PROJECT: CAB FARE PREDICTION

Submitted By: MEGHA TAPALI

TABLE OF CONTENTS

Chapter 1: Introduction

a) Problem Statement

The objective of this project is to predict Cab Fare amount. 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.

b) Data:

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.

CHAPTER 2: METHODOLOGY

According to industry standards, the process of Data Analysing mainly includes 6 main steps and this process is abbreviated as CRISP DM Process, which is Cross-Industry Process for Data Mining. And the Six main steps of CRISP DM Methodology for developing a model are:

1. Business understanding
2. Data understanding
3. Data preparation/Data Pre-processing
4. Modelling
5. Evaluation
6. Deployment

2.1 Business Understanding

It is important to understand the idea of business behind the data set. The given data set is asking us to predict fare amount. And it really becomes important for us to predict the fare amount accurately. Else, there might be great loss to the revenue of the firm. Thus, we have to concentrate on making the model most efficient.

2.2 Data Understanding

To get the best results, to get the most effective model it is really important to our data very well. Here, the given train data is a CSV file that consists 7 variables and 16067 Observation. A snapshot of the data provided.

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.84161	40.712278	1
16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.97927	40.782004	1
5.7	2011-08-18 00:35:00 UTC	-73.982738	40.76127	-73.99124	40.750562	2
7.7	2012-04-21 04:30:42 UTC	-73.98713	40.733143	-73.99157	40.758092	1
5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.95666	40.783762	1

Table 2.1: Train Data

The different variables of the data are:

- fare_amount: fare of the given cab ride.
- pickup_datetime: timestamp value explaining the time of ride start.
- pickup_longitude: a float value explaining longitude location of the ride start.
- pickup_latitude: a float value explaining latitude location of the ride start.
- dropoff_longitude: a float value explaining longitude location of the ride end.
- dropoff_latitude: a float value explaining latitude location of the ride end
- passenger_count: an integer indicating the number of passengers

Following table explains how the variables are categorized.

Independent Variables
pickup_datetime
pickup_longitude
pickup_latitude
dropoff_longitude
dropoff_latitude
passenger_count

Table 2.2: Independent Variables

Dependent Variables
fare_amount

Table 2.3: Dependent/Target Variable

From the given train data, it is understood that, we have to predict fare amount, and other variables will help me achieve that, here pickup_latitude/longitude, dropoff_latitude/longitude this data are signifying the location of pick up and drop off. It is explaining starting point and end point of the ride. So, these variables are crucial for us. Passenger_count is another variable, that explains about how many people or passenger boarded the ride, between the pickup and drop off locations. And pick up date time gives information about the time the passenger is picked up and ride has started. But unlike pick up and drop off locations has start and end details both in given data. The time data has only start details and no time value or time related information of end of ride. So, during pre-processing of data we will drop this variable. As it seems the information of time is incomplete.

Also, there is a separate test data given, in the format of CSV file containing 9514 observations and 6 variables. All of them are the Independent variables. An in these data at the end we have to predict the fare or the target variable. Following is a snap of the test data provided.

pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2015-01-27 13:08:24 UTC	-73.97332001	40.76380539	-73.98143005	40.74383545	1
2015-01-27 13:08:24 UTC	-73.98686218	40.71938324	-73.99888611	40.73920059	1
2011-10-08 11:53:44 UTC	-73.982524	40.75126	-73.979654	40.746139	1
2012-12-01 21:12:12 UTC	-73.98116	40.767807	-73.990448	40.751635	1
2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	1

Table 2.4: Test Data

2.3 Data Preparation

The next step in the CRISP DM Process is, Data pre-processing. It is a data mining process that involves transformation of raw data into a format that helps us execute our model well. As, the data often we get are incomplete, inconsistent and also may contain many errors. Thus, Data pre-processing is a generic method to deal with such issues and get a data format that is easily understood by machine and that helps developing our model in best way. In this project also we have followed data pre-processing methods to rectify errors and issues in our data. And this is done by popular data pre-processing techniques, this are following below.

