Project Report

On

Cab fare Prediction

AUTHOR: Moulieswaran R
Date : 20.07.2019

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.

1.1 Problem Statement

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

1.2 Data

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

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

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 Analysis
- Outlier Analysis
- Feature Selection
- Features Scaling

➢ 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
- KNeighbors regressor
- Decision Tree
- Random forest
- Gradient Boosting

❖ We have also used hyper parameter tunings to check the parameters on which our model runs best. Following techniques of hyper parameter tuning we have used:

- 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

3.1 Data exploration and Cleaning

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 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.

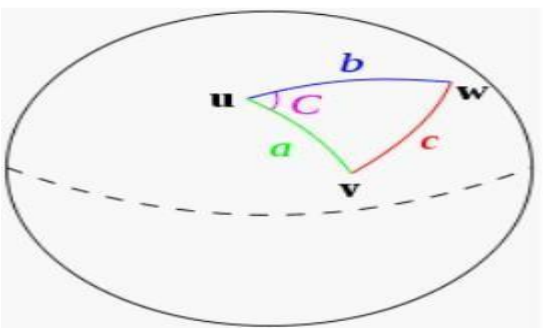
3.2 Creating some new variables from the given variables.

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

3.3 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

Now only following variables we will use for further steps:

VariableNames	Variable DataTypes
fare_amount	float64
passenger_count	object
Year	object
Month	object
Date	object
Day of Week	object
Hour	object
Distance	float64

3.4 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.

3.4.1 Below are the names of Independent variables:

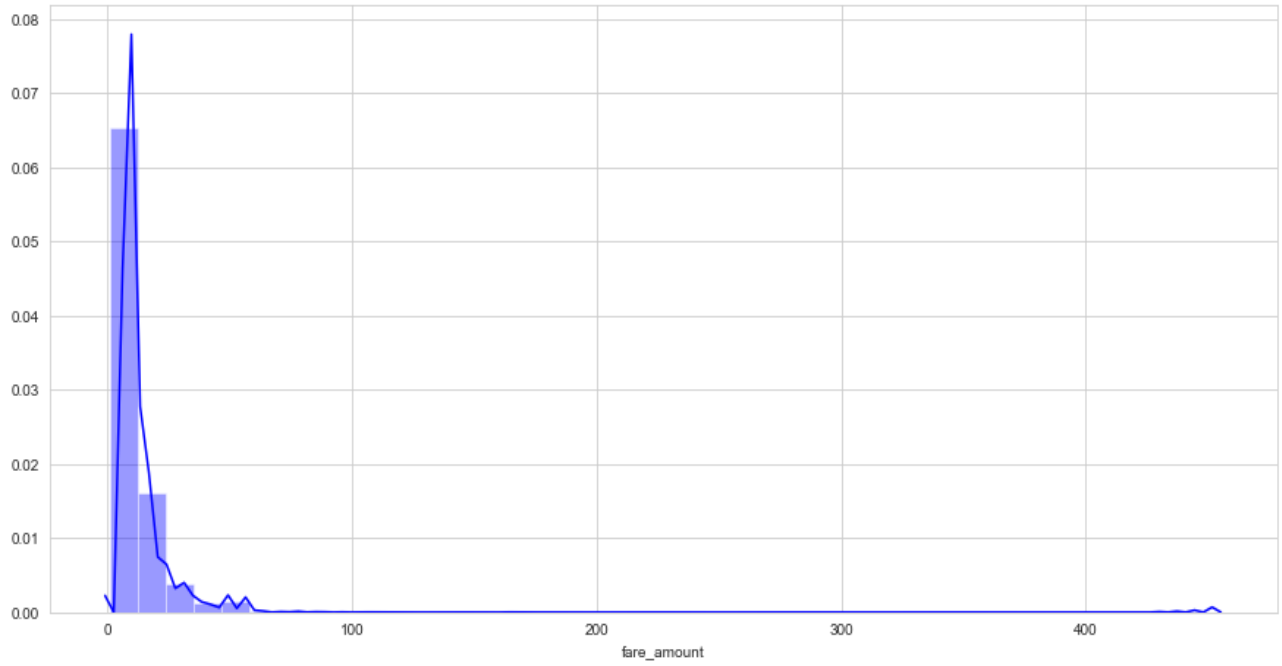
passenger_count, year, Month, Date, Day of Week, Hour, distance

Our Dependent variable is: **fare_amount**

3.4.2 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'

Distribution of target variable



3.5 Missing value analysis

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

Below table illustrate no missing value present in the data.

Fig 2.1 missing values

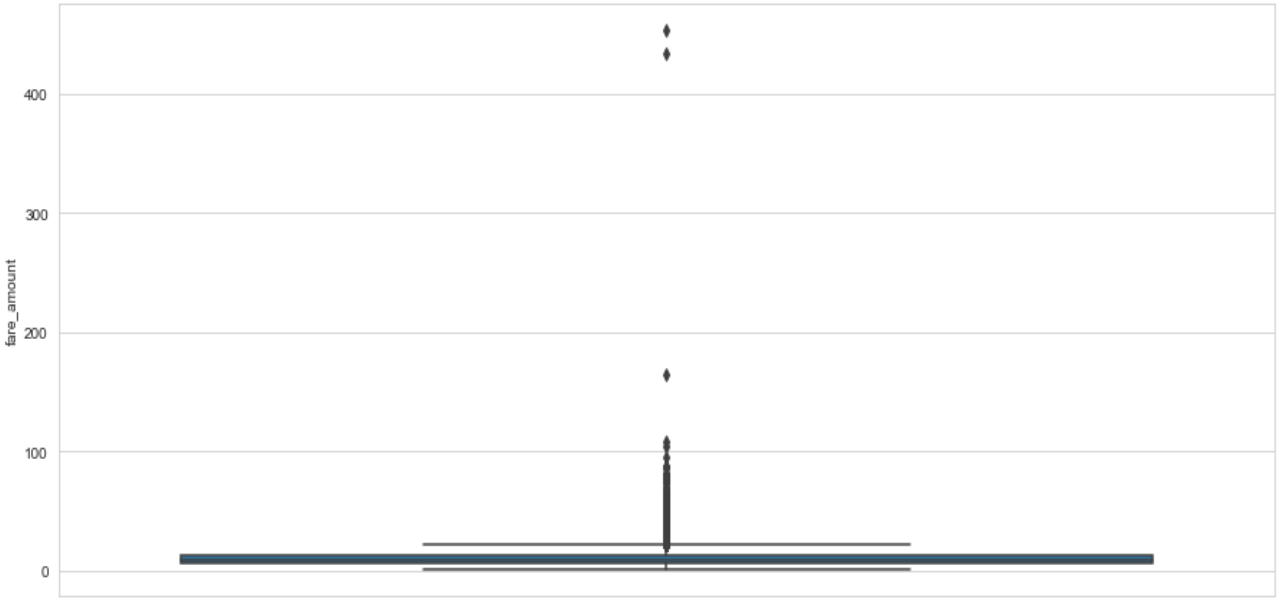
s.no	Variables	missing values
1	fare_amount	0
2	pickup_datetime	0
3	pickup_longitude	0
4	pickup_latitude	0
5	dropoff_longitude	0
6	dropoff_latitude	0
7	passenger_count	0
8	Year	0
9	Month	0
10	Date	0
11	Day of Week	0
12	Hour	0

