Project Report

Cab Fare Prediction

Data Science

(Submitted By: Ankit Anand)

***Abstract*—** In this report, the fare amount of a cab ride is detected using machine learning algorithms. From the given train_data, a feature set of distance is created from the pickup and drop-off latitude and longitude. From the datetime variable features like year and hour of travel are extracted which affect the fare prices. Using this feature data, different models are built: Decision Tree, Linear Regression, Random Forest, KNN classifier. Final prediction is of the fare amount of the given test_data, which only have latitude and longitude columns. From the given model's best error rates was with Random Forest model, which is then implemented on the given test data for the final fare prediction.

Contents

Introduction

CRISP-DM Methodology....1-1
Problem Statement……..1-1

Business Understanding

2.1 Define Objective....2-1
2.2 Identify Data Source....2-1

Data Understanding

3.1 Train_cab data....3-1
3.2 Test Data....3-2

Data Preparation

4.1 Missing value....4-1
4.2 Outlier Analysis....4-2
4.3 Feature Selection....4-3

Modelling

5.1 Decision Tree....5-1
5.2 Random Forest....5-1
5.3 KNN…….5-2
5.4 Linear Regression……...5-3

Evaluation

6.1 Error Matrix……6.1
6.1 Regression Evaluation measure of error....6-1

Model Selection

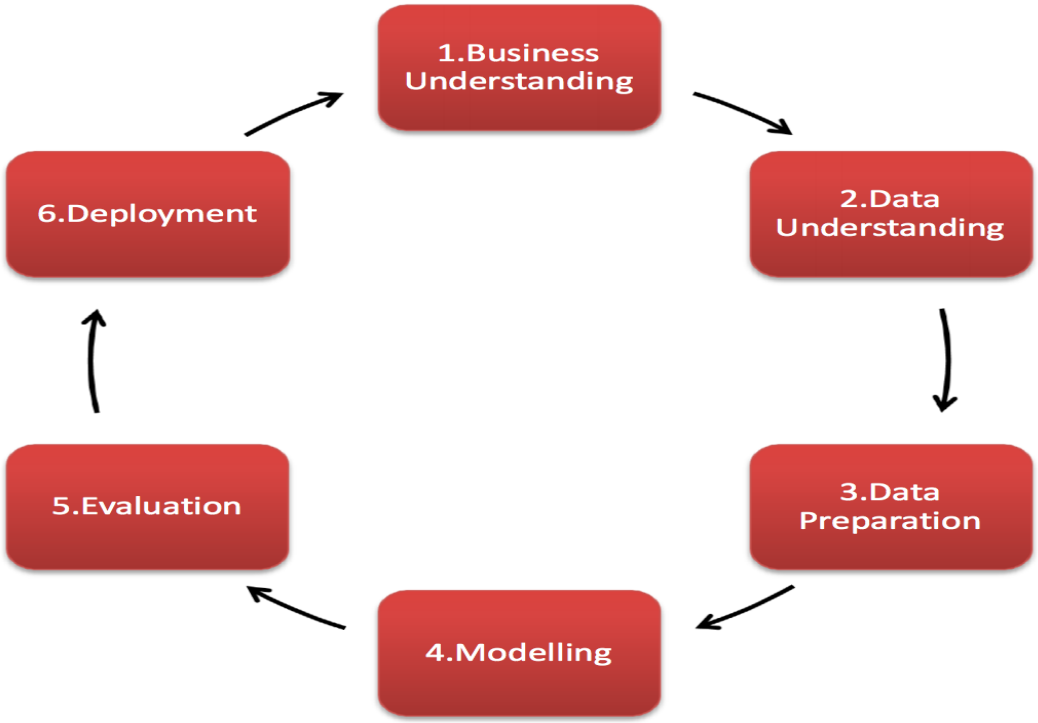
7.1 Selecting Model…….....7-1

Appendix

A. Python Code

B. R code

1.Introduction: CRISP-DM Methodology



CRISP-DM Methodology: Describes commonly used approach to solve Business problem

CRISP-DM includes 6 major phases

- Business Understanding
- Data understanding
- Data Preparation
- Data modelling
- Evaluation
- Deployment

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. Business Understanding

There are two main tasks addressed in this stage:

- **Define objectives**: Work with our customer and other stakeholders to understand and identify the business problems. Formulate questions that define the business goals that the data science techniques can target.
- **Identify data sources:** Find the relevant data that helps to answer the questions that define the objectives of the project.

Define objective

1. A central objective of this step is to identify the key business variables that the analysis needs to predict. These variables are referred as the *model targets*, and the metrics associated with them is used to determine the success of the project.
2. Define the project goals by asking and refining "sharp" questions that are relevant, specific, and unambiguous. Data science is a process that uses names and numbers to answer such questions. We typically use data science or machine learning to answer five types of questions:
    - How much or how many? (regression)
    - Which category? (classification)
    - Which group? (clustering)
    - Is this weird? (anomaly detection)
    - Which option should be taken? (recommendation)

    We determine which of these questions we're asking and how answering it achieves our business goals.

Identify data sources

Identify data sources that contain known examples of answers to our sharp questions. Look for the following data:

- Data that's relevant to the question. Do we have measures of the target and features that are related to the target?
- Data that's an accurate measure of our model target and the features of interest.

With the advancement of technology there are new ideas coming up. Cab rental methodology in one among them. It has become one of the leading market shares for the economy. The main reason for its success is cabs easy availability and also its accurate fare predictions.

Our main task for this project is dig into the historical data provided and looks into the features which affect the fare amount. This is very important in business point of view because this prediction will decide the myth of the company progress. If the predicted fare is greater than actual, then it will it affect the customer and will give negative impact to the company. If the predicted fare is less than the actual, then it will results loss to the company.

So, from both Company as well Customer point of view cab fare prediction should have to be accurate.

3.Data Understanding

There are two sets of data:

Data Set : 1) train_cab 2) test

Number of attributes:

- pickup_datetime - timestamp value indicating when the cab ride started.
- pickup_longitude - float for longitude coordinate of where the cab ride started.
- pickup_latitude - float for latitude coordinate of where the cab ride started.
- dropoff_longitude - float for longitude coordinate of where the cab ride ended.
- dropoff_latitude - float for latitude coordinate of where the cab ride ended.
- passenger_count - an integer indicating the number of passengers in the cab ride.

Missing values = yes

1) **Train_cab data**: Count-16067, Variables- 7

Here, **Target Variable** = fare_amount

Independent Variables= pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, passenger_count

```
train.head()
```

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

```
train.describe()
```

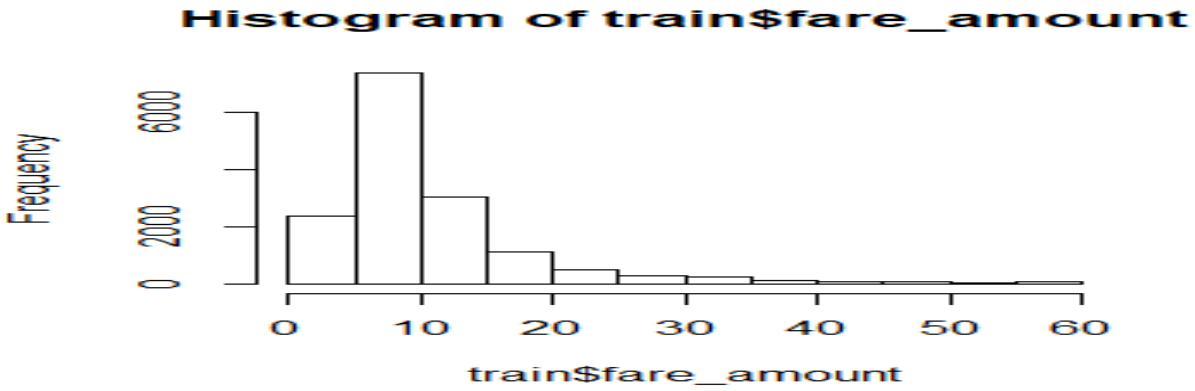
	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	10.578384	6.826587	10.575062	6.187087	60.844122
min	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	40.766125	401.083332	40.802437	41.366138	5345.000000

