

Contents

Sr. no		Topics
1		Introduction
	1.1	Problem Statements
	1.2	Understanding Data
2		Methodology Overview
	2.1	Data Preprocessing
	2.2	Data Modeling
	2.3	Evaluation Data
	2.4	Find predictable output
3		Data Preprocessing
	3.1	Exploratory Analytics
	3.2	Missing Value Analysis
	3.3	Outlier Analysis
	3.4	Some Exploratory Operations
	3.5	Data Visualization
	3.6	Feature Selection
	3.7	Feature Scaling
4		Data Modeling
	4.1	Linear Regression Modeling
	4.2	Decision Tree Regression
	4.3	Random Forest Regression
	4.4	Matrices to find accuracy
	4.5	Create predicted variable on test data
5		Conclusion
6		References

INTRODUCTION

In current state cab rental services are expanding in fast. This services are easy to use and flexible so customer get to use for that.

1.1 Problem Statement

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

1.2 Understanding Data

Libraray Use: Pandas

For understanding any business problem we have to first understand dataset we have given. So here we understanding our dataset features by following:

Size of trian_cab.csv : - 16067 rows, 7 Columns (including dependent variable)

Missing Values: Yes

Size of test_cab.csv : - 9914 rows, 6 Columns

Missing Values: No

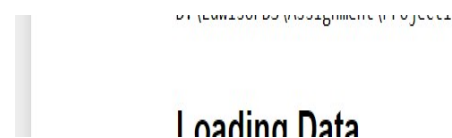
Variable list in Train and Test Dataset:

Variables	Data_Type	Train_cab	Test_cab
fare_amount	Object	No	Yes
pickup_datetime	Object	Yes	Yes

pickup_longitude	Float64	Yes	Yes
pickup_latitude	Float64	Yes	Yes
dropoff_longitude	Float64	Yes	Yes
dropoff_latitude	Float64	Yes	Yes
passenger_count	Float64\int64	Yes	Yes

Load Dataset:

Python



R

```

Train_Cab = read.csv("train_cab.csv")
Test_Cab = read.csv("test.csv")

```

Description of datasets:

```

In [11]: train_cab.describe()
Out[11]:

```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
count	16067.000000	16067.000000	16067.000000	16067.000000
mean	-72.462787	39.914725	-72.462328	39.891725
std	10.578384	6.826587	10.575062	6.181725
min	-74.438233	-74.006893	-74.429332	-74.006893
25%	-73.992156	40.734927	-73.991182	40.734927
50%	-73.981698	40.752603	-73.980172	40.752603
75%	-73.966838	40.767381	-73.963643	40.767381
max	40.766125	401.083332	40.802437	41.361111

```

In [12]: test_cab.describe()
Out[12]:

```

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
count	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751041

R

Check class of the data

```
class(Train_Cab)
```

#Check the dimensions(no of rows and no of columns)

```
dim(Train_Cab)
```

#Check top(first) rows of dataset

```
head(Train_Cab)
```

#Check structure of dataset(data structure of each variable)

```
str(Train_Cab)
```

#Check summary of dataset

```
summary(Train_Cab)
```

#Same for Test Dataset

METHODOLOY

2.1 Data Preprocessing

Here we required to built predictive model, therefore before moving towards modeling we have to manipulate data by applying multiple preprocessing techniques like follows:

- 1) **Exploratory analytics**
- 2) **Missing Value Analysis**
- 3) **Outlier Analysis**
- 4) **Feature Selection**
- 5) **Feature Scaling**
- 6) **Data Visualization**

2.2 Data Modeling

After Pre-Processing steps , we will move towards data modeling steps. Data modeling is a set of tools and techniques used to understand and analyse how an organization should collect, update, and store data. It is a critical skill for the business analyst who is involved with discovering, analyzing, and specifying changes to how software systems create and maintain information.

As our dataset in **continuous** format we will go with **regression modeling** techniques like:

- 1) **Linear Regression**
- 2) **Decision Tree Regression**
- 3) **Random Forest Regression**

2.3 Evaluation of Model

As our dataset tells about continuous variable and follows regression modeling , so for evaluate best fit model we have to find accuracy of model and for that we choose to go with regression matrices like:

- 1) **Root Mean Squared Error (RMSE)**
- 2) **Mean Absolute Error (MAE)**
- 3) **Mean Absolute Percentage Error (MAPE)**

From that we choose to simple follow MAPE.

2.4 Find Predictable output

After selection of best match model we will apply this model on test dataset and find predictable output of fare amount.

DATA PREPROCESSING

Data preprocessing is a **data** mining technique that involves transforming raw **data** into an understandable format. Real world **data** is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. **Data preprocessing** is a proven method of resolving such issues.

3.1 Data Exploration

Library use: pandas, numpy

Exploratory Data Analysis (EDA) is the first step in your data analysis process. This process makes deeper analysis easier because it can help target future searches and begin the process of excluding irrelevant data points and search paths that may turn up no results.

Steps we applied in our dataset:

- Convert our fare_amount variable from object to numeric data type.
train_cab['fare_amount'] =
pd.to_numeric(train_cab['fare_amount'], errors = "coerce")
- When we try to convert pickup_datetime variable to date format it was throwing an error because of strange value in the variable observation found which is of '43'. Therefore, first we removed this amount make that rows as null. Then we can drop null observation from that variable pickup_datetime.

Python code:

'43'. Therefore, first we removed this amount make that rows as null. Then we can drop null observation from that variable

```
In [15]: # when we tried convert pickup_datetime variable to date format it was throwing error coz of a st  
# So first treat it as NA and drop
```

R Code:

```
Train_Cab$fare_amount =  
as.numeric(as.character(Train_Cab$fare_amount))  
str(Train_Cab$fare_amount)
```

```
# We need to change pickup_datetime from factor to datetime  
# But first, let's replace UTC in pickup_datetime variable with  
"(space)
```

```
Train_Cab$pickup_datetime[Train_Cab$pickup_datetime== '43'  
]= NA  
Missing_val = sum(is.na(Train_Cab$pickup_datetime))  
Missing_val  
#na.omit(Train_Cab$pickup_datetime)
```

- pickup_datetime variable is in object so we need to change its data type to datetime, then we separate this variable by year, months, date ,day, hour and minute.

Here pickup_datetime variable is in object so we need to change its data type to datetime

```
In [16]: # Here pickup_datetime variable is in object so we need to change its data type to datetime
train_cab['pickup_datetime'] = pd.to_datetime(train_cab['pickup_datetime'], format='%Y-%m-%d %H:%M:%S')

In [17]: train_cab['pickup_datetime'].dtypes
Out[17]: dtype('<M8[ns]')

In [18]: ### we will saperate the Pickup_datetime column into separate field like year, month, day of the week
train_cab['year'] = train_cab['pickup_datetime'].dt.year
train_cab['Month'] = train_cab['pickup_datetime'].dt.month
train_cab['Date'] = train_cab['pickup_datetime'].dt.day
train_cab['Day'] = train_cab['pickup_datetime'].dt.dayofweek
train_cab['Hour'] = train_cab['pickup_datetime'].dt.hour
train_cab['Minute'] = train_cab['pickup_datetime'].dt.minute

In [19]: train_cab.dtypes #Re-checking datatypes after conversion
Out[19]: fare_amount          float64
pickup_datetime      datetime64[ns]
pickup_longitude      float64
pickup_latitude       float64
passenger_count        int64
trip_distance          float64
tip_amount             float64
total_amount           float64
trip_start_timestamp    datetime64[ns]
trip_end_timestamp      datetime64[ns]
```

R Code:

```
Train_Cab$pickup_datetime = gsub('/',
UTC',",Train_Cab$pickup_datetime)
```

```
##### For Train dataset #####
```

```
# Now convert variable pickup_datetime to date time format by creating
```

```
# new variable with name Date
```

```
Train_Cab$date = as.Date(Train_Cab$pickup_datetime)
```

```
# Lets split this new variable Date into year,month,weekday
```

```
# Extract the year
```

```
Train_Cab$year = substr(as.character(Train_Cab$date),1,4)
```

```
# Extract the month
```

```
Train_Cab$month =substr(as.character(Train_Cab$date),6,7)
```

```
# Extract the weekday

Train_Cab$day = weekdays(as.POSIXct(Train_Cab$date),abbreviate =
F)

# Extract the date

Train_Cab$date = substr(as.character(Train_Cab$date),9,10)

# Extract the time / hour

Train_Cab$hour = substr(as.factor(Train_Cab$pickup_datetime),12,13)
str(Train_Cab)

#Recheck dimensions of test and train data

dim(Train_Cab)

####Same for test dataset####
```