Missing Value Analysis

Missing value is availability of incomplete observations in the dataset. This is found because of reasons like, incomplete submission, wrong input, manual error etc. These Missing values affect the accuracy of model. So, it becomes important to check missing values in our given data.

In the given dataset it is found that there are lot of values which are missing. It is found in the following types:

1. Blank spaces: Which are converted to NA and NaN in R and Python respectively for further operations
2. Zero Values: This is also converted to NA and Nan in R and python respectively prior further operations
3. Repeating Values: there are lots of repeating values in pickup_longitude, pickup_latitude, dropoff_longitude and dropoff_latitude. This will hamper our model, so such data is also removed to improve the performance.

Following the standards of percentage of missing values we now have to decide to accept a variable or drop it for further operations. Industry standards ask to follow following standards:

1. Missing value percentage < 30% : Accept the variable
2. Missing value percentage > 30 % : Drop the variable

Missing values can be treated in two ways: Delete or Impute

a) Outlier Analysis

Outlier is an abnormal observation that stands or deviates away from other observations. These happens because of manual error; poor quality of data and it is correct but exceptional data. But it can cause an error in predicting the target variables. So we have to check for outliers in our data set and also remove or replace the outliers wherever required

Missing Values and Outliers in this project:

1. **Fare_Amount:** Fare amount contains negative and NANs. Also, we can see below that two fares above 453 are very high compared to whole dataset. Hence, we remove all outliers and NANs, as missing percentage is only 1.9%.

```
##finding decending order of fare to get to know whether the outliers are present or not
train["fare_amount"].sort_values(ascending=False)
```

```
1015    54343.0
1072     4343.0
607       453.0
980       434.0
1335      180.0
          ...
1712        NaN
2412        NaN
2458        NaN
8178        NaN
8226        NaN
Name: fare_amount, Length: 16067, dtype: float64
```

2. **Passenger_count:** We can see maximum number of passenger count is 5345 which is actually not possible. So, reducing the passenger count to 6 (even if we consider the SUV). Also, passenger count cannot be 0. So, we delete all values greater than 6 and values equals to and less than 0.

```
train["passenger_count"].describe()
```

```
count    15980.000000
mean         2.623693
std         60.903563
min          0.000000
25%          1.000000
50%          1.000000
75%          2.000000
max       5345.000000
Name: passenger_count, dtype: float64
```

3. **Location points:** As we know, latitude ranges from -90 to 90 and Longitude ranges from -180 to 180. We need to drop rows exceeding the above range.

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
3.3	2011-07-30 11:15:00 UTC	-73.947235	401.083332	-73.951392	40.778927	1

We can see one inappropriate data. So we drop that row.

c) Feature Engineering:

Sometimes it happens that, all the variables in our data may not be accurate enough to predict the target variable, in such cases we need to analyse our data, understand our data and select the dataset variables that can be most useful for our model. In such cases we follow feature selection. Feature selection helps by reducing time for computation of model and also reduces the complexity of the model.

After understanding the data, pre-processing and selecting specific features, there is a process to engineer new variables if required to improve the accuracy of the model.

In this project the data contains only the pick up and drop points in longitude and latitude. The fare_amount will mainly depend on the distance covered between these two points. Thus, we have to create a new variable prior further processing the data. And in this project the variable created is Distance variable (distance), which is a numeric value and explains the distance covered between the pick up and drop of points.

We use Haversine function to do so. This function calculates the shortest distance between two points in a sphere.