3.6 Outlier Analysis

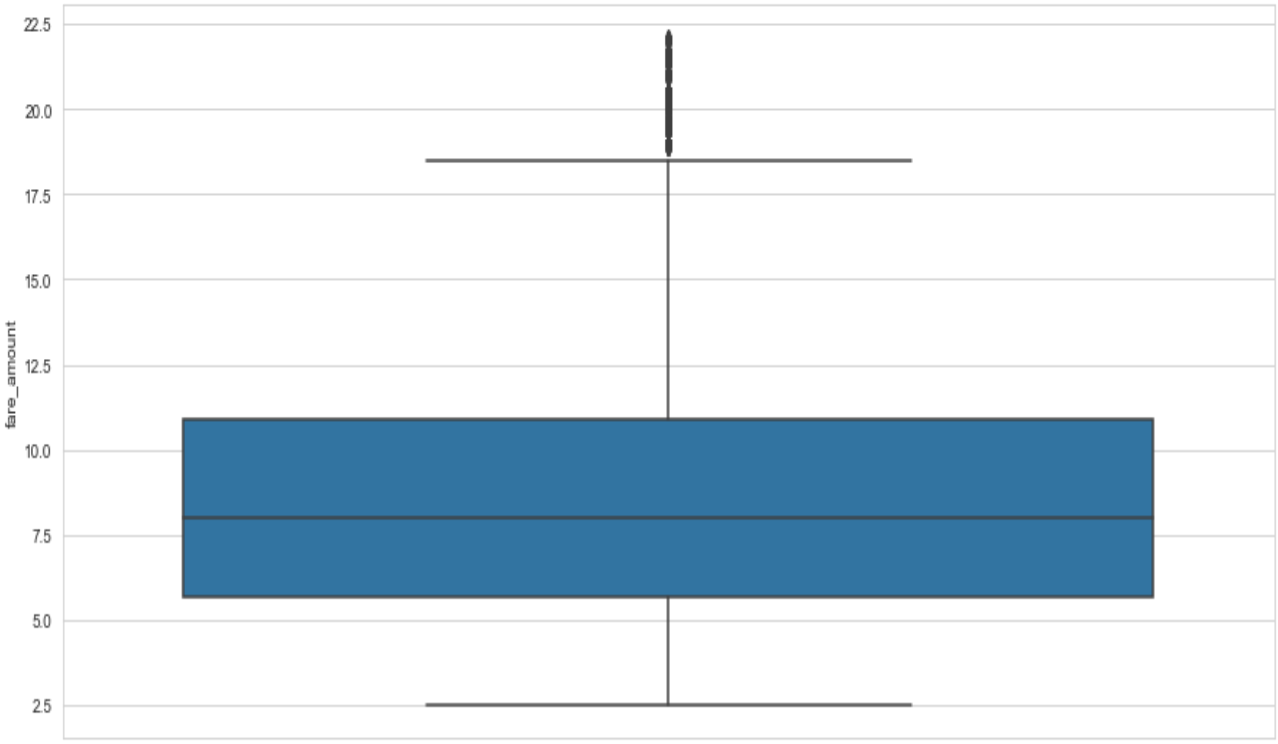
The Other steps of Preprocessing Technique is Outliers analysis, an outlier is an observation point that is distant from other observations. Outliers in data can distort predictions and affect the accuracy, if you don't detect and handle them appropriately especially in regression models.

As we observed , there is presence of outlier in independent variable 'distance' and 'fare_amount',

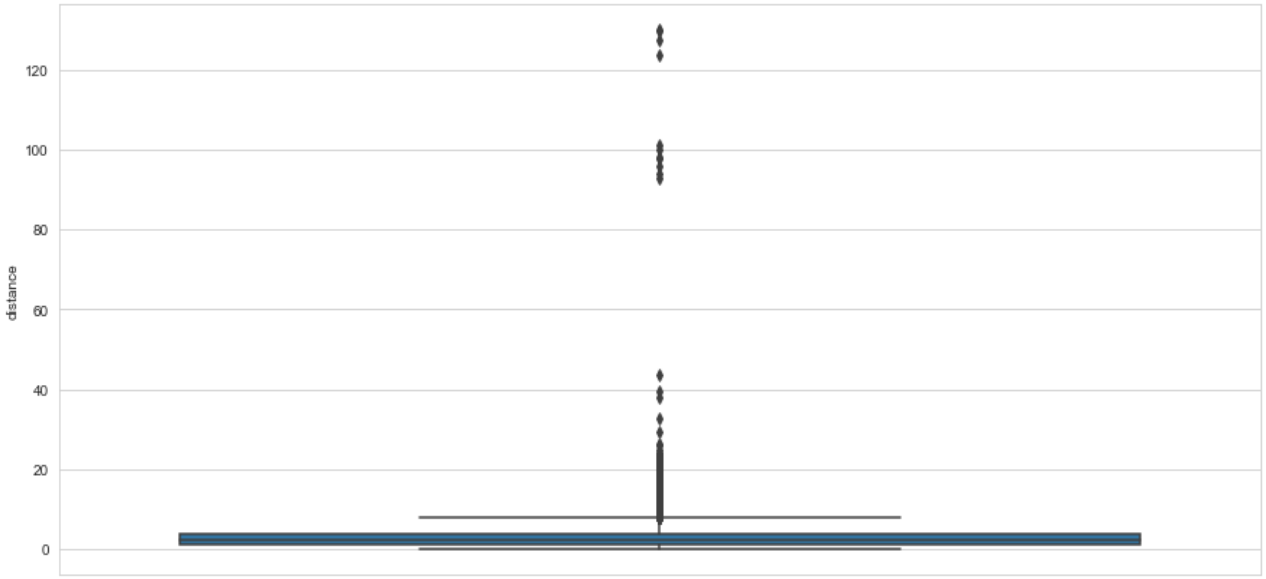
Show the presence of outlier in the variable ‘fare_amount’



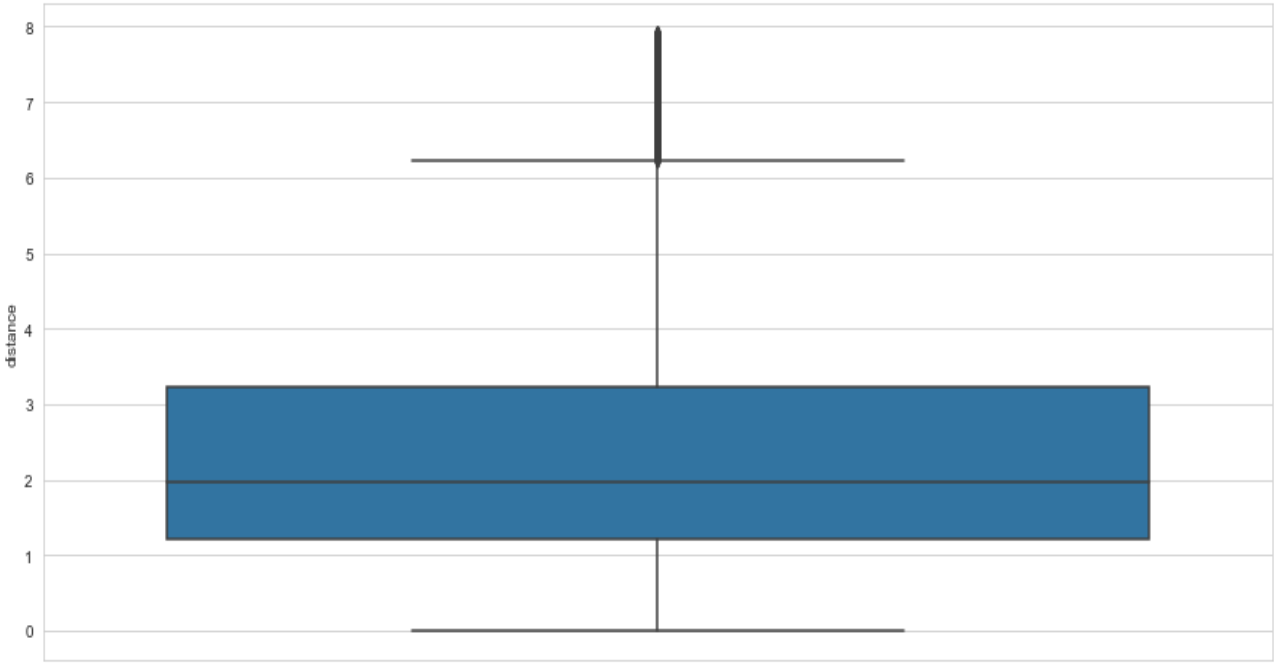
Show the Box plot of ‘fare_amount’ after removal of outlier