3.2 Missing Value Analysis

Python Library use: pandas, numpy

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

In our dataset we found an missing values in train_cab data not in test_cab data.

```

In [23]: #lets see missing values in our train data
missing_val = train_cab.isnull().sum()
missing_val

Out[23]: fare_amount      25
pickup_datetime      0
pickup_longitude      0
pickup_latitude      0
dropoff_longitude      0
dropoff_latitude      0
passenger_count     55
year      0
Month      0
Date      0
Day      0
Hour      0
Minute      0
dtype: int64

In [24]: #lets see missing values in our test data
missing_val = test_cab.isnull().sum()
missing_val

Out[24]: pickup_datetime      0
pickup_longitude      0

```

Now we seen that there are missing data available in train dataset so now find how much percentage of data is missing.

```

dtype: int64

There are no missing values in test data

In [25]: ((train_cab.isnull().sum())/len(train_cab))*100 #here we are finding percentage of missing value

Out[25]: fare_amount      0.155608
pickup_datetime      0.000000
pickup_longitude      0.000000
pickup_latitude      0.000000
dropoff_longitude      0.000000
dropoff_latitude      0.000000
passenger_count      0.342338
year      0.000000
Month      0.000000
Date      0.000000
Day      0.000000
Hour      0.000000
Minute      0.000000

```

So here we can observed that the missing data is not more than 35% of all observed data , so we can direct remove it from our data. We found missing data in fare_amount and passenger_count so we now going to remove it from our data.

```

In [26]: #first we removing passenger_count missing values rows
train_cab = train_cab.drop(train_cab[train_cab['passenger_count'].isnull()].index, axis=0)
print(train_cab.shape)
print(train_cab['passenger_count'].isnull().sum())

(16011, 13)
0

In [27]: #second we will removing fare_amount missing values
# eliminating rows for which value of "fare_amount" is missing
train_cab = train_cab.drop(train_cab[train_cab['fare_amount'].isnull()].index, axis=0)
print(train_cab.shape)
print(train_cab['fare_amount'].isnull().sum())

(15986, 13)
0

```

R Code:

```

# Total number of missing values present in whole dataset

Missing_val = sum(is.na(Train_Cab))

Missing_val #86

#here we are removing all missing values from train datasets

Train_Cab = na.omit(Train_Cab)

Missing_val = sum(is.na(Train_Cab))

Missing_val

#lets check the dimension after removing missing values

dim(Train_Cab)

```

3.3 Outlier Analysis

Python Library use: pandas, numpy, collections

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

For variable : passenger_Count

```
For passenger_Count variable

In [28]: train_cab["passenger_count"].describe()
Out[28]: count    15986.000000
         mean      2.623272
         std       60.892140
         min       0.000000
```

R code:

```
summary(Train_Cab$passenger_count)
```

When we got detail of passenger data we found that minimum number of passenger is 0 and maximum number of passenger is 5345 which not logically accepted. In a natural way cabs are taking atleast 1 passenger to ride and maximum 6.

So, as per manual observation we can say that above 6 and below 1 there would be outlier. So, we make that observation 'NaN' and then can remove it.

```
So we manually remove this outlier from passenger variable by considering maximum

In [29]: #here we get count of passenger which is less than 1 and greater than 7
print(Counter(train_cab['passenger_count'] < 1))
print(Counter(train_cab['passenger_count'] > 6))

Counter({False: 15928, True: 58})
Counter({False: 15967, True: 19})

In [30]: #Let's check same for the test data
print(Counter(test_cab['passenger_count'] < 1))
print(Counter(test_cab['passenger_count'] > 6))

#No outlier found

Counter({False: 9914})
Counter({False: 9914})

In [31]: #here passenger count has totally 77 outliers where 58 variables are 0 and
#there is no use of having these data. hence we drop it
train_cab = train_cab.drop(train_cab[train_cab['passenger_count'] < 1 | train_cab['passenger_count'] > 7].index)
```

R code:

```
Train_Cab[Train_Cab$passenger_count < 1, "passenger_count"] =
NA
```

```
#Train_Cab[Train_Cab$passenger_count > 6,"passenger_count"]  
<- NA
```

When I applying above statements it was giving me error

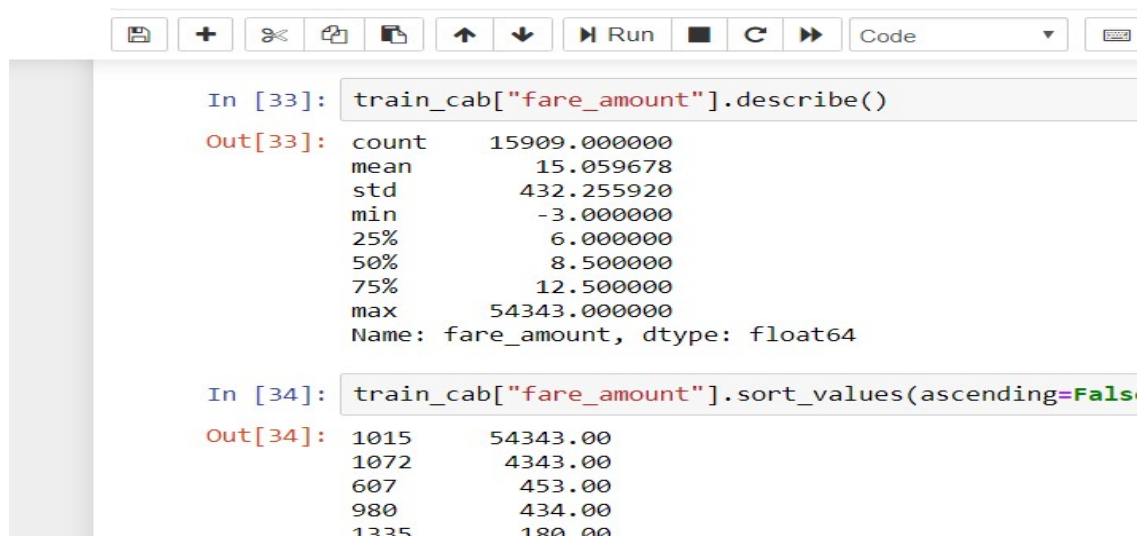
#ERROR: missing values are not allowed in subscripted
assignments of data frames

#so we go by below solution

```
Train_Cab$passenger_count<- ifelse(Train_Cab$passenger_count  
> 6, NA,Train_Cab$passenger_count )
```

For variable: fare_amount

Python code:



The screenshot shows a Jupyter Notebook interface with a toolbar at the top containing icons for file operations, navigation, and execution. The main area displays two code cells. The first cell, labeled 'In [33]:', contains the code `train_cab["fare_amount"].describe()`. The output, labeled 'Out[33]:', shows a statistical summary for the 'fare_amount' variable, including count, mean, standard deviation, minimum, and various percentiles. The second cell, labeled 'In [34]:', contains the code `train_cab["fare_amount"].sort_values(ascending=False)`. The output, labeled 'Out[34]:', shows the top five highest fare amounts in descending order.

```
In [33]: train_cab["fare_amount"].describe()  
Out[33]: count      15909.000000  
         mean         15.059678  
         std         432.255920  
         min          -3.000000  
         25%           6.000000  
         50%           8.500000  
         75%          12.500000  
         max         54343.000000  
         Name: fare_amount, dtype: float64  
  
In [34]: train_cab["fare_amount"].sort_values(ascending=False)  
Out[34]: 1015      54343.00  
         1072      4343.00  
         607       453.00  
         980       434.00  
         1225       180.00
```

R code:

```
summary(Train_Cab$fare_amount)
```

```
sort(Train_Cab$fare_amount,decreasing = TRUE)
```

#In order of fare amount we can see that there is a huge difference
in 1st 2nd and 3rd position

#so we will remove the rows having fare amounting more than 454 as considering them as outliers

#And also, from the above description, we can say that the fare amount for a cab cannot be in negative

#as well as it cannot exceed 500 above and below the values will be considered as outliers and drop it.

```
Train_Cab$fare_amount[Train_Cab$fare_amount < 1] = NA
```

```
Train_Cab$fare_amount[Train_Cab$fare_amount > 454] = NA
```

```
summary(Train_Cab$fare_amount)
```

#removing all nan values after overcoming outliers of fare_amount and passenger_count

```
sum(is.na(Train_Cab)) #77+
```