2) Test Data: Count=9914, Variables= 6

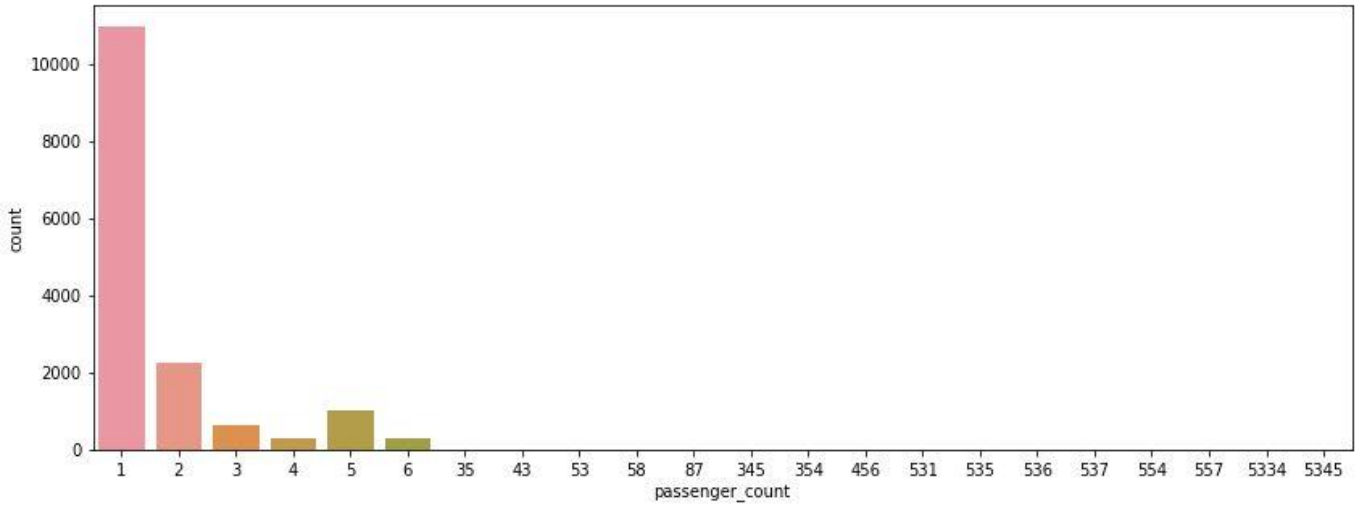
	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	1
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	1
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	1
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	1
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273
std	0.042774	0.033541	0.039072	0.035435	1.278747
min	-74.252193	40.573143	-74.263242	40.568973	1.000000
25%	-73.992501	40.736125	-73.991247	40.735254	1.000000
50%	-73.982326	40.753051	-73.980015	40.754065	1.000000
75%	-73.968013	40.767113	-73.964059	40.768757	2.000000
max	-72.986532	41.709555	-72.990963	41.696683	6.000000

Distribution of target Variable(fare_amount)



Distribution of Passengr_count



4.Data Preparation

Goals

- Produce a clean, high-quality data set whose relationship to the target variables is understood. Locate the data set in the appropriate analytics environment so that it is ready to model.

Before we train our models, we need to develop a sound understanding of the data. Real-world data sets are often noisy, are missing values, or have a host of other discrepancies. We can use data summarization and visualization to audit the quality of your data and provide the information we need to process the data before it's ready for modelling. This process is often iterative.

Missing Value analysis.

The concept of missing values is important to understand in order to successfully manage data. If the missing values are not handled properly, then we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from ones where the missing values are present.

Percent of missing values in our data before imputation

```
missingval(train)
```

	Variables	count	Missing_percentage
0	passenger_count	112	0.717719
1	fare_amount	24	0.153797
2	pickup_longitude	12	0.076898
3	pickup_latitude	12	0.076898
4	dropoff_longitude	11	0.070490
5	dropoff_latitude	9	0.057674
6	pickup_datetime	0	0.000000

Also, there are many observations with value of 0, so we should impute it with missing value NA. In our data there are many observations where pickup_latitude is equal to dropoff_latitude, so we should remove these observation as they don't provide any information.

```
train[train['pickup_longitude']==train['dropoff_longitude']].count()
```

```
fare_amount          460
pickup_datetime      462
pickup_longitude     462
pickup_latitude      462
dropoff_longitude    462
dropoff_latitude     462
passenger_count      462
dtype: int64
```

Impute missing values:

- **Fill with central statistics**: Mean Median Mode
- **Distance based or Data mining method**: KNN imputation
- **Prediction Method**

To impute missing values with this method, a sample observation is picked. Its value is noted and then it is filled with NA. After performing these methods, the method whose value is close to the chosen sample, is picked and applied to the missing value.
For example:

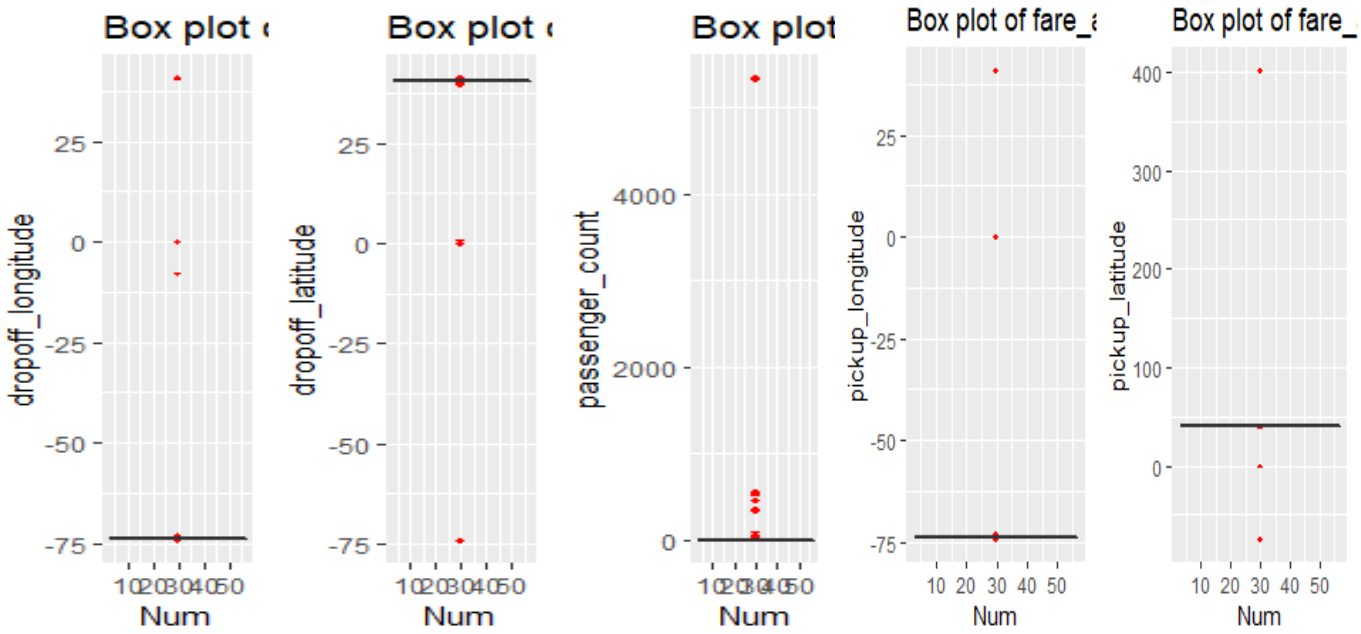
actual, train['pickup_longitude'].loc[50]	-**73.985582**
KNN train['pickup_longitude'].loc[50]	-73.9777079999999
median train['pickup_longitude'].loc[50]	**-73.9820605**
mean train['pickup_longitude'].loc[50]	-73.91160183651274

