Project Report

On

Cab fare Prediction

AUTHOR : Subba Reddy

INDEX

Chapter 1

Introduction

Now a day's cab rental services are expanding with the multiplier rate. The ease of using the services and flexibility gives their customer a great experience with competitive prices.

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

Understanding of data is the very first and important step in the process of finding solution of any business problem. Here in our case our company has provided a data set with following features, we need to go through each and every variable of it to understand and for better functioning.
Size of Dataset Provided: - 16067 rows, 7 Columns (including dependent variable)
Missing Values: Yes
Outliers Presented: Yes
Below mentioned is a list of all the variable names with their meanings:

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

Chapter 2

Methodology

Pre-Processing

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

- Data exploration and Cleaning
- Missing values treament
- Outlier Analysis
- Feature Selection
- Features Scaling
    - Skewness and Log transformation
- Visualization

Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested:

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

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

Model Selection

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

Chapter 3

Pre-Processing

Data exploration and Cleaning (Missing Values and Outliers)

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

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

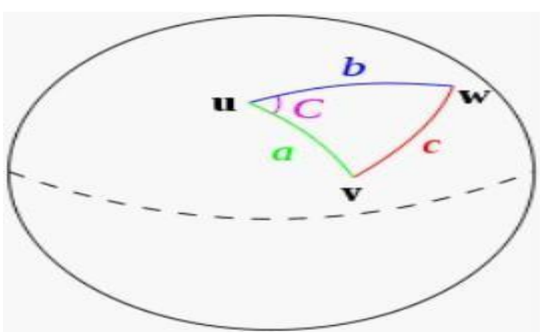
Creating some new variables from the given variables.

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

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

Also, we tried to find out the distance using the haversine formula which says:

The **haversine formula** determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles.



So our new extracted variables are:

- fare_amount
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count
- year
- Month
- Date
- Day of Week
- Hour
- Minute
- Distance

Selection of variables

Now as we know that all above variables are of now use so we will drop the redundant variables:

- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- Minute

Now only following variables we will use for further steps:

	fare_amount	passenger_count	year	Month	Date	Day of Week	Hour	distance
0	4.5	1.0	2009.0	6.0	15.0	0.0	17.0	1.030764
1	16.9	1.0	2010.0	1.0	5.0	1.0	16.0	8.450134
2	5.7	2.0	2011.0	8.0	18.0	3.0	0.0	1.389525
3	7.7	1.0	2012.0	4.0	21.0	5.0	4.0	2.799270
4	5.3	1.0	2010.0	3.0	9.0	1.0	7.0	1.999157
5	12.1	1.0	2011.0	1.0	6.0	3.0	9.0	3.787239
6	7.5	1.0	2012.0	11.0	20.0	1.0	20.0	1.555807
8	8.9	2.0	2009.0	9.0	2.0	2.0	1.0	2.849627
9	5.3	1.0	2012.0	4.0	8.0	6.0	7.0	1.374577
10	5.5	3.0	2012.0	12.0	24.0	0.0	11.0	0.000000

VariableNames	Variable Data Types
fare_amount	float64
passenger_count	object
year	object
Month	object
Date	object
Day of Week	object
Hour	object
distance	float64

Some more data exploration

In this report we are trying to predict the fare prices of a cab rental company. So here we have a data set of 16067 observations with 8 variables including one dependent variable.

Below are the names of Independent variables:

passenger_count, year, Month, Date, Day of Week, Hour, distance

Our Dependent variable is: **fare_amount**

Uniqueness in Variable

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

Variable Name	Unique Counts
fare_amount	450
passenger_count	7
year	7
Month	12
Date	31
Day of Week	7
Hour	24
distance	15424

Dividing the variables into two categories basis their data types:

Continuous variables - 'fare_amount', 'distance'.

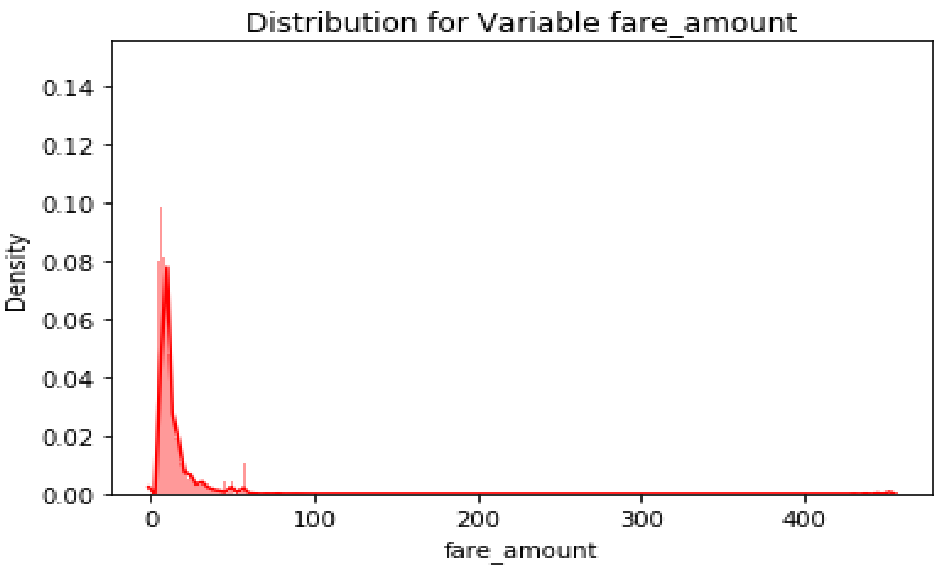
Categorical Variables - 'year', 'Month', 'Date', 'Day of Week', 'Hour', 'passenger_count'

