Project Report
On
Cab Fare Prediction

MONIKA N. WADHWANI
19-12-19

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.

1. Problem Statement

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

2. Data

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

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

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 visualising the data through graph and plots, all these steps is combined under one shed which is **Exploratory Data Analysis**, which includes following steps:

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

➢ 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

1. **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.

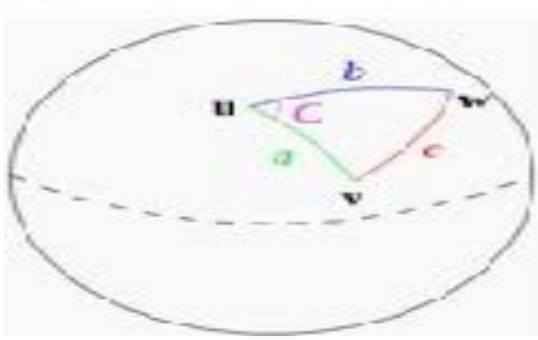
2. **Creating some new variables from the given variables.**

   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

3. 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	hour	distance
0	4.5	1.0	2009	6	15	0	17	1.030764
1	16.9	1.0	2010	1	5	1	16	8.450134
2	5.7	2.0	2011	8	18	3	0	1.389525
3	7.7	1.0	2012	4	21	5	4	2.799270
4	5.3	1.0	2010	3	9	1	7	1.999157

VariableNames Variable DataTypes	
fare_amount	float64
passenger_count	float64
year	int64
Month	int64
Date	int64
Day of Week	int64
Hour	int64
distance	float64

4. 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.

1. **Below are the names of Independent variables:**
passenger_count, year, Month, Date, Day of Week, Hour, distance

Our Dependent variable is: **fare_amount**

2. 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

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

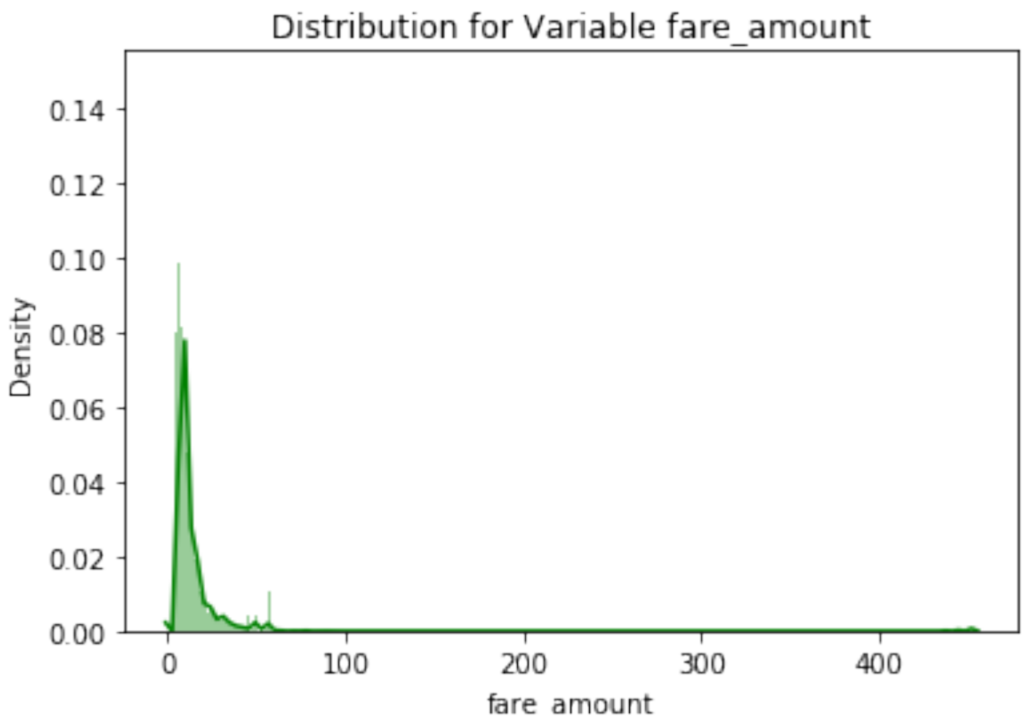
Continuous variables - 'fare_amount', 'distance'.

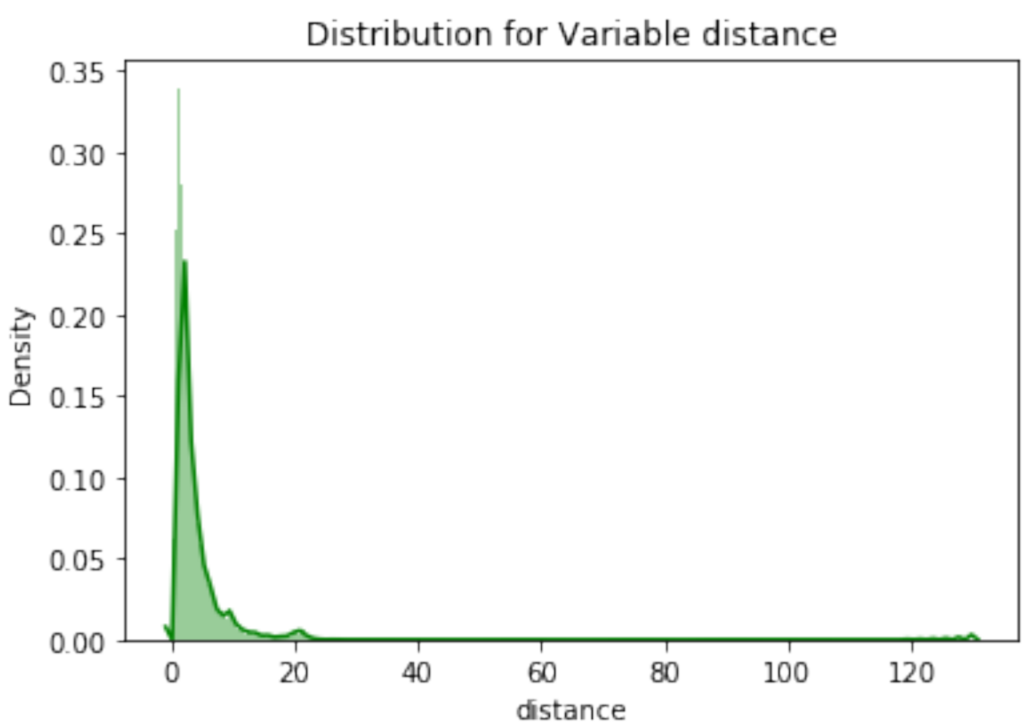
Categorical Variables - 'year', 'Month', 'Date', 'Day of Week', 'Hour', 'passenger_count'