From the result Median provides the best result. So, Impute the missing values with this method.
Also ,variable **passenger_count** is a categorical variable, so we should impute it with mode method.

Outlier Analysis:

Observations inconsistent with rest of the dataset Global Outlier.
Causes of Outliers:
• Poor data quality / contamination
• Low quality measurements, malfunctioning equipment, manual error
• Correct but exceptional data
To detect the Outlier in the data – Graphical approach- Box Plot method.

The outliers in our data are.



Outlier in Fare_amount:

```
train[train['fare_amount']<0]
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2039	-2.9	2010-03-09 23:37:10 UTC	-73.789450	40.643498	-73.788665	40.641952	1.0
2486	-2.5	2015-03-22 05:14:27 UTC	-74.000031	40.720631	-73.999809	40.720539	1.0
13032	-3.0	2013-08-30 08:57:10 UTC	-73.995062	40.740755	-73.995885	40.741357	4.0

Outlier in passenger_count:

```
train[train['passenger_count']>6]
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
233	8.5	2011-07-24 01:14:35 UTC	0.000000	0.000000	0.000000	0.000000	236.0
263	4.9	2010-07-12 09:44:33 UTC	-73.983249	40.734655	-73.991278	40.738918	456.0
293	6.1	2011-01-18 23:48:00 UTC	-74.006642	40.738927	-74.010828	40.717907	5334.0
356	8.5	2013-06-18 10:27:05 UTC	-73.992108	40.764203	-73.973000	40.762695	535.0
386	8.1	2009-08-21 19:35:05 UTC	-73.960853	40.761557	-73.976335	40.748361	354.0
413	NaN	2013-09-12 11:32:00 UTC	-73.982060	40.772705	-73.956213	40.771777	55.0
971	10.1	2010-11-21 01:41:00 UTC	-74.004500	40.742143	-73.994330	40.720412	554.0
1007	3.7	2010-12-14 14:46:00 UTC	-73.969157	40.759000	-73.968763	40.764617	53.0
1043	5.7	2012-08-22 22:08:29 UTC	-73.973573	40.760184	-73.953564	40.767392	35.0

After detecting the outlier, we should replace it with NA. The NA values after removing the outlier is:

fare_amount	35
pickup_datetime	0
pickup_longitude	779
pickup_latitude	497
dropoff_longitude	893
dropoff_latitude	737
passenger_count	17

After substituting outliers with NA, we should impute all the variables with the best possible method, in our case median. The variable fare-amount is target variable so instead of imputing it we should drop the NA available.

Feature Selection:

Selecting a subset of relevant features (variables, predictors) for use in model construction. Subset of a learning algorithm's input variables upon which it should focus attention, while ignoring the rest.
Feature Selection is the process where we automatically or manually select those features which contribute most to our prediction variable or output in which we are interested in.

The main criterial of deciding the features between the variables is the correlation analysis. It is performed on the numerical variable. Whether to select a variable or remove it depends on how they are corelated. Ideally there should be no correlation between two independent variables but high correlation between the dependent and independent variable. After performing the correlation analysis different features which can be added are:

1. Distance Variable.

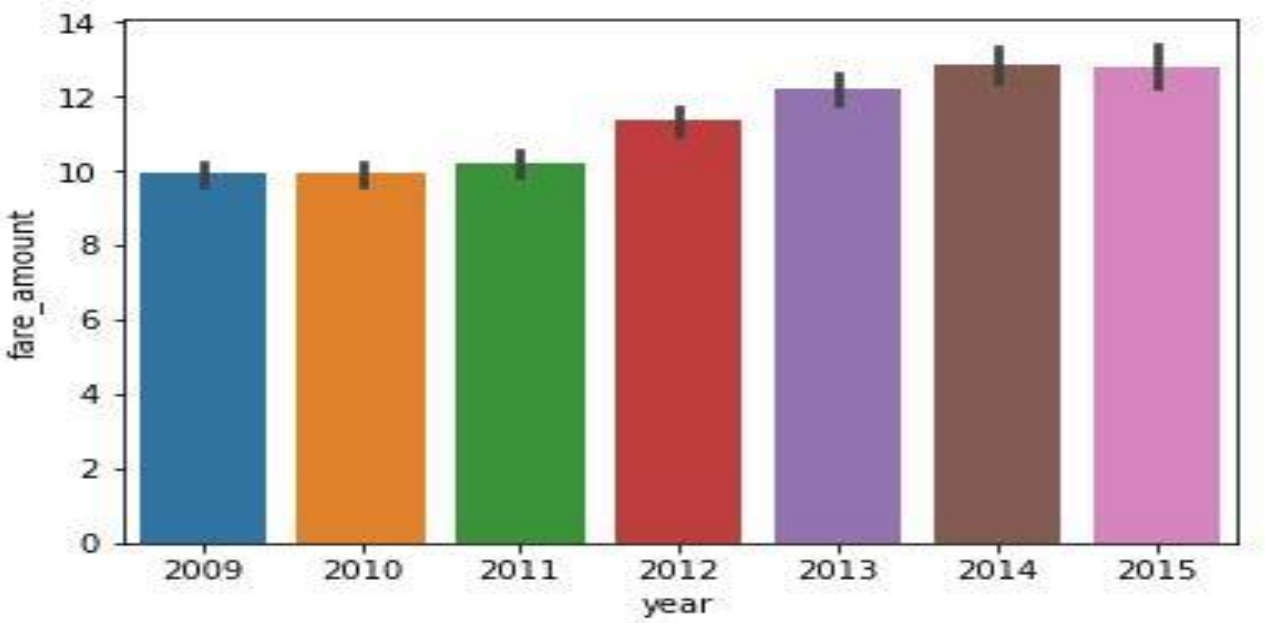
Our data contains the pickup and drop-off longitude and latitude which basically indicate the distance travelled. So, using these variables we can add a new variable called the Distance variable. In R there is a package available called geosphere which can be used to calculate distance between two points using their longitude and latitude.