Feature Scaling

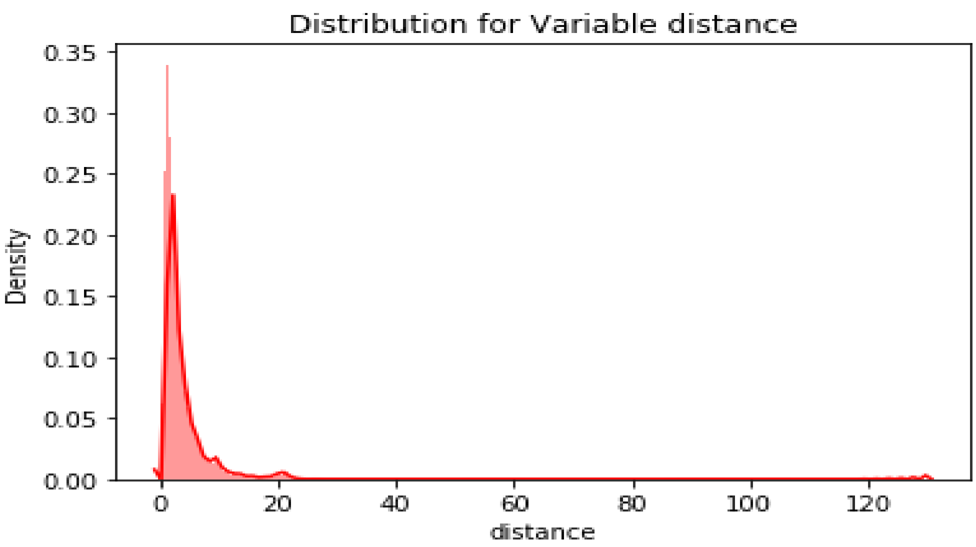
Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we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

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

```
fare_amount
```

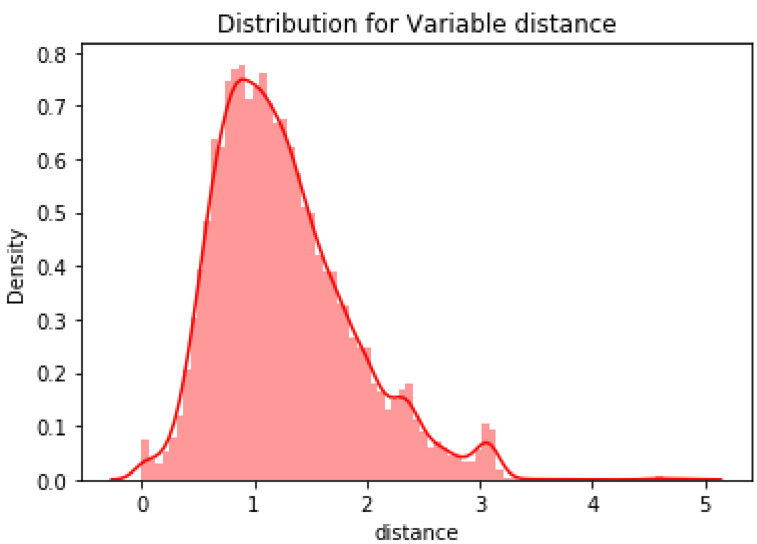



```
distance
```

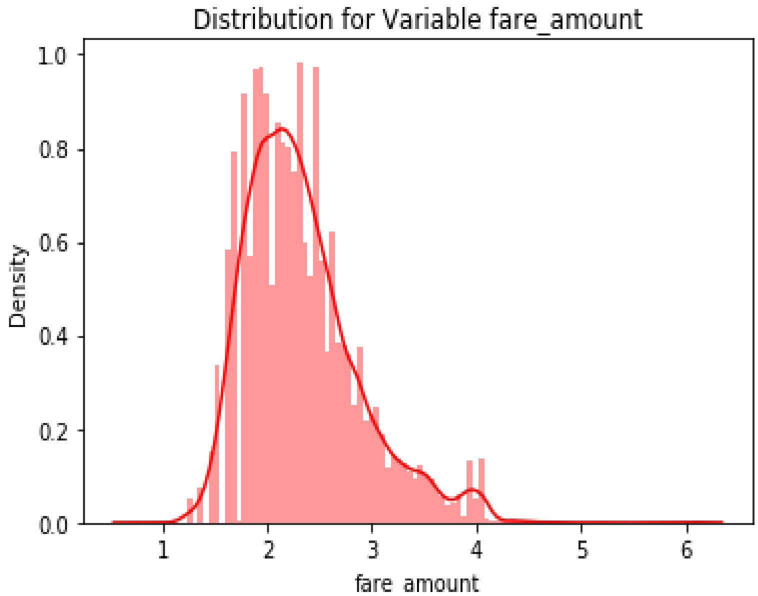


Below mentioned graphs shows the probability distribution plot to check distribution <u>after log transformation:</u>

distance



`fare_amount`



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

Chapter 4

Modelling

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

- Linear Regression
- Decision Tree
- Random Forest
- Gradient Boosting

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

Preparing the input features and target label matrices

In [90]:

```
X=np.array(train.iloc[:,1:])
y=np.array(train.iloc[:,0])
```

In [91]:

```
#Splitting the dataset into train and test dataset

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
```

Linear Regression

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

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

Linear Regression Model

```
In [92]: #Training the data based on Linear Regression model
         model_lr=LinearRegression()
         model_lr.fit(X_train,y_train)

Out[92]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [93]: #Predicting the model on train data
         train_pred_lr=model_lr.predict(X_train)
```

```
In [94]: test_pred_lr=model_lr.predict(X_test)
```

```
In [95]: ##calculating RMSE for test data
         test_rmse_lr = np.sqrt(mean_squared_error(y_test, test_pred_lr))

         ##calculating RMSE for train data
         train_rmse_lr= np.sqrt(mean_squared_error(y_train, train_pred_lr))
```

```
In [96]: print("Root Mean Squared Error For Training data = ",train_rmse_lr)
         print("Root Mean Squared Error For Test data = ",test_rmse_lr)

         Root Mean Squared Error For Training data =  0.2762019841074451
         Root Mean Squared Error For Test data =  0.24628672006420102
```

```
In [97]: print("R2 score for training data is",r2_score(y_train,train_pred_lr))
         print("R2 score for testing data is",r2_score(y_test,test_pred_lr))

         R2 score for training data is 0.7479265933156389
         R2 score for testing data is 0.7811405225793193
```