We have built in haversine function in both python and R:

Python:

```
Signature: haversine(point1, point2, unit=<Unit.KILOMETERS: 'km'>)
Docstring:
Calculate the great-circle distance between two points on the Earth surface.

Takes two 2-tuples, containing the latitude and longitude of each point in decimal degrees,
and, optionally, a unit of length.

:param point1: first point; tuple of (latitude, longitude) in decimal degrees
:param point2: second point; tuple of (latitude, longitude) in decimal degrees
:param unit: a member of haversine.Unit, or, equivalently, a string containing the
             initials of its corresponding unit of measurement (i.e. miles = mi)
             default 'km' (kilometers).
```

R:

```
distHaversine(p1, p2, r=6378137)
```

Arguments

p1	longitude/latitude of point(s). Can be a vector of two numbers, a matrix of 2 columns (first one is longitude, second is latitude) or a SpatialPoints* object
p2	as above; or missing, in which case the sequential distance between the points in p1 is computed
r	radius of the earth; default = 6378137 m

After executing the haversine function in our project, we got new variable distance and some instances of data are mentioned below.

pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	distance
-73.97332001	40.76380539	-73.98143005	40.74383545	1	1.200947
-73.98686218	40.71938324	-73.99888611	40.73920059	1	1.231205
-73.982524	40.75126	-73.979654	40.746139	1	0.481579
-73.98116	40.767807	-73.990448	40.751635	1	1.085539
-73.966046	40.789775	-73.988565	40.744427	1	1.854313

Here, we check for outliers in distance column

```
#finding decending order of distance to get to know whet
train['distance'].sort_values(ascending=False).head(50)
```

```
9147     8667.554076
8647     8667.509484
2397     8667.466393
472      8667.316940
11653    8666.713475
13340    8666.625616
10215    8666.596677
4597     8666.578001
10458    8665.988191
10672    8665.714360
10488    8665.567603
1260     8665.280557
4278     8665.235736
6188     8664.203456
12983    8664.143776
6302     8663.051088
12705    8661.374116
14197    8657.148577
15783    8656.726125
15749    6028.935107
2280     6026.502540
5864     5420.996446
7014     4447.092840
10710     129.950662
14536     129.560634
11619     127.509437
12228     123.561327
```

```
train["distance"].describe()
```

```
count    15902.000000
mean        15.070804
std        311.732831
min          0.000000
25%          1.215752
50%          2.125953
75%          3.851274
max       8667.554076
Name: distance, dtype: float64
```

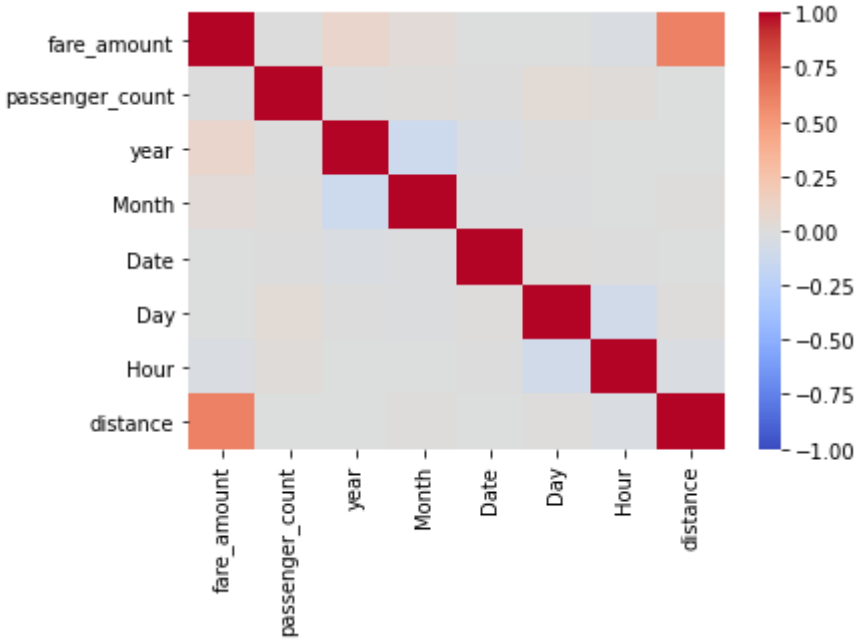
We can see that distance above 129 are outliers, as 8000km is practically inappropriate distance to travel in a cab. So, we delete rows having distance above 130 and below 0.01 as 0 distance doesn't make any sense

d) Feature Selection:

i) Correlation Analysis:

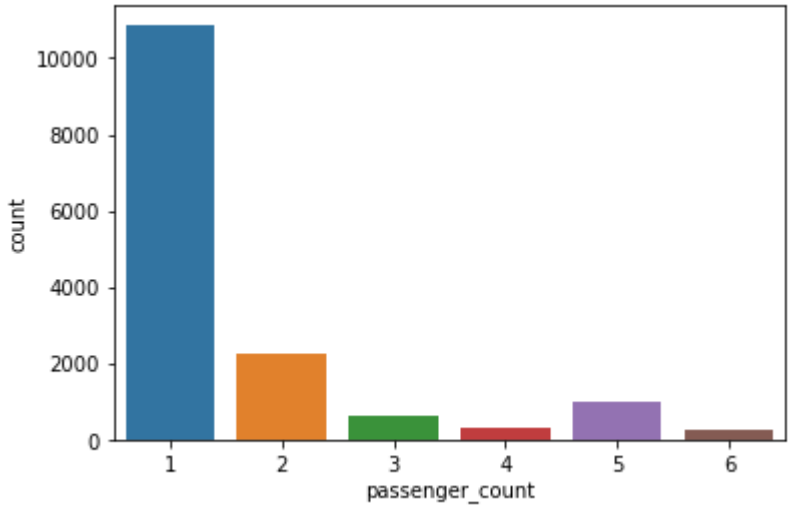
In some cases, it is asked that models require independent variables free from collinearity issues. This can be checked by correlation analysis for the categorical variables and continuous variables. Correlation analysis is a process that is defined to identify the level of relation between two variables.

In this project, our Predictor variable is continuous, so we will plot a correlation table that will predict the correlation strength between independent variables and the 'fare_amount' variable

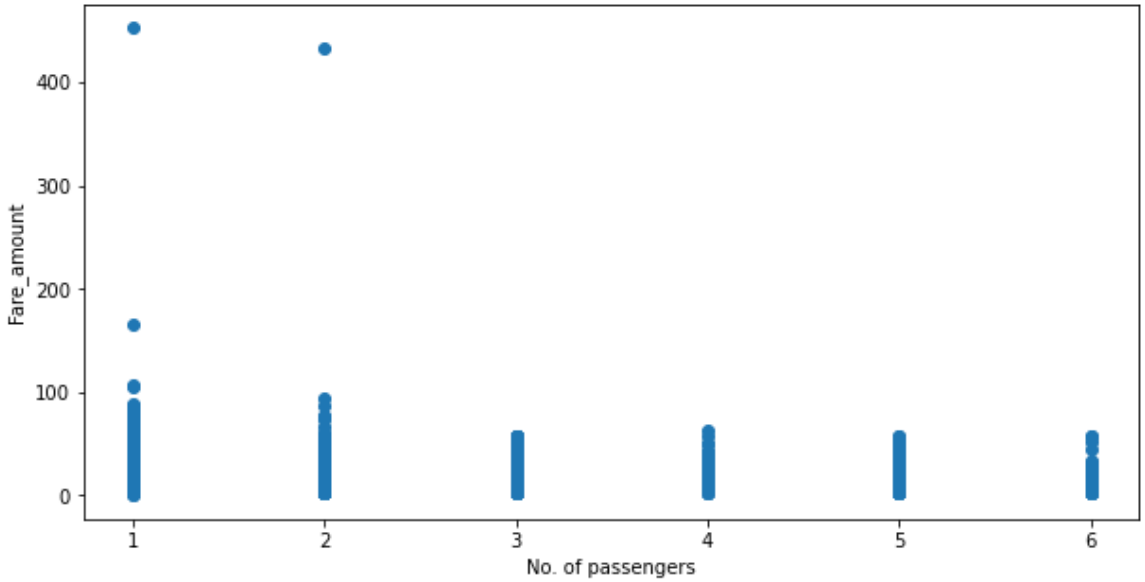


From above plot we can see that features are not highly correlated with each other. And label "fare_amount" is highly correlated with distance feature. Since label is continuous variable, it is a regression problem. And for regression, features should be not correlated and there should be high collinearity between feature and label.