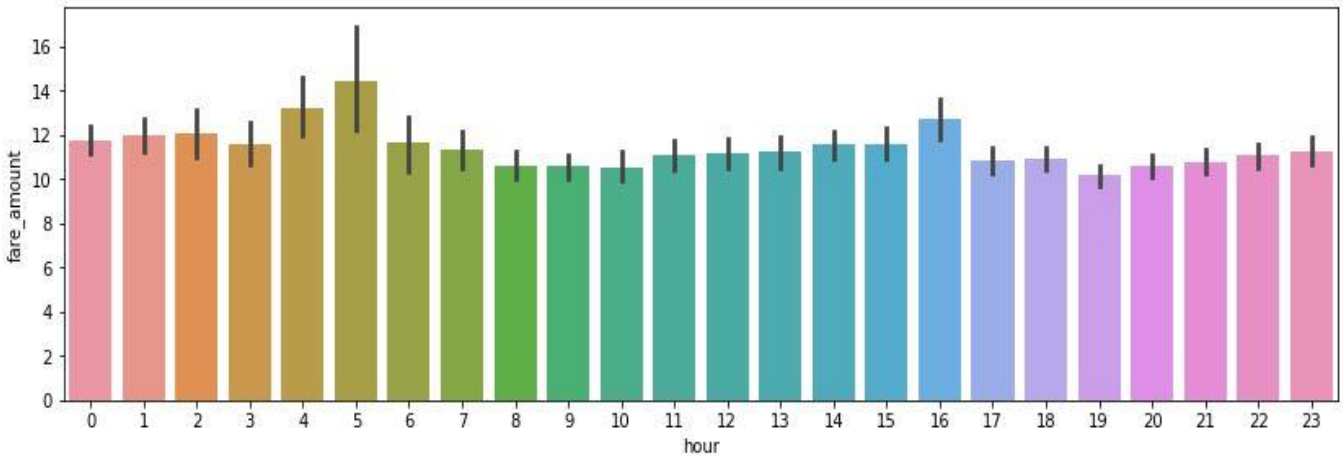
A function in python to calculate the distance between two points using its longitude and latitude is given below .It is also called as haversine function. In python it is defined as shown below.

```
def distance(s_lat, s_lng, e_lat, e_lng):

    # approximate radius of earth in km
    R = 6373.0

    s_lat = s_lat*np.pi/180.0
    s_lng = np.deg2rad(s_lng)
    e_lat = np.deg2rad(e_lat)
    e_lng = np.deg2rad(e_lng)

    d = np.sin((e_lat - s_lat)/2)**2 + np.cos(s_lat)*np.cos(e_lat) * np.sin((e_lng - s_lng)/2)**2

    return 2 * R * np.arcsin(np.sqrt(d))
```

2. DateTime Variable.

From the variable in our data called pickup_datetime we can create a new date time objects like Year, Month, Hour, DayOfWeek, etc. In python by using pandas to_datetime we can create new variables Year ,Hour. Dependences of these variables on target variables is as shown:





In R we can create the new variable using the date time using these commands.

```
# creating additional column from datetime to year, month,day,dayOfWeek,hour,partOfDay
train <- mutate(train,
                pickup_datetime = ymd_hms(pickup_datetime),
                month = as.factor(month(pickup_datetime)),
                year = as.numeric(year(pickup_datetime)),
                day = day(pickup_datetime),
                dayOfWeek = as.factor(wday(pickup_datetime)),
                hour = hour(pickup_datetime),
                partOfDay = as.factor(round(hour * 2 / 10)),
                hour = as.factor(hour(pickup_datetime))

)
```

After adding the new variables, the correlation plot between these variables is shown:

5. Modelling

In our data we have the knowledge of the output and also learning is in presence of the independent variables, so this modelling can be classified under Supervised Learning. Also, the variables are labelled, and we have to predict the values of the continuous variable i.e. fare_amount, so the modelling method to be used is of Regression. Different model under Supervised Learning for Regression problem should have to be checked. On the basic of the performance of the model, the best model will be chosen for prediction of the fare_amount of our test data.

1.Decision Tree.

A predictive model based on a branching series of Boolean tests. It Can be used for classification and regression. Decision tree is a rule. Each branch connects nodes with “and” and multiple branches are connected by “or”. A decision tree is drawn upside down with its root at the top.

The train data is first split into train and test set of data in a ratio of 80:20.
In python from sklearn library, train_test_split and DecisionTreeRegressor are imported. Using the fit command, the X_train and y_train data are fitted and using predict X_test the final prediction is made.

In R : fit = rpart(fare_amount ~. , data = train, method = "anova", minsplit=10), here “anova ” is used for regression model.

```
Node number 1: 12436 observations,    complexity param=0.1372755
  mean=11.19334, MSE=80.84198
  left son=2 (9528 obs) right son=3 (2908 obs)
  Primary splits:
      distance          < 3.480889  to the left,  improve=0.137275500, (0 missing)
      year              < 2011.5    to the left,  improve=0.014227840, (0 missing)
      dropoff_latitude  < 40.71716  to the right, improve=0.008007053, (0 missing)
      dropoff_longitude < -73.94488 to the left,  improve=0.007406592, (0 missing)
      pickup_latitude   < 40.77435  to the right, improve=0.003411593, (0 missing)
  Surrogate splits:
      dropoff_latitude  < 40.71278  to the right, agree=0.775, adj=0.039, (0 split)
      pickup_latitude   < 40.70808  to the right, agree=0.770, adj=0.015, (0 split)
      dropoff_longitude < -73.94612 to the left,  agree=0.770, adj=0.014, (0 split)
      pickup_longitude  < -74.01076 to the right, agree=0.769, adj=0.011, (0 split)

This shows the observations of the node 1 using decisionTree. Here the observations are
split into primary split and Surrogate splits
```

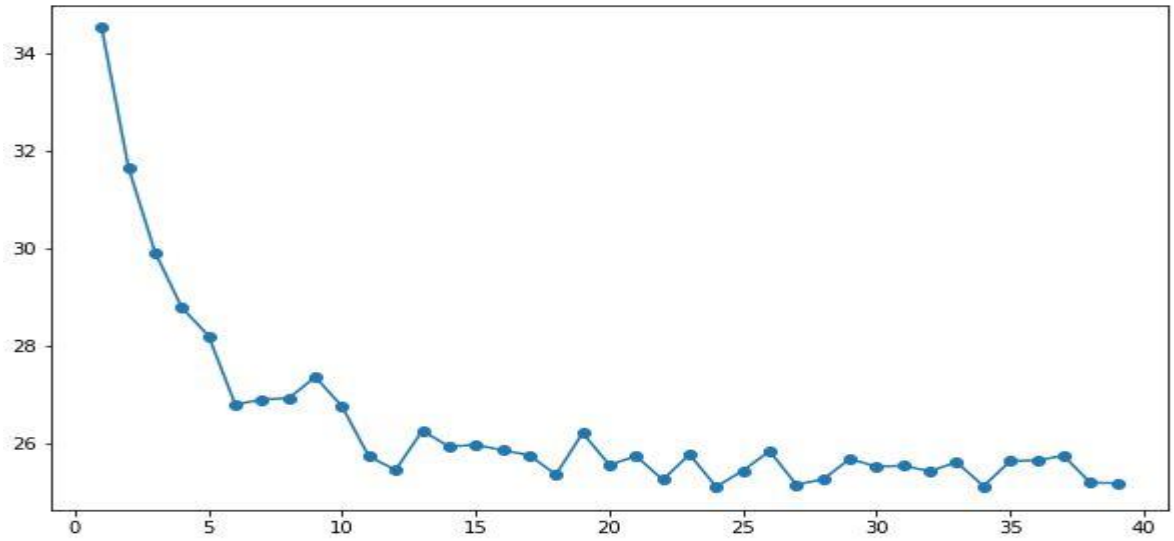
Random Forest:

Random forest is an ensemble that consists of many decision trees. The method combines Breiman's "bagging" idea and the random selection of features. Outputs the class that is the mode of the class's output by individual trees. Mean for regression.
For a better result , a function is developed which will increment the number of estimators one by one and calculate the error rate in the graphical form. From this result we can calculate which estimator will give least error rate.