Show the presence of outlier in the variable ‘distance’



Show the Box plot of distance after removal of outlier



3.7 Feature Selection

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

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

We should consider the selection of feature for model based on below criteria

i. The relationship between two independent variable should be less and
ii. The relationship between Independent and Target variables should be high.

Below fig 2.6 illustrates that relationship between all numeric variables using Corrgram plot.

Correlation plot of numeric variables

Color pink indicates there is strong positive relationship and if darkness is decreasing indicates relation between variables are decreasing.

Color black indicates there is strong negative relationship and if darkness is decreasing indicates relationship between variables are decreasing.

Corrgram : it help us visualize the data in correlation matrices. correlograms are implimented through the **corrgram(x, order = , panel=, lower.panel=, upper.panel=, text.panel=, diag.panel=)**

Dimensionality Reduction for numeric variables

Above Fig 2.6 is showing

And it is also showing there is almost no relationship between independent variables

‘pickup_datetime’,’pickup_latitude’,’pickup_longitude’,‘dropoff_latitude’,‘dropoff_longitude’

and dependent variable ‘fare_amount’. So, these variable is not so important to predict.

Subsetting two independent features ‘pickup_datetime’,’pickup_latitude’,’pickup_longitude’

‘dropoff_latitude’ and ‘dropoff_longitude’ from actual dataset.

3.8 Feature Scaling

The word “normalization” is used informally in statistics, and so the term normalized data can have multiple meanings. In most cases, when you normalize data you eliminate the units of measurement for data, enabling you to more easily compare data from different places. Some of the more common ways to normalize data include:

Transforming data using a z-score or t-score. This is usually called standardization. In the vast majority of cases, if a statistics textbook is talking about normalizing data, then this is the definition of “normalization” they are probably using.

Rescaling data to have values between 0 and 1. This is usually called feature scaling. One possible formula to achieve this is.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

In rental dataset numeric variables like ‘temp’ , ‘atemp’ ,’hum’ and ‘ windspeed’ are in normalization form so , we have to Normalize two variables ‘casual’ and ‘registered’

After normalize ‘casual’ and ‘registered’ variables look like in table below where all values between 0 and 1.

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
- KNeighbors Regressor
- 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.

```
from sklearn.model_selection import train_test_split,GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(train_feature_selection.iloc[:,1:],train_feature_selection['fare_amount'],
                                                    test_size=0.2)
```

4.1 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:

```
from sklearn.linear_model import LinearRegression
linear_regression_model = LinearRegression().fit(X_train,y_train)


print (linear_regression_model.intercept_)
# print(linear_regression_model.coef_ )

# Lets print the summary model
summary_LR_Model = pd.DataFrame(linear_regression_model.coef_,X_train.columns,columns=['coeff'])
print(summary_LR_Model)
```

```
predict_LR_train = linear_regression_model.predict(X_train)
RMSE(y_train,predict_LR_train)

print()
predict_LR = linear_regression_model.predict(X_test)
RMSE(y_test,predict_LR,False)
```

```
 Trained Model performance:
 MAPE:   18.327231081823285
RMSE:    2.1280716515861955
 r^2 :   0.7057068553335915

 Predicted Model performance:
 MAPE:   18.629217786722208
RMSE:    2.1840829973158913
 r^2 :   0.7036986939542891
```

4.2 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

Decision tree

```
from sklearn.tree import DecisionTreeRegressor

# DT = DecisionTreeRegressor(max_depth=10,min_samples_split=9,max_leaf_nodes=49)
DT = DecisionTreeRegressor()
DT.fit(X_train,y_train)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

```
predict_DT_train = DT.predict(X_train)
RMSE(y_train,predict_DT_train)

print()

predictions_DT = DT.predict(X_test)
RMSE(y_test,predictions_DT,False)
```

```
 Trained Model performance:
 MAPE:   1.3057802049526498e-16
RMSE:    8.272820146126499e-17
 r^2 :   1.0

 Predicted Model performance:
 MAPE:   25.578430503654843
RMSE:    3.0907792298976555
 r^2 :   0.4072297587266779
```

Using GridSearchCV method, to improve the performance of Decision Tree.

Decision Tree Algorithm using GridSearchCV

```
##Overfitting and Grid search

from sklearn.model_selection import GridSearchCV

params = {'max_leaf_nodes': list(range(2,20)),
          'min_samples_split': list(range(2,10)),
          'max_depth': list(range(2,10))}


gs_CV_DT = GridSearchCV(DecisionTreeRegressor(random_state =42),params,n_jobs=1, verbose= 1)

gs_CV_DT.fit(X_train,y_train)

print(gs_CV_DT.best_estimator_)
```

```
predict_DT_train = gs_CV_DT.predict(X_train)
RMSE(y_train,predict_DT_train)

print()


predict_GS = gs_CV_DT.predict(X_test)

RMSE(y_test,predict_GS,False)
```

```
 Trained Model performance:
 MAPE:   17.959177450018736
RMSE:    2.0989720832482965
 r^2 :   0.7137002441905126

 Predicted Model performance:
 MAPE:   18.755934749723615
RMSE:    2.18974047977231
 r^2 :   0.7021974136459344
```

4.3 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.

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:

```
from sklearn.ensemble import RandomForestRegressor

RF_model = RandomForestRegressor(n_estimators=500, random_state= 43 ).fit(X_train,y_train)

print(RF_model)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=500,
                      n_jobs=None, oob_score=False, random_state=43, verbose=0,
                      warm_start=False)
```

```
# Predict the model using predict funtion
RF_predict_train = RF_model.predict(X_train)
RMSE(y_train,RF_predict_train)

print()

RF_predict= RF_model.predict(X_test)
RMSE(y_test,RF_predict,False)
```

```
 Trained Model performance:
 MAPE:   6.51955594728393
RMSE:    0.7817736287679398
 r^2 :   0.9603687859870876

 Predicted Model performance:
 MAPE:   18.036187950758332
RMSE:    2.1373090153634156
 r^2 :   0.7167744353477197
```

Check results after using GridSearch CV on Random forest