Decision Tree

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

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

Decision Tree Algorithm

Decision Tree Model

In [98]:
```
#Training the data using Decision Tree model
model_dt=DecisionTreeRegressor(max_depth=2)
model_dt.fit(X_train,y_train)
train_pred_dt=model_dt.predict(X_train)
test_pred_dt=model_dt.predict(X_test)
```

In [99]:
```
##calculating RMSE for test data
test_rmse_dt = np.sqrt(mean_squared_error(y_test, test_pred_dt))
##calculating RMSE for train data
train_rmse_dt= np.sqrt(mean_squared_error(y_train, train_pred_dt))
```

In [100]:
```
print("Root Mean Squared Error For Training data = ",train_rmse_dt)
print("Root Mean Squared Error For Test data = ",test_rmse_dt)
```

```
Root Mean Squared Error For Training data =  0.2996210902077019
Root Mean Squared Error For Test data =  0.2867460617158616
```

In [101]:
```
print("R2 score for training data is",r2_score(y_train,train_pred_dt))
print("R2 score for testing data is",r2_score(y_test,test_pred_dt))
```

```
R2 score for training data is 0.7033678616157003
R2 score for testing data is 0.7033268167661038
```

Random Forest

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

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

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

Random Forest Regressor Model

In [102]:
```
#Training the data using Random Forest Regressor
model_rf=RandomForestRegressor(n_estimators=101)  #n_esimators means No.of Trees
model_rf.fit(X_train,y_train)
train_pred_rf=model_rf.predict(X_train)
test_pred_rf=model_rf.predict(X_test)
```

In [103]:
```
##calculating RMSE for test data
test_rmse_rf = np.sqrt(mean_squared_error(y_test, test_pred_rf))
##calculating RMSE for train data
train_rmse_rf= np.sqrt(mean_squared_error(y_train, train_pred_rf))

```

In [104]:
```
print("Root Mean Squared Error For Training data = ",train_rmse_rf)
print("Root Mean Squared Error For Test data = ",test_rmse_rf)
```

```
Root Mean Squared Error For Training data =  0.09656085532166817
Root Mean Squared Error For Test data =  0.23954534141794698
```

In [105]:
```
print("R2 score for training data is",r2_score(y_train,train_pred_rf))
print("R2 score for testing data is",r2_score(y_test,test_pred_rf))
```

```
R2 score for training data is 0.9691911384352145
R2 score for testing data is 0.792957822573146
```

Gradient Boosting

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

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

Gradient Boosting Regressor Model

In [106]:

```
#training the data using Gradient Boosting model
model_gb=GradientBoostingRegressor()
model_gb.fit(X_train,y_train)
train_pred_gb=model_gb.predict(X_train)
test_pred_gb=model_gb.predict(X_test)
```

In [107]:

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

##calculating RMSE for train data
train_rmse_gb= np.sqrt(mean_squared_error(y_train, train_pred_gb))
```

In [108]:

```
print("Root Mean Squared Error For Training data = ",train_rmse_gb)
print("Root Mean Squared Error For Test data = ",test_rmse_gb)
```

```
Root Mean Squared Error For Training data =  0.2278465768661526
Root Mean Squared Error For Test data =  0.22802380053723245
```

In [109]:

```
print("R2 score for training data is",r2_score(y_train,train_pred_gb))
print("R2 score for testing data is",r2_score(y_test,test_pred_gb))
```

```
R2 score for training data is 0.8284627438023151
R2 score for testing data is 0.8123952942871889
```

Hyper Parameters Tunings for optimizing the results

Model hyperparameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter because this is set by the data scientist. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation dataset - for a problem.

Here we have used two hyper parameters tuning techniques

- ➢ Random Search CV
- ➢ Grid Search CV

1. **Random Search CV**: This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.
2. **Grid Search CV**: This algorithm set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient.

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

In [111]:
```
from sklearn.model_selection import train_test_split,RandomizedSearchCV
##Random Search CV on Random Forest Model

model_rrf = RandomForestRegressor(random_state = 0) #rrf=Rondom forest regressor
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

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

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

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)

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

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

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.8.
RMSE =  0.23748527661309576
```

```
In [113]:
##Random Search CV on gradient boosting model

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

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

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

view_best_params_gb = randomcv_gb.best_params_

best_model = randomcv_gb.best_estimator_

predictions_gb = best_model.predict(X_test)

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

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

```
Random Search CV Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.77.
RMSE =  0.25256717919910787
```

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

Final parameter extraction for Random Forest Model

```
In [114]:
from sklearn.model_selection import GridSearchCV
## Grid Search CV for random Forest model
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

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

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

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

#R^2
grf_r2 = r2_score(y_test, predictions_grf)
#Calculating RMSE
grf_rmse = np.sqrt(mean_squared_error(y_test,predictions_grf))

print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(grf_r2))
print('RMSE = ',(grf_rmse))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 17}
R-squared = 0.8.
RMSE =  0.2364371463281487
```

Final parameter extraction for Gradient Boosting Model

In [115]:

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

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

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

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

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

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

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
R-squared = 0.8.
RMSE =  0.23739342063769528
```

Chapter 5

Conclusion

Model Evaluation

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

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

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

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

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

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

Below table shows the model results before applying hyper tuning:

Model Name	**RMSE**		**R Squared**	
	Train	**Test**	**Train**	**Test**
`Linear Regression	0.27	0.25	0.74	0.77
Decision Tree	0.30	0.28	0.70	0.70
Random Forest model	0.09	0.23	0.96	0.79
Gradient Boosting	0.22	0.22	0.82	0.81

Below table shows results post using hyper parameter tuning techniques:

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

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

Model Selection

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

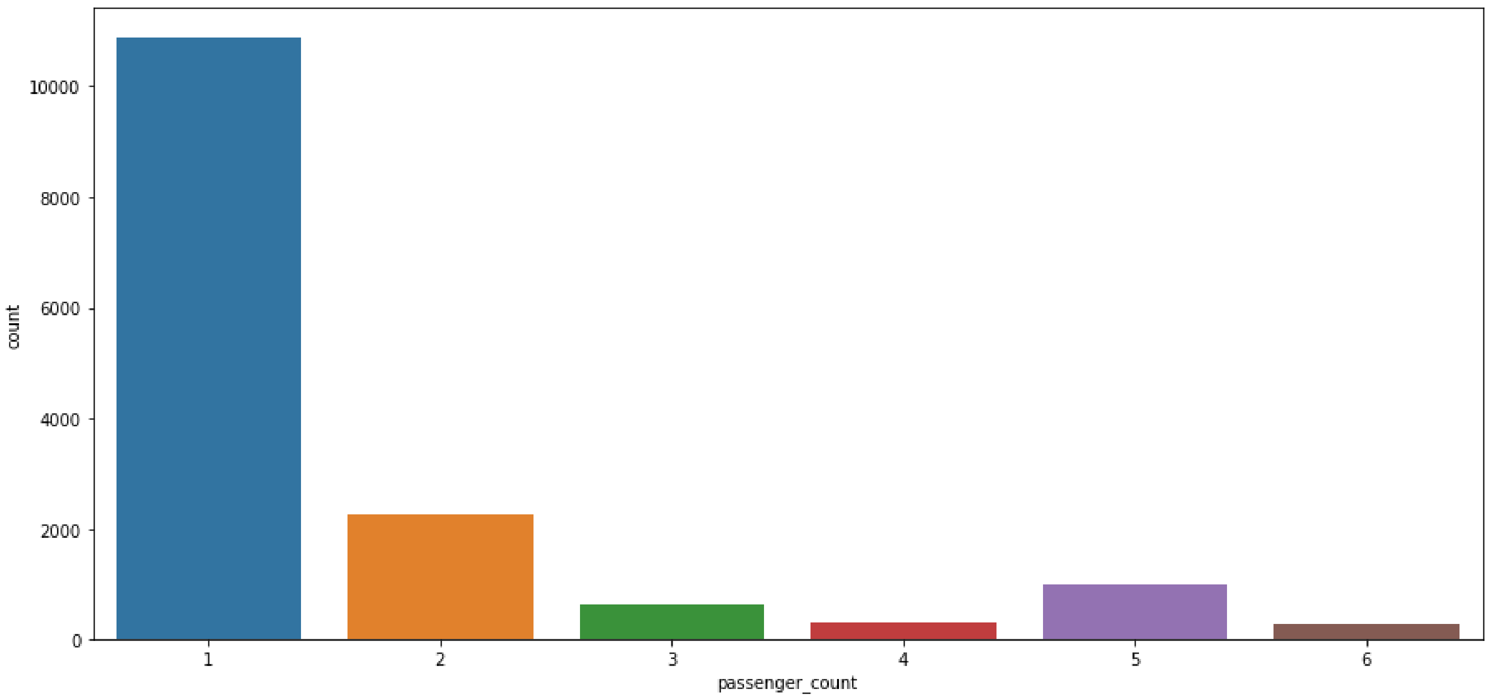