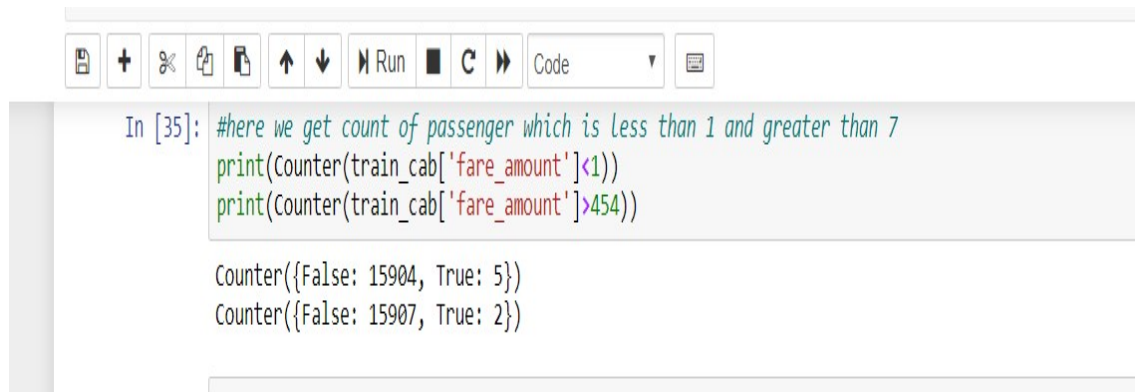
```
Train_Cab = na.omit(Train_Cab)
```

#recheck dimension of Train dataset

```
dim(Train_Cab)
```

When we observed detail of fare amount and when we sort a values in descending order we found that first 3 observations have huge difference in between and after **3rd observation(453)** it's going somehow in good range. So here we observed third observation as maximum, and remove all above that observation considering **outer outlier**.

Similarly, there is obvious situation where fare amount could not be less than or equal to 0. Therefore here we can say amount should not be **minimum 1** so below 1 we can conclude as **inner outlier**.



```
In [35]: #here we get count of passenger which is less than 1 and greater than 7
print(Counter(train_cab['fare_amount']<1))
print(Counter(train_cab['fare_amount']>454))

Counter({False: 15904, True: 5})
Counter({False: 15907, True: 2})
```

For variable: pickup_latitude, pickup_latitude and dropoff_longitude , dropoff_longitude

Latitude lines run east-west and are parallel to each other. If you go further north, latitude values increase. Finally, latitude values (Y-values) range between -90 and +90 degrees

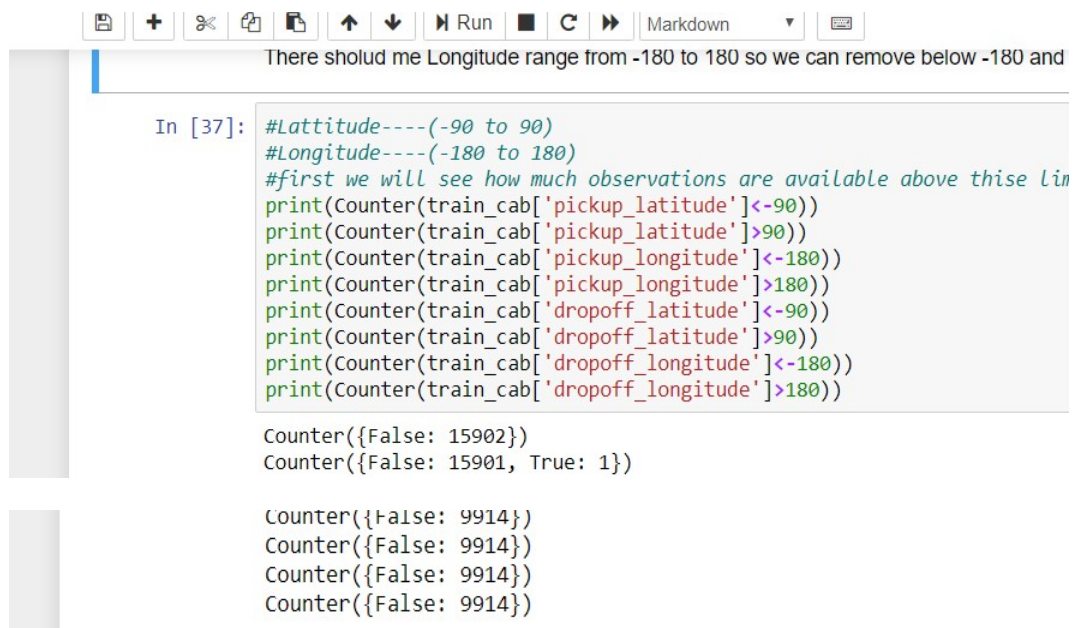
But **longitude** lines run north-south. They converge at the poles. And its X-coordinates are between -180 and +180 degrees.

Latitude and longitude coordinates make up our geographic coordinate system.

So, here we can say that for pickup_latitude and dropoff_latitude range should be between -90 and +90 i.e below -90 we can consider as inner outlier and +90 will be outer outlier. Similarly, for pickup_longitude and dropoff_longitude in range between -180 to +180.

So here we will remove observation manually by considering general fact about latitude and longitude.

Python Code:



```
There shold me Longitude range from -180 to 180 so we can remove below -180 and

In [37]: #Latitude---(-90 to 90)
#Longitude---(-180 to 180)
#first we will see how much observations are available above thise lin
print(Counter(train_cab['pickup_latitude']<-90))
print(Counter(train_cab['pickup_latitude']>90))
print(Counter(train_cab['pickup_longitude']<-180))
print(Counter(train_cab['pickup_longitude']>180))
print(Counter(train_cab['dropoff_latitude']<-90))
print(Counter(train_cab['dropoff_latitude']>90))
print(Counter(train_cab['dropoff_longitude']<-180))
print(Counter(train_cab['dropoff_longitude']>180))

Counter({False: 15902})
Counter({False: 15901, True: 1})

Counter({False: 9914})
Counter({False: 9914})
Counter({False: 9914})
Counter({False: 9914})
```

R code:

```
Train_Cab[Train_Cab$pickup_longitude < -180  
,"pickup_longitude"] = NA
```

```
Train_Cab[Train_Cab$pickup_longitude > 180  
,"pickup_longitude"] = NA
```

```
sum(is.na(Train_Cab))
```

```
Train_Cab[Train_Cab$pickup_latitude < -90,"pickup_latitude"] =  
NA
```

```
Train_Cab[Train_Cab$pickup_latitude > 90,"pickup_latitude"] =  
NA
```

```
sum(is.na(Train_Cab))#1
```

```

Train_Cab[Train_Cab$dropoff_longitude < -
180,"dropoff_longitude"] = NA

Train_Cab[Train_Cab$dropoff_longitude >
180,"dropoff_longitude"] = NA

sum(is.na(Train_Cab))# 1 found

Train_Cab$dropoff_latitude<- ifelse(Train_Cab$dropoff_latitude
< -90, NA,Train_Cab$dropoff_latitude )

Train_Cab$dropoff_latitude<- ifelse(Train_Cab$dropoff_latitude
> 90, NA,Train_Cab$dropoff_latitude )

sum(is.na(Train_Cab))#1 found

Train_Cab = na.omit(Train_Cab)

sum(is.na(Train_Cab))

```

3.4 Some Exploratory Operation

Python Library use: numpy, pandas, collections, math

R library use: purrr , geosphere, rlist

Before applying further future selection process, we will do some operations on our data which will help to find minimum and best match variable for our further process.

Here we choose to apply haversine algorithm to find distance by considering variable pickup_latitude , pickup_longitude, dropoff_latitude and dropoff_longitude.

Haversine formula

The **haversine formula** determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the **law of haversines**, that relates the sides and angles of spherical triangles.