5. Feature Scaling

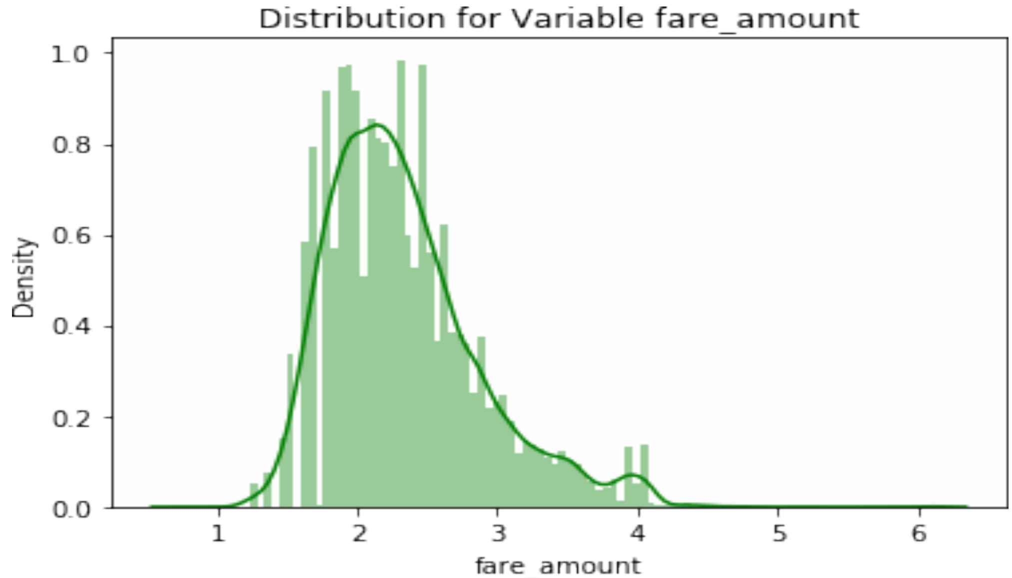
Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we 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.

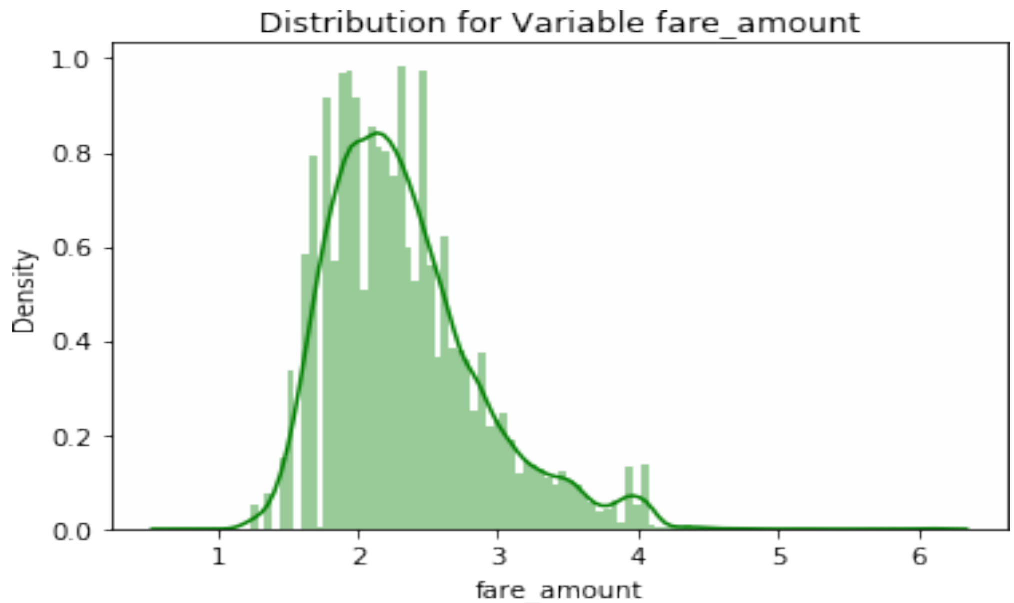
Below mentioned graphs shows the probability distribution plot to check distribution <u>before log transformation:</u>





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





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

☐
☐
☐
☐

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.

We need to split our train data into two parts

```
In [56]: from sklearn.tree import DecisionTreeRegressor
         from sklearn.metrics import mean_squared_error

In [60]: ##train test split for further modelling
         X_train, X_test, y_train, y_test = train_test_split( train_df.iloc[:, train_df.columns != 'fare_amount'],
                                   train_df.iloc[:, 0], test_size = 0.20, random_state = 1)

In [61]: print(X_train.shape)
         print(X_test.shape)

(12339, 7)
(3085, 7)
```

1. Linear Regression

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

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

1. Linear Regeression Model

```
In [62]: # Importing libraries for Linear Regression
         from sklearn.linear_model import LinearRegression

In [63]: # Building model on top of training dataset
         fit_LR = LinearRegression().fit(X_train , y_train)

In [64]: #prediction on train data
         pred_train_LR = fit_LR.predict(X_train)

In [65]: #prediction on test data
         pred_test_LR = fit_LR.predict(X_test)

In [66]: ##calculating RMSE for test data
         RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

In [67]: print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))

         Root Mean Squared Error For Test data = 0.2503511796785927

In [68]: from sklearn.metrics import r2_score
         #calculate R^2 for train data
         r2_score(y_train, pred_train_LR)

Out[68]: 0.746855951097612

In [69]: r2_score(y_test, pred_test_LR)

Out[69]: 0.7778537029821875
```