- From the observation of all RMSE Value and R-Squared Value we have concluded that,
- Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
- After this, I chose Random Forest CV and Grid Search CV to apply cross validation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

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

Some more visualization facts:

1. **Number of passengers and fare**

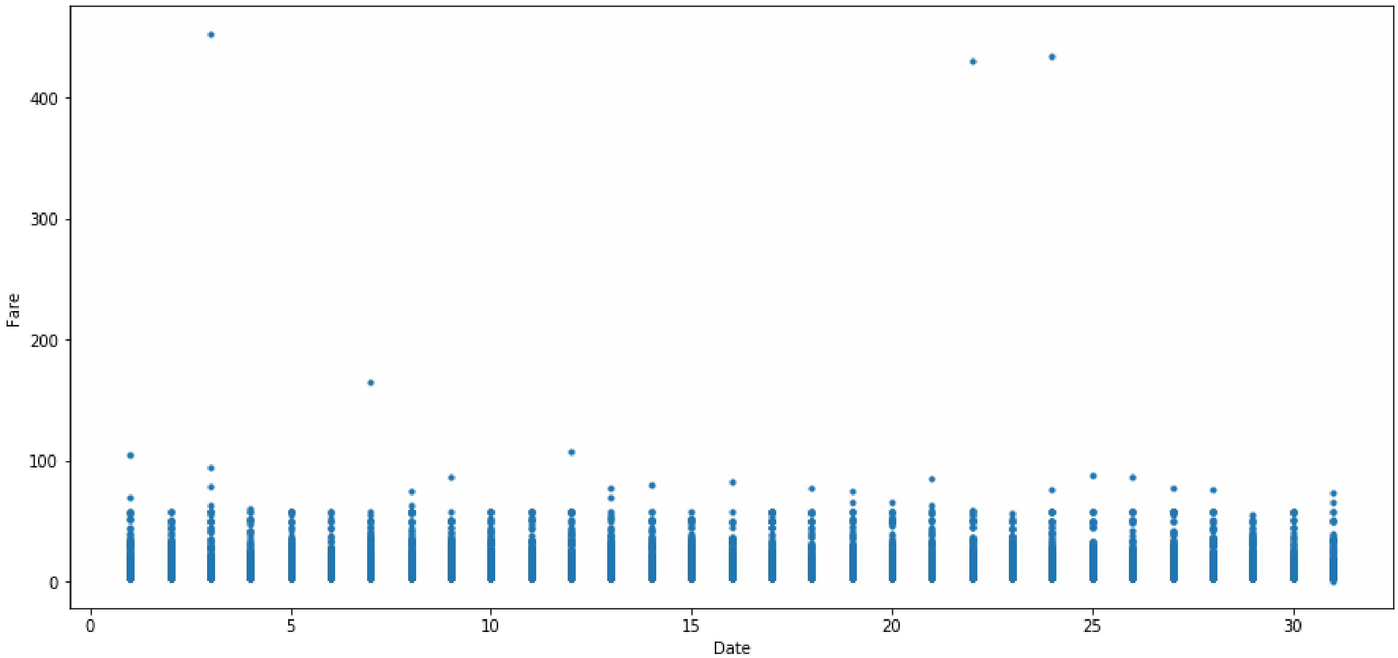
We can see in below graph that single passengers are the most frequent travelers, and the highest fare also seems to come from cabs which carry just 1 passenger.



10000
8000
6000
4000
2000
0
count
1
2
3
4
5
6
passenger_count

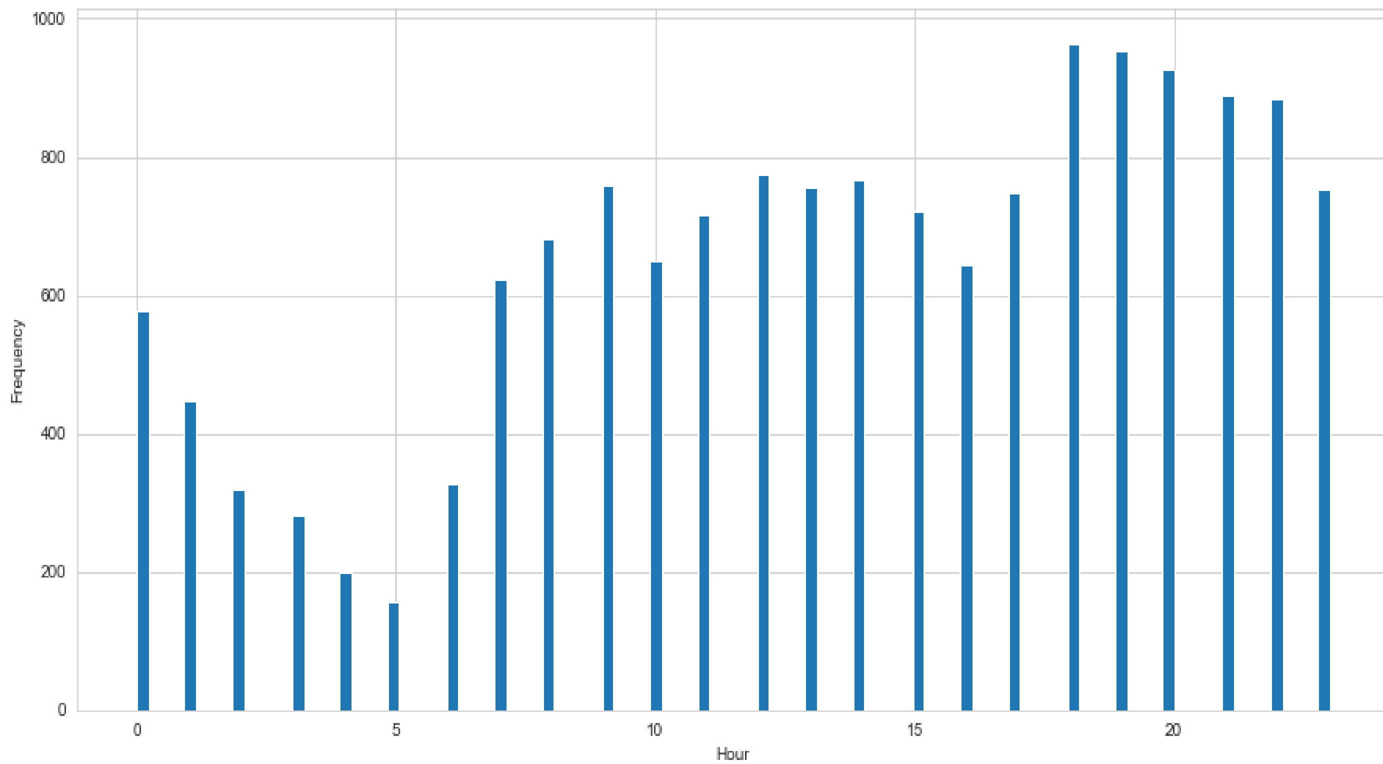
2. Date of month and fares

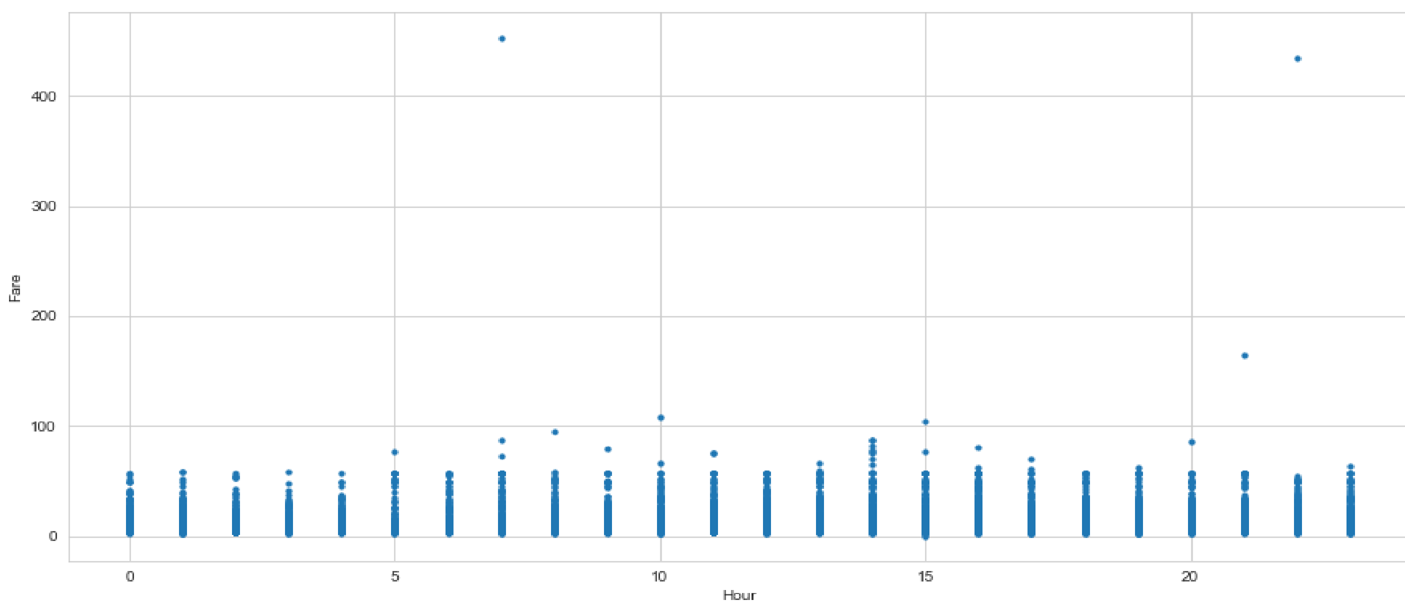
The fares throughout the month mostly seem uniform.



3. **Hours and Fares**

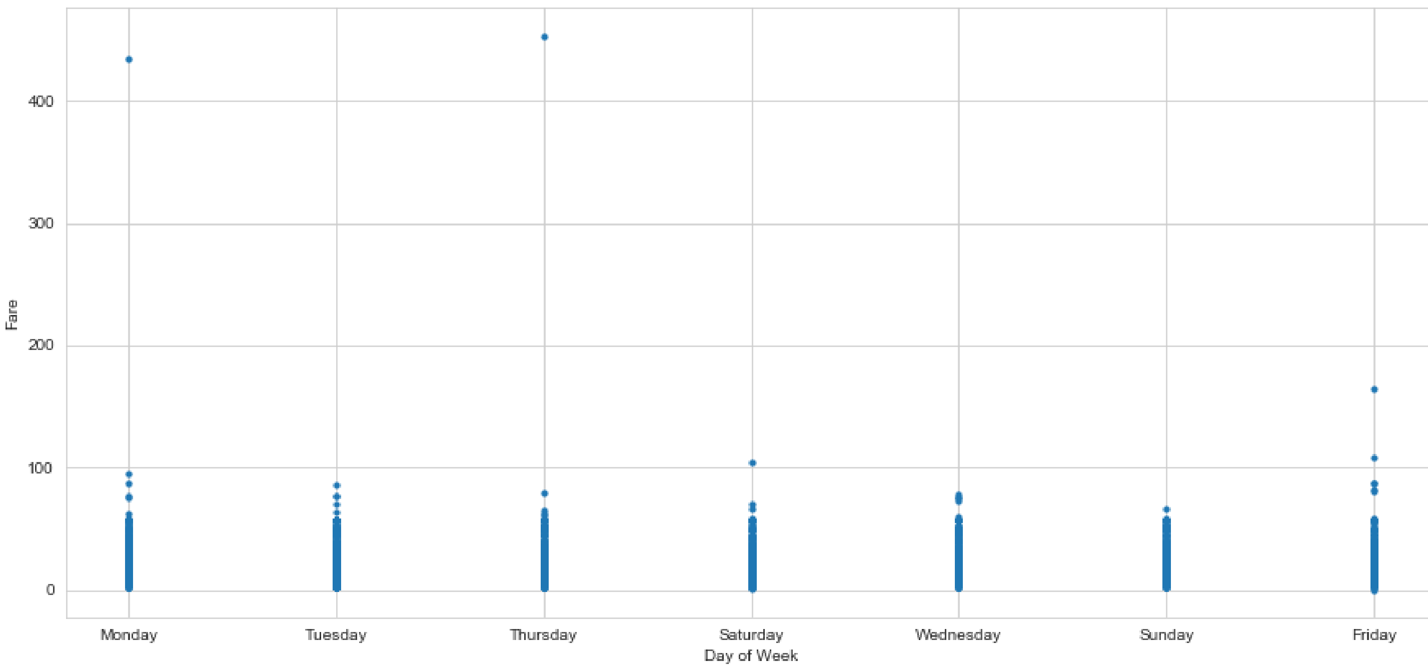
- During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours
- Fare prices during 2PM to 8PM is bit high compared to all other time might be due to high demands.



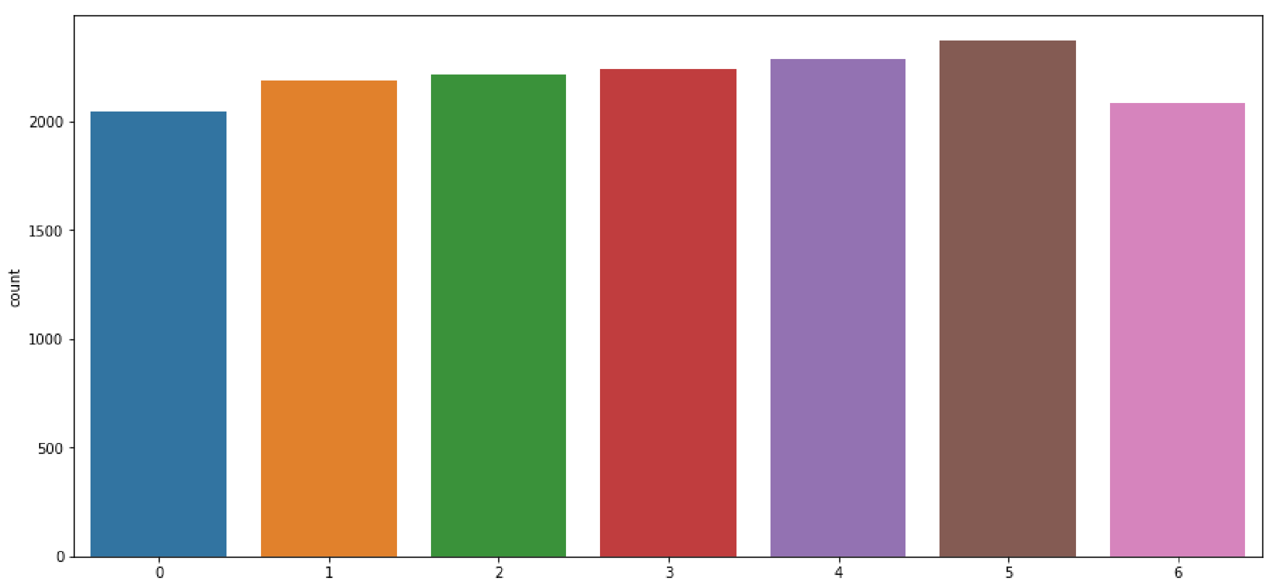


4. **Week Day and fare**

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



5. **Impact of Day on the Number of Cab rides :**



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

End of Report

References

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

Appendix

R code:

```
# Cab Fare
Prediction
rm(list = ls())
setwd("C:/Users/User/Desktop/Cab Fare Prediction Project")
getwd()
# #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)
# 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.
# loading datasets
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test.csv")
test_pickup_datetime = test["pickup_datetime"]
# Structure of data
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(test,5)
#############                    Exploratory Data Analysis              #######################
# Changing the data types of variables
train$fare_amount = as.numeric(as.character(train$fare_amount))
train$passenger_count=round(train$passenger_count)
### 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.
train[which(train$fare_amount < 1 ),]
nrow(train[which(train$fare_amount < 1 ),])
train = train[-which(train$fare_amount < 1 ),]
#2.Passenger_count variable
for (i in seq(4,11,by=1)){
print(paste('passenger_count above ' ,i,nrow(train[which(train$passenger_count > i ),])))
}
# so 20 observations of passenger_count is consistenly above from 6,7,8,9,10 passenger_counts, let's
check them.
train[which(train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])
# We will remove these 58 observations and 20 observation which are above 6 value because a cab
cannot hold these number of passengers.
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(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(train[which(train$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(train[which(train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(train[which(train$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(train[which(train$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(train[which(train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(train[which(train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(train[which(train$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(train[which(train$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(train[which(train$pickup_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
nrow(train[which(train$dropoff_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
# there are values which are equal to 0. we will remove them.
train = train[-which(train$pickup_latitude > 90),]
train = train[-which(train$pickup_longitude == 0),]
train = train[-which(train$dropoff_longitude == 0),]
# Make a copy
df=train
# train=df
#############                    Missing Value Analysis                #############
missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val
unique(train$passenger_count)
unique(test$passenger_count)
train[,'passenger_count'] = factor(train[,'passenger_count'], labels=(1:6))
test[,'passenger_count'] = factor(test[,'passenger_count'], labels=(1:6))
# 1.For Passenger_count:
# Actual value = 1
# Mode = 1
# KNN = 1
train$passenger_count[1000]
train$passenger_count[1000] = NA
getmode <- function(v) {
uniqv <- unique(v)
uniqv[which.max(tabulate(match(v, uniqv)))]
}
# Mode Method
getmode(train$passenger_count)
# We can't use mode method because data will be more biased towards passenger_count=1
# 2.For fare_amount:
# Actual value = 18.1,
# Mean = 15.117,
# Median = 8.5,
# KNN = 18.28
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.968236      4635.700531         2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
```