The ***haversine formula*** allows the haversine of Θ (that is, $\text{hav}(\theta)$) to be computed directly from the latitude and longitude of the two points:

$$\text{hav}(\theta) = \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)$$

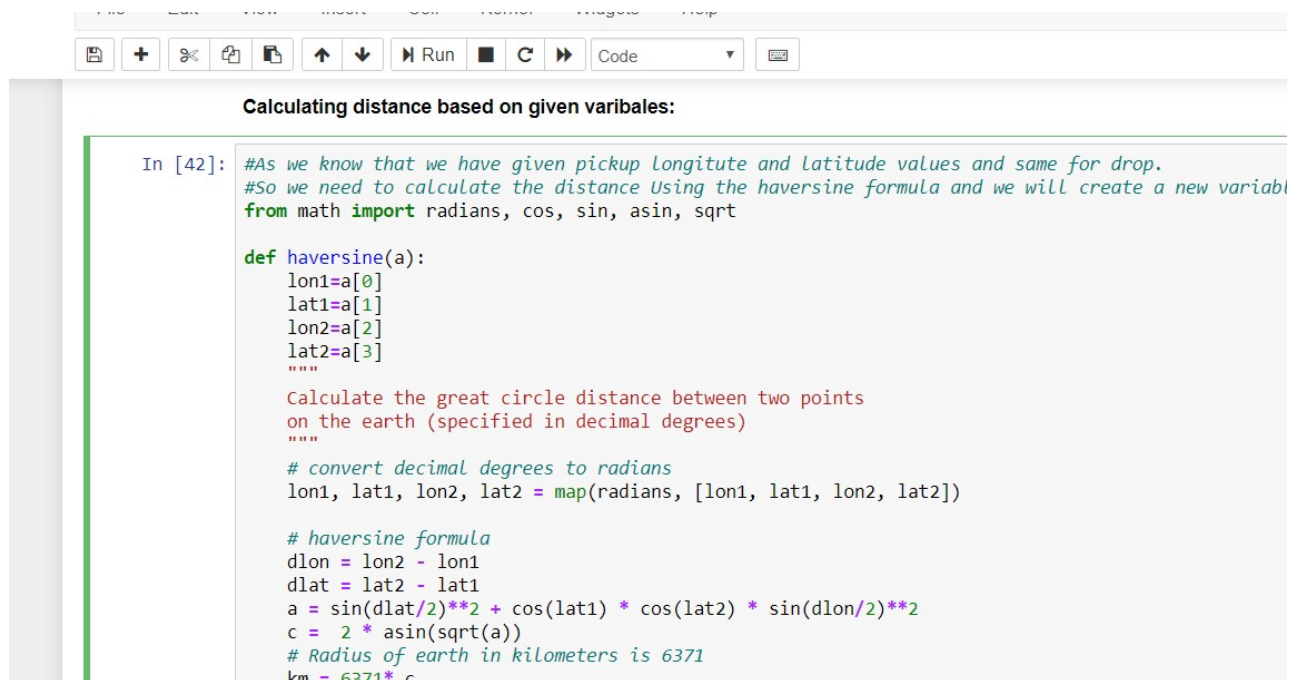
where

- φ_1, φ_2 : latitude of point 1 and latitude of point 2 (in radians),
- λ_1, λ_2 : longitude of point 1 and longitude of point 2 (in radians).

Finally, the haversine function $\text{hav}(\theta)$ applied above to both the central angle Θ and the differences in latitude and longitude, is

$$\text{hav}(\theta) = \sin^2(\theta/2) = (1 - \cos(\theta)) / 2$$

Python code:



The screenshot shows a Jupyter Notebook window with a toolbar at the top containing icons for saving, adding, deleting, and running code. Below the toolbar, the notebook title is "Calculating distance based on given variables:". The code cell contains the following Python code:

```
In [42]: #As we know that we have given pickup longitude and latitude values and same for drop.
#So we need to calculate the distance Using the haversine formula and we will create a new variable
from math import radians, cos, sin, asin, sqrt

def haversine(a):
    lon1=a[0]
    lat1=a[1]
    lon2=a[2]
    lat2=a[3]
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371 * c
```

R code:

```
get_geo_distance = function(long1, lat1, long2, lat2) {
  loadNamespace("purrr")
  loadNamespace("geosphere")
  longlat1 = purrr::map2(long1, lat1, function(x,y) c(x,y))
  longlat2 = purrr::map2(long2, lat2, function(x,y) c(x,y))
  distance_list = purrr::map2(longlat1, longlat2, function(x,y)
    geosphere::distHaversine(x, y))
  distance_m = list.extract(distance_list, position = 1)
  #if (units == "km") {
    distance = distance_m / 1000.0;
```

```

distance

}

# Applying distance formula for train data

for(i in (1:nrow(Train_Cab)))

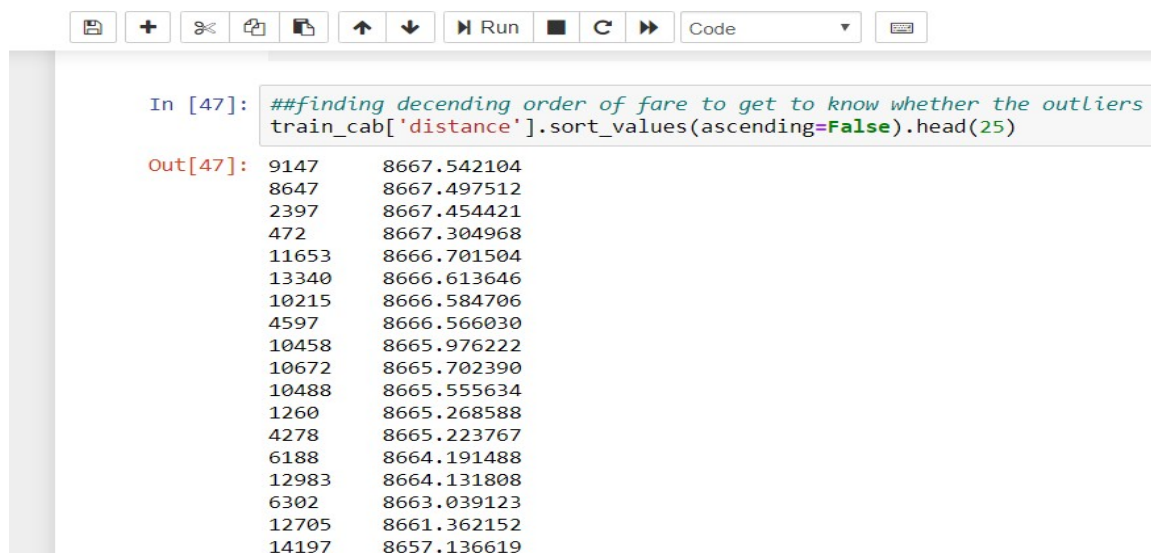
{

  Train_Cab$distance[i]=
  get_geo_distance(Train_Cab$pickup_longitude[i],Train_Cab$pickup_latitude[i],Train_Cab$dropoff_longitude[i],Train_Cab$dropoff_latitude[i])

}

```

Now, in our dataset new variable created named 'Distance'. Now we will just check whether there any outlier is there are not. So for performing outlier analysis we first see descending order of distance and also verify there detailed description.



The screenshot shows a Jupyter Notebook interface. At the top, there is a toolbar with icons for saving, adding, undo, redo, and running code. Below the toolbar, there is a code cell with the following text:

```
In [47]: ##finding decending order of fare to get to know whether the outliers
train_cab['distance'].sort_values(ascending=False).head(25)
```

Below the code cell, the output is displayed. It shows the first 25 rows of the 'distance' column from the 'train_cab' dataset, sorted in descending order. The output is presented as a table with two columns: the first column contains the index of the row, and the second column contains the distance value.

9147	8667.542104
8647	8667.497512
2397	8667.454421
472	8667.304968
11653	8666.701504
13340	8666.613646
10215	8666.584706
4597	8666.566030
10458	8665.976222
10672	8665.702390
10488	8665.555634
1260	8665.268588
4278	8665.223767
6188	8664.191488
12983	8664.131808
6302	8663.039123
12705	8661.362152
14197	8657.136619

R code:

Lets check whether distance variables has any outlier

```
sort(Train_Cab$distance,decreasing = TRUE)
```

```
2200      5420.988959
5864      4447.086698
7014      129.950482
10710     129.560455
14536
Name: distance, dtype: float64

In [48]: train_cab['distance'].describe()
Out[48]: count    15901.000000
         mean      15.071717
```

So as per observing a order of descending we seen that above 23 orbservations are in quite same range but after 23rd observation its drastically down to 129, here we can conclude that first 23 observation which is above 129 create outer outlier in this case.

And as per considering description of distance we found minimum value is 0 which is not logically valid for distance.

So here we remove **Above 130** observation considering **outer outlier** and **0** as **inner outlier**.

```

# Now we can see that top 20 values in the distance variables are very high, no more
# the top, the distance goes down to 129, which means these values are showing :

In [49]: #For train dataset
print(Counter(train_cab['distance']==0))
print(Counter(train_cab['distance']>130))

Counter({False: 15447, True: 454})
Counter({False: 15878, True: 23})

In [50]: #same for test dataset
print(Counter(test_cab['distance']==0))
print(Counter(test_cab['distance']>130))

Counter({False: 9829, True: 85})
Counter({False: 9914})

In [51]: #Now we will remove the rows whose distance value is zero and more
#For training dataset
train_cab = train_cab.drop(train_cab[train_cab['distance']== 0],i
train_cab = train_cab.drop(train_cab[train_cab['distance'] > 130
train_cab.shape
```

R code:

```
Train_Cab[Train_Cab$distance == 0,'distance'] <- NA
```

```
Train_Cab$distance<- ifelse(Train_Cab$distance > 130,  
NA,Train_Cab$distance )
```

```
sum(is.na(Train_Cab))# 478 found
```

```
Train_Cab = na.omit(Train_Cab)
```

Now our data after applying above preprocessing techniques:

```
In [53]: train_cab.head()
```

```
Out[53]:
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	Month	Date	Day	Hour	Minu
0	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.712278	1.0	2009	6	15	0	17	2
1	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.782004	1.0	2010	1	5	1	16	!
2	5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	40.750562	2.0	2011	8	18	3	0	!
3	7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	40.758092	1.0	2012	4	21	5	4	!
4	5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	40.783762	1.0	2010	3	9	1	7	!

