Date: 05/06/2019

Cab Fare Prediction Project Report

Submitted by Purushottam Potnuru

Contents

Chapter 1

1. Introduction

1.1 Problem Statement

We are a cab rental start-up company. We have successfully run the pilot project and now want to launch our cab service across the country.

We have collected the historical data from our pilot project and now have a requirement to apply analytics for fare prediction.

Our objective of this Cab Fare Prediction Project is to **design a system that predicts the fare amount for a cab ride taken in the city.**

1.2 Dataset

Image 1

	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

Image 2

	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

In image 1, we can see our historical data. It has 16066 observations and 7 features, including fare_amount which is our target variable.

In image 2, we can see data, whose target variable i.e. fare_amount values to be predicted. It has 9914 observations and 6 features, where, fare_amount is not available, as we predict those values later in the journey.

As said above, our primary focus is to determine the values for our target variable fare_amount for future test cases using the above shown dataset in image 1.

1.3 Exploratory Data Analysis

In Exploratory Data Analysis, we go through different things, like:

- **Brainstorming** – Here, all the team members (if you have a group working together) or only you alone, will actually draw a rough sketch which talks about – the steps you are going to follow to achieve your given objective.

- **Defining Problem Statement** – Although, in this project we are given with problem statement to work with, however, when dealing with real time scenarios, we are sometimes only given with limited information, and the client may ask us in reverse – just find some insights from the given data, to grow my business.

  In that case, we are responsible to define the problem statement by ourselves. We have to understand the complete in and out operations of the client's business and take decisions accordingly.

- Knowing the types of variables in the given dataset, whether they are factor, character or numeric. In our project we changed the pickup_datetime variable from object to datetime, the same way, we converted the datatype of fare_amount from object to numeric.

- Changing and removing datatypes if required. In our project, we removed four variables i.e. pickup_latitude, pickup_longitude, dropoff_latitude and dropoff_longitude after we obtained distance from them.

- We also split the pickup_datetime variable into year, month, date, day, hour, and minutes as we wanted not to deal with time series analysis.

Chapter 2

2. Methodology

Now, we have the dataset and also, we discussed about Exploratory Data Analysis, let's talk about the **methodology** we are going to follow to achieve our goal.

We will be going through:

- Pre-processing which includes missing value analysis, outlier analysis, feature selection and feature scaling.
- Model development, where we will choose what machine learning algorithms to apply.

2.1 Pre-processing

In pre-processing, we actually apply few techniques like missing value analysis, outlier analysis, feature selection, feature scaling.

Why we do that? Well, actually we never get a structured data to work with. Always messy data is handed to us, and we need to clean that data.

The data may have many observations (rows in dataset), where values in few fields will be absent. We can also say, there may be some inconsistent values in a variable (column in dataset), when compared with other values.

When we go for model development, we should have a structured dataset. We can't go forward for model development, if we don't apply pre-processing techniques on data and convert it into structured format.

2.1.1 Missing Value Analysis

Missing value analysis, as the name suggests, we face with situations, where we are given with dataset, and we have missing values in the observations.

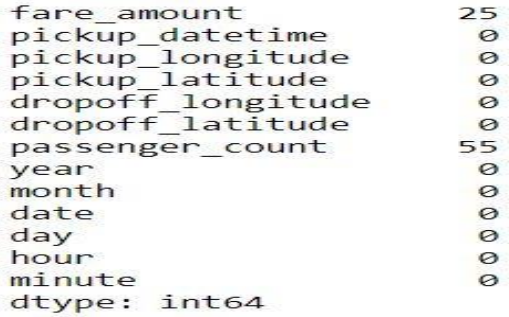
The reason why we have missing values may be plenty, like human error, the user didn't want to share his complete information, the one who was supposed to fill the data may didn't work properly.

But, as discussed earlier, we have to give structured data to the model. In order to achieve that, we perform missing value analysis on top of the data to clean the data, to transform the data from unstructured format to structured format.

We give a line of code and it gives us the total number of missing values in each column. Later, we impute those missing values using mean, median or KNN method for numerical variables or using mode for categorical variables. In some cases, we may delete the observations with missing values, only when we have a case where we got few observations with missing values.

After imputing the missing values, we proceed further with outlier analysis. In our project, we had missing values in fare_amount i.e. 25 missing values and passenger_count i.e. 55 missing values. We dropped these observations from the dataset, as mentioned above, we may go for deletion of observations with missing values when we have a low number of it.

Image 3



Another important thing is, we apply missing value techniques only on numerical variables.

2.1.2 Outlier Analysis

Outliers may be defined as the inconsistent values in a variable. For example, a = 1,2,3,4, 20. In object a, 20 is inconsistent, in terms of mean. Another important thing is, we apply outlier analysis only on numerical variables.

Outliers are used for fraud detection. Let's say, in one bank account, consistently, we observe an amount which ranges from 50,000 to 1,00,000, but in one case 10 lakhs gets deposited. In that case, simply using outlier technique, we can get the inconsistent values.

In our project, we found outliers in fare_amount, pickup_datetime i.e. 43 (refer to image attached below), pickup_latitude, and passenger_count variables. In other variables the values are consistent with each other.

We deleted the outliers in these variables as the number of outliers were minimum. We went for ascending order and knew the inliers and went for descending order and learnt about outliers and finally sometimes we used to describe function to know minimum and maximum.

For example, in pickup_latitude, we came to know about an outlier present in that simply using describe function.

Image 4

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16041.000000	16066.000000	16066.000000	16066.000000	16066.000000	16011.000000
mean	15.015223	-72.462693	39.914675	-72.462233	39.897852	2.625171
std	430.474362	10.578707	6.826797	10.575384	6.187276	60.846021
min	-3.000000	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	6.000000	-73.992156	40.734935	-73.991182	40.734647	1.000000
50%	8.500000	-73.981697	40.752605	-73.980170	40.753566	1.000000
75%	12.500000	-73.966837	40.767381	-73.963642	40.768015	2.000000
max	54343.000000	40.766125	401.083332	40.802437	41.366138	5345.000000

You can check about that in third column, at the end of pickup_latitude variable. We understood that 401.083332 is an outlier present after learning that latitudes value lies in between from -90 to +90.

2.1.3 Data Understanding

Understanding visually is easier sometimes and for that purpose we have few libraries in python which allows us to plot awesome visualizations.

We get data, and in order to talk about how few attributes relationship with each other, we can use these visualizations.

You can give two variables, to know how they are related with each other. You can also give three variables to understand the relationship between three variables. You got infinite possibilities to plot. Let's have a view on few of the plots we used in our project.