```
##Random forest with GridSearchCV

params = {'max_depth': list(range(2,20)),
          'min_samples_split': list(range(2,10)) ,
          'max_leaf_nodes': list(range(2,10))}

gs_CV_RF = GridSearchCV(RandomForestRegressor(random_state =42),params,n_jobs=1, verbose= 1)

gs_CV_RF.fit(X_train,y_train)

#print(gs_CV_RF.best_estimator_)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=42,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
                                       14, 15, 16, 17, 18, 19],
                         'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9],
                         'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```
# Predict the model using predict funtion
predict_GS_RF = gs_CV_RF.predict(X_train)
RMSE(y_train,predict_GS_RF)

print()


predict_GS_RF = gs_CV_RF.predict(X_test)
RMSE(y_test,predict_GS_RF,False)
```

```
 Trained Model performance:
 MAPE:   18.33018828124112
RMSE:    2.1383312163003314
 r^2 :   0.702869468764465

 Predicted Model performance:
 MAPE:   19.05387160261999
RMSE:    2.218932086064252
 r^2 :   0.6942283381020575
```

4.4 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:

```
from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor().fit(X_train,y_train)
print(GBR)

predict_GBR_train = GBR.predict(X_train)
RMSE(y_train, predict_GBR_train)

print()
predict_GBR = GBR.predict(X_test)
RMSE(y_test,predict_GBR,False)
```

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='auto',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
 Trained Model performance:
 MAPE:   16.16060653123229
RMSE:    1.9245340038115144
 r^2 :   0.7593095657260064

 Predicted Model performance:
 MAPE:   17.24788255983299
RMSE:    2.050954207420539
 r^2 :   0.7387140774859371
```

Chapter 5

Conclusion

5.1 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.

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

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

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

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	**MAPE, RMSE**		**R Squared**	
	Train	**Test**	**Train**	**Test**
`Linear Regression	18.32, 2.18	18.62, 2.18	0.705	0.703
Decision Tree	1.30e-16, 8.27e-17	25.5, 3.09	1	0.41
Random Forest model	6.52, 0.78	18.03, 2.13	0.96	0.71
Gradient Boosting	16.16, 1.92	17.24, 2.05	0.75	0.7387

Below table shows results post using hyper parameter tuning techniques:

Model Name	**Parameter**	**MAPE ,RMSE (Test)**	**RSquared**
Grid Search CV	Decision Tree	18.75, 2.18	0.70
	Random Forest	19.05, 2.22	0.69
	Gradient boosting	17.28, 2.05	0.7388

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 Grid Search CV under which we have given the range of n_estimators, depth and CV folds.

5.2 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:

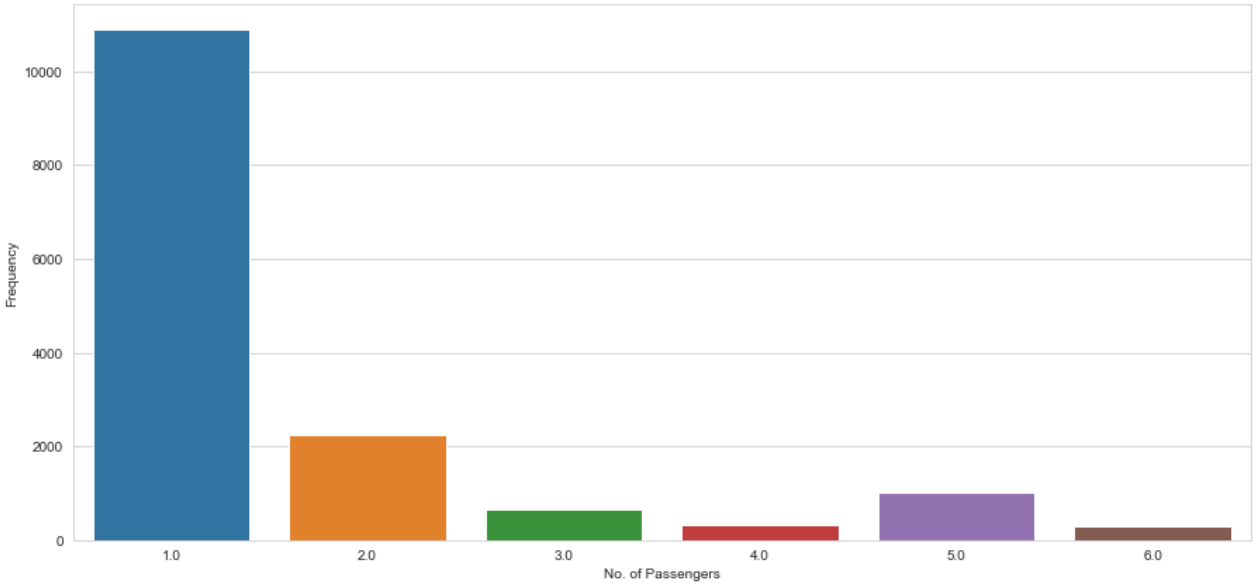
- From the observation of all RMSE Value and R-Squared Value we have concluded that,
- Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
- Compare to all other model, gradient boost model has 84.2% accuracy and RMSE is 2.01 and r^2 is 0.74. Hence, Gradient boost is best model for cab fare prediction

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.

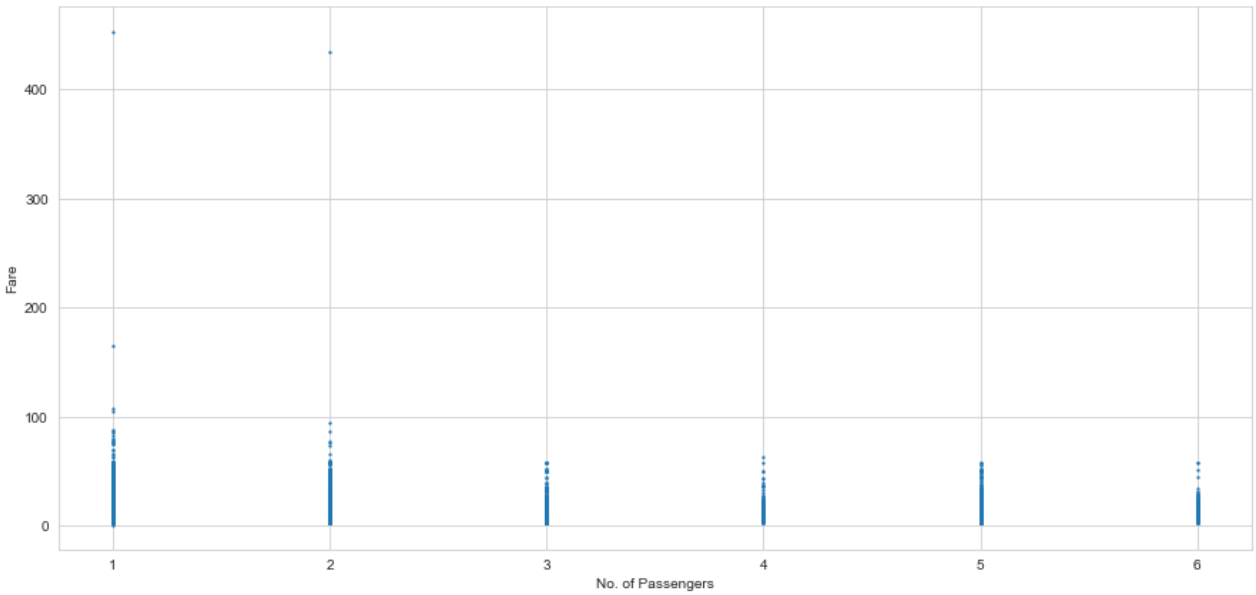
5.3 Some more visualization facts:

1. Does the number of passengers affect the fare?

Frequency of passenger count



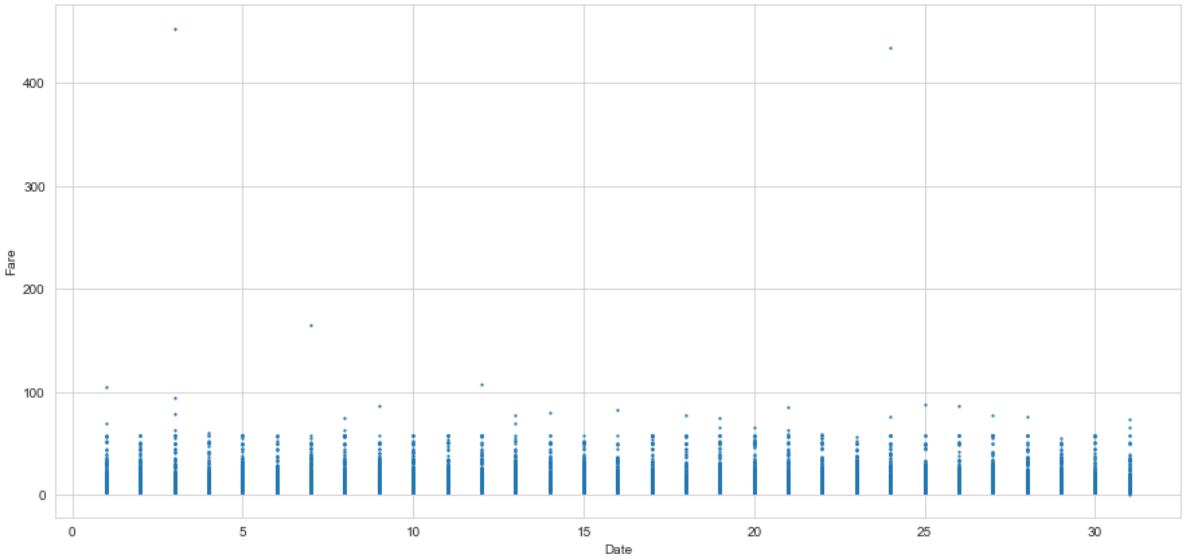
No of Passenger vs Fare



From the above 2 graphs we can see that single passengers are the most frequent travellers, and the highest fare also seems to come from cabs which carry just 1 passenger.

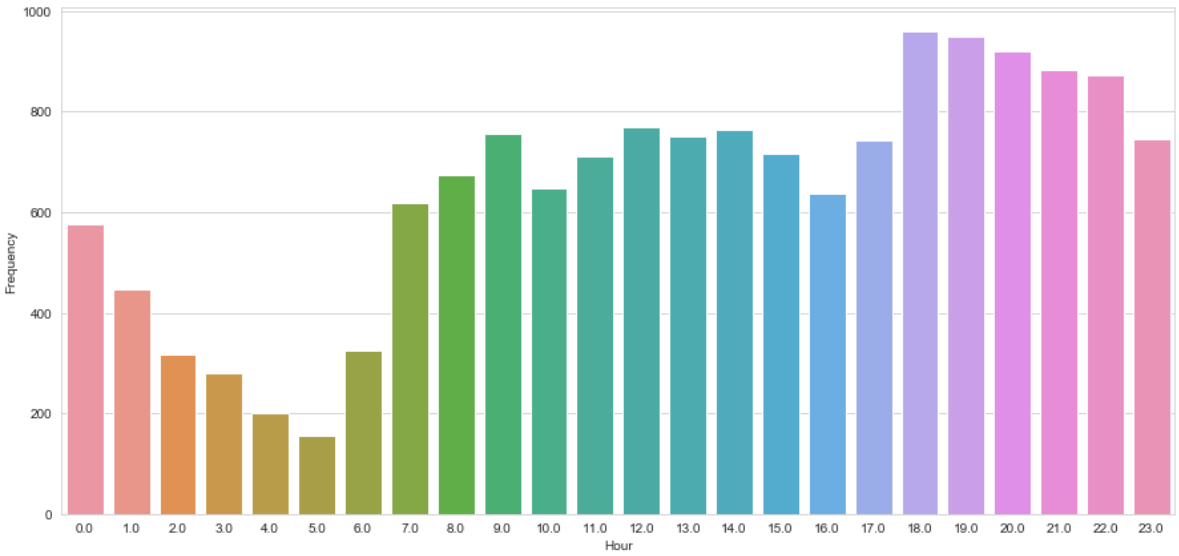
2. Does the date and time of pickup affect the fare?

Date vs Fare



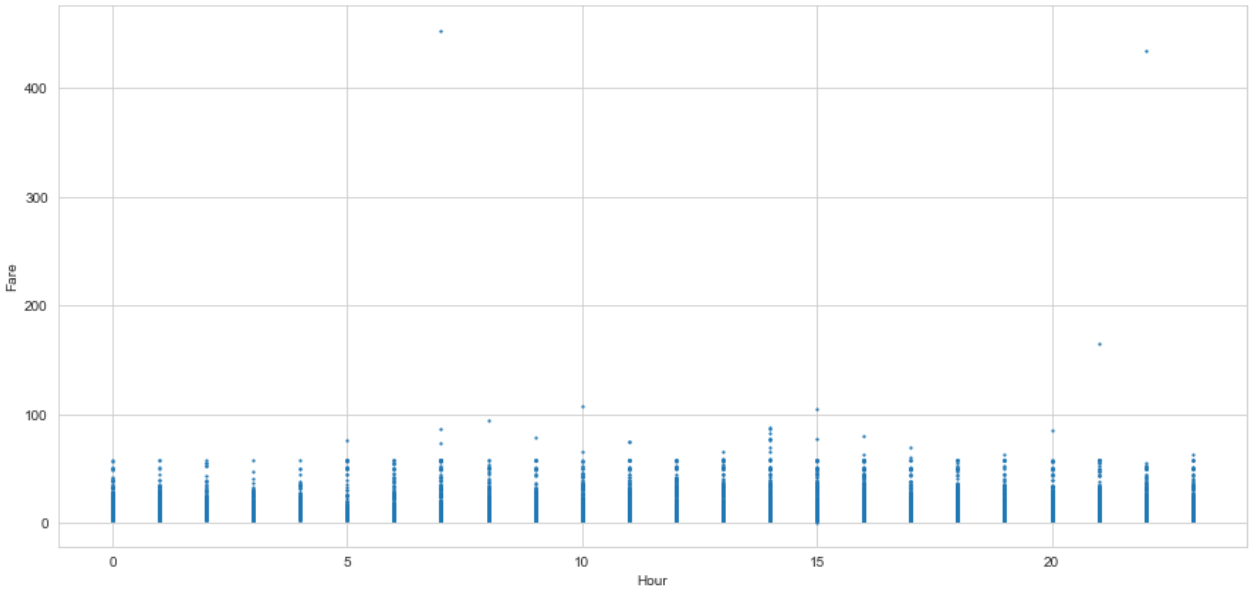
The fares throughout the month mostly seem uniform, with the maximum fare received on the 3rd.