```
In [54]: test_cab.head()
```

```
Out[54]:
```

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	Month	Date	Day	Hour	Minute	distance
0	2015-01-27 13:08:24	-73.973320	40.763805	-73.981430	40.743835	1	2015	1	27	1	13	8	2.323259
1	2015-01-27 13:08:24	-73.986862	40.719383	-73.998886	40.739201	1	2015	1	27	1	13	8	2.425353
2	2011-10-08 11:53:44	-73.982524	40.751260	-73.979654	40.746139	1	2011	10	8	5	11	53	0.618628
3	2012-12-01 21:12:12	-73.981160	40.767807	-73.990448	40.751635	1	2012	12	1	5	21	12	1.961033
4	2012-12-01 21:12:12	-73.966046	40.789775	-73.988565	40.744427	1	2012	12	1	5	21	12	5.387301

3.5 Data visualization operation on variable and with there relationship

We are observing one by one variable visualization and try to pass conclusion on that.

Libraray use: Seaborn and Matplot.

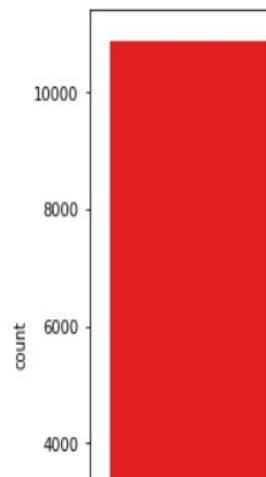
1] for variable passenger_count



Observation on Passenger_Count variable

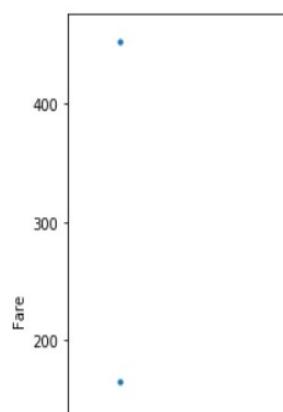
```
In [65]: # Count plot on passenger count
plt.figure(figsize=(15,7))
sns.countplot(x="passenger_count", data=train_cab, color='red')
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb200d4e48>



1) Number of passenger affect the fare

```
In [66]: #Relationship between number of passengers and Fare
plt.figure(figsize=(15,7))
plt.scatter(x=train_cab['passenger_count'], y=train_cab['fare_amount'], s=10)
plt.xlabel('No. of Passengers')
plt.ylabel('Fare')
plt.show()
```



R code:

Visualization between fare_amount and passenger count.

```
ggplot(data = Train_Cab, aes(x = passenger_count,y = fare_amount))+
  geom_bar(stat = "identity",color ="green")+
  labs(title = "Fare Amount Vs. year", x = "Passenger Count", y =
"Fare")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="green", size=6, angle=45))
```

Here we are observing number of passenger with fare amount.

By observing about visualization for varibale passanger_count we can conclude that:

1. single passanger have most hight frequency
2. also we can see single and double passanger gives us high fare amount.

2] Relationship Between Date and Fare

2] Relationship between Date and Fare

```
In [67]: plt.figure(figsize=(15,7))
```

R code:

Visualization between fare_amount and date

```
ggplot(data = Train_Cab, aes(x = date= fare_amount))+
  geom_bar(stat = "identity",color = "yellow")+
  labs(title = "Fare Amount Vs.hour", x = "hour", y = "Fare")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
```

```
theme(axis.text.x = element_text( color="black", size=6, angle=45))
```

```
plt.scatter(x=train_cab['Date'], y=train_cab['fare_amount'], s=10)  
plt.xlabel('Date')  
plt.ylabel('Fare')  
plt.show()
```



Oberseved that:

On beginning oof month or between date 0-5 our fare amount is then its goes down and fluctuating up-high but aroud 23-25 again its go high.

3] Lets observed Time variable

R code:

Visualization between fare_amount and time.

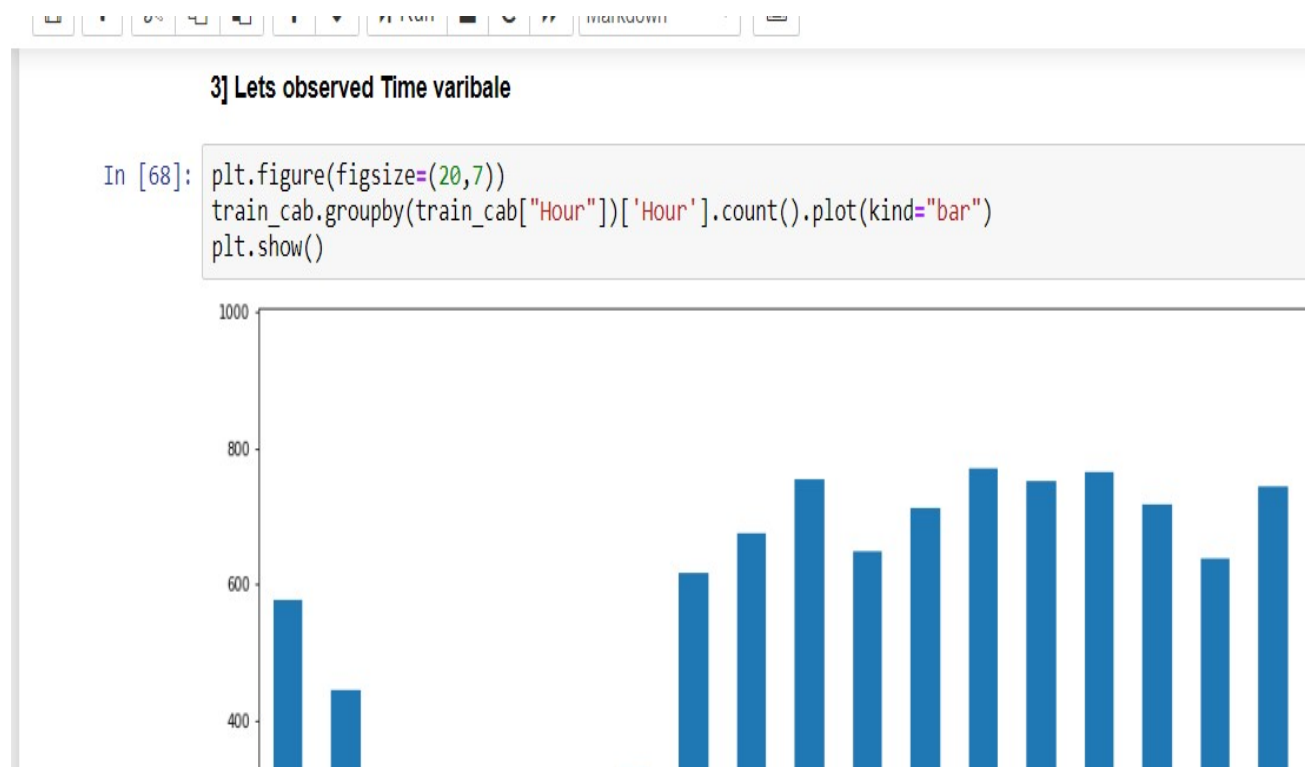
```
ggplot(data = Train_Cab, aes(x = hour, y = fare_amount))+
```

```
  geom_bar(stat = "identity",color = "yellow")+
```

```
  labs(title = "Fare Amount Vs.hour", x = "hour", y = "Fare")+
```

```
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
```