Image 4

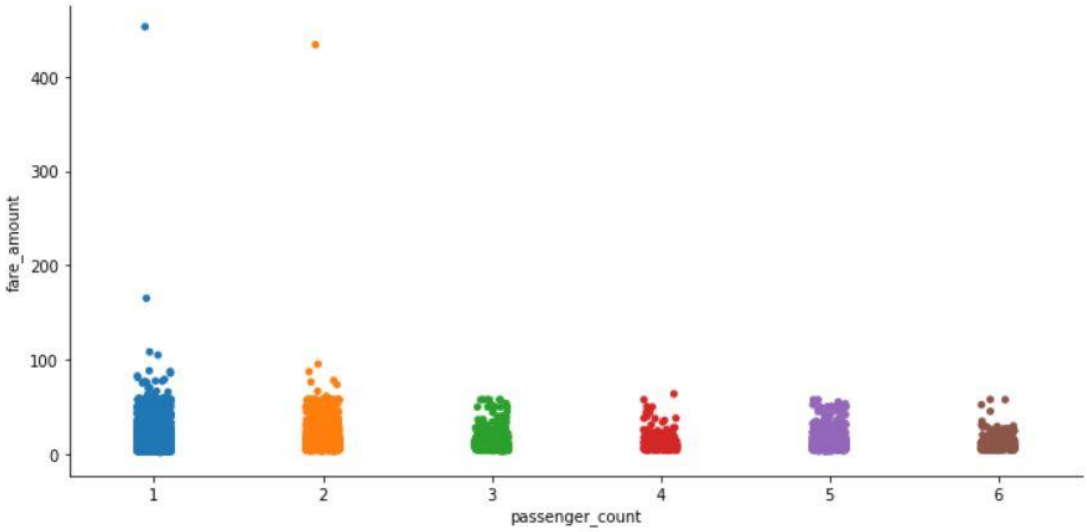


Image 5

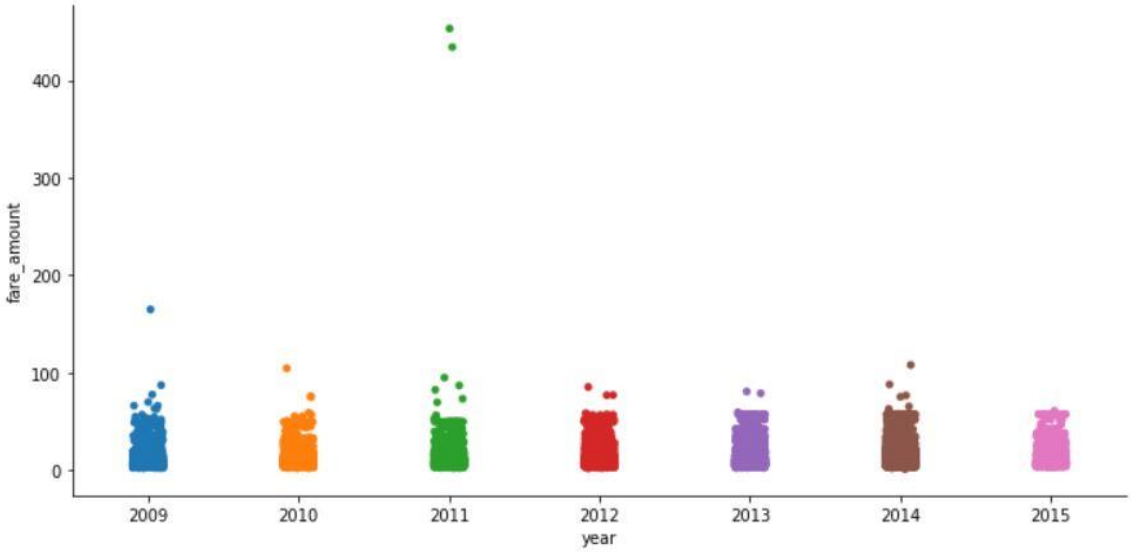


Image 6

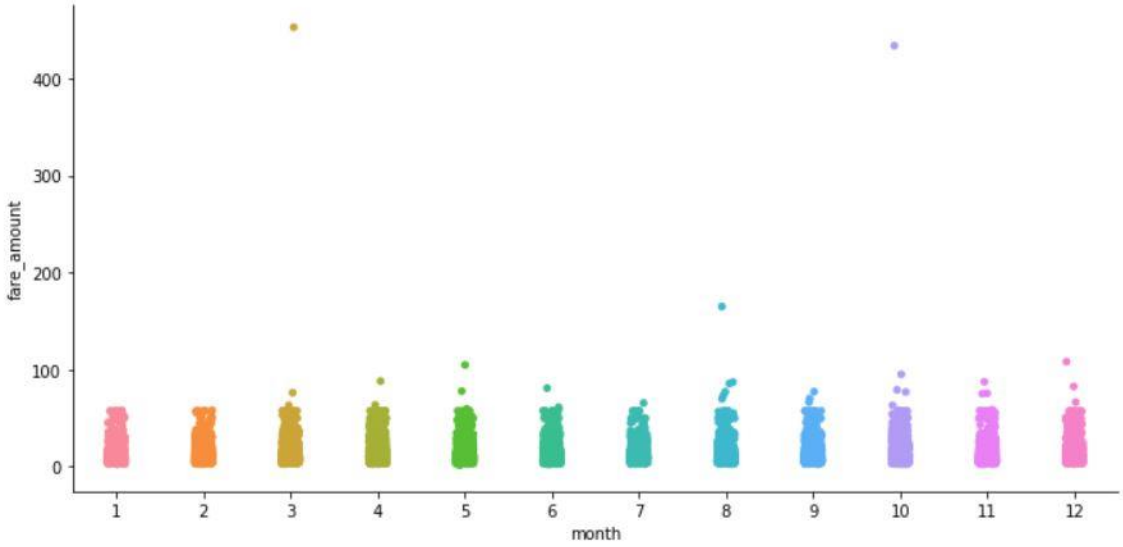


Image 7

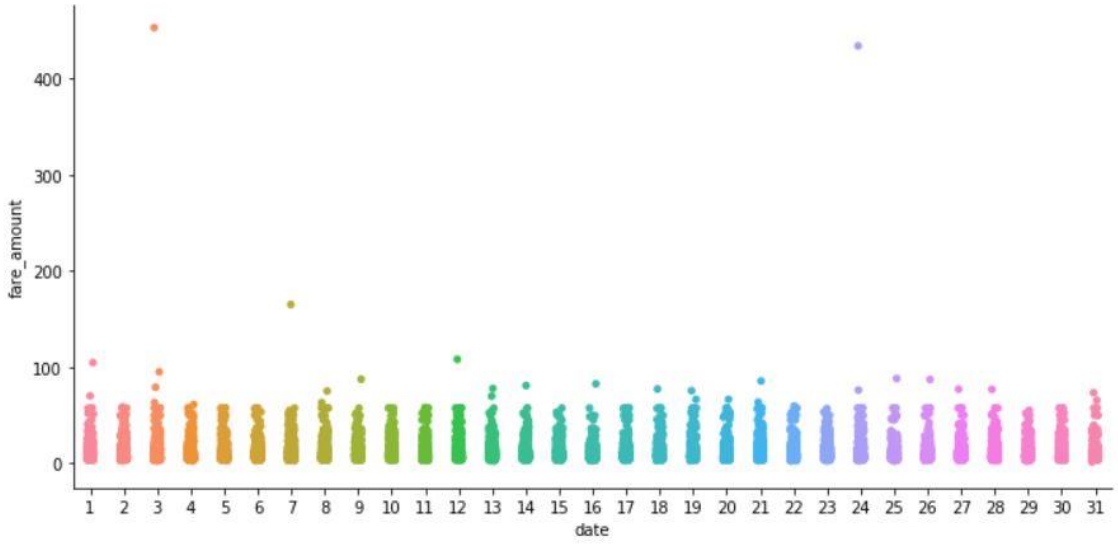


Image 8

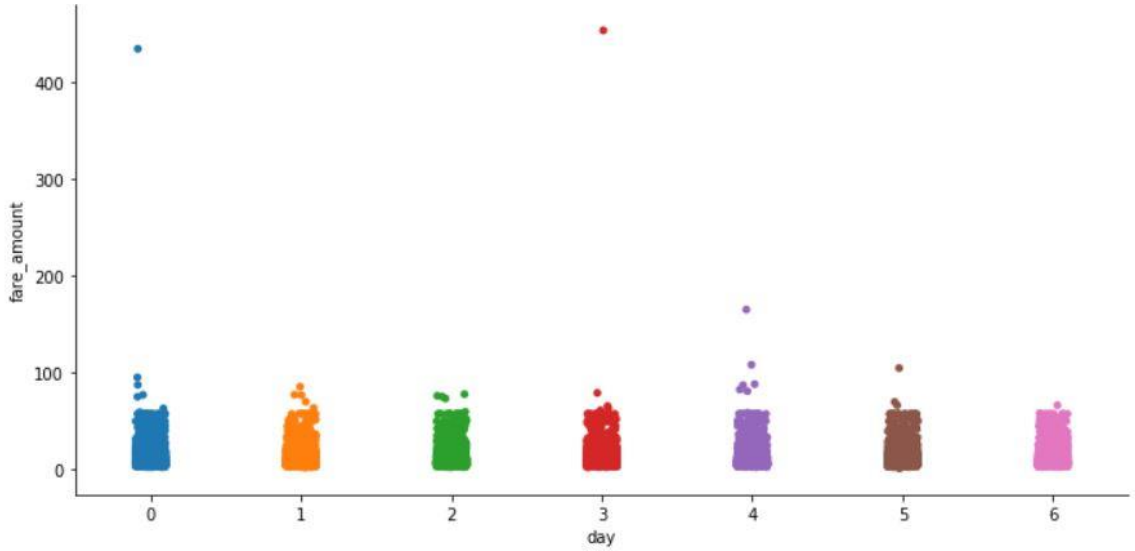


Image 9

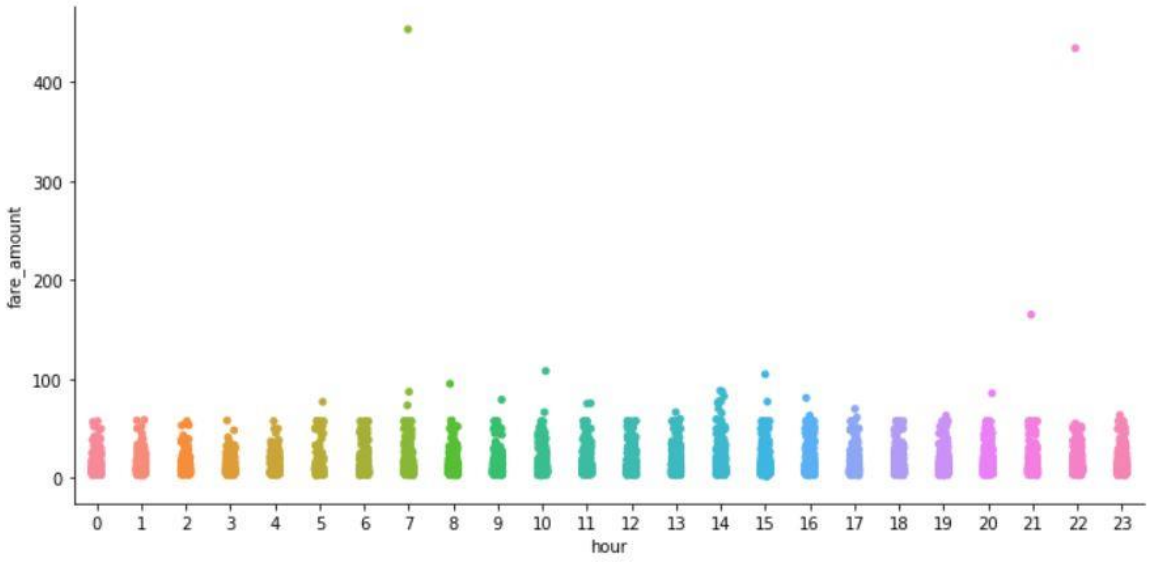


Image 10

Lets talk about the relationship of fare_amount and each variable ¶

> Single passenger pays high fare and 6 passengers at once contributes less fare

> In 2011, the fare is at peak and in 2015 fare is at lowest

> March has high fare, july got low fare

> On monday, thursday and friday fare is high, remaining like, sunday has low fare

> Morning 7 am and night 10 pm are the expensive hours, 3 am and 12 pm are not expensive

2.1.4 Feature Selection

In Feature Selection, we use few techniques to know which variables are important to us, it's all about selecting subset of variables from all variables. Actually, when we are given with a dataset, we perform exploratory data analysis, later data pre-processing to clean the data and transform it from unstructured to structured data to feed into model.

However, after this, we may face a situation where we may have variables which have same information with them about the target variable. Let's talk about an example, in a situation where we are sending five people on a mission. Later, you came to know that, two individuals have the same exact information with them about the mission. Definitely you would drop one, in order to reduce infrastructure and complexity.

The same way, we also drop few variables if they have same information. We always aim that, there should be no independent variable which talks the same as other independent variables but, we appreciate those variables which talks more about the target variable.

In our project, we did correlation analysis between target variable (continuous variable) and other continuous variable i.e. distance. Let's check it:

Image 11

Now, we can see, distance got pretty good information about fare_amount. We are not going to drop it.

Next, we did Anova test between categorical variables and target variable (continuous variable)

Let's check that:

Image 12

	sum_sq	df	F	PR(>F)
passenger_count	7.412790e+01	1.0	0.650559	0.419925
Residual	1.757260e+06	15422.0	NaN	NaN
	sum_sq	df	F	PR(>F)
year	1.765442e+04	1.0	156.503813	9.795817e-36
Residual	1.739680e+06	15422.0	NaN	NaN
	sum_sq	df	F	PR(>F)
month	2.393332e+03	1.0	21.032028	0.000005
Residual	1.754941e+06	15422.0	NaN	NaN
	sum_sq	df	F	PR(>F)