No of trips vs hours



The frequency of cab rides seem to be the lowest at 5AM and the highest at 6PM.

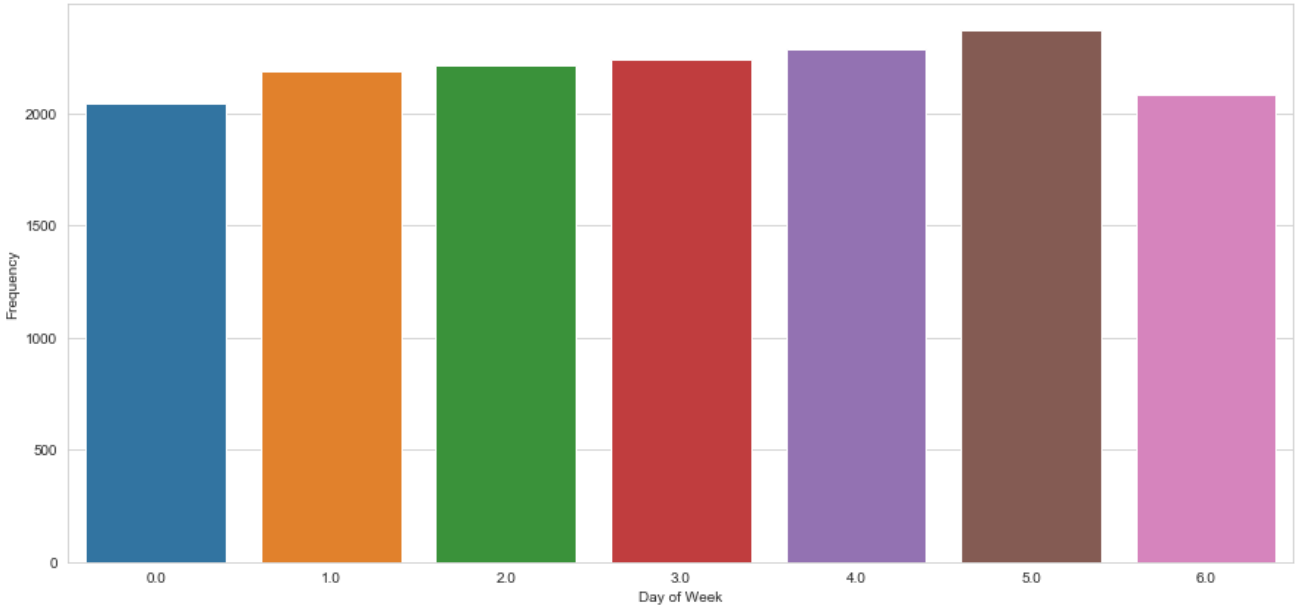
Hour vs Fare



The highest fare receive at 7 am and 10pm. Other than, fare rate seems to high between 7am to 10am

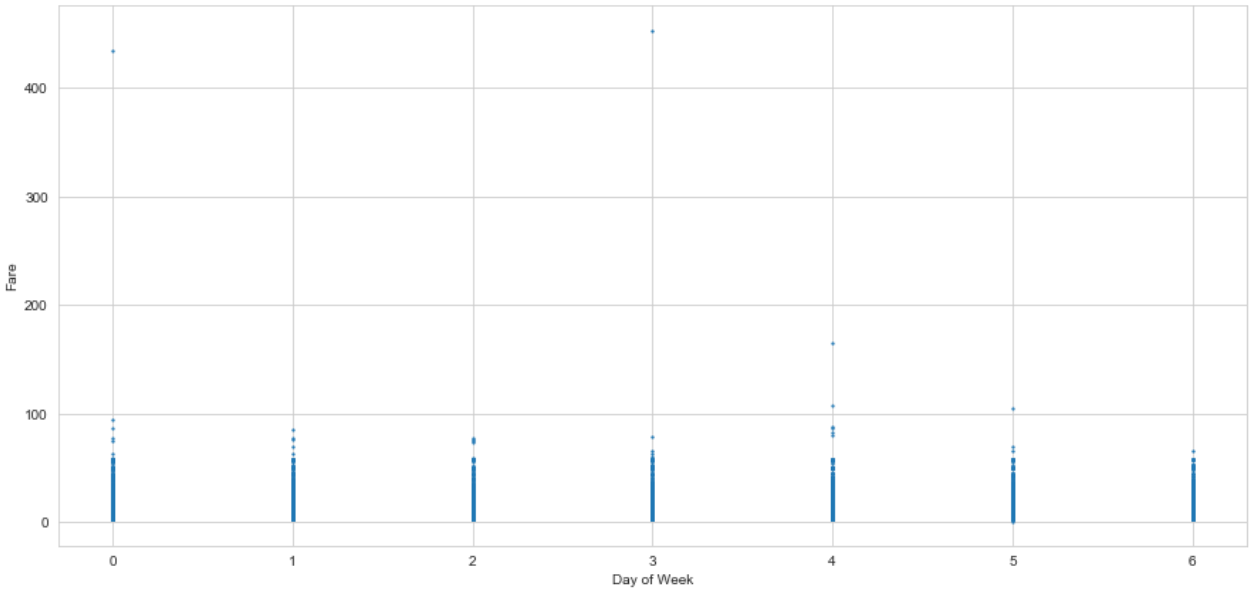
3. Does the day of the week affect the fare?

No of trip vs Day of Week



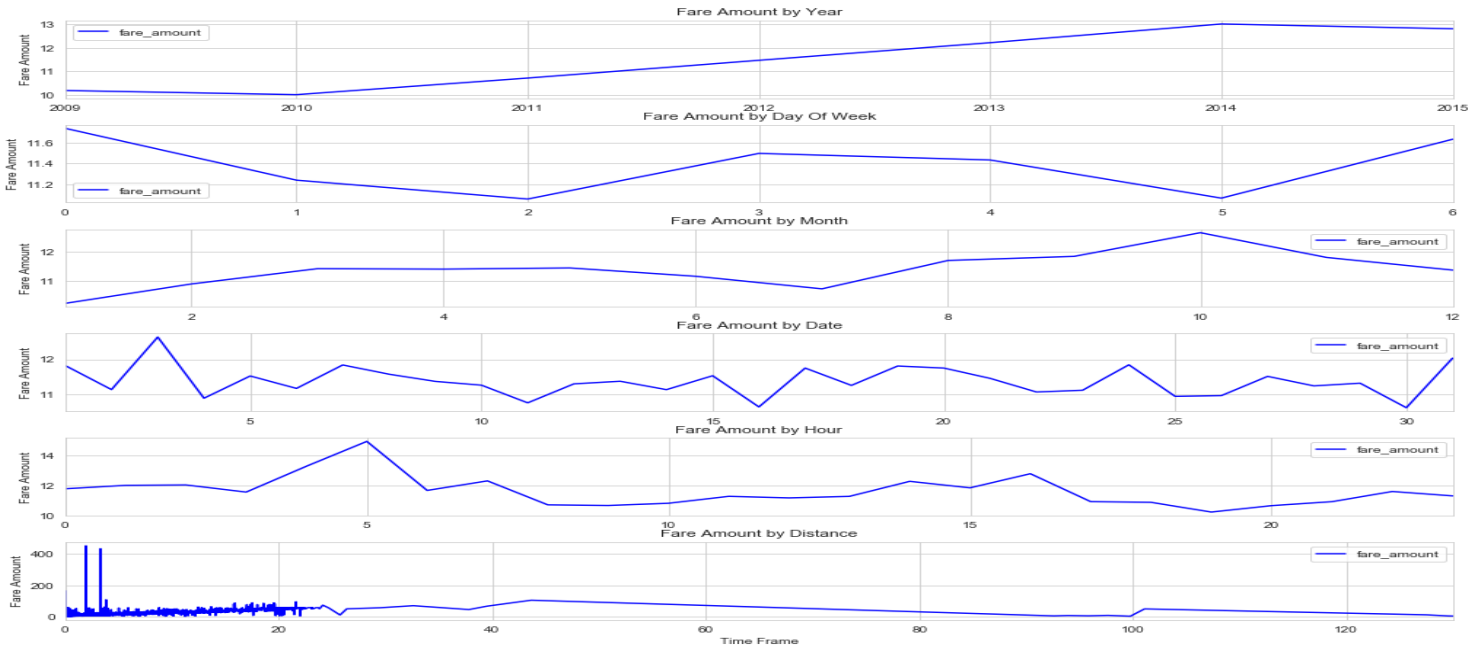
Day of week doesn't seem to have that much of an influence on the number of cab rides. Mostly saturday have maximum number of rides