2. 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

2. Decision Tree Model

```
In [71]: fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)

In [72]: #prediction on train data
         pred_train_DT = fit_DT.predict(X_train)

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

In [73]: ##calculating RMSE for train data
         RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

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

In [74]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))

         Root Mean Squared Error For Training data = 0.30120630747129796
         Root Mean Squared Error For Test data = 0.28969521517125973

In [75]: ## R^2 calculation for train data
         r2_score(y_train, pred_train_DT)

Out[75]: 0.70012186420221

In [76]: ## R^2 calculation for test data
         r2_score(y_test, pred_test_DT)

Out[76]: 0.7025442022580384
```

3. 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:

3. Random Forest Model

```
In [77]: # Importing libraries for Random Forest
         from sklearn.ensemble import RandomForestRegressor

In [78]: fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)

In [79]: #prediction on train data
         pred_train_RF = fit_RF.predict(X_train)
         #prediction on test data
         pred_test_RF = fit_RF.predict(X_test)

In [80]: ##calculating RMSE for train data
         RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
         ##calculating RMSE for test data
         RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))

In [81]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))

         Root Mean Squared Error For Training data = 0.09504886483675674
         Root Mean Squared Error For Test data = 0.2391865396547728

In [82]: ## calculate R^2 for train data

         r2_score(y_train, pred_train_RF)

Out[82]: 0.9695764079865043

In [83]: #calculate R^2 for test data
         r2_score(y_test, pred_test_RF)

Out[83]: 0.797225534315765
```

4. 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:

4. Gradient Boosting

```
In [86]: # Importing library for GradientBoosting
         from sklearn.ensemble import GradientBoostingRegressor
```

```
In [87]: # Building model on top of training dataset
         fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
```

```
In [88]: #prediction on train data
         pred_train_GB = fit_GB.predict(X_train)

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

```
In [89]: ##calculating RMSE for train data
         RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
         ##calculating RMSE for test data
         RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

```
In [90]: print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
         print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))

         Root Mean Squared Error For Training data = 0.2292168048250226
         Root Mean Squared Error For Test data = 0.22939164285908767
```

```
In [92]: #calculate R^2 for test data
         r2_score(y_test, pred_test_GB)

Out[92]: 0.813493068270751
```

```
In [93]: #calculate R^2 for train data
         r2_score(y_train, pred_train_GB)

Out[93]: 0.8263361773771449
```

5. 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 [101]: ##Random Search CV on Random Forest Model

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

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

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

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)

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

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

Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.79.
RMSE =  0.2434649308921136
```

```
In [103]: ##Random Search CV on gradient boosting model

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

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

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

view_best_params_gb = randomcv_gb.best_params_

best_model = randomcv_gb.best_estimator_

predictions_gb = best_model.predict(X_test)

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

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

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

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

```
In [104]: 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': 5, 'n_estimators': 12}
R-squared = 0.8.
RMSE =  0.2398346306918429
```

```
In [105]: AA Grid Seorrh rV for grod{net boosting
          gb = GradientBoostingRegressor(random state
          = 0) n estimator = list(range(11,2B,1))
          depth  list(range(5,15,2))

          # Create the grid
          grid search = (' n est 1mato rs ' : n estimator,
                         ' nax depth ' : depth}

          #A Grid Seorrh Cross-kotidotion with S hold CP
          gridcv b = GridSRarchCV(gb, param grid = grid search, cv
          = 5) gridcv b = gridcv gb.fit(X train,y train)
          view best params Ggb = gridcv    b.best params

          #App Ly nodeL on test data
          predictions Ggb = gridcv gb.predict(X test)

          #R^2
          Ggb r2  = r2  score(y  test   pred1ctions Ggb)
          #COLcu£O£1ng /WSE
          Ggb rmse  np.sqrt(mean squared error(y test,predictions Ggb))

          print('Grid Search CV Gradient Boosting regression Model
          PerfDrmance:') print('Best Parameters = ',view bRst arams Ggb)
          print('R-squared = (:0.2}.'.format(Ggb r2))
          print('RM5E  = ',(Ggb rmse))

          Grid Search CV Gradient Boosting regression Model Performance:
          Best Parameters =   (' max depth' : 5, ' n est1uators ' : 19}
            sEuare0
          RS         . 2417391489664249
```

Chapter 5

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

Conclusion

1. 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.

$$RMSE =$$

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.