date	1.051343e+02	1.0	0.922692	0.336784
Residual	1.757229e+06	15422.0	NaN	NaN
	sum_sq	df	F	PR(>F)
day	6.935365e+00	1.0	0.060864	0.805139
Residual	1.757327e+06	15422.0	NaN	NaN
	sum_sq	df	F	PR(>F)
hour	1.203834e+03	1.0	10.571838	0.001151
Residual	1.756130e+06	15422.0	NaN	NaN

It's clear, that passenger_count, date, and day has p value > 0.05, by which we accept null hypothesis (all means are equal, they carry same information). So, we are going to drop them even.

That's all about Feature Selection.

2.1.5 Feature Scaling

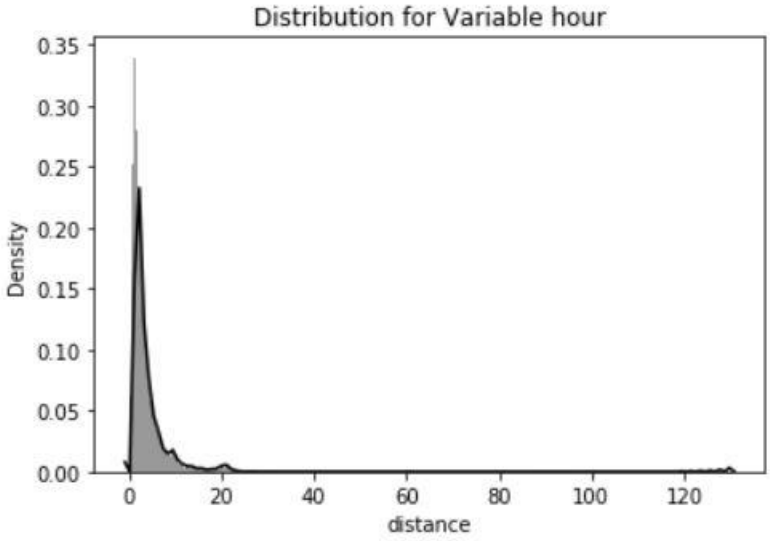
In Feature Scaling, we try to limit the ranges of variables, so that they can be compared on the same ground.

Let's talk about one example, consider, two variables, age and income. Age is varied from 1 to 100 but coming to income, it ranges in a large scale.

At this situation, the higher values bias the result towards themselves, in order to overcome a situation like this, we use Feature Scaling, where we limit the ranges of variables.

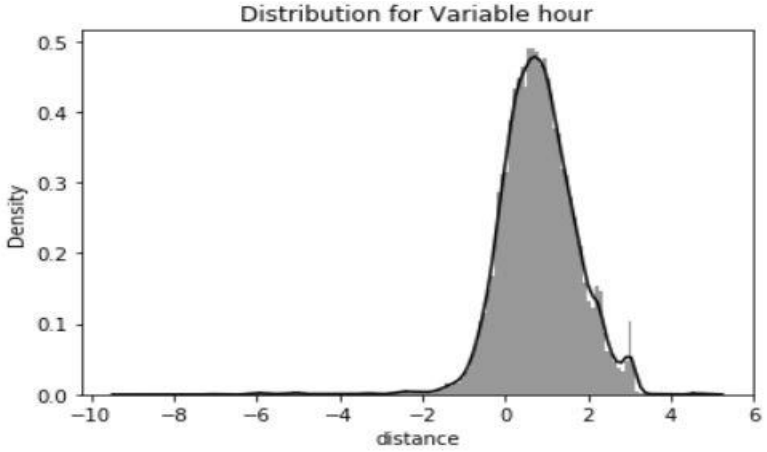
In our project, we did go for feature scaling for distance variable as the given data was left skewed. We also went for distribution check, so let's have look on that.

Image 13



Now, as observed, the data points of distance variable are left skewed. Now, we are going to apply log to the data points and let's check again, its distribution using histogram.

Image 14



As we can see now, the distance variable got a pretty desired distribution. Finally, we are done with Feature Scaling. Now, let's step into Model Development.

2.2 Model Development

Model Development, is the phase which comes after we are done with applying the exploratory data analysis, data pre-processing techniques, on the top of data.

The data, will be in structured format, which was our goal, is now ready to develop model.

After we defined our objective and received the data, we transformed it into our required form, we enter into model development, but before that, let's discuss about model selection.

2.2.1 Model Selection

Model Selection particularly depends on the objective, the problem statement. We have to know at first hand, that, under which category, the problem statement falls.

We have four categories:

- Forecasting
- Classification
- Optimization
- Unsupervised Learning

Our problem statement is to **design a system that predicts the fare amount for a cab ride taken in the city.**

Our problem statement is a regression problem (target variable is a continuous variable) and it falls under Forecasting category.