 # function to calculate error rate.

```
error_rate=[]
for i in range(1,40):
   ran =RandomForestRegressor(n_estimators=i)
   ran.fit(X_train,y_train)
   pred_i=ran.predict(X_test)
   error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)
```

#program to plot the error rate.
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,marker='o')



importance(RF_model, type = 1)

	%IncMSE
pickup_longitude	28.811396
pickup_latitude	28.577160
dropoff_longitude	29.565171
dropoff_latitude	25.346114
passenger_count	1.762296
distance	56.792483
year	15.811514
hour	7.896068

This table shows that of all the variable distance is one of the most important and the least is the passenger count from the given data set.

3.KNN

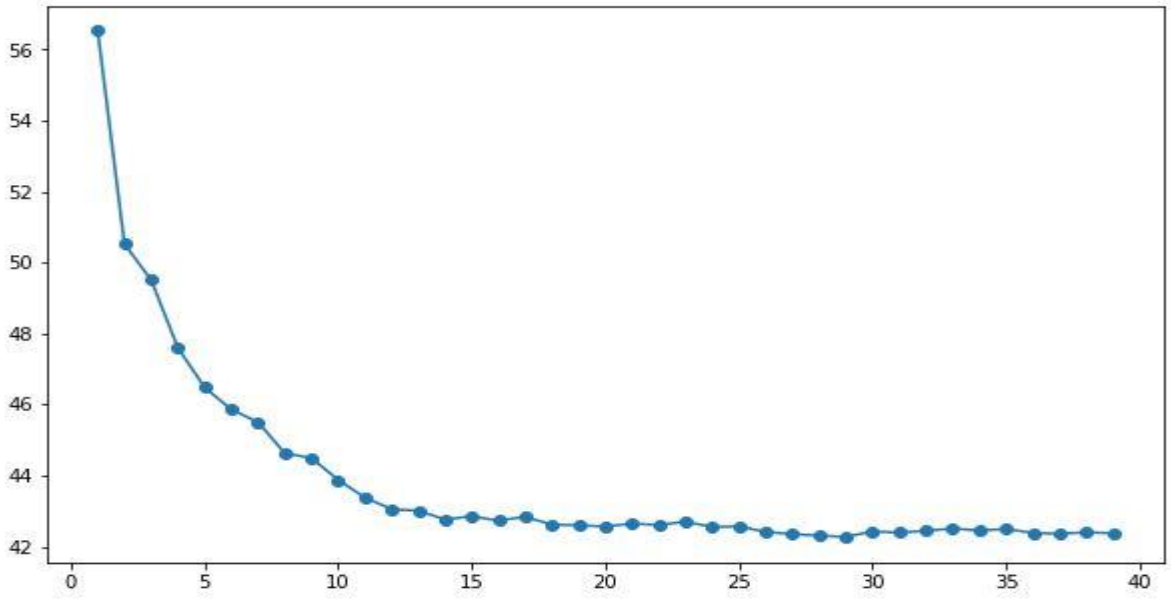
Stands for K-Nearest Neighbour. KNN is simple algorithm that stores all available cases and classifies new cases based on a similarity measure. It is also called as lazy learning algorithm because whenever a new test data come then it will calculate the distance between test case vs all the training cases. It does store any pattern and so calculation time is increased. This algorithm Pick a number of neighbours we want to use for classification or regression. It then Choose a method to measure distances.
In python it is imported through:
from sklearn.neighbors import KNeighborsRegressor. Then we have to choose a number of neighbours for which we want to impute the method.

For a better result , a function is developed which will increment the number of neighbours one by one and calculate the error rate in the graphical form. From this result we can calculate which neighbours will give least error rate.

 # function to calculate error rate.

```
error_rate=[]
for i in range(1,40):
   knn =KNeighborsRegressor(n_neighbors=i)
   knn.fit(X_train,y_train)
   pred_i=knn.predict(X_test)
   error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)
```

#program to plot the error rate.
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,marker='o')



4.Linear Regression:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering, and the number of independent variables being used. Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
Linear regression model for R:

```
> #Linear Regression
> lm_model = lm(fare_amount ~. , data = train)
> summary(lm_model)

Call:
lm(formula = fare_amount ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max
-21.306  -3.379  -2.020  -0.263  51.580

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        1.928e+03  6.936e+02   2.779 0.005458 **
pickup_longitude   1.317e+01  6.317e+00   2.086 0.037027 *
pickup_latitude   -5.861e+00  4.880e+00  -1.201 0.229821
dropoff_longitude  1.818e+01  5.476e+00   3.320 0.000902 ***
dropoff_latitude  -1.367e+01  4.297e+00  -3.180 0.001475 **
passenger_count    5.049e-01  1.390e-01   3.633 0.000281 ***
distance           2.209e+00  4.126e-02  53.544  < 2e-16 ***
year               5.928e-01  3.846e-02  15.416  < 2e-16 ***
hour1             -1.695e-01  5.588e-01  -0.303 0.761724
hour2              3.359e-04  6.162e-01   0.001 0.999565
hour3             -1.401e-01  6.533e-01  -0.214 0.830210
hour4              5.947e-01  7.279e-01   0.817 0.413899
hour5              2.383e+00  8.084e-01   2.948 0.003203 **
hour6              2.330e-01  6.211e-01   0.375 0.707590
hour7              1.610e-01  5.169e-01   0.311 0.755478
hour8             -5.464e-02  5.109e-01  -0.107 0.914824
hour9              1.098e-02  4.924e-01   0.022 0.982206
hour10             4.759e-02  5.158e-01   0.092 0.926481
hour11             7.475e-01  5.011e-01   1.492 0.135807
```

```
hour12             1.051e+00  4.908e-01   2.141 0.032283 *
hour13             1.104e+00  4.935e-01   2.237 0.025324 *
hour14             9.366e-01  4.933e-01   1.899 0.057647 .
hour15             8.171e-01  4.997e-01   1.635 0.102017
hour16             1.767e+00  5.089e-01   3.473 0.000516 ***
hour17             1.321e-01  4.935e-01   0.268 0.788940
hour18             1.652e-01  4.730e-01   0.349 0.726920
hour19            -6.771e-01  4.723e-01  -1.434 0.151671
hour20            -3.981e-01  4.727e-01  -0.842 0.399646
hour21            -4.230e-01  4.770e-01  -0.887 0.375164
hour22            -6.840e-01  4.795e-01  -1.427 0.153744
hour23            -4.080e-01  4.962e-01  -0.822 0.410905
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7.991 on 12405 degrees of freedom
Multiple R-squared:  0.212,    Adjusted R-squared:  0.2101
F-statistic: 111.3 on 30 and 12405 DF,  p-value: < 2.2e-16
```

This table shows the results obtained by Linear Regression Model. The row of residual shows the error. The coefficient shows how much information each variable stores. Estimate, for example pickup_longitude=1.317e+01 signifies1-unit change in the pickup_longitude changes the fare_amount by 1.317e+01.The column Std. Error measures the average amount that the coefficient estimates varying from actual average value of the target variable. The column t-value measures how much our std Error away from 0.R-squared and adjusted R-square of 0.21 shows that our independent variable can only be able to show 21.2% of variance of the target variable.