2. Model Selection

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

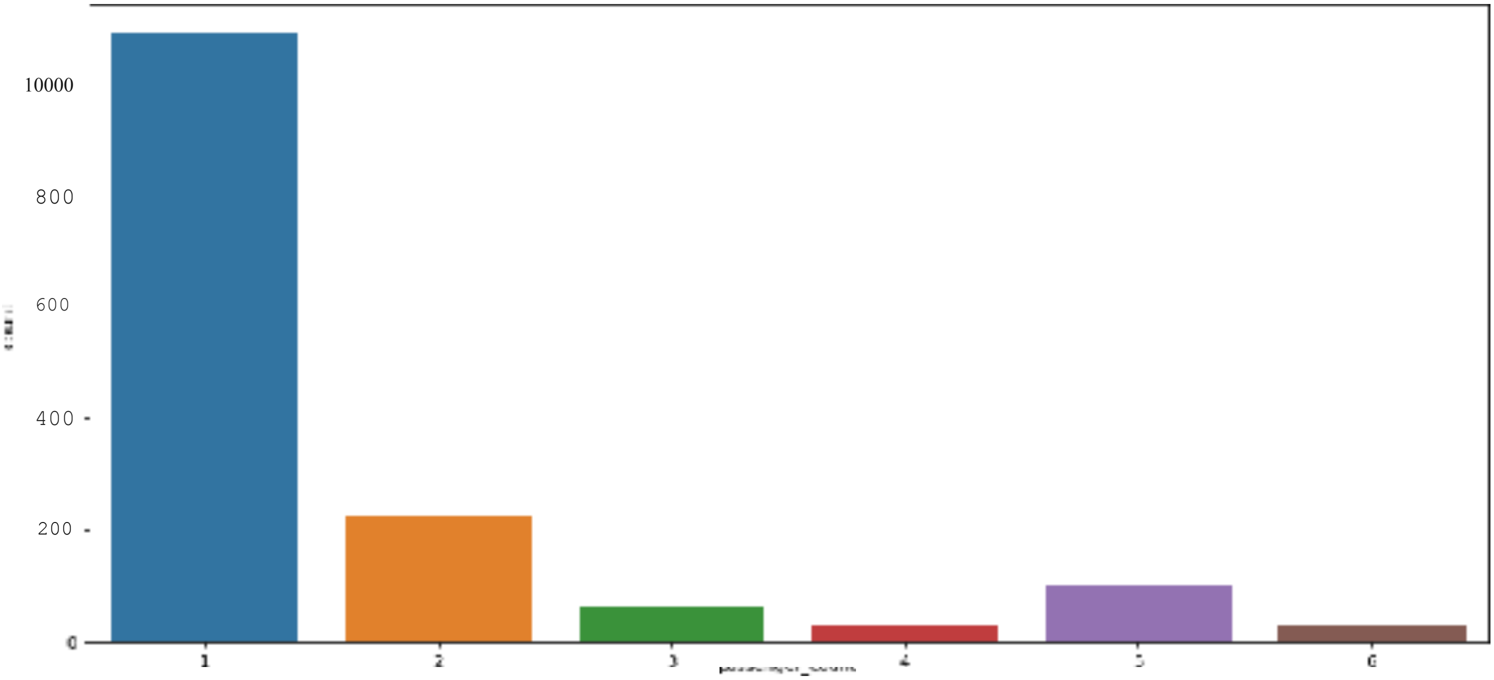
- From the observation of all RMSE Value and R-Squared Value we have concluded that,
- Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
- After this, I chose Random Forest CV and Grid Search CV to apply cross validation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

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

3. Some more visualization facts:

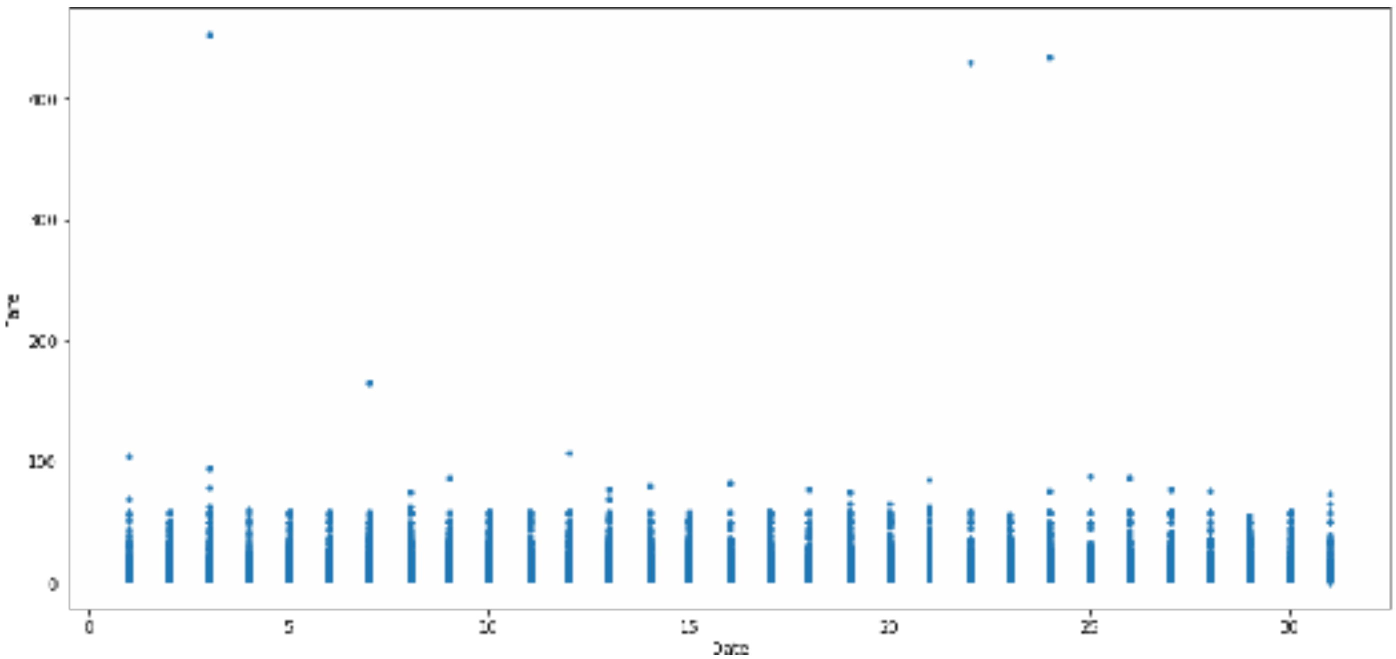
1. Number of passengers and fare

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
800
600
400
200
0
1
2
3
4
5
6