In our project we decided to go with, Linear Regression, Decision Tree, Random Forest, parameters hyper tuning of Random Forest with Randomized Search CV and Grid Search CV.

2.2.2 Linear Regression

Linear Regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It's used to predict values within a continuous range, (e.g. sales, price) rather than trying to classify them into categories (e.g. cat, dog).

Linear Regression, unlike other algorithms, stores information in terms of coefficients. It is a statistical model. We cannot use this for classification. It describes relationship among variables.

In our project, we get RMSE as 6.219815 and Rsquare as 0.484300. We are rejecting this model as RMSE is high and Rsquare is low when compared with all other models. Our aim is – we always want a model with low RMSE value i.e. minimum calculated errors and high R square

value i.e. the independent variables should have maximum potential to explain about the target variable.

Image 15

Linear Regression

In [58]:

```
# Building LRModel on training data
LRModel = LinearRegression().fit(X_train, y_train)

# Predicting LRModel on train data
LRTrain = LRModel.predict(X_train)

# Predicting LRModel on test data
LRTest = LRModel.predict(X_test)

# Calculating RMSE for train data
LRTrain_RMSE = np.sqrt(mean_squared_error(y_train, LRTrain))
print('RMSE for train data = ' +str(LRTrain_RMSE))

# Calculating RMSE for test data
LRTest_RMSE = np.sqrt(mean_squared_error(y_test, LRTest))
print('RMSE for test data = ' +str(LRTest_RMSE))

# Calculating Rsquare for train data
LRTrain_rsquare = r2_score(y_train, LRTrain)
print('Rsquare for train data = ' +str(LRTrain_rsquare))

# Calculating Rsquare for test data
LRTest_rsquare = r2_score(y_test, LRTest)
print('Rsquare for test data = ' +str(LRTest_rsquare))
```

```
RMSE for train data = 9.264227551993713
RMSE for test data = 6.219814989887592
Rsquare for train data = 0.3057507904124328
Rsquare for test data = 0.4843001126743429
```

2.2.3 Decision Tree

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter.

The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

In our project, we get RMSE as 4.504301 and Rsquare as 0.729544. Although, Decision Tree performed better than Linear Regression, we are rejecting this model as RMSE is high and Rsquare is low when compared with Random Forest. Our aim is – we always want a model with low RMSE value i.e. minimum calculated errors and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Image 16

Decision Tree

In [59]:

```
# Building DTModel on training data
DTModel = DecisionTreeRegressor(max_depth = 2).fit(X_train, y_train)

# Predicting DTModel on train data
DTTrain = DTModel.predict(X_train)

# Predicting DTModel on test data
DTTest = DTModel.predict(X_test)

# Calculating RMSE for train data
DTTrain_RMSE = np.sqrt(mean_squared_error(y_train, DTTrain))
print('RMSE for train data = ' +str(DTTrain_RMSE))

# Calculating RMSE for test data
DTTest_RMSE = np.sqrt(mean_squared_error(y_test, DTTest))
print('RMSE for test data = ' +str(DTTest_RMSE))

# Calculating Rsquare for train data
DTTrain_rsquare = r2_score(y_train, DTTrain)
print('Rsquare for train data = ' +str(DTTrain_rsquare))

# Calculating Rsquare for test data
DTTest_rsquare = r2_score(y_test, DTTest)
print('Rsquare for test data = ' +str(DTTest_rsquare))
```

```
RMSE for train data = 7.6015436393085505
RMSE for test data = 4.504301201426911
Rsquare for train data = 0.5325872525943498
Rsquare for test data = 0.7295437941743977
```

2.2.4 Random Forest

The Random Forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a “forest”), this model uses two key concepts that gives it the name *random*:

- Random sampling of training data points when building trees
- Random subsets of features considered when splitting nodes

The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

In our project, we get RMSE as 3.870582 and Rsquare as 0.796779. Although, Random Forest performed better than Decision Tree, we are rejecting this model as RMSE is high at a small margin and Rsquare is low at a small margin when compared with Randomized Search CV. Our aim is – we always want a model with low RMSE value i.e. minimum calculated errors

and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Image 17

Random Forest

In [60]:

```
# Building RFModel on training data
RFModel = RandomForestRegressor(n_estimators = 200).fit(X_train, y_train)

# Predicting RFModel on train data
RFTrain = RFModel.predict(X_train)

# Predicting RFModel on test data
RFTest = RFModel.predict(X_test)

# Calculating RMSE for train data
RFTrain_RMSE = np.sqrt(mean_squared_error(y_train, RFTrain))
print('RMSE for train data = ' +str(RFTrain_RMSE))

# Calculating RMSE for test data
RFTest_RMSE = np.sqrt(mean_squared_error(y_test, RFTest))
print('RMSE for test data = ' +str(RFTest_RMSE))

# Calculating Rsquare for train data
RFTrain_rsquare = r2_score(y_train, RFTrain)
print('Rsquare for train data = ' +str(RFTrain_rsquare))

# Calculating Rsquare for test data
RFTest_rsquare = r2_score(y_test, RFTest)
print('Rsquare for test data = ' +str(RFTest_rsquare))
```

```
RMSE for train data = 2.939819266900513
RMSE for test data = 3.890833217190048
Rsquare for train data = 0.9300901725901756
Rsquare for test data = 0.7981971325019972
```

2.2.5 Grid Search CV

Before talking about Grid Search CV, let me convey about Hyperparameter Tuning, which can be said as – Hyperparameter Tuning is choosing a set of optimal hyperparameters for a learning algorithm. Now, what is a hyperparameter - well, it can be defined as, it is a parameter whose value is set before the learning process begins.

There are two ways to do this – one is Grid Search and other one is Random Search. We will first discuss about Grid Search.

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters. Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.

The benefit of grid search is that it is guaranteed to find the optimal combination of parameters supplied. The drawback is that it can be very time consuming and computationally expensive. We can combat this with random search.

In our project, we get RMSE as 3.830760 and Rsquare as 0.804381. We will finally accept this model, as we got the best RMSE and Rsquare values when compared with all other models.

Our aim is – we always want a model with low RMSE value i.e. minimum calculated errors and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Image 18

In [63]:

```
# Building GridSearchCV
GRF = RandomForestRegressor(random_state = 0)

# Create grid
ran_sear2 = {'n_estimators': list(range(1,20,2)),
             'max_depth': list(range(1,10,2))}

gridcv_rf = GridSearchCV(GRF, param_grid = ran_sear2, 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)

# Rsquare
GRF_rsquare = 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_rsquare))
print('RMSE = ',(GRF_RMSE))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 11}
R-squared = 0.8.
RMSE =  3.830759661692142
```

2.2.6 Randomized Search CV

Random search differs from grid search mainly in the matter that it searches the specified subset of hyperparameters randomly instead of exhaustively. The major benefit being decreased processing time.

There is a trade-off to decreased processing time. However, we aren't guaranteed to find the optimal combination of hyperparameters.

Very similar to grid search above, we define the hyperparameters to search over before running the search.

An important additional parameter to specify here is n_iter. This specifies the number of combinations to randomly try.

Selecting too low of a number will decrease our chance of finding the best combination. Selecting too large of a number will increase our processing time.

In our project, we get RMSE as 3.887366 and Rsquare as 0.798557. It gave equal results with a little undesirable margin when compared with Random Forest, but we reject this model, as we found out that Grid Search CV is giving the best optimized values. Our aim is – we always want a model with low RMSE value i.e. minimum calculated errors and high R square value i.e. the independent variables should have maximum potential to explain about the target variable.

Image 19

In [62]:

```
# Building RandomizedSearchCV
RRF = RandomForestRegressor(random_state = 0)

# Create grid
ran_sear = {'n_estimators': list(range(1,20,2)),
            'max_depth': list(range(1,100,2))}

randcv_rf = RandomizedSearchCV(RRF, param_distributions = ran_sear, n_iter = 5, cv = 5, random_state = 0)
randcv_rf = randcv_rf.fit(X_train, y_train)
view_best_params_RRF = randcv_rf.best_params_

#Apply model on test data
predictions_RRF = randcv_rf.predict(X_test)

# Calculating Rsquare
RRF_rsquare = r2_score(y_test, predictions_RRF)

# Calculating RMSE
RRF_RMSE = np.sqrt(mean_squared_error(y_test, predictions_RRF))

print('Randomized Search CV Result:')
print('Best Parameters =', view_best_params_RRF)
print('R-Squared = {:0.2}.'.format(RRF_rsquare))
print('RMSE = ' ,RRF_RMSE)
```

```
Randomized Search CV Result:
Best Parameters = {'n_estimators': 15, 'max_depth': 9}
R-Squared = 0.8.
RMSE =  3.8873663314404117
```

Chapter 3

Conclusion

3.1 Model Evaluation

We always need a metric to evaluate the work we did. So, the same way, after we developed our models, we need a metric to validate the model we developed.