```
# 2.613305         2.710835         2.632400
# passenger_count
# 1.266104
train$fare_amount[1000]
train$fare_amount[1000]= NA
# Mean Method
mean(train$fare_amount, na.rm = T)
#Median Method
median(train$fare_amount, na.rm = T)
# kNN Imputation
train = knnImputation(train, k = 181)
train$fare_amount[1000]
train$passenger_count[1000]
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.661952       4635.700531         2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305         2.710835         2.632400
# passenger_count
# 1.263859
sum(is.na(train))
str(train)
summary(train)
df1=train
# train=df1
#####################                    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(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 = train[,"fare_amount"] %in% boxplot.stats(train[,"fare_amount"])$out
train[which(vals),"fare_amount"] = NA
#lets check the NA's
sum(is.na(train$fare_amount))
#Imputing with KNN
train = knnImputation(train,k=3)
# lets check the missing values
sum(is.na(train$fare_amount))
str(train)
df2=train
# train=df2
##################                 Feature Engineering                   ##########################
# 1.Feature Engineering for timestamp variable
# we will derive new features from pickup_datetime variable
# new features will be year,month,day_of_week,hour
#Convert pickup_datetime from factor to date time
train$pickup_date = as.Date(as.character(train$pickup_datetime))
train$pickup_weekday = as.factor(format(train$pickup_date,"%u"))# Monday = 1
train$pickup_mnth = as.factor(format(train$pickup_date,"%m"))
train$pickup_yr = as.factor(format(train$pickup_date,"%Y"))
pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
train$pickup_hour = as.factor(format(pickup_time,"%H"))
#Add same features to test set
test$pickup_date = as.Date(as.character(test$pickup_datetime))
test$pickup_weekday = as.factor(format(test$pickup_date,"%u"))# Monday = 1
```