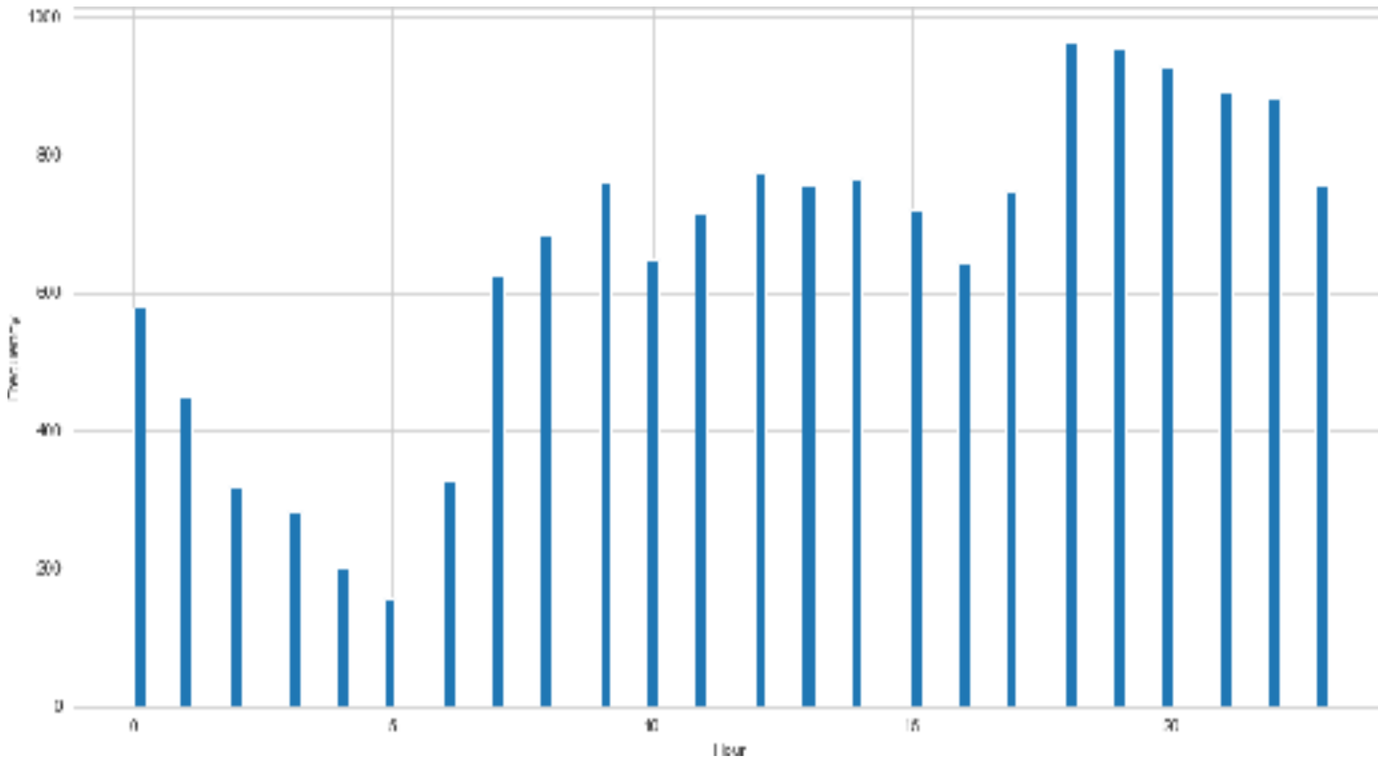
2. Date of month and fares

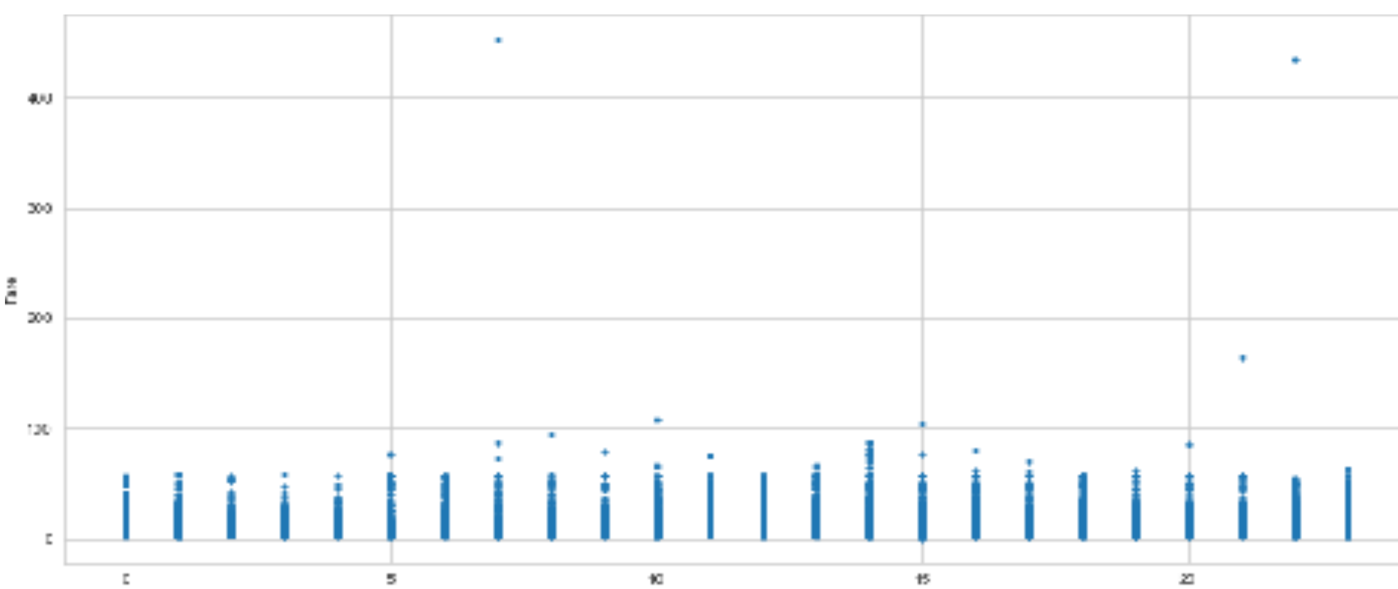
The fares throughout the month mostly seem uniform.