DATA VISUALIZATION:



We can observe that there are lot of single passengers, followed by 2,5,3,4 and 6

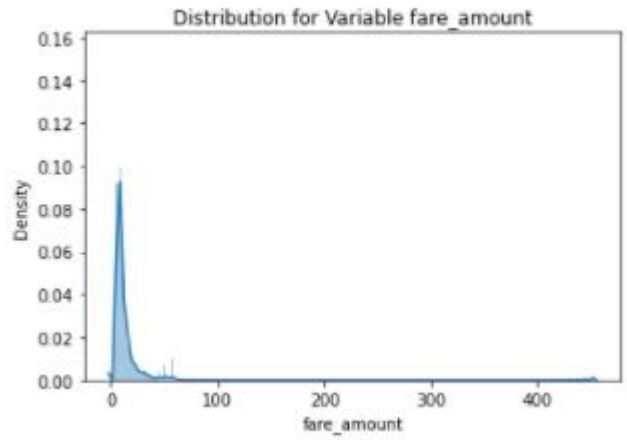


Single travelling and 2 people travelling passengers contribute a lot to the fare amount.

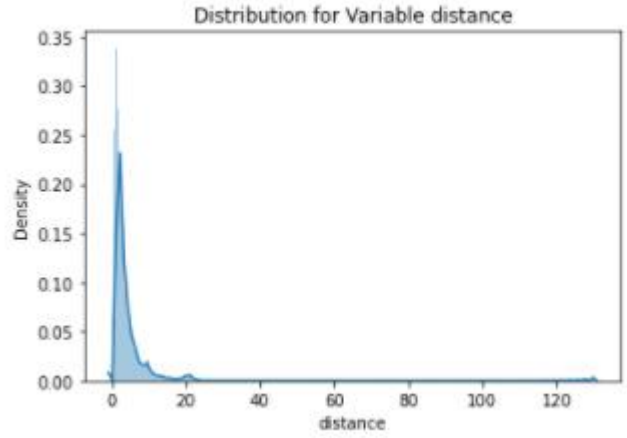
Distribution of the features:

Let's check the distribution of the features.

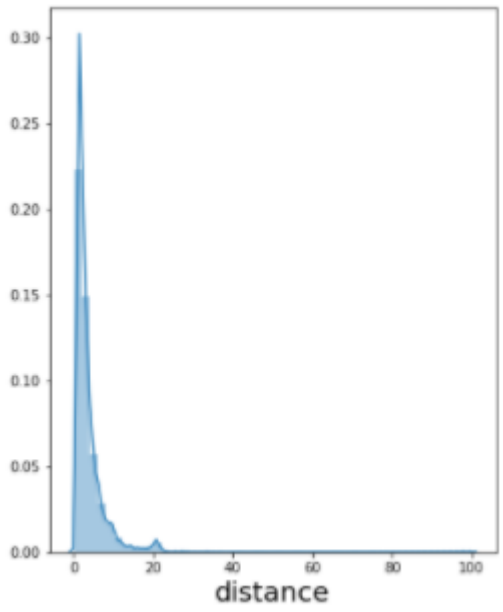
fare_amount



distance



The distribution in fare amount and distance of train dataset are skewed.



The distribution in distance of train dataset is skewed.

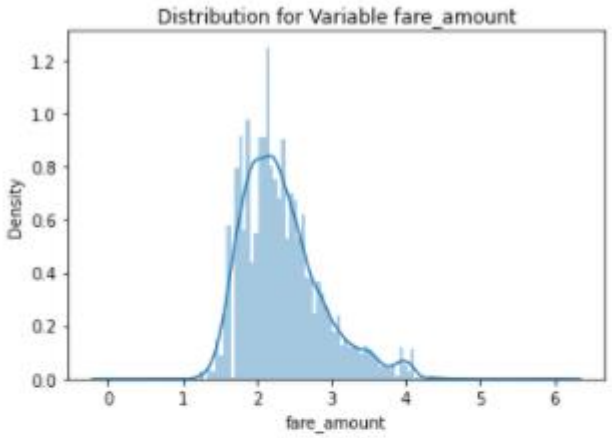
e) Feature Scaling

Feature scaling is the process of dealing with skewed data, making the feature to set with in a scale of 0-10, by applying the log transformation.