There are many metrics to evaluate, even, we have different metrics for classification problem and different metrics for regression problems.

For classification problems, we have metrics like:

- Confusion Matrix.
- Accuracy.
- Recall.
- Specificity.

For regression problems, we have metrics like:

- MSE.
- RMSE.
- MAPE.
- Rsquare.

We are choosing RMSE and Rsquare for our project. **Why RMSE over MAPE?**

Because, in RMSE, as the errors are squared before they are averaged, the RMSE gives a relatively high weightage to large errors, another reason is, RMSE penalizes large errors. For the above-mentioned reasons, we choose RMSE over MAPE.

3.1.1 Root Mean Squared Error (RMSE) & Rsquare

We are going to use RMSE and Rsquare as our error metrics to evaluate our models.

RMSE – Simply said, it is the sum of calculated errors.

Rsquare – Simply defined, correlation of original and predicted values.

3.2 Model Selection

Finally, it's our Model selection time. We developed five models. Linear Regression, Decision Tree, Random Forest, GridSearchCV on Random Forest and RandomizedSearchCV on Random Forest.

We are going to freeze GridSearchCV with RMSE as 3.830760 and Rsquare as 0.804381, even though RandomizedSearchCV on Random Forest has low RMSE value than Random Forest, because Rsquare is our priority, it's because, high Rsquare is equivalent to minimizing sum of squared errors, on the other hand, minimizing RMSE may yield biased point predictions.

Python Result:

Image 20

	RMSE	Rsquare
Linear Regression	6.219815	0.484300
Decision Tree	4.504301	0.729544
Random Forest	3.929988	0.794115
RandomizedSearch CV	3.887366	0.798557
GridSearch CV	3.830760	0.804381

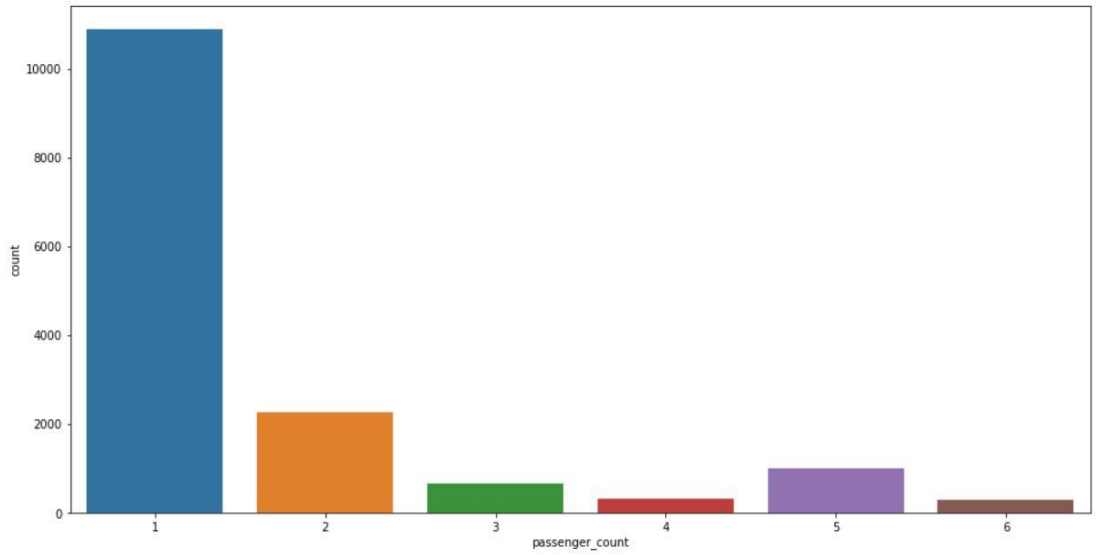
R Result:

Image 21

	Model	RMSE_Values_Train	RMSE_Values_Test	Rsquare_Values_Train	Rsquare_Values_Test
1	Linear Regression	160.4500	157.7057	0.1720470	0.1921982
2	Decision Tree	135.7522	128.3161	0.4073203	0.4663231
3	Random Forest	111.4996	126.8245	0.6442312	0.4907095

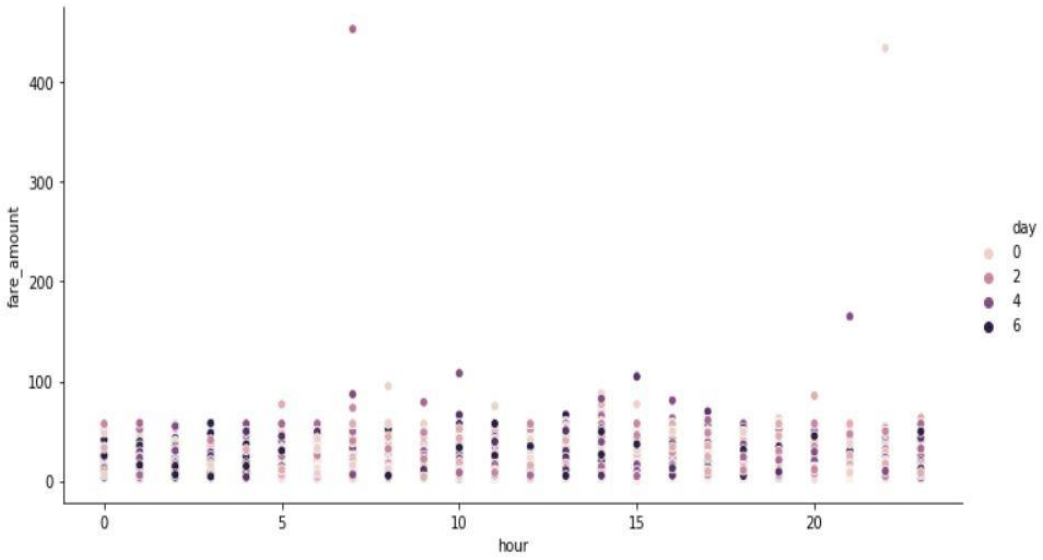
Appendix A – Extra Figures

Image 22



Single Passengers are the most frequent travellers

Image 23



The fare is highest, when hour is 7 am and 10 pm, and when days are Monday, Thursday and Friday

Image 24

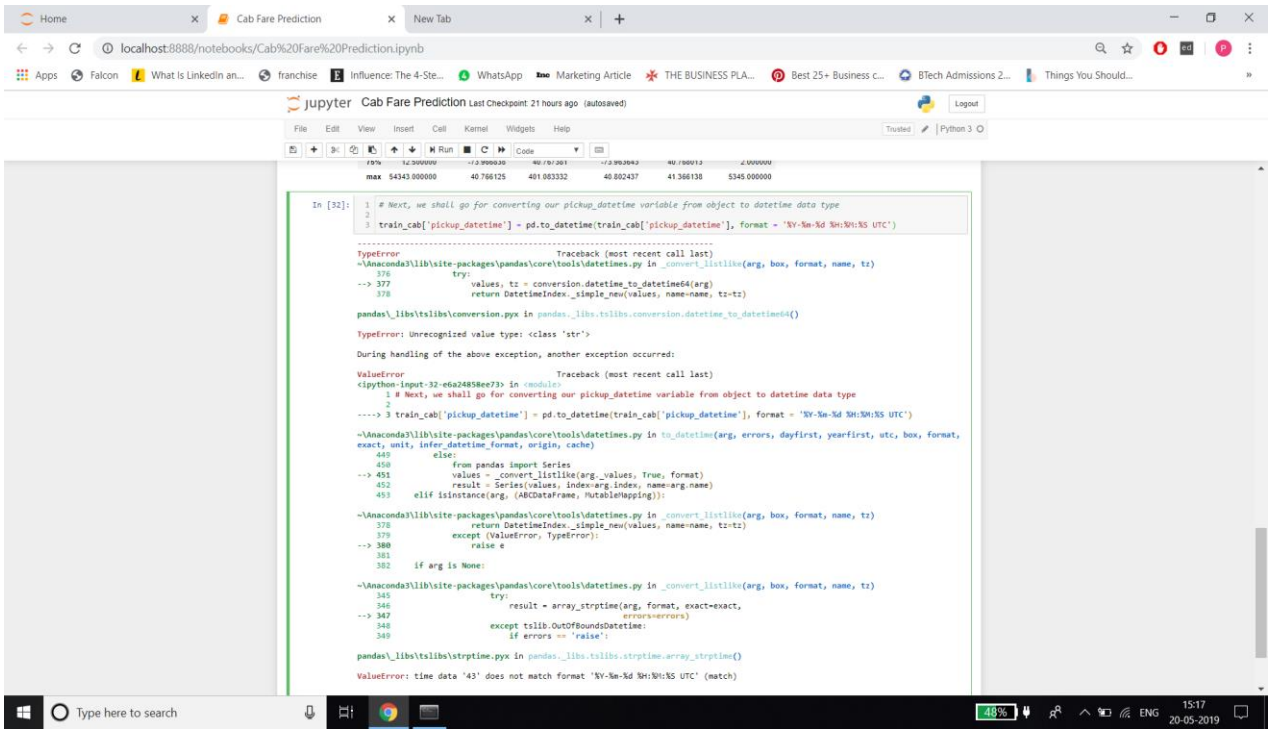
```
# Here's the output. It's a great journey

test_cab.head()
```