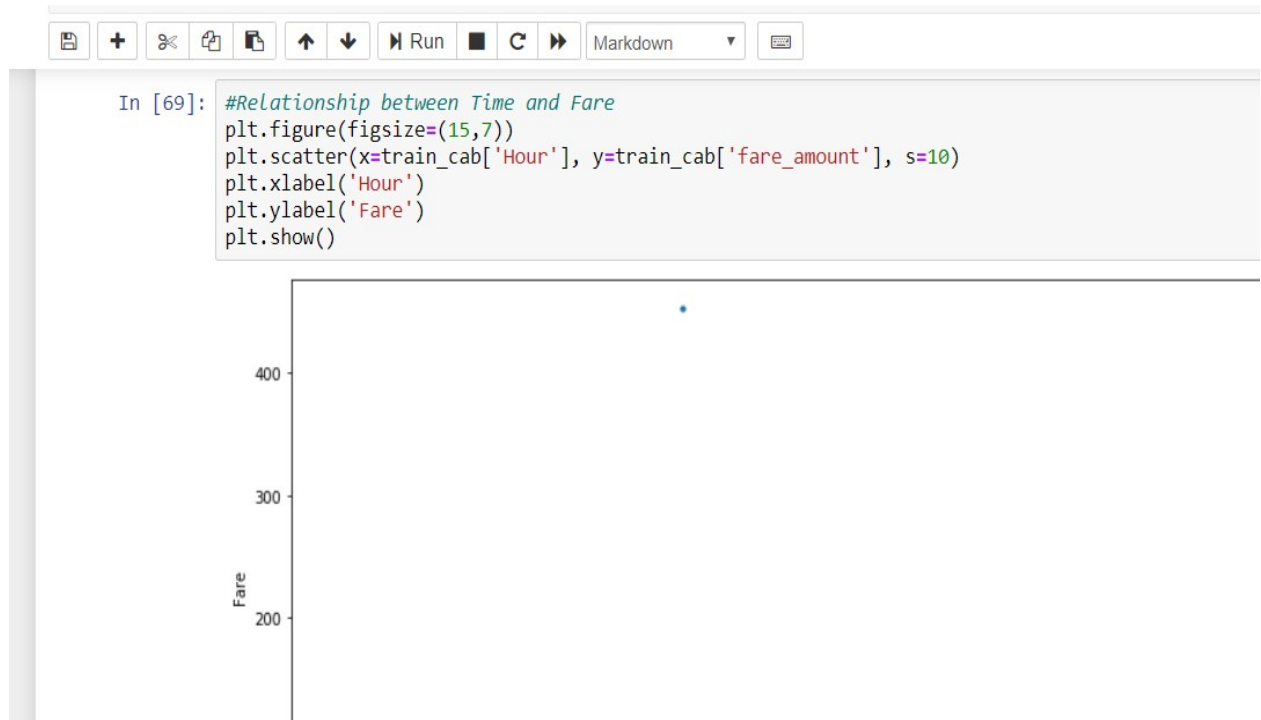
```
  theme(axis.text.x = element_text( color="black", size=6, angle=45))
```



Observed:

While morning around 7 count going high and start fluctuating till 17 i.e ti evening 5pm as this office hours and after 17 office rush again start to travelling back to there home, so this time after 5 effect lot i.e it going high.

Relationship between Time and Fare



R code:

Visualization between fare_amount and time.

```
ggplot(data = Train_Cab, aes(x = hour, y = fare_amount))+
  geom_bar(stat = "identity",color = "yellow")+
  labs(title = "Fare Amount Vs.hour", x = "hour", y = "Fare")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme(axis.text.x = element_text( color="black", size=6, angle=45))
```

Obseravtion we can colclude by considering time:

From the above plot We can say that the cabs taken at 7am and 23Pm effecting on fare highly . Hence we can assume that cabs taken early in morning and late at night are costliest.

4] Observing Day Variable

4] Observing day variable

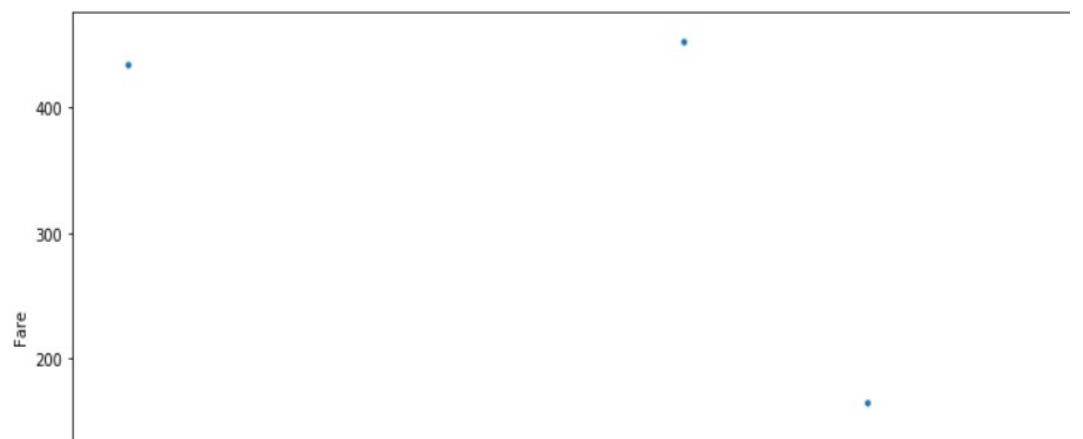
```
In [70]: #impact of Day on the number of cab rides
plt.figure(figsize=(15,7))
sns.countplot(x="Day", data=train_cab)
```

```
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1cb21471208>
```



Relationships between day and Fare

```
In [71]: #Relationships between day and Fare
plt.figure(figsize=(15,7))
plt.scatter(x=train_cab['Day'], y=train_cab['fare_amount'], s=10)
plt.xlabel('Day')
plt.ylabel('Fare')
plt.show()
```



R code:

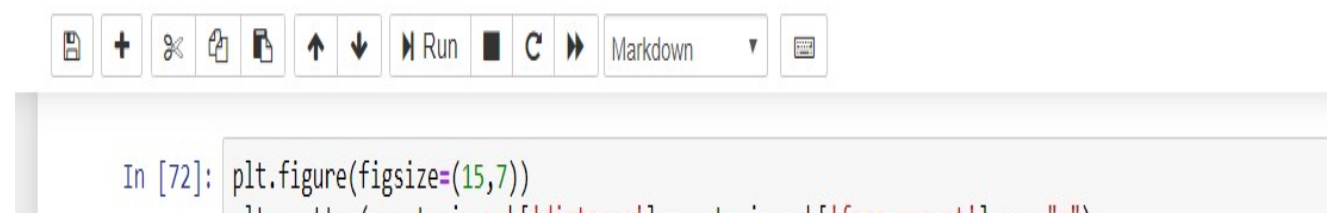
```
# Visualization between fare_amount and day.

ggplot(data = Train_Cab, aes(x = day, y = fare_amount))+
  geom_bar(stat = "identity", color = "green")+
  labs(title = "Fare Amount Vs. weekday", 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))
```

Observation:

The highest fares seem to be on a Sunday, Monday and Thursday, and the low on Wednesday and Saturday. May be due to low demand of the cabs on saturday the cab fare is low and high demand of cabs on sunday and monday shows the high fare prices.

5] Relationship between distance and fare



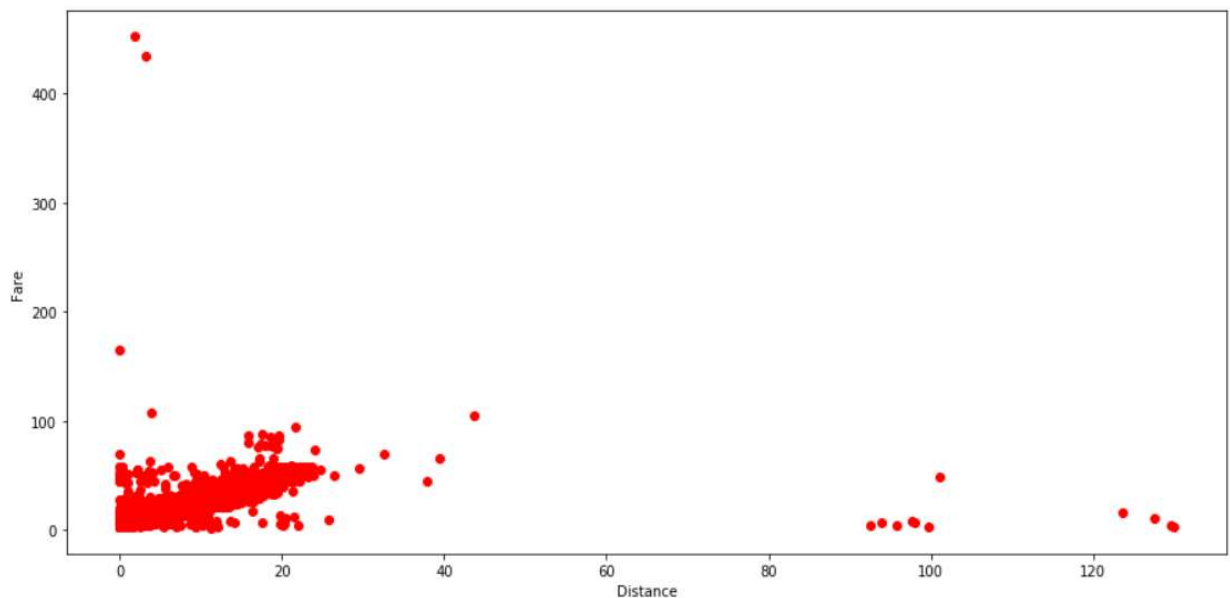
R code:

```
# Visualization between fare_amount and distance.

ggplot(data = Train_Cab, aes(x = distance, y = fare_amount))+
  geom_bar(stat = "identity", color = "yellow")+
  labs(title = "Fare Amount Vs. distance", x = "Distance", y = "Fare")+

```

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



Observation:

Distance always affect Fare. As distance increases fare also inscreses.