```
test$pickup_mnth = as.factor(format(test$pickup_date,"%m"))
test$pickup_yr = as.factor(format(test$pickup_date,"%Y"))
pickup_time = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
test$pickup_hour = as.factor(format(pickup_time,"%H"))
sum(is.na(train))# there was 1 'na' in pickup_datetime which created na's in above feature engineered
variables.
train = na.omit(train) # we will remove that 1 row of na's
train = subset(train,select = -c(pickup_datetime,pickup_date))
test = subset(test,select = -c(pickup_datetime,pickup_date))
# Now we will use month,weekday,hour to derive new features like sessions in a day,seasons in a
year,week:weekend/weekday
# f = function(x){
#   if ((x >=5)& (x <= 11)){
#     return ('morning')
#   }
#   if ((x >=12) & (x <= 16)){
#     return ('afternoon')
#   }
#   if ((x >=17) & (x <= 20)){
#     return ('evening')
#   }
#   if ((x >=21) & (x <= 23)){
#     return ('night (PM)')
#   }
#   if ((x >=0) & (x <= 4)){
#     return ('night (AM)')
#   }
# }
# 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
train$dist =
haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longitude,train$dropoff_latitude)
test$dist =
haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude,test$dropoff_latitude)
# We will remove the variables which were used to feature engineer new variables
train = subset(train,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
test = subset(test,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
str(train)
summary(train)
################                    Feature selection            ###################
numeric_index = sapply(train,is.numeric) #selecting only numeric
numeric_data = train[,numeric_index]
```

```
cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")
#ANOVA for categorical variables with target numeric variable
#aov_results = aov(fare_amount ~ passenger_count * pickup_hour * pickup_weekday,data = train)
aov_results = aov(fare_amount ~ passenger_count + pickup_hour + pickup_weekday + pickup_mnth +
pickup_yr,data = train)
summary(aov_results)
# pickup_weekdat has p value greater than 0.05
train = subset(train,select=-pickup_weekday)
#remove from test set
test = subset(test,select=-pickup_weekday)
##################################            Feature Scaling
################################################
#Normality check
# qqnorm(train$fare_amount)
# histogram(train$fare_amount)
library(car)
# dev.off()
par(mfrow=c(1,2))
qqPlot(train$fare_amount)                              # qqPlot, it has a x values derived from gaussian
distribution, if data is distributed normally then the sorted data points should lie very close to the solid
reference line
truehist(train$fare_amount)                            # truehist() scales the counts to give an estimate of the
probability density.
lines(density(train$fare_amount))  # Right skewed      # lines() and density() functions to overlay a
density plot on histogram
#Normalisation
print('dist')
train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/
(max(train[,'dist'] - min(train[,'dist'])))
# #check multicollearity
# library(usdm)
# vif(train[,-1])
#
# vifcor(train[,-1], th = 0.9)
#################### Splitting train into train and validation subsets ###################
set.seed(1000)
tr.idx = createDataPartition(train$fare_amount,p=0.75,list = FALSE) # 75% in trainin and 25% in
Validation Datasets
train_data = train[tr.idx,]
test_data = train[-tr.idx,]
rmExcept(c("test","train","df",'df1','df2','df3','test_data','train_data','test_pickup_datetime'))
###################Model Selection################
#Error metric used to select model is RMSE
#############           Linear regression              #################
lm_model = lm(fare_amount ~.,data=train_data)
summary(lm_model)
str(train_data)
plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",
xlab = "Predicted values of fare_amount",
ylab = "standardized residuals")
lm_predictions = predict(lm_model,test_data[,2:6])
qplot(x = test_data[,1], y = lm_predictions, data = test_data, color = I("blue"), geom = "point")
regr.eval(test_data[,1],lm_predictions)
# mae        mse       rmse       mape
# 3.5303114 19.3079726  4.3940838  0.4510407
#############                           Decision Tree           #####################
```

```r
Dt_model = rpart(fare_amount ~ ., data = train_data, method = "anova")
summary(Dt_model)
#Predict for new test cases
predictions_DT = predict(Dt_model, test_data[,2:6])
qplot(x = test_data[,1], y = predictions_DT, data = test_data, color = I("blue"), geom = "point")
regr.eval(test_data[,1],predictions_DT)
# mae       mse      rmse      mape
# 1.8981592 6.7034713 2.5891063 0.2241461
#############                              Random forest           #####################
rf_model = randomForest(fare_amount ~.,data=train_data)
summary(rf_model)
rf_predictions = predict(rf_model,test_data[,2:6])
qplot(x = test_data[,1], y = rf_predictions, data = test_data, color = I("blue"), geom = "point")
regr.eval(test_data[,1],rf_predictions)
# mae       mse      rmse      mape
# 1.9053850 6.3682283 2.5235349 0.2335395
############         Improving Accuracy by using Ensemble technique ---- XGBOOST
###########################
train_data_matrix = as.matrix(sapply(train_data[-1],as.numeric))
test_data_data_matrix = as.matrix(sapply(test_data[-1],as.numeric))
xgboost_model = xgboost(data = train_data_matrix,label = train_data$fare_amount,nrounds =
15,verbose = FALSE)
summary(xgboost_model)
xgb_predictions = predict(xgboost_model,test_data_data_matrix)
qplot(x = test_data[,1], y = xgb_predictions, data = test_data, color = I("blue"), geom = "point")
regr.eval(test_data[,1],xgb_predictions)
# mae       mse      rmse      mape
# 1.6183415 5.1096465 2.2604527 0.1861947
#############                        Finalizing and Saving Model for later use
####################
# In this step we will train our model on whole training Dataset and save that model for later use
train_data_matrix2 = as.matrix(sapply(train[-1],as.numeric))
test_data_matrix2 = as.matrix(sapply(test,as.numeric))
xgboost_model2 = xgboost(data = train_data_matrix2,label = train$fare_amount,nrounds = 15,verbose
= FALSE)
# Saving the trained model
saveRDS(xgboost_model2, "./final_Xgboost_model_using_R.rds")
# loading the saved model
super_model <- readRDS("./final_Xgboost_model_using_R.rds")
print(super_model)
# Lets now predict on test dataset
xgb = predict(super_model,test_data_matrix2)
xgb_pred = data.frame(test_pickup_datetime,"predictions" = xgb)
# Now lets write(save) the predicted fare_amount in disk as .csv format
write.csv(xgb_pred,"xgb_predictions_R.csv",row.names = FALSE)
```