	year	month	hour	distance	Predicted fare_amount
0	2015	1	13	0.842971	9.062778
1	2015	1	13	0.885977	9.325481
2	2011	10	11	-0.480251	5.476132
3	2012	12	21	0.673471	8.595973
4	2012	12	21	1.684045	14.408911

Appendix B – Errors

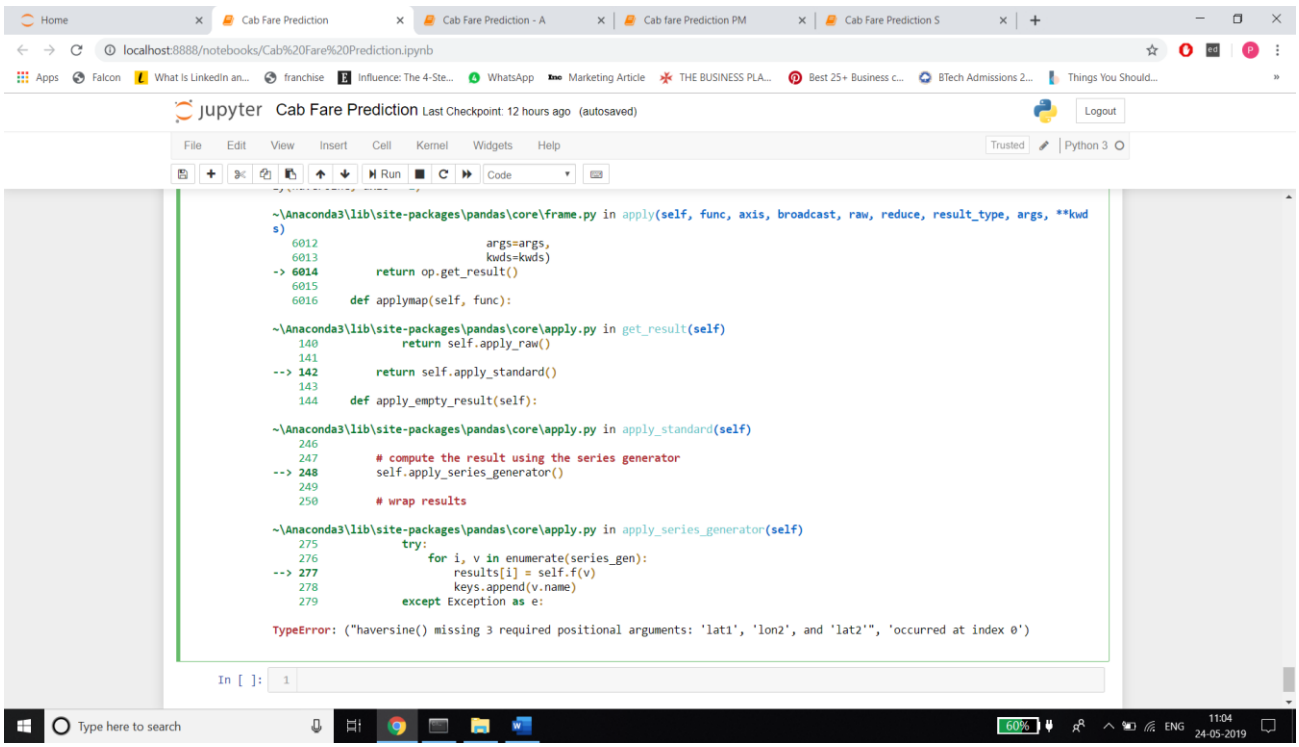
1. As I started the project, I observed that pickup_datetime variable format is in object, so, decided to convert it into datetime type. At that point of time, I faced an error, which you can see in Image 20. Later, I went back to original file, I found the 43 value in row number 1329. Later, went back to jupyter notebook and while importing the data, I skipped observation 1329, which allowed me to proceed further.

Image 25



2. While defining Haversine function to calculate distance, I got an error, check image 21 for that. I solved it by defining one object a, where I passed latitude and longitude variables in form of a list.

Image 26



3. Another one, which was massive, is when I faced while predicting the fare for test data given. I actually created dummy variables for train data. Later while applying the model to test data, it didn't take and said, number of features of train data does not match test data. In train, there were 45 variables because of dummies, but in test data, there were just 5. It was a learning point.

