amount for test dataset

Cab\_Fare\_Prediction

**Output:**



#writing csv file

Cab\_Fare\_Prediction.to\_csv("Cab\_Fare\_Prediction\_By\_Python.csv",index=False)

**Note: The predicted fare\_amount for the test data is present in the file "Cab\_Fare\_Prediction\_By\_Python.csv" which is attached with the report.**