Day of Week vs Fare



The fares throughout the Day of week mostly seem uniform, with the maximum fare received on Sunday and Wednesday

Time slicer graph for fare amount vs Independent variables



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/

Instructions to deploy and run the code.

- Save the python file as .py extension
- Open cmd prompt
- Before run the python file, make sure to set the path variable and install sklearn, numpy, mathplotlib, seaborn, pandas, fancyimpute library in the system
- Type cd <path of the folder contains python file> and enter
  EX: cd C:\Users\MOULIESWARAN\Downloads
- python <name of the file with extension> and then enter
  EX: python cab_fare_final.py
- The file will run in the cmd prompt

Appendix

R- code

```
rm(list =  ls(all=TRUE))
setwd("G:/Data science/Statistics/R Code")
getwd()


#load library
x = c('ggplot2','corrgram','DMwR','caret','randomForest','unbalanced','C50','dummies','e1071','Information',
     'MASS', 'rpart','gbm', 'ROSE','sampling', 'DataCombine', 'inTrees')
unlist(lapply(x, require, character.only = TRUE))

y = c("dplyr","plyr","ggplot2","data.table","GGally","magrittr","lubridate","tidyr","geosphere")
unlist(lapply(y, require, character.only = TRUE))


###Load the csv file
train=read.csv("train_cab.csv",header=T,na.strings = c(""," ","NA"))
# View(train)
test=read.csv("test_cab.csv",header=T,na.strings = c(""," ","NA"))
# View(test)\




#####################################Explore the data##########################################

str(train)
str(test)

#convert fare_amount to numeric type
# train$fare_amount=as.numeric(as.character(train$fare_amount))
train$fare_amount=as.numeric(as.character(train$fare_amount))
str(train$fare_amount)



#Expand pickup_date attribute
prepare_datetime = function (x){
```

```
  return(x %>%
           mutate(
             pickup_datetime = ymd_hms(pickup_datetime),
             month = month(pickup_datetime),
             year = year(pickup_datetime),
             day = day(pickup_datetime),
             dayOfWeek = wday(pickup_datetime),
             hour = hour(pickup_datetime)
           ))
}

train =  prepare_datetime(train)
test = prepare_datetime(test)




#passenger data can't be decimal so make it to floor value
# In given dataset, there are 57 data with 0 passenger count and remove it
# max 6 numbers allowed in a cab, Remove the data with passenger count greater than 7

prepare_passenger_count = function(x){
  x = x %>%
    filter(passenger_count>=1 , passenger_count <=6 ) %>%
    mutate( passenger_count = floor(passenger_count)   )
  # it removes na values as well
  return(x)
}

train = prepare_passenger_count(train) #133 records dropped (55 na + 78 condition unsatisfied)
test = prepare_passenger_count(test)




#Checking if any longitude is < -180 or > 180
summary(train$pickup_longitude)
summary(train$dropoff_longitude)

#Checking if any latitude is < -90 or > 90
summary(train$pickup_latitude)
```

```
summary(train$dropoff_latitude)

filter_lat_long = function(x){
  x = x %>%
    filter(pickup_latitude<=90 , pickup_latitude>=-90)
  x = x %>%
    filter(dropoff_latitude<=90 , dropoff_latitude>=-90)

  x = x %>%
    filter(pickup_longitude<=180 , pickup_longitude>=-180)

  x = x %>%
    filter(dropoff_longitude<=180 , dropoff_longitude>=-180)

  return(x)
}

train = filter_lat_long(train) # 1 data removed
test = filter_lat_long(test)



# ###################################Missing Values
Analysis#################################################
# missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
# missing_val
# apply(train,2, function(x) { sum(is.na(x))} )
sum(is.na(train))
##Total no of NA's:24
train = na.omit(train)




################################ Calculating distance from pickup and drop coordinates.
prepare_distance = function(x){

  x = x %>%
    mutate(distance.in.KM = by(x, 1:nrow(x), function(row) {
      distHaversine(c(row$pickup_longitude, row$pickup_latitude),
c(row$dropoff_longitude,row$dropoff_latitude))/1000}))
```

```
  #Removing the distance which is 0
  x=  x[-which(x$distance.in.KM == 0),]

  return(x)

}
train = prepare_distance(train)
test = prepare_distance(test)
#Getting Summary of distance
summary(train$distance.in.KM)
summary(test$distance.in.KM)


#Removing the distance which are > 150
train = train[-which (train$distance.in.KM > 150 ),]

#Removing the fare_amount >1000
train = train[- which (train$fare_amount > 1000 | train$fare_amount < 1), ]




#1. Does the number of passengers affect the fare?
######################Distribution of passenger count#########################

#Frequency of 1 passenger is high.
hist(train$passenger_count,xlab = "Passenger count ", main=paste("Hist of Passenger count"),col = "Yellow")


######################Distribution of passenger_count over fare################


#PLoting the graph for passenger count and Fare
gplot_p <- ggplot(data=train, aes(x=train$passenger_count, y=train$fare_amount)) + geom_point()+ geom_line()+
  ggtitle("Time and Fare Plot") +
  xlab("Passenger Count ") +
  ylab("Fare")
gplot_p

# From the Graph, passenger count is not affecting the fare.
# we can see that single passengers are the most frequent travellers, and the highest fare also seems to come from cabs which carry just 1 passenger.
```

#2. Does the date and time of pickup affect the fare?

```
######################Distribution of no of trips over the years################
train %>%
  dplyr::count(year)

train %>%
  dplyr::group_by(year) %>%
  dplyr::summarise(count = n() ) %>%
  ggplot( aes(x = year, y = count, fill = year)) +
  geom_bar(stat="identity") +
  theme(legend.position = "none") +
  labs(title = "Distribution of no of trips over the years",
       x = "passenger count",
       y = "Total Count")
```

#From the graph,the no of trips over the year is uniform,maximum : 2012 and minimum : 2015

```
######################Distribution of fare_amount over the years################
train %>%
  dplyr::count(year)

train %>%
  dplyr::group_by(year)%>%
  dplyr::summarise(fare_amount = mean(fare_amount) ) %>%
  ggplot( aes(x = year, y =fare_amount, fill = year)) +
  geom_bar(stat="identity") +
  theme(legend.position = "none") +
  labs(title = "Distribution of fare amount",
       x = "Years",
       y = "Fare amount")
```