Appendix C – R Script

```
# Objective of our PROJECT - 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 requi
# -rement to apply analytics for fare prediction. You need to design a system that predicts fare amount for a cab ride in city.

# First step, as always, let's clean the environment
rm(list = ls())

# Setting the working directory
setwd("C:/Users/Purushottam/Desktop/Data Science/Cab Fare Prediction Project")

# checking the working directory
getwd()

# Loading the data into our R environment
train_cab = read.csv("train_cab.csv")

#~~~~~~~~~~~~~~~~~Let's interact with our data and perform Exploratory Data Analysis~~~~~~~~~~~~~~~~~~~~~~~~~~

class(train_cab) # Its DataFrame
head(train_cab) # Let's have a look on first 6 observations
dim(train_cab) # 16067 observations & 7 variables
str(train_cab) # Have a look on the structure, fare_amount and pickup_datetime are as factors
summary(train_cab) # With a glance, we can get that, pickup_latitude and passenger_count have outliers
names(train_cab) # In names, we can get to see as perfect naming for respective variables.

# As observed, we have to change fare_amount from factor to numeric
train_cab$fare_amount = as.numeric(train_cab$fare_amount)

# Fare_amount has some missing values around 25, but they got replaced with 1, so we are removing those observations with 1
train_cab = subset(train_cab, fare_amount != 1)

# Similarly, we are going to change pickup_datetime from factor to datetime
# But first, let's replace UTC in pickup_datetime variable with "
train_cab$pickup_datetime = gsub('\\ UTC', '', train_cab$pickup_datetime)
```

```
train_cab$Date = as.Date(train_cab$pickup_datetime)

# We are familiar with working with variables like, hour, day, date, year but not a complete datetime variable.
# We are also not dealing with time series analysis, so, now, we are going to split the pickup_datetime into its subsets
train_cab$year = substr(as.character(train_cab$Date),1,4)
train_cab$month = substr(as.character(train_cab$Date),6,7)
train_cab$weekday = weekdays(as.POSIXct(train_cab$Date), abbreviate = F)
train_cab$Date = substr(as.character(train_cab$Date),9,10)
train_cab$time = substr(as.factor(train_cab$pickup_datetime),12,13)

# Now, we are going to delete pickup_datetime variable, as we have already have its substitutes
train_cab = subset(train_cab, select = -c(pickup_datetime))

#~~~~~~~~~~~~~~~~~~Here comes Missing Values~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Line of code to know the sum of missing values in dataset
sum(is.na(train_cab)) # Total number of missing values are 59

# Line of code to know missing values in respective columns
apply(train_cab, 2, function(x) {sum(is.na(x))})

# Let's create a Missing Values dataframe for more clarity
missing_values = data.frame(apply(train_cab, 2, function(x){sum(is.na(x))}))
missing_values$Columns = row.names(missing_values)
names(missing_values)[1] = "Missing_values_percentage"

# Now, moving to calculate % of missing values
missing_values$Missing_values_percentage =
(missing_values$Missing_values_percentage/nrow(train_cab)) * 100

# Have a glance at descending order of missing values percentage
missing_values = missing_values[order(-missing_values$Missing_values_percentage),]
row.names(missing_values) = NULL

# Re-ordering columns for better understanding
missing_values = missing_values[,c(2,1)]

# We got idea that the missing values percentage is negligible, so we are dropping the observations with missing values
train_cab = na.omit(train_cab)

# Line of code to know the sum of missing values in dataset
sum(is.na(train_cab)) # Total number of missing values are 0

#~~~~~~~~~~~~~~~~~~Its time for OUTLIER analysis~~~~~~~~~~~~~~~~~~~~~~

# We are going to delete manually using ascending and descending options
# We are starting with fare_amount variable
```

```
# Descending order to know outliers
train_cab = train_cab[order(-train_cab$fare_amount),]

# Ascending order to know inliers
train_cab = train_cab[order(train_cab$fare_amount),]

# There are no outliers in fare_amount

# Next coming to latitudes and longitudes
# According to our learning, latitudes must range from -90 to +90 and longitudes must range from -180 to +180
# We are simply going to use summary function to find outliers in these four variables
summary(train_cab)

# In console we can see, only pickup_longitude has one value out of range i.e Max. :401.08
# Now, we are going to remove the outlier in pickup_latitude

train_cab = subset(train_cab, (pickup_latitude > -90 & pickup_latitude < +90))

# There are no outliers in pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude

# Next, we have passenger_count in our hands. Practical case, we can have minimum of 1 and maximum of 6 passengers.
# Descending order to know outliers
train_cab = train_cab[order(-train_cab$passenger_count),] # Lot of values are present which are > 6
# Ascending order to know inliers
train_cab = train_cab[order(train_cab$passenger_count),] # Lot of values are 0 and one value is 0.12.
# Line of code to remove the outliers in passenger_count
train_cab = subset(train_cab, passenger_count > 0 & passenger_count < 7) # Removed the outliers and inliers
train_cab = subset(train_cab, passenger_count != 0.12) # Removed observation with passenger_count value as 0.12

# There are no outliers in passenger_count

# We have longitudes and latitudes. With the concept of feature engineering, we decided to create distance variable
# Using Haversine Formula, we can create distance variable.

# Calculates the geodesic distance between two points specified by radian latitude/longitude using the Haversine formula (hf)
gcd.hf = function(long1, lat1, long2, lat2) {
  R = 6371 # 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 * asin(min(1,sqrt(a)))
  d = R * c
  return(d) # Distance in km
}

# Now, we are going to apply the function, over all the rows to create a new variable - distance
for (i in 1:nrow(train_cab)){
  train_cab$distance[i] = gcd.hf(train_cab$pickup_longitude[i], train_cab$pickup_latitude[i],
                                 train_cab$dropoff_longitude[i], train_cab$dropoff_latitude[i])
}

# Dimension Reduction - we are going to delete latitude and longitude variables as we obtained
distance from these four variables
train_cab = subset(train_cab, select = -c(pickup_latitude, pickup_longitude, dropoff_latitude,
dropoff_longitude))

# Now, its time to check outliers in distance
train_cab = train_cab[order(-train_cab$distance),]
train_cab = train_cab[order(train_cab$distance),]

# Removing the outliers from the distance variable
train_cab = subset(train_cab, distance > 0 & distance <500)

#~~~~~~~~~~~~~~Let's do some visualizations~~~~~~~~~~~~~~~~~~~~

# Time to plot some graphs, so let's install few libraries

library(ggplot2)
library(scales)
library(psych)
library(gplots)

# Visualization between fare_amount and weekday.
ggplot(data = train_cab, 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))
#--> We can understand that - Thursday and Saturday rides obtains the highest fare_amount

# Visualization between fare_amount and months.
ggplot(train_cab, aes(x = reorder(month,-fare_amount), y = fare_amount))+
  geom_bar(stat = "identity")+
  labs(title = "Fare Amount Vs. Month", x = "Month", y = "Fare")+
  theme(axis.text.x = element_text( color="navy blue", size=8))
#--> We can observe that - specific months liks January, March, June collects the highest
fare_amount

# Visualization between fare_amount and years.
ggplot(data = train_cab, aes(x = reorder(year,-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))
#--> We can say, in year 2009 and 2010 there were rides which got high fare_amount

# Visualization between fare_amount and time.
ggplot(data = train_cab, aes(x = time, 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))
#--> Rides taken during 6 pm to 8 pm gives highest fare_amount

# Visualization between fare_amount and passenger_count.
ggplot(data = train_cab, aes(x = passenger_count, 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))
#--> Rides with single passenger will lead to highest fares

# Visualization between fare_amount and distance.
ggplot(train_cab,aes(distance,fare_amount)) +
  geom_point(alpha=0.5) +
  labs(title = "Fare amount based on distance", x = "Distance", y = "Fare Amount")+
  scale_color_gradientn(colors=c('blue','light blue','dark blue','light green','yellow','dark
orange','black')) +
  theme_bw()
#--> It's an obvious fact, fare_amount is directly proportional to distance

#~~~~~~~~~~~~~~~~~~~Time for Feature Selection~~~~~~~~~~~~~~~~~~
# Storing continuous variables
num_var = c('fare_amount', 'distance')

# Storing categorical variables
cat_var = c('Date', 'year', 'month', 'weekday', 'time', 'passenger_count')

# Keeping data in another object
train_cab2 = train_cab
train_cab = train_cab2

# In Feature Selection, we perform Correlation Analysis and Anova test to find out the varaibles
which are to be excluded
# before feeding to the model

# Correlation Analysis is performed between num_var (continuous independent variables) &
fare_amount (continuous target variable)

library(corrgram)
```

```
corrgram(train_cab[,num_var],order=FALSE,upper.panel = panel.pie,
        text.panel = panel.txt,
        main= "Correlation Analysis between numeric variables")

#--> We can include distance in model development phase

# Anova Test is performed between cat_var (categorical independent variables) & fare_amount
(continuous target variable)

for(i in cat_var){
  print(i)
  Anova_test_result = summary(aov(formula = fare_amount~train_cab[,i],train_cab))
  print(Anova_test_result)
}

#-->> From the result, we can observe, passenger_count, weekday and Date has p value > 0.05,
by which, we accept null hypothesis.

# Deleting the below given continuous and categorical variables from day, as we found out
they won't add any value to the model.
train_cab = subset(train_cab, select = -c(passenger_count, weekday, Date))

#~~~~~~~~~~~~~~~~~Let's jump to FEATURE SCALING~~~~~~~~~~~~~~~~~~