3.6 Feature Selection

Library use: numpy, pandas, seaborn, statsmodels

In machine learning and statistics, **feature selection**, also known as **variable selection**, **attribute selection** or **variable subset selection**, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons:

- simplification of models to make them easier to interpret by researchers/users,
- shorter training times,

- to avoid the curse of dimensionality,
- enhanced generalization by reducing overfitting (formally, reduction of variance)

Here in our project we remove some unwanted variable from our dataset like:

1. We have splitted the pickup_datetime variable into different variables like month, year, day etc so now we dont need to have that pickup_Date variable now. Hence we can drop that,
2. Also we have created distance variable using pickup and drop longitudes and latitudes so we will also drop pickup and drop longitudes and latitudes variables.



For train dataset

```
In [55]: drop = ['pickup_datetime', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude']
train_cab = train_cab.drop(drop, axis = 1)
```

```
In [56]: train_cab.head()
```

Out[56]:

	fare_amount	passenger_count	year	Month	Date	Day	Hour	distance
0	4.5	1.0	2009	6	15	0	17	1.030764
1	16.9	1.0	2010	1	5	1	16	8.450134
2	5.7	2.0	2011	8	18	3	0	1.389525
3	7.7	1.0	2012	4	21	5	4	2.799270
4	5.3	1.0	2010	3	9	1	7	1.999157

```
In [57]: train_cab.shape
```

Out[57]: (15424, 8)

```
In [58]: train_cab.dtypes
```

R Code:

```
Train_Cab[,c('pickup_datetime',  
'pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude')] <- list(NULL)
```

After Feature selection we observed that our shape of train data got reduced to 15424 columns and 8 rows. This optimized data we can use to preprocess further.

In our train dataset we find that passenger_count variable is in float64 format while in test dataset it's in int64 format so we will change train passenger_count variable into int64 format.

```
Hour          int64  
distance      float64  
dtype: object  
  
In [63]: #As in test dataset passenger_count data type is int64 and in train dataset we found passenger_count  
#so for avoiding any on mismatch further operations we convert it into int64  
train_cab['passenger_count'] = train_cab['passenger_count'].astype('int64')  
  
In [64]: train_cab.dtypes  
  
Out[64]: fare_amount      float64  
passenger_count      int64  
year                 int64
```

For further feature selection process we choose to with

1. Correlation Analysis
2. ANOVA testing

1. Correlation analysis

Correlation analysis is a method of statistical evaluation used to study the strength of a relationship between two, numerically measured, continuous variables (e.g. height and weight). This particular type of analysis is useful when a researcher wants to establish if there are **possible connections** between variables.

If correlation is found between two variables it means that when there is a systematic change in one variable, there is also a systematic change in the other; the variables alter together over a certain period of time. If there is correlation found, depending upon the numerical values measured, this can be either **positive** or **negative**.

Correlation Coefficient Formula

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

When a correlation of two variable is highly positive or neative correlated we remove this variable from our dataset only if this variable not create any impact our final model development process.



```
In [75]: ##Correlation analysis
#Correlation plot

df_corr = train_cab.loc[:,cnames]
```

```
In [76]: #Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
corr = df_corr.corr()
```



R code:

#CORRELATION ANALYSIS

```
numeric_index = sapply(Train_Cab,is.numeric)# Selecting only
numeric
```

```
numeric_index #fare_amount, passenger_count, distance
```

```
numeric_data =Train_Cab[,numeric_index]
```

```
cnames = colnames(numeric_data)
```

```
cnames
```

```
# Correlation Plot for to select significant continous variables
```

```
#correlation matrix
```

```
correlation_matrix = cor(Train_Cab[,cnames])
```

```
correlation_matrix
```

```
library(corrgram)
```

```
#correlation plot
```

```
corrgram(Train_Cab[,numeric_index],order = F,upper.panel =
panel.pie,
```

```
text.panel = panel.txt,main = 'Correlation plot')
```

In our train dataset we observed that the highly negatively correlated data 'passenger_count' and 'distance' are important for further model development process, so we considered it as less correlated and not drop from our process.

2. ANOVA testing

Analysis of variance (ANOVA) is a collection of statistical models and their associated estimation procedures (such as the "variation" among and between groups) used to analyze the differences among group means in a sample. The ANOVA is based on the law of total variance, where the observed variance in a particular variable is partitioned into components attributable to different sources of variation. In its simplest form, ANOVA provides a statistical test of whether two or more population means are equal, and therefore generalizes the *t*-test beyond two means.

In our case there are five categorical variable: 'year', 'Month', 'Date', 'Day', 'Hour'

Code

```
In [77]: # Anova Test is performed between catnames (categorical independent variables) & fare_amount(cont

import statsmodels.api as sm

from statsmodels.formula.api import ols

for i in catnames:
    mod = ols('fare_amount' + '~' + i, data = train_cab).fit()
    aov_table = sm.stats.anova_lm(mod, typ = 2)
    print(aov_table)

# From the anova result, we can observe Date ,weekday

# has p value > 0.05, so delete these variables not consider in model.
```

	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)

As we are considering p-value as 0.05. Therefore if variable of p-value greater than 0.05 then that variable we will remove.

So, here we will remove 'day' and 'date' variable from our dataset.

	sum_sq	df	F	PR(>F)
Hour	1.203834e+03	1.0	10.571838	0.001151
Residual	1.756130e+06	15422.0	NaN	NaN

```
In [78]: # After correlation and ANOVA test we need to remove these variables
# Cat variables day and date(which has p-value>0.05)
```

```
train_cab = train_cab.drop('Date', axis = 1)
train_cab = train_cab.drop('Day', axis = 1)
```

```
In [79]: # Lets cross check shape of dataset after feature selection
train_cab.shape
```

R code:

```
cat_var = c("date", "year", "month", "day", "hour")
```

```
# aov(Train_Cab$fare_amount~Train_Cab$year)
```

```
# for all categorical variables
```

```
for(i in cat_var){
  print(i)
```

```
  Anova_test_result = summary(aov(formula =
fare_amount~Train_Cab[,i], Train_Cab))
```

```
print(Anova_test_result)
}
names(Train_Cab)
# From the anova result, we can observe Date and day
# has p value > 0.05, so delete this variable not consider in model.
# lets delete date and day variable
Train_Cab$day = NULL
Train_Cab$date = NULL
```

3.7 Feature Scalling

Python Library use: numpy, seaborn, Matplotlib

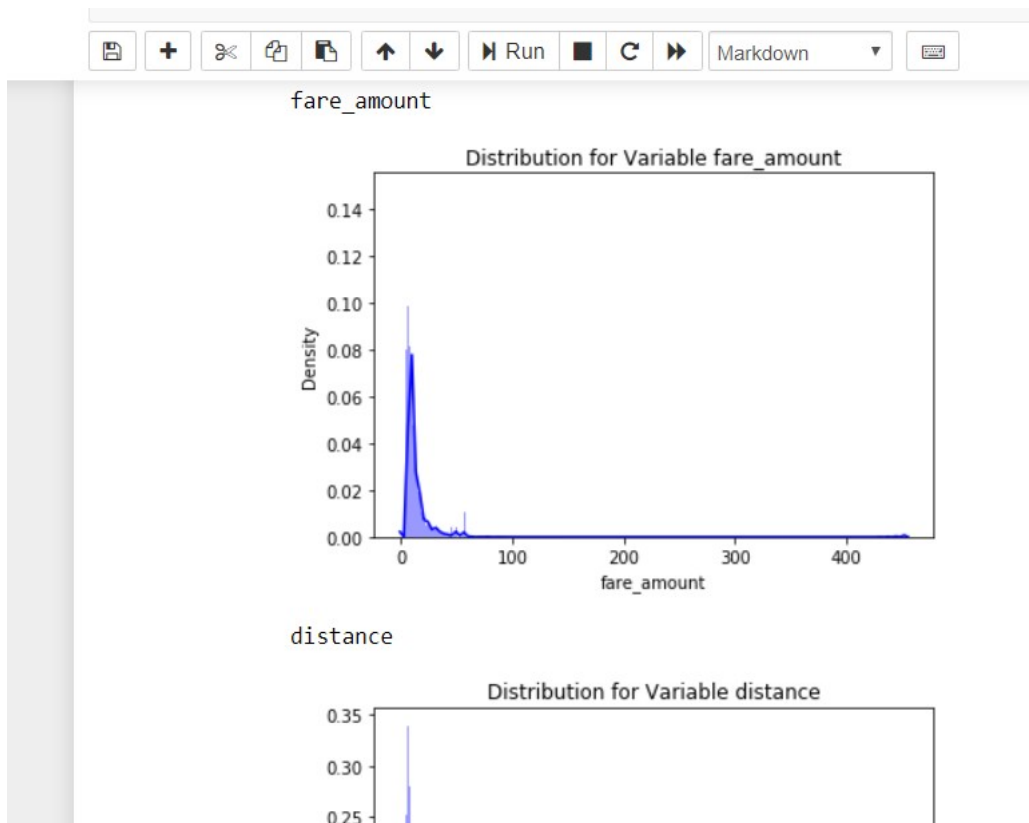
R Libraray use: ggplot2

The range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, many classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours having is one sided skewed so by

using **log transform** technique we tried to reduce the skewness of the same.