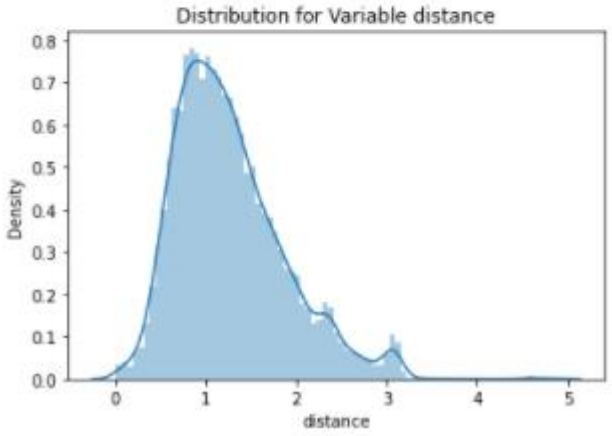
Data Transformation: - In statistics, data transformation is the application of a deterministic mathematical function to each point in a data set — that is, each data point z_i is replaced with the transformed value $y_i = f(z_i)$, where f is a function. Transforms are usually applied so that the data appear to more closely meet the assumptions of a statistical inference procedure that is to be applied, or to improve the interpretability or appearance of graphs.

Log Transformation: - The log transformation can be used to make highly skewed distributions less skewed. This can be valuable both for making patterns in the data more interpretable and for helping to meet the assumptions of inferential statistics. After applying normalization, the data representation is here below

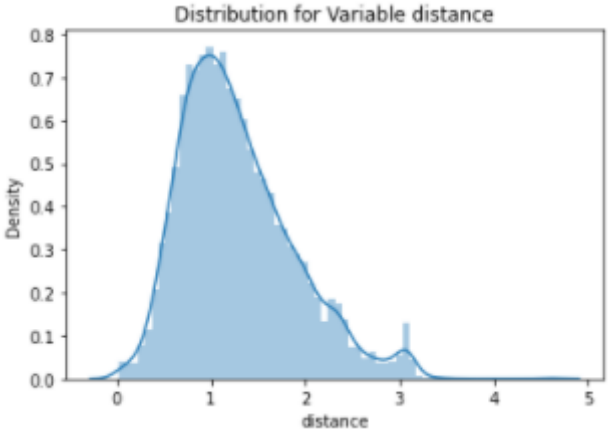
```
fare_amount
```




```
distance
```



Test Data:



CHAPTER 3: EVALUATION OF MODEL

3.1 Splitting train and Validation Dataset

a) We have used sklearn's train_test_split() method to divide whole Dataset into train and validation dataset.
b) 20% is in validation dataset and 80% is in training data.
c) We will test the performance of model on validation dataset.
d) The model which performs best will be chosen to perform on test dataset provided along with original train dataset.
e) x_train y_train--are train subset.
f) x_test y_test--are validation subset.

3.2 Model Evaluation Metrics:

Evaluating a model is a core part of building an effective machine learning model There are several evaluation metrics, like confusion matrix, cross-validation, AUC-ROC curve, etc. Different evaluation metrics are used for different kinds of problems.

1. R-squared: R^2 is a statistic that will give some information about the goodness of fit of a model. In regression, the R^2 coefficient of determination is a statistical measure of how well the regression predictions approximate the real data points. Therefore, we are using this to measure the performance of model in this project. A R^2 of 1 indicates that the regression predictions perfectly fit the data.

2. RMSE: Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. It is in the range of the data and hence is more interpretable as compared to MSE (Mean Square Error) It also penalize large errors have the result be in the same units as the outcome variable. It's a great choice for a loss metric when hyperparameter tuning.