6.Evaluation

Once we have developed different models, the main task is to detect the performance of the model and also how it can accomplish the need of the problem statement. Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

Error matrix

One of the best tools for the evaluation is the Error matrix. It helps to evaluate our analytical model. It also helps us to make trade-off between multiple matrixes.

Choice of the matrix depends upon :

- Type of model
- Implementation plan of model

Also, the matrix selection depends on the types of problem we are dealing with like: Classification, Regression , Optimization, Unsupervised.

Since, in our problem the target variable is continuous in nature we will go for **Regression Matrix evaluation**.

Regression Evaluation measure of error:

Mean Absolute Error:

The mean absolute error(MAE) has the same unit as the original data, and it can only be compared between models whose errors are measured in the same units. It is similar in magnitude to RMSE, but slightly smaller.

Mean Absolute Percentage Error: It measures accuracy as a percentage of error. It is obtained by multiplying MAE by 100.

Root Mean Square Error: It is the square root of the square of difference of the given value and true value.

Which Error method to choose among these three methods:

If our dataset contains a transition data or time-based which is also called as time series analysis or time series data , then it is better to go for RMSE

If we want to convert our error number into percentage then MAPE is best method.

In our analysis of error, it is better to go for the percentage error method. So MAPE error method is used.

Accuracy: Is obtained as: 100-MAPE

A function to calculate MAPE is obtained as:

```
MAPE = function(y, yhat)
{
mean(abs((y - yhat)/y)*100)
}
```

From the MAPE function created , two input passed to it is y_test and the predicted value. We pass this function to all the model developed and calculate the respected MAPE and Accuracy for each model. The model with least error and highest accuracy among all will be the chosen model for our final test.

For Python

Model	MAPE	Accuracy
Decision Tree	32.520410679150345	67.46959
Random Forest	25.834387313308692	74.16562
KNN	42.75859923591262	57.24141
Linear Regression	42.6133964475484	57.38661

For R

Model	MAPE	Accuracy
Decision Tree	41.88119	58.11881
Random Forest	18.18387	81.81613
KNN	33.02988	66.97012
Linear Regression	38.87821	61.12179

7.MODEL SELECTION

From the above table, we see error and accuracy calculation of different model. Based on this, the model with least error rate or highest accuracy will be selected.
In python, Random Forest model gives the least **Error Rate i.e. 25.834387313308692** and highest Accuracy. So, this model is selected for our final analysis.
Similarly , **In R also Random Forest** model gives the least **Error Rate i.e. 18.18387** and highest Accuracy.

After Selection of the model , we use our test data to analysis final fare_amount.
Test Data: 9914 rows and 6 columns. With the use of feature selection three more column are made: Distance, Year and Hour.
And finally using the Random Tree model final fare_amount is predicted and added as a column to the test data.

Test Data after fare_amount prediction.

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	distance	year	hour	fare_amount
0	2015-01-27 13:08:24	-73.973320	40.763805	-73.981430	40.743835	1	2.323989	2015	13	9.550000
1	2015-01-27 13:08:24	-73.986862	40.719383	-73.998886	40.739201	1	2.426114	2015	13	10.591667
2	2011-10-08 11:53:44	-73.982524	40.751260	-73.979654	40.746139	1	0.618822	2011	11	4.191667
3	2012-12-01 21:12:12	-73.981160	40.767807	-73.990448	40.751635	1	1.961648	2012	21	10.125000
4	2012-12-01 21:12:12	-73.966046	40.789775	-73.988565	40.744427	1	5.388992	2012	21	14.808333