#Avg Fare amount has beern increasing over the years.

```
######################Distribution of fare_amount over the months################
train %>%
  dplyr::count(month)
```

```
train %>%
  dplyr::group_by(month)%>%
  dplyr::summarise(fare_amount = mean(fare_amount) ) %>%
  ggplot( aes(x = month, y =fare_amount, fill = month)) +
  geom_bar(stat="identity") +
  theme(legend.position = "none") +
  labs(title = "Distribution of fare_amount over the months",
       x = "Months",
       y = "fare amount")


#The fares throught the month mostly seem uniform, with the maximum fare received on the 10th

######################Distribution of fare_amount over hours#################

#Hist of Hours
hist(as.numeric(train$hour), xlab = "Hours", main=paste("Hist of Hours"),col = "Red")

#from the graph,low frequency of the cab is found on  morning hours

#PLoting the graph for Hours and Fare
gplot_H <- ggplot(data=train, aes(x=train$hour, y=train$fare_amount)) + geom_point()+ geom_line()+
  ggtitle("Time and Fare Plot") +
  xlab("Hours ") +
  ylab("Fare")
gplot_H

#From the above graph we can see that the timeing is not affecting too much. Maximin dots are below 100.




#3. Does the day of the week affect the fare?

hist(as.numeric(train$dayOfWeek), xlab = "DayOfWeek", main=paste("Hist of Dayofweek"),col = "Red")

#day of the week doesn't seem to have that much of an influence on the number of cab rides

#Plotting the graph for passenger Dayofweek and Fare
```

```
#gplot_d <- ggplot(data=train, aes(x=train$dayOfWeek, y=train$fare_amount)) + geom_point()+ geom_line()+
# ggtitle("Day count and Fare Plot") +
# xlab("Day of Week ") +
#ylab("Fare")
#gplot_d

#The highest fares seem to be on a Sunday and Monday, and the lowest on Wednesday and Friday.




#4. Does the distance affect the fare?

######################Distribution of fare_amount over distance################

gplot <- ggplot(data=train, aes(x=train$distance.in.KM, y=train$fare_amount)) + geom_point()+ geom_line()+
  ggtitle("Distance and Fare Plot") +
  xlab("Distance in KM ") +
  ylab("Fare")
gplot

# From the above graph, distance is found to be  an important independent variable.



###################################Missing Values
Analysis#################################################
missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val

# No missing value is present in the data



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

# # ## BoxPlots - Distribution and Outlier Check
# numeric_index = sapply(train,is.numeric) #selecting only integer
```

```
# numeric_data = train[,numeric_index]
# cnames  = colnames(numeric_data)
#
# for( i in 1:length(cnames)) {
#   print(i)
#   assign(paste0('gn',i), ggplot( data = train,aes_string( y= cnames[i], x = 'fare_amount'),)+
#            stat_boxplot(geom = "errorbar", width = 0.5) +
#            geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
#                         outlier.size=1, notch=FALSE) +
#            theme(legend.position="bottom")+
#            labs(y=cnames[i],x="fare_amount")+
#            ggtitle(paste("Box plot of cnt for",cnames[i])))
# }
# # ## Plotting plots together
# gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)
# gridExtra::grid.arrange(gn3,gn4,gn5,ncol=3)
# gridExtra::grid.arrange(gn6,gn7,gn8,ncol=3)
# gridExtra::grid.arrange(gn9,gn10,gn11,ncol=3)
# gridExtra::grid.arrange(gn12,ncol=1)

cnames = c("distance.in.KM", "fare_amount")
for( i in cnames){
  print(i)
  val = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]
  train = train[ which(! train[,i] %in% val),]
}




###################################Feature
Selection################################################
## Correlation Plot

corrgram(train[,c('pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_longitude','dropoff_latitude','passen
ger_count','month', 'year', 'day' ,'hour' ,'dayOfWeek', 'distance.in.KM')],order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")




# names(train)
```

```
#  dimensionality  reduction

test = subset(test,select=-c(pickup_datetime,dropoff_longitude,dropoff_latitude,pickup_latitude,pickup_longitude))
train = subset(train,select=-c(pickup_datetime,dropoff_longitude,dropoff_latitude,pickup_latitude,pickup_longitude))




###################################Feature Scaling#################################################

#Normality check
#Train Data
cnames = c("distance.in.KM")
for( i in cnames){
  train[,i] = (train[,i] - min(train[,i])) / ( max(train[,i]) -min( train[,i]))
}
#Test data
for( i in cnames){
  test[,i] = (test[,i] - min(test[,i])) / ( max(test[,i]) -min( test[,i]))
}




################################### Evaluation matrics#########################################
EVM = function( y_actual, y_predict) {
  print("MAPE")
  m = mean(abs( (y_actual - y_predict) / y_actual) )
  print(m  )
  difference = y_actual - y_predict
  root_mean_square = sqrt(mean(difference^2))
  print("RMSE")
  print(root_mean_square)
  print("R square value")
  rss <- sum((y_predict - y_actual) ^ 2)  ## residual sum of squares
  tss <- sum((y_actual - mean(y_actual)) ^ 2)  ## total sum of squares
  rsq <- 1 - (rss/tss)
  print(rsq)
}
```

```
#####################################Model Development#########################################
#Clean the environment
rmExcept(c("train","test","EVM"))
set.seed(1234)
train.index = createDataPartition(train$fare_amount,p= 0.8,list = F)
train_data = train[train.index,1:8]
test_data = train[c(-train.index),1:8]

################################### LINEAR REGRESSION
#########################################

linerModel <- lm(fare_amount ~., data = train_data)
summary(linerModel)

linear_predict <- predict(linerModel,test_data)

EVM(test_data[,1],linear_predict)

# "MAPE" =0.1804
# "RMSE" =1.9441
# R-squared = 0.7070



################################### DECISION TREE MODEL
#########################################

DT = rpart(fare_amount ~ ., data = train_data, method = 'anova')


predictions_DT = predict( DT, test_data[,-1])
print(DT)

EVM(test_data[,1], predictions_DT)

# "MAPE" = 0.209386
# "RMSE" = 2.138644
# R-squared = 0.6455116
```

```
################################### RANDOM FOREST ##########################################

RF = randomForest(fare_amount ~ .,train_data , importance = TRUE, ntree = 500)

predictions_RF = predict( RF, test_data[,-1])
# plot(RF)

EVM(test_data[,1], predictions_RF)
# "MAPE" = 0.2233
# "RMSE" = 2.1763
# R-squared = 0.632



################################### GRADIENT BOOST ##########################################

model_gbm <- caret::train(fare_amount ~ .,
                          data = train_data,
                          method = "gbm",
                          distribution="gaussian",
                          trControl = trainControl(method = "repeatedcv",
                                                   number = 5,
                                                   repeats = 3,
                                                   verboseIter = FALSE),
                          verbose = 0)

data_gbm = predict(model_gbm, test_data)
EVM(test_data[,1], data_gbm)

# "MAPE" = 0.1729488
# "RMSE" = 1.896542
# R-squared = 0.7212275




test$fare_amount <- predict(model_gbm,test)
```

```
write.csv(test, file="cab_fare_prediction.csv", row.names = FALSE)
submission=read.csv("cab_fare_prediction.csv",header=T,na.strings = c("","  ","NA"))
```