We are using RMSE over MAE (Mean Absolute Error) because the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE is more useful when large errors are particularly undesirable. Hence, we are using RMSE to evaluate the model.

3.3: Model Development:

After all the above processes the next step is developing the model based on our prepared data.

In this project we got our target variable as "fare_amount". The model has to predict a numeric value. Thus, it is identified that this is a **Regression** problem statement. And to develop a regression model, the various models that can be used are Decision trees, Random Forest and Linear Regression.

1. Linear Regression

It is used to predict the value of variable Y based on one or more input predictor variables X. The goal of this method is to establish a linear relationship between the predictor variables and the response variable. Such that, we can use this formula to estimate the value of the response Y, when only the predictors (X- Values) are known. In this project Linear Regression is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.7943085750383834
RMSE	0.24272749872139385

Error metrics in R:

MAE	0.17515868
MSE	0.07527370
RMSE	0.27436053
MAPE	0.07621996

2. Decision Tree

Decision Tree is a supervised learning predictive model that uses a set of binary rules to calculate the target value/dependent variable. Decision trees are divided into three main parts this are:

- Root Node: performs the first split
- Terminal Nodes: that predict the outcome, these are also called leaf nodes
- Branches: arrows connecting nodes, showing the flow from root to other leaves.

In this project Decision tree is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.7257890718782061
RMSE	0.28025493351242536

Error metrics in R: using regr.eval function

MAE	0.19168951
MSE	0.07668177
RMSE	0.27691474
MAPE	0.08409804

3. Random Forest

The next model to be followed in this project is Random forest. It is a process where the machine follows an ensemble learning method for classification and regression that operates by developing a number of decision trees at training time and giving output as the class that is the mode of the classes of all the individual decision trees. For regression, it takes mean of output of all individual decision trees

In this project Random Forest is applied in both R and Python, details are described following.

Error Metrics in Python:

R- squared	0.7818258146860362
RMSE	0.24998420473121494

Error metrics in R:

MAE	0.21834016
MSE	0.09150787
RMSE	0.30250268
MAPE	0.09690444

As observed in above 3 models, random forest model has better performance. We will now tune the hyper parameters of Random Forest Model for more accuracy

4. Grid Search CV for Random Forest:

Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. GridSearchCV is a library function that is a member of sklearn's model_selection package. In addition to that, we can specify the number of times for the cross-validation for each set of hyperparameters.

Here we are giving below ranges for the parameters:

n_estimator = list(range(11,20,1))

depth = list(range(5,15,2))

And cross validation with 5 fold CV

And we got below parameters as best parameters:

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.81.
RMSE =  0.2312184435662826
```

As we can see random forest model performed better when max_depth = 7 and n_estimators= 19

Here, n_estimators: the number of trees in the forest,

max_depth: The maximum depth of the tree.

For R, we got better performance when paramters ntree = 15 and mtry= 2
Here, ntree: Number of trees to grow

mtry: Number of variables randomly sampled as candidates at each split.

Error metrics in R:

MAE	0.17583970
MSE	0.06578149
RMSE	0.25647903
MAPE	0.07780042

3.4 Model Selection

After comparison of the error matrix, the next step we come to is Selection of the most effective model. From the values of Error and accuracy, it is found that all the models perform close to each other. In this case any model can best used for further processes, but Random forest with hyper parameter tuning gives better results compared to all other methods. So, we will prefer Random Forest Model with hyper parameter tuning to be used for further processes.

CHAPTER 4: REFERENCES

Websites:

- www.edwisor.com : Videos from Mentor
- https://rpubs.com/ : Coding Doubts
- https://pypi.org/project/haversine/ : Haversine in python
- https://www.rdocumentation.org/packages/geosphere/versions/1.5-10/topics/distHaversine : Haversine in R
- https://www.r-bloggers.com/ : Doubts in R
- https://scikit-learn.org/ : Doubts in Python