Python Code…

```
# In[1]:
import os
import pandas as pd
import numpy as np
from fancyimpute import KNN
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:
train = pd.read_csv('D:\\Data Science\\Project1\\train_cab.csv')
# In[3]:
train.head()
# In[4]:
train.info()
# In[5]:
train.describe()
# In[6]:
train.columns
# In[7]:
train.dtypes
# In[8]:
train.shape
# In[9]:
train.isna().sum()
# In[10]:
train['fare_amount']= train['fare_amount'].apply(pd.to_numeric, errors='coerce')
# In[11]:
train[train['fare_amount']<0]
# In[12]:
sns.pairplot(train)
# In[13]:
train[train['fare_amount'].isnull()]
# In[14]:
train[train['passenger_count'].isnull()]
# In[15]:
train[train['passenger_count']>6]
# In[16]:
train[train['pickup_longitude']==train['dropoff_longitude']]
# In[17]:
train[train['pickup_longitude']==train['dropoff_longitude']].count()
# In[18]:
train.drop(train[train['pickup_longitude']==train['dropoff_longitude']].index,inplace=True)
# In[19]:
#replace 0 with NA in the variables
train['fare_amount']= train['fare_amount'].apply(pd.to_numeric, errors='coerce')
train['fare_amount']= train['fare_amount'].replace({0:np.nan})
train['passenger_count']=train['passenger_count'].fillna(0)
train['passenger_count']= train['passenger_count'].astype(int)
train['passenger_count']=train['passenger_count'].replace({0: np.nan})
train['pickup_longitude']= train['pickup_longitude'].replace({0:np.nan})
train['pickup_latitude']= train['pickup_latitude'].replace({0:np.nan})
train['dropoff_longitude']= train['dropoff_longitude'].replace({0:np.nan})
train['dropoff_latitude']= train['dropoff_latitude'].replace({0:np.nan})
```

```
# In[20]:
train.isna().sum()
# In[21]:
train.head()
# In[22]:
#creaet a function to calculate missing values
def missingval(data):
   missing_val = pd.DataFrame(data.isnull().sum())

#Reset index
   missing_val = missing_val.reset_index()

#Rename variable
   missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'count'})

#Calculate percentage
   missing_val['Missing_percentage'] = (missing_val['count']/len(data)*100)

#descending order
   missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
   print(missing_val)
# In[23]:
missingval(train)
# In[24]:
df = train.copy()
#train = df.copy()
# In[ ]:
#actual--train['pickup_longitude'].loc[50]==-73.985582
#KNN----73.97770799999999
#median-   -73.9820605
#mean   -73.91160183651274
# In[ ]:
#fare_amount =9.7
#Mean-15.127673940949794
#median-8.5
#knn- 7.5
# In[25]:
train[train['passenger_count']<=8]['passenger_count'].hist(bins=100, figsize=(14,3))
# In[26]:
#Since Mean is the best method, we impute missing values with mean

train['pickup_longitude'] = train['pickup_longitude'].fillna(train['pickup_longitude'].median())
train['pickup_latitude'] = train['pickup_latitude'].fillna(train['pickup_latitude'].median())
train['dropoff_longitude'] = train['dropoff_longitude'].fillna(train['dropoff_longitude'].median())
train['dropoff_latitude'] = train['dropoff_latitude'].fillna(train['dropoff_latitude'].median())

#for category variables we impute with mode
train['passenger_count'] = train['passenger_count'].fillna(int(train['passenger_count'].mode()))
# In[27]:
train.isna().sum()
# In[28]:
train.shape
# In[29]:
#Imputing the NAs in target variables will bias the model, hence remove them
train=train.dropna()
# In[30]:
```

```
train.shape
# In[31]:
#conert into proper data type
pro_data={'fare_amount' : 'float','passenger_count': 'int'}
train=train.astype(pro_data)
#train['passenger_count'] = train['passenger_count'].astype('category')
# In[32]:
plt.figure(figsize=(14,5))
sns.countplot(x='passenger_count', data=train)
# In[33]:
train['passenger_count'].value_counts()
# In[34]:
train[train['fare_amount']<60]['fare_amount'].hist(bins=30, figsize=(14,3))
plt.xlabel('fare')
plt.title('Histogram')
# In[35]:
train[train['fare_amount']>60].count()
# In[36]:
train[train['fare_amount']<0]
# # Outlier Analysis
# In[37]:
df1 =train.copy()
#train = df1.copy()
# In[38]:
# #Plot boxplot to visualize Outliers
  plt.boxplot(train['passenger_count'])
# In[39]:
plt.boxplot(train['pickup_latitude'])
# In[40]:
plt.boxplot(train['fare_amount'])
# In[41]:
cnames = ['pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude']
# In[42]:
#Detect and delete outliers from data
for i in cnames:
    print(i)
    q75, q25 = np.percentile(train.loc[:,i], [75 ,25])
    iqr = q75 - q25

    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)
    print(min)
    print(max)
    train.loc[train[i] < min,i] = np.nan
    train.loc[train[i] > max,i] = np.nan

# In[43]:
train.isna().sum()
# In[44]:
# fare_amount at the higher end are converted to na
train.loc[train['fare_amount']<0 , 'fare_amount']=np.nan
train.loc[train['fare_amount'] > 60, 'fare_amount']=np.nan
# In[45]:
train.isna().sum()
# In[46]:
#there are few passenger counts that are greater than 8. convert them intoNAN
train.loc[train['passenger_count'] > 6,'passenger_count'] = np.nan
```

```
# In[47]:
train.isna().sum()
# In[48]:
#Since Median is the best method, we impute missing values with Median

train['pickup_longitude'] = train['pickup_longitude'].fillna(train['pickup_longitude'].median())
train['pickup_latitude'] = train['pickup_latitude'].fillna(train['pickup_latitude'].median())
train['dropoff_longitude'] = train['dropoff_longitude'].fillna(train['dropoff_longitude'].median())
train['dropoff_latitude'] = train['dropoff_latitude'].fillna(train['dropoff_latitude'].median())

#for category variables we impute with mode
train['passenger_count'] = train['passenger_count'].fillna(int(train['passenger_count'].mode()))
# In[49]:
train.isna().sum()
# In[50]:
train=train.dropna()
# In[51]:
train.isna().sum()
# In[52]:
df2= train.copy()
#train = df2.copy()
# In[53]:
train.shape
# In[54]:
train.iloc[1264]= np.nan
# In[55]:
train.dropna(inplace=True)
# In[56]:
def distance(s_lat, s_lng, e_lat, e_lng):

    # approximate radius of earth in km
    R = 6373.0

    s_lat = s_lat*np.pi/180.0
    s_lng = np.deg2rad(s_lng)
    e_lat = np.deg2rad(e_lat)
    e_lng = np.deg2rad(e_lng)

    d = np.sin((e_lat - s_lat)/2)**2 + np.cos(s_lat)*np.cos(e_lat) * np.sin((e_lng - s_lng)/2)**2

    return 2 * R * np.arcsin(np.sqrt(d))
# In[57]:
train['distance'] =
distance(train['pickup_latitude'],train['pickup_longitude'],train['dropoff_latitude'],train['dropoff_longitude'])
# In[58]:
train.head()
# In[59]:
train.shape
# In[60]:
train['pickup_datetime']=pd.to_datetime(train['pickup_datetime'], format= "%Y-%m-%d %H:%M:%S UTC")
# In[61]:
train['year'] = train.pickup_datetime.apply(lambda t: t.year)
train['hour'] = train.pickup_datetime.apply(lambda t: t.hour)
# In[62]:
train.head()
# In[63]:
sns.barplot(x='year', y='fare_amount', data=train)
```

```
# In[64]:
#plt.figure(figsize=(14,4))
#sns.barplot(x='hour', y='fare_amount', data=train)
# In[65]:
#plt.figure(figsize=(14,4))
#sns.scatterplot(x='fare_amount',y ='distance', data=train,hue='passenger_count')
# In[66]:
#sns.pairplot(train)
# In[67]:
cnames1=['fare_amount', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude',
     'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'distance',
     'year', 'hour']
# In[68]:
df_corr = train.loc[:,cnames1]
# In[69]:
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(16, 6))
#Generate correlation matrix
corr = df_corr.corr()
#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap='viridis',linewidths=1,linecolor='black',annot=True)
# In[70]:
sns.distplot(train['fare_amount'])
# # Model Development
# In[71]:
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
# In[72]:
train.columns
# In[73]:
X=train[['pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude',
'passenger_count','distance','year','hour']]
# In[74]:
y=train['fare_amount']
# In[75]:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
# In[76]:
dtree = DecisionTreeRegressor()
# In[77]:
dtree.fit(X_train,y_train)
# In[78]:
predictions = dtree.predict(X_test)
# In[79]:
#Calculate MAPE
def MAPE(y_true, y_pred):
   mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
   return mape
# In[80]:
MAPE(y_test, predictions)
# In[81]:
#Random Forest
from sklearn.ensemble import RandomForestRegressor
# In[84]:
RF_model = RandomForestRegressor(n_estimators = 23).fit(X_train,y_train)
# In[85]:
RF_Predictions = RF_model.predict(X_test)
```

```
# In[86]:
MAPE(y_test, RF_Predictions)
# In[82]:Function to calculate error rate by incrementing n_estimators
error_rate=[]
for i in range(1,40):
    ran =RandomForestRegressor(n_estimators=i)
    ran.fit(X_train,y_train)
    pred_i=ran.predict(X_test)
    error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)
# In[126]:
plt.figure(figsize=(8,4))
plt.plot(range(1,40),error_rate,marker='o')
# In[87]:
from sklearn.neighbors import KNeighborsRegressor
# In[90]:
KNN_model = KNeighborsRegressor(n_neighbors = 14)
# In[91]:
KNN_model.fit(X_train, y_train)
# In[92]:
KNN_Predictions = KNN_model.predict(X_test)
# In[93]:
MAPE(y_test, KNN_Predictions)
# In[88]:
error_rate=[]
for i in range(1,40):
    knn =KNeighborsRegressor(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i=knn.predict(X_test)
    error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)
# In[127]:
plt.figure(figsize=(8,4))
plt.plot(range(1,40),error_rate,marker='o')
# In[94]:
from sklearn.linear_model import LinearRegression
# In[95]:
lm = LinearRegression()
# In[96]:
lm.fit(X_train, y_train)
# In[97]:
pred =lm.predict(X_test)
# In[98]:
MAPE(y_test, pred)
# In[ ]:
# # test data
# In[111]:
final = pd.read_csv('D:\\Data Science\\Project1\\test.csv')
# In[112]:
final.head()
# In[113]:
final.describe()
# In[114]:
final.info()
# In[115]:
final.isna().sum()
# In[116]:
final[final['pickup_longitude']==final['dropoff_longitude']].count()
# In[117]:
```

```
final.drop(final[final['pickup_longitude']==final['dropoff_longitude']].index,inplace=True)
# In[118]:
final['distance'] =
distance(final['pickup_latitude'],final['pickup_longitude'],final['dropoff_latitude'],final['dropoff_longitude'])
# In[119]:
final['pickup_datetime']=pd.to_datetime(final['pickup_datetime'], format= "%Y-%m-%d %H:%M:%S UTC")
# In[120]:
final['year'] = final.pickup_datetime.apply(lambda t: t.year)
final['hour'] = final.pickup_datetime.apply(lambda t: t.hour)
# In[121]:
RF_model = RandomForestRegressor(n_estimators = 24).fit(X_train,y_train)
# In[122]:
final.columns
# In[123]:
Xf_test =final[['pickup_longitude', 'pickup_latitude','dropoff_longitude', 'dropoff_latitude','passenger_count',
'distance',
      'year', 'hour']]
# In[124]:
final['fare_amount']= RF_model.predict(Xf_test)
# In[125]:
final.head()
```