# Checking distance variable distribution using histogram
hist(train_cab$distance)
#--> The diagram represents, it is left skewed

# Checking summary of distance
summary(train_cab$distance)
#--> Maximum value is 499.8405. We will go for normilisation

# We are going to define function using log
signedlog10 = function(x) {
  ifelse(abs(x) <= 1, 0, sign(x)*log10(abs(x)))
}

# Applying log function to distance variable
train_cab$distance = signedlog10(train_cab$distance)

# Checking distance distribution after applying function
hist(train_cab$distance)

# Let's look at summary again
summary(train_cab$distance)

# Let's manually remove the outliers in distance variable
train_cab = subset(train_cab, distance > 0)
```

```
# Till now, we completed Exploratory Data Analysis, plotted visualizations, worked on Data Pre Processing, defined error metrics

#~~~~~~~~~~~~~~~Now, We can go for MODEL DEVELOPMENT~~~~~~~~~~~~~~~~~

# Let's clean R Environment, as it uses RAM which is limited

library(DataCombine)
rmExcept("train_cab")

# Defining Error Metric, which are used to evaluate the model

# Defining R Square function - Correlation of original and predicted values.

Rsquare = function(y,y1){
  cor(y,y1)^2
}

# Defining RMSE function - Root Mean Squared Errors (Calculated Errors)
# RMSE over MAPE, Why? - Because, RMSE gives weight to large errors during calculation and it penalizes large errors

RMSE = function(y,y1){
  sqrt(mean((y - y1)^2))
}

# Let's create copy of data for further reference

train_cab2 = train_cab
train_cab = train_cab2

# Now, as we know, we have to divide the data into train and test. So, let's go for that.
set.seed(123)
train_index = sample(1:nrow(train_cab),0.8*nrow(train_cab)) # Using Simple Random Sampling Technique
train= train_cab[train_index,]
test= train_cab[-train_index,]

#--> We can see our train and test data are available in the environment

#~~~~~~~~~~~~~~~We are going for LINEAR REGRESSION MODEL~~~~~~~~~~~~~~~~~

# Code for development of model
LRModel = lm(fare_amount~., train)

# Let's have a view on the summary
summary(LRModel)

# Predicting model on train data
LRTrain = predict(LRModel, train[-1])
```

```
# Predicting model on test data
LRTest = predict(LRModel, test[-1])

# Calculating RMSE for Train Data
LRRMSE_Train = RMSE(LRTrain, train[,1])

# Calculating RMSE for Test Data
LRRMSE_Test = RMSE(LRTest, test[,1])

# Calculating Rsquare for Train Data
LRR2_Train = Rsquare(train[,1], LRTrain)

# Calculating Rsquare for Test Data
LRR2_Test = Rsquare(test[,1], LRTest)

#~~~~~~~~~~~~~~~It's time for DECISION TREE~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Code to build Decision Tree
library(rpart)

DTModel = rpart(fare_amount~., train, method = "anova")

# Predicting model on train data
DTTrain = predict(DTModel, train[-1])

# Predicting model on test data
DTTest = predict(DTModel, test[-1])

# Calculating PMSE for Train Data
DTRMSE_Train = RMSE(DTTrain, train[,1])

# Calculating RMSE for Test Data
DTRMSE_Test = RMSE(DTTest, test[,1])

# Calculating Rsquare for Train Data
DTR2_Train = Rsquare(train[,1], DTTrain)

# Calculating Rsquare for Test Data
DTR2_Test = Rsquare(test[,1], DTTest)

#~~~~~~~~~~~~~~~Here we go for RANDOM FOREST~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Code for development of model

library(randomForest)

RFModel = randomForest(fare_amount~., train, ntree = 500, method = "anova")

# Predicting model on train data
```

```
RFTrain = predict(RFModel, train[-1])

# Predicting model on test data
RFTest = predict(RFModel, test[-1])

# Calculating RMSE for Train Data
RFRMSE_Train = RMSE(RFTrain, train[,1])

# Calculating RMSE for Test Data
RFRMSE_Test = RMSE(RFTest, test[,1])

# Calculating Rsquare for Train Data
RFR2_Train = Rsquare(train[,1], RFTrain)

# Calculating Rsquare for Test Data
RFR2_Test = Rsquare(test[,1], RFTest)

#~~~~~~~~~~~~~~~~Let's save all the values in a data frame~~~~~~~~~~~~~~~~~~~~~

Result = data.frame("Model" = c("Linear Regression", "Decision Tree", "Random Forest"),
                    "RMSE_Values_Train" = c(LRRMSE_Train, DTRMSE_Train,
RFRMSE_Train),
                    "RMSE_Values_Test" = c(LRRMSE_Test, DTRMSE_Test, RFRMSE_Test),
                    "Rsquare_Values_Train" = c(LRR2_Train, DTR2_Train, RFR2_Train),
                    "Rsquare_Values_Test" = c(LRR2_Test, DTR2_Test, RFR2_Test))

#--> Random Forest gives the most optimized values. We are going to freeze Random Forest

# Now, it's time to bring the test data and perform the steps as performed on train_cab and
predict the fare_amount
# Loading the data into our R environment
test_cab = read.csv("test.csv")

#~~~~~~Let's interact with our data and perform Exploratory Data Analysis~~~~~~~~~~
class(test_cab) # Its DataFrame
head(test_cab) # Let's have a look on first 6 observations
dim(test_cab) # 9914 observations & 6 variables
str(test_cab) # Have a look on the structure, pickup_datetime is in factor
summary(test_cab) # With a glance, we can get that, pickup_latitude and passenger_count don't
have outliers
names(test_cab) # In names, we can get to see as perfect naming for respective variables.

# We are going to change pickup_datetime from factor to datetime
# But first, let's replace UTC in pickup_datetime variable with "
test_cab$pickup_datetime = gsub('\\ UTC', '', test_cab$pickup_datetime)
test_cab$Date = as.Date(test_cab$pickup_datetime)

# We are familiar with working with variables like, hour, day, date, year but not a complete
datetime variable.
```

```
# We are also not dealing with time series analysis, so, now, we are going to split the pickup_datetime into its subsets
test_cab$year = substr(as.character(test_cab$Date),1,4)
test_cab$month = substr(as.character(test_cab$Date),6,7)
test_cab$weekday = weekdays(as.POSIXct(test_cab$Date), abbreviate = F)
test_cab$Date = substr(as.character(test_cab$Date),9,10)
test_cab$time = substr(as.factor(test_cab$pickup_datetime),12,13)

# Now, we are going to delete pickup_datetime variable, as we have already have its substitutes
test_cab = subset(test_cab, select = -c(pickup_datetime))

#~~~~~~~~~~~~~~~~Here comes Missing Values~~~~~~~~~~~~~~~~~~~~

# Line of code to know the sum of missing values in dataset
sum(is.na(test_cab)) # Total number of missing values are 0

#~~~~~~~~~~~~~~~~Its time for OUTLIER analysis~~~~~~~~~~~~~~~~~

# Coming to outliers, let's check summary and check for outliers
summary(test_cab)
#--> It clearly shows, there are no outliers.

# Now, let's create distance using Haversine Formula as did in train_cab

# Calculates the geodesic distance between two points specified by radian latitude/longitude using the Haversine formula (hf)
gcd.hf = function(long1, lat1, long2, lat2) {
  R = 6371 # 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 * asin(min(1,sqrt(a)))
  d = R * c
  return(d) # Distance in km
}

# Now, we are going to apply the function, over all the rows to create a new variable - distance
for (i in 1:nrow(test_cab)){
  test_cab$distance[i] = gcd.hf(test_cab$pickup_longitude[i], test_cab$pickup_latitude[i],
                                test_cab$dropoff_longitude[i], test_cab$dropoff_latitude[i])
}

# Dimension Reduction - we are going to delete latitude and longitude variables as we obtained distance from these four variables
test_cab = subset(test_cab, select = -c(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude))

# Now, its time to check outliers in distance
test_cab = test_cab[order(-test_cab$distance),]
test_cab = test_cab[order(test_cab$distance),]
```

```
# Removing the outliers from the distance variable
test_cab = subset(test_cab, distance > 0 & distance <500)

# To match the number of features of test data to the model, we are going to delete few variables
test_cab = subset(test_cab, select = -c(passenger_count, weekday, Date))

# We are going to normalize the data in distance variable

# Checking distance variable distribution using histogram
hist(test_cab$distance)
#--> The diagram represents, it is left skewed

# Checking summary of distance
summary(test_cab$distance)
#--> Maximum value is 499.8466. We will go for normilisation

# We are going to define function using log
signedlog10 = function(x) {
  ifelse(abs(x) <= 1, 0, sign(x)*log10(abs(x)))
}

# Applying log function to distance variable
test_cab$distance = signedlog10(test_cab$distance)

# Checking distance distribution after applying function
hist(test_cab$distance)

# Let's look at summary again
summary(test_cab$distance)

# Let's manually remove the outliers in distance variable
test_cab = subset(test_cab, distance > 0)

# We are near to predict our cab fare using Random Forest

# Code for development of model

RFModel = randomForest(fare_amount~., train, ntree = 500, method = "anova")

# Predicting model on test_cab data
RFTest_cab = predict(RFModel, test_cab)

# Adding our obtained predictions as Predicted Fare Amount variable to test_cab dataset
test_cab$Predicted_fare_amount = RFTest_cab

# Here's a glance of predicted values
head(test_cab)

# Exporting the output to hard disk for further use
```

```
write.csv(test_cab, "C:/Users/Purushottam/Desktop/Data Science/Cab Fare Prediction Project/Cab Fare Prediction Output 2.csv",
          row.names = FALSE)
```

#--> Finally, we designed a model, which predicts the cab fare. Our objective achieved.

References

- Edwisor.com
- Edwisor Community.
- Google
- Pasting few useful links here:
- To get idea to deal with pickup_datetime variable, we used this source - https://stackoverflow.com/questions/46428870/how-to-handle-date-variable-in-machine-learning-data-pre-processing
- Coming to latitude and longitude variables, we got idea how to use them to go forward using these links - https://gist.github.com/rochacbruno/2883505 & https://gis.stackexchange.com/questions/119846/calculating-distance-between-latitude-and-longitude-points-using-python/119854
- In normalisation, we used this link to deal with normal distribution - https://towardsdatascience.com/transforming-skewed-data-73da4c2d0d16