3. Hours and Fares

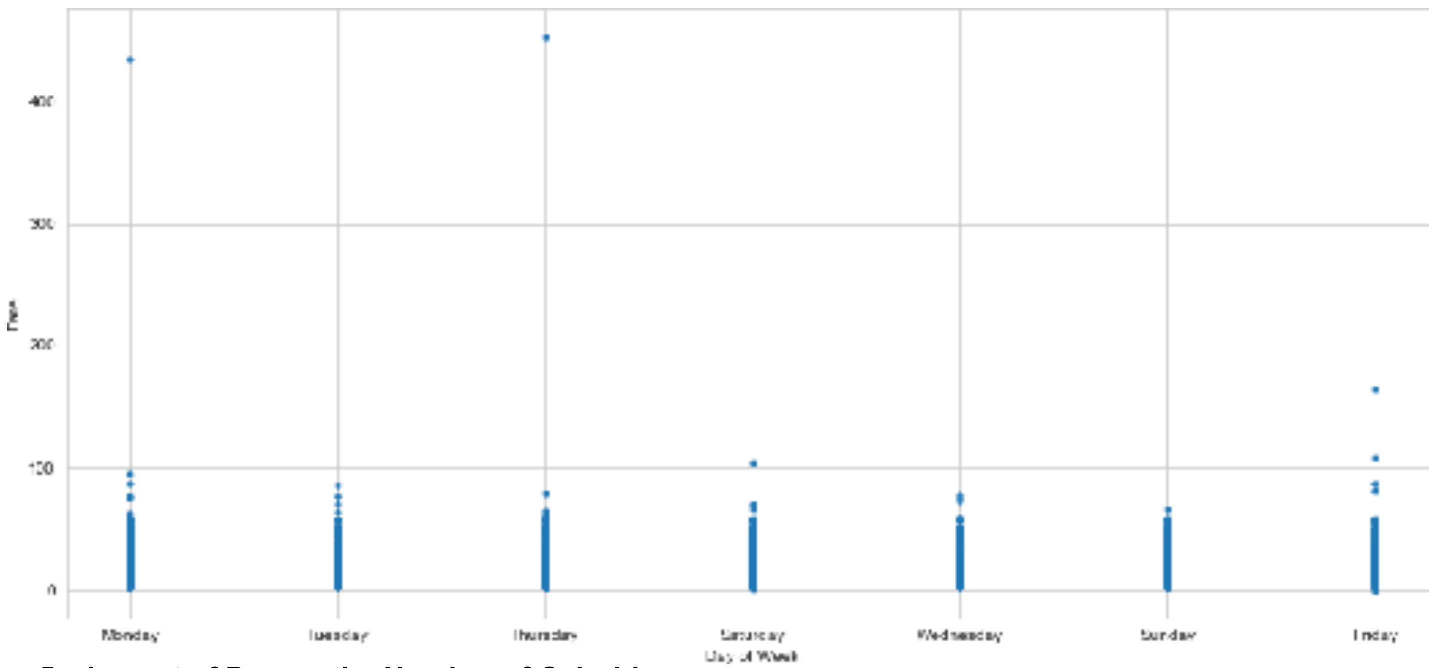
- During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours
- Fare prices during 2PM to 8PM is bit high compared to all other time might be due to high demands.