R Code

```
rm(list = ls())

setwd("D:/Data Science/Project1")
getwd()


#Load Libraries
x = c("ggplot2","dplyr","lubridate","tidyverse", "geosphere", "corrgram", "DMwR", "caret", "randomForest",
"unbalanced", "C50", "dummies", "e1071", "Information",
    "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')
lapply(x, require, character.only = TRUE)
rm(x)

# load the data
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
summary(train)


# converting into proper datatype
train$fare_amount=as.numeric(as.character(train$fare_amount))
train$passenger_count=as.integer(train$passenger_count)
summary(train)

# filtering out the rows where pickup_longitude = dropoff_longitude and pick_up latitude = drop_off latitude
train = filter(train,
          train$pickup_longitude!= train$dropoff_longitude &
          train$pickup_latitude!=train$dropoff_latitude
)

#replace all "0" with NA
train[train==0]= NA
summary(train)


# fare_amount is target variable, so remove NA by filtering out all fare_amount>0, this will also filter out NA
if available
train = filter(train,
          fare_amount>0&
          fare_amount<60
)


#missing Value

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)]

ggplot(data = missing_val[1:3,], aes(x=reorder(Columns, -Missing_percentage),y = Missing_percentage))+
geom_bar(stat = "identity",fill = "grey")+xlab("Parameter")+
ggtitle("Missing data percentage (Train)")



#passenger_count is categorical, so replace it by mode
#function to calculate mode
mode= function(x){
  y=unique(x)
  as.numeric(as.character(y[which.max(tabulate(match(x,y)))]))

}

#impute with the mode
train$passenger_count[is.na(train$passenger_count)] = mode(train$passenger_count)
train$passenger_count = as.integer(train$passenger_count)

#Method
#Actual train$pickup_latitude[48]= 40.77205
#Mean =train$pickup_latitude[48]= 40.71222
#Median =train$pickup_latitude[48]= 40.75332
#KNN = train$pickup_latitude[48]= 40.77271


#KNN imputation
train = knnImputation(train, k = 5)

# Outlier Analysis
train1 = train
summary(train)


# ## BoxPlots - Distribution and Outlier Check


cnames = colnames(train[,c(3:7)])
#
for (i in 1:length(cnames))
 {
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "fare_amount", group = 1), data = train)+
         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="Num")+
         ggtitle(paste("Box plot of fare_amount for",cnames[i])))
 }
```

```
#
# ## Plotting plots together
gridExtra::grid.arrange(gn1,gn2,ncol=2)
gridExtra::grid.arrange(gn3,gn4,gn5, ncol=3 )

# # #Remove outliers using boxplot method
train_data = train
#train = train_data

# replacing outlier with NA
for(i in cnames){
  val = train[,i][train[,i] %in% boxplot.stats(train[,i])$out]
  #print(length(val))
  train[,i][train[,i] %in% val] = NA
  }



#KNN imputation
train = knnImputation(train, k = 5)


####################
train2 = train
#train = train2

# library(geosphere)
#creating a distance column using distHaversine function
train$distance = distHaversine(train[,c(3,4)],train[,c(5,6)])
train$distance = as.numeric(train$distance)/1000

#which(is.na(train$pickup_datetime))
# removing a row which has pickup_datatime value not in proper format, pickup_datetime=43
train = train[-c(1265),]

# creating aditional column from datetime to year, month,day,dayOfWeek,hour,partOfDay
train <- mutate(train,
                pickup_datetime = ymd_hms(pickup_datetime),
                month = as.factor(month(pickup_datetime)),
                year = as.numeric(year(pickup_datetime)),
                day = day(pickup_datetime),
                dayOfWeek = as.factor(wday(pickup_datetime)),
                hour = hour(pickup_datetime),
                partOfDay = as.factor(round(hour * 2 / 10)),
                hour = as.factor(hour(pickup_datetime))

)
#converting into proper datatype
train$passenger_count = as.integer(train$passenger_count)


## Correlation Plot
```

```
  corrgram(train[,-c(7)], order = F,
                   upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot"
)

## drop unwanted columns
train <- train[,-c(2,9,11,12,14)]


qqnorm(train$fare_amount)

hist(train$fare_amount)

train3 = train
#Clean the environment
rmExcept("train")

train3 = train


#Divide data into train and test using stratified sampling method
set.seed(1234)

train.index = createDataPartition(train$fare_amount, p = .80, list = FALSE)
train = train[ train.index,]
test  = train[-train.index,]


# Define Mape
MAPE = function(x, y){
  mean(abs((x - y)/x*100))
}

#Decision Tree
fit = rpart(fare_amount ~. , data = train, method = "anova", minsplit=10)
summary(fit)
predictions_DT = predict(fit, test[,-1])
MAPE(test[,1], predictions_DT)
#41.88119

#Linear Regression
lm_model = lm(fare_amount ~. , data = train)
summary(lm_model)
predictions_LR = predict(lm_model, test[,-1])
MAPE(test[,1], predictions_LR)
#38.87821




##KNN Implementation
library(class)
#Predict test data
```

```
KNN_Predictions = knn(train[, 2:9], test[, 2:9], train$fare_amount, k = 5)
#convert the values into numeric
KNN_Predictions=as.numeric(as.character((KNN_Predictions)))
#Calculate MAPE
MAPE(test[,1], KNN_Predictions)
#33.02988

#Random Forest
RF_model = randomForest(fare_amount ~.  , train, importance = TRUE, ntree=200, mtry=2)
RF_Predictions = predict(RF_model, test[,-1])
MAPE(test[,1], RF_Predictions)
importance(RF_model, type = 1)
#18.18387

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

# Loading the test data
test_cab = read.csv("test.csv", header = T)

# filtering out the rows where pickup_longitude = dropoff_longitude and pick_up latitude = drop_off latitude
test_cab = filter(test_cab,
          test_cab$pickup_longitude!= test_cab$dropoff_longitude &
          test_cab$pickup_latitude!=test_cab$dropoff_latitude
)


#creating a distance column using distHaversine function
test_cab$distance = distHaversine(test_cab[,c(2,3)],test_cab[,c(4,5)])
test_cab$distance = as.numeric(test_cab$distance)/1000


# creating aditional column from datetime to year,hour

test_cab <- mutate(test_cab,
           pickup_datetime = ymd_hms(pickup_datetime),

           year = as.numeric(year(pickup_datetime)),

           hour = hour(pickup_datetime),

           hour = as.factor(hour(pickup_datetime))

)


# droping out unwanted columns
test_cab = test_cab[,-c(1)]
#Random Forest
RF_model = randomForest(fare_amount ~.  , train, importance = TRUE, ntree=200, mtry=2)
test_cab$fare_amount = predict(RF_model, test_cab)
```

References:

https://edwisor.com/
https://stackoverflow.com
https://www.geeksforgeeks.org/k-nearest-neighbours/
https://www.udemy.com/

Book:
ISLR: An Introduction to Statistical Learning with Applications in R by Gareth James • Daniela Witten • Trevor Hastie Robert Tibshirani