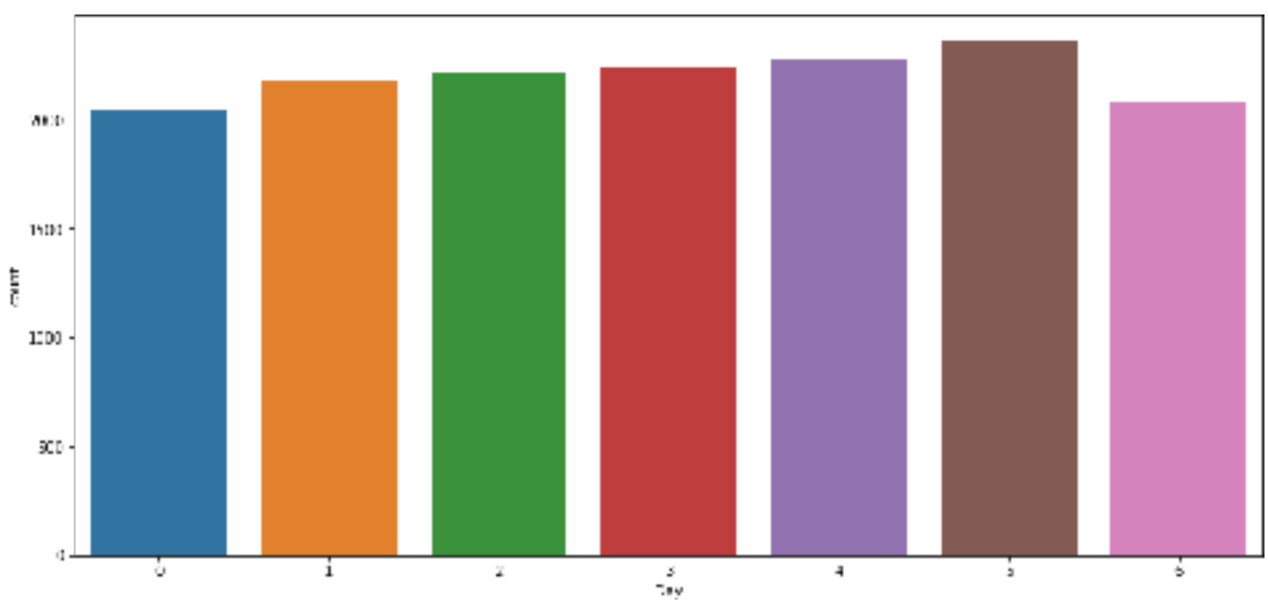


4. **<u>Week Day and fare</u>**

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



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



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

End of Report

References

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

Appendix

R code:

```
rm(list = ls())

setwd("/Users/monikawadhwani/Desktop/PROJECT")

getwd()

# #loading Libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "e1071","geosphere",
    "DataCombine", "pROC", "doSNOW", "class", "readxl","ROSE","dplyr", "plyr", "reshape","xlsx",
    "pbapply", "unbalanced", "dummies", "MASS" , "gbm" ,"Information", "rpart", "tidyr", "miscTools")


# #install.packages if not

lapply(x, install.packages)


# #load libraries

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

rm(x)


#Input Data Source

df = data.frame(read.csv('train_cab.csv'))

df2 = data.frame(read.csv('test.csv'))



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

#               EXPLORING DATA                                                        #

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


#viewing the data

head(df)


##########################################################################################
######################
```

```
#structure of data or data types
str(df)

#Summary of data
summary(df)

#unique value of each count
apply(df, 2,function(x) length(table(x)))

df$pickup_datetime <- gsub('\\ UTC','',df$pickup_datetime)

#Splitting Date and time
df$Date <- as.Date(df$pickup_datetime)
df$Year <- substr(as.character(df$Date),1,4)
df$Month <- substr(as.character(df$Date),6,7)
df$Weekday <- weekdays(as.POSIXct(df$Date), abbreviate = F)
df$Date <- substr(as.character(df$Date),9,10)
df$Time <- substr(as.factor(df$pickup_datetime),12,13)

#Now we can drop the column pickup_datetime as we have different columns
df = subset(df, select = -c(pickup_datetime))



##############################################################################
#          Checking Missing data                                             #
##############################################################################
apply(df, 2, function(x) {sum(is.na(x))}) # in R, 1 = Row & 2 = Col

#Creating dataframe with missing values present in each variable
```

```
null_val = data.frame(apply(df,2,function(x){sum(is.na(x))}))
```

```
null_val$Columns = row.names(null_val)
names(null_val)[1] =  "null_percentage"

#Calculating percentage missing value
null_val$null_percentage = (null_val$null_percentage/nrow(df)) * 100



# Sorting null_val in Descending order
null_val = null_val[order(-null_val$null_percentage),]
row.names(null_val) = NULL

# Reordering columns
null_val = null_val[,c(2,1)]

#viewing the % of missing data for all variales
null_val

#We have seen that null values are very less in our data set i.e. less than 1%.
#So we can delete the columns having missing values
df <- DropNA(df)

#Verifying missing values after deletion
sum(is.na(df))

names(df)

# Convert degrees to radians- Our data is already in radians, so skipping this step
#deg2rad <- function(deg) return(deg*pi/180)

# Calculates the geodesic distance between two points specified by
```

radian latitude/longitude using the Haversine formula

```
lat1 = df['pickup_latitude']

lat2 = df['dropoff_latitude']

long1 =df['pickup_longitude']

long2 = df['dropoff_longitude']


##### Function to calculate distance ######


gcd_hf <- function(long1, lat1, long2, lat2) {
  R <- 6371.145 # Earth mean radius [km]
  delta.long <- (long2 - long1)
  delta.lat <- (lat2 - lat1)
  a <- sin(delta.lat/2)^2 + cos(lat1) * cos(lat2) * sin(delta.long/2)^2
  c <- 2 * atan2(sqrt(a),sqrt(1-a))
  d = R * c
  return(d) # Distance in km
}

#Running the function for all rows in dataframe
for (i in 1:nrow(df))
{
  df$distance[i]= gcd_hf(df$pickup_longitude[i], df$pickup_latitude[i], df$dropoff_longitude[i],
                        df$dropoff_latitude[i])
}

#Now we can drop the columns for latitude/longitude as we have new column- Distance
df = subset(df, select = -c(pickup_latitude,dropoff_latitude,pickup_longitude,dropoff_longitude))

##############################################################################################
##########
```

```
##########################################################################################
##########

#We have seen that fare_amount has negative values which should be removed


df$fare_amount[df$fare_amount<=0] <- NA

df$fare_amount[df$fare_amount>500] <- NA



sum(is.na(df))


#So we can delete the columns having missing values

df <- DropNA(df)


#Verifying missing values after deletion

sum(is.na(df))


summary(df)



###removing passangers count more than 6

df$passenger_count[df$passenger_count<1] <- NA

df$passenger_count[df$passenger_count>6] <- NA


sum(is.na(df))


df <- DropNA(df)


sum(is.na(df))


summary(df)
```

```
###removing outliers in distance

df$distance[df$distance <= 0] <- NA
df$distance[df$distance > 500] <- NA

sum(is.na(df))

df <- DropNA(df)

sum(is.na(df))

summary(df)




# From the above EDA and problem statement categorizing data in 2 categories "continuous" and "categorical"
#Fare_amount being our target variable is excluded from the list.
cont = c( 'distance')

cata = c('Weekday', 'Month', 'Year' , 'Time', 'Date' , 'passenger_count')

##############################################################################
#                    Visualizing the data                               #
##############################################################################

#library(ggplot2)

#Plot fare amount Vs. the days of the week.
ggplot(data = df, aes(x = reorder(Weekday,-fare_amount), y = fare_amount))+
  geom_bar(stat = "identity")+
```

```
labs(title = "Fare Amount Vs. days", x = "Days of the week", y = "Fare")+
```

```
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="black", size=6, angle=45))

#Plot Fare amount Vs. months
ggplot(df,aes(x = reorder(Month,-fare_amount), y = fare_amount))+
  geom_bar(stat = "identity")+
  #ylim = c(0,1000) +
  labs(title = "Fare Amount Vs. Month", x = "Month", y = "Fare")+
  theme(axis.text.x = element_text( color="#993333", size=8))



######################################################################
#               Outlier Analysis                                   #
######################################################################

#We have done manual updation so we will skip this step

######################################################################
#               Feature Selection                         #
######################################################################

## Dimension Reduction
#We have already excluded the below columns that were redundant:
#pickup_datetime,
#pickup_latitude,
#dropoff_latitude,
#pickup_longitude,
#dropoff_longitude
#pickup_datetime
```

```
#We will remove Time column also as it is not required
##df = subset(df, select = -c(Time))

#####################################################################
#               Feature Scaling                              #
#####################################################################
#We will go for Normalization.

#Viewing data before Normalization.
head(df)




signedlog10 = function(x) {
  ifelse(abs(x) <= 1, 0, sign(x)*log10(abs(x)))
}



df$fare_amount = signedlog10(df$fare_amount)
df$distance = signedlog10(df$distance)

##checking distribution
hist(df$fare_amount)
hist(df$distance)






#Normalization
for(i in cont)
```

{

```
  print(i)
  df[,i] = (df[,i] - min(df[,i]))/(max(df[,i])-min(df[,i]))
}

hist(df$distance)

#Viewing data after Normalization.
head(df)

#Creating dummy variables for categorical variables
library(mlr)
df1 = dummy.data.frame(df, cata)

#Viewing data after adding dummies
head(df1)

#df1 = df

######################################################################
#                          Sampling of Data                          #
######################################################################

# #Divide data into trainset and testset using stratified sampling method

#install.packages('caret')
library(caret)
set.seed(101)
```

```
split_index = createDataPartition(df1$fare_amount, p = 0.7, list = FALSE)

trainset = df1[split_index,]

testset  = df1[-split_index,]


#Checking df Set Target Class

table(trainset$fare_amount)


####FUNCTION to calculate MAPE####

MAPE = function(y, yhat){

  mean(abs((y - yhat)/y))*100

}


##################################################################################################################################################################
##                                     Basic approach for ML - Models                                                                                      ##
##               We will first get a basic idea of how different models perform on our preprocesed data and then select the best model and make it          ##
##                                     more efficient for our Dataset                                                                                 ##
##################################################################################################################################################################


#_____________________________Decision tree_______________________________#

#Develop Model on training data

fit_DT = rpart(fare_amount ~., data = trainset, method = "anova")


#Variable importance

fit_DT$variable.importance


#          distance          Time05 passenger_count

##       725793.64246       431.82787        13.85704
```

```
#Lets predict for test data
pred_DT_test = predict(fit_DT, testset)

# For test data
print(postResample(pred = pred_DT_test, obs = testset$fare_amount))

#Compute R^2
dt_r2 = rSquared(testset$fare_amount, testset$fare_amount - pred_DT_test)
print(dt_r2)

#Compute MSE
dt_mse = mean((testset$fare_amount - pred_DT_test)^2)
print(dt_mse)

#Compute MAPE
dt_mape = MAPE(testset$fare_amount, pred_DT_test)
print(dt_mape)

#  RMSE      Rsquared       MAE
 # 0.12        0.59      0.01




#______________________________Linear Regression______________________________#

#Develop Model on training data
fit_LR = lm(fare_amount ~ ., data = trainset)
```

#Lets predict for test data

```
pred_LR_test = predict(fit_LR, testset)

# For test data
print(postResample(pred = pred_LR_test, obs = testset$fare_amount))

#Compute R^2
lr_r2 = rSquared(testset$fare_amount, testset$fare_amount - pred_LR_test)
print(lr_r2)

#Compute MSE
lr_mse = mean((testset$fare_amount - pred_LR_test)^2)
print(lr_mse)

#Compute MAPE
lr_mape = MAPE(testset$fare_amount, pred_LR_test)
print(lr_mape)

##RMSE      Rsquared    MAE
##0.13      0.53        0.01



#_____________________________Random Forest________________________________#

#Develop Model on training data
fit_RF = randomForest(fare_amount~., data = trainset)

#Lets predict for test data
pred_RF_test = predict(fit_RF, testset)
```

```
# For test data
print(postResample(pred = pred_RF_test, obs = testset$fare_amount))

#Compute R^2
rf_r2 = rSquared(testset$fare_amount, testset$fare_amount - pred_RF_test)
print(rf_r2)

#Compute MSE
rf_mse = mean((testset$fare_amount - pred_RF_test)^2)
print(rf_mse)

#Compute MAPE
rf_mape = MAPE(testset$fare_amount, pred_RF_test)
print(rf_mape)

#     RMSE   Rsquared        MAE
#

#R2
#

#MSE
#

#MAPE
#

#_______________________________XGBoost_______________________________#
### for xgboost it is required to make date variable as factor.
trainset$Date <- as.factor(trainset$Date)
```

```
#Develop Model on training data
fit_XGB = gbm(fare_amount~., data = trainset, n.trees = 500, interaction.depth = 2)

#Lets predict for test data
pred_XGB_test = predict(fit_XGB, testset, n.trees = 500)

# For test data
print(postResample(pred = pred_XGB_test, obs = testset$fare_amount))

#Compute R^2
xgb_r2 = rSquared(testset$fare_amount, testset$fare_amount - pred_XGB_test)
print(xgb_r2)

#Compute MSE
xgb_mse = mean((testset$fare_amount - pred_XGB_test)^2)
print(xgb_mse)

#Compute MAPE
xgb_mape = MAPE(testset$fare_amount, pred_XGB_test)
print(xgb_mape)

#     RMSE   Rsquared        MAE
#

#R2
#

#MSE
#
```

#MAPE

```
#

##################-------------------------------Viewing summary of all models------------------------------
################

# Create variables

MSE <- c(dt_mse, lr_mse, rf_mse, xgb_mse)

r2 <- c(dt_r2, lr_r2, rf_r2, xgb_r2)

MAPE <- c(dt_mape, lr_mape, rf_mape, xgb_mape)


# Join the variables to create a data frame

results <- data.frame(MSE,r2,MAPE)

results


#       MSE   r2     MAPE

#1

#2

#3

#4


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

#                                     Saving output to file
            #

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


#swrite.csv(submit,file = '/Users/monikawadhwani/Desktop/PROJECT/final.csv',row.names = F)

#rm(list = ls())
```