Before log transformation data look like:



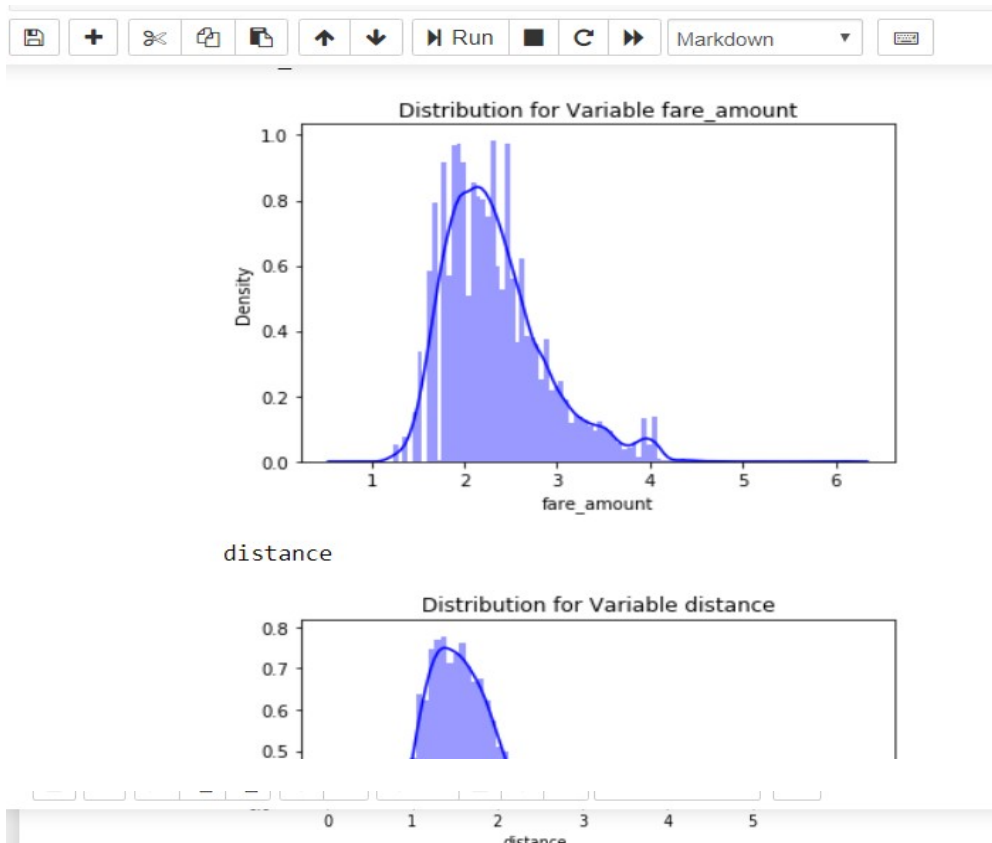
Here we apply log transformation:

```
In [74]: #since skewness of target variable is high, apply log transform to reduce the skewness-
train_cab['fare_amount'] = np.log1p(train_cab['fare_amount'])

#since skewness of distance variable is high, apply log transform to reduce the skewness-
train_cab['distance'] = np.log1p(train_cab['distance'])

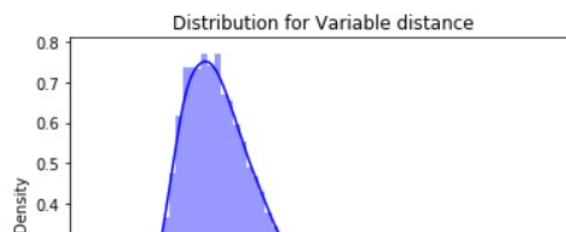
In [75]: #Normality Re-check to check data is uniformly distributed or not after log transformation
for i in ['fare_amount', 'distance']:
```

Now look at data after log transformation



```
In [76]: #since skewness of distance variable is high, apply log transform to reduce the skewness-
test_cab['distance'] = np.log1p(test_cab['distance'])
```

```
In [77]: #rechecking the distribution for distance
sns.distplot(test_cab['distance'],bins='auto',color='blue')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()
```



R code:

Checking distance variable distribution using histogram

ggplot(Train_Cab, aes_string(x = Train_Cab\$distance)) +

```
geom_histogram(fill="skyblue", colour = "black") + geom_density() +  
  theme_bw() + xlab("distance") + ylab("Frequency")+ggtitle("  
distribution of distance ")
```

```
# this variable is right skewed
```

```
ggplot(Train_Cab, aes_string(x = Train_Cab$fare_amount)) +  
  geom_histogram(fill="skyblue", colour = "black") + geom_density() +  
  theme_bw() + xlab("distance") + ylab("Frequency")+ggtitle("  
distribution of distance ")
```

```
# Lets take log transformation to remove skewness
```

```
# Lets define function for log transformation of variables
```

```
signedlog10 = function(x) {  
  ifelse(abs(x) <= 1, 0, sign(x)*log1p(abs(x)))  
}
```

```
# Applying log function to distance variable
```

```
Train_Cab$distance = signedlog10(Train_Cab$distance)
```

```
# Checking distance distribution after applying function
```

```
ggplot(Train_Cab, aes_string(x = Train_Cab$distance)) +  
  geom_histogram(fill="skyblue", colour = "green") + geom_density() +  
  theme_bw() + xlab("distance") + ylab("Frequency")+ggtitle("  
distribution of distance after log transformation")
```

```
# Applying log function to fare_amount variable
```

```
Train_Cab$fare_amount = signedlog10(Train_Cab$fare_amount)
```

```
# Checking fare_amount distribution after applying function  
ggplot(Train_Cab, aes_string(x = Train_Cab$fare_amount)) +  
  geom_histogram(fill="skyblue", colour = "green") + geom_density() +  
  theme_bw() + xlab("distance") + ylab("Frequency")+ggtitle("  
distribution of distance after log transformation")
```

As we can see a bell shaped distribution. Hence our continuous variables are now normally distributed, so we will not use any Feature Scalling technique like Normalization or Standarization for our test data and train data.

DATA MODELING

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Data models provide a framework for data to be used within information systems by providing specific definition and format. If a data model is used consistently across systems then compatibility of data can be achieved. If the same data structures are used to store and access data then different applications can share data seamlessly.

In **machine learning**, there are **three types**:

1. Supervised Learning

This algorithm consist of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: **Regression, Decision Tree, Random Forest, KNN, Logistic Regression** etc.

2. Unsupervised Learning

In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: **Apriori algorithm, K-means**.

3. Reinforcement Learning:

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions.

As observing dataset, we find output on based on pattern defined in training dataset, so here we go with **Supervised Learning**.

In **supervised learning**, we have two type:

1] Clasification Models

On the other hand, classification algorithms attempt to estimate the mapping function (f) from the input variables (x) to discrete or categorical output variables (y). In this case, y is a category that the mapping function predicts. If provided with a single or several input variables, a classification model will attempt to predict the value of a single or several conclusions.

For example, when provided with a dataset about houses, a classification algorithm can try to predict whether the prices for the houses “sell more or less than the recommended retail price.”

2] Regression Models

In machine learning, regression algorithms attempt to estimate the mapping function (f) from the input variables (x) to numerical or continuous output variables (y). In this case, y is a real value, which can be an integer or a floating point value. Therefore, regression prediction problems are usually quantities or sizes.

For example, when provided with a dataset about houses, and you are asked to predict their prices, that is a regression task because price will be a continuous output.

As per observing on dataset we can conclude that our dataset is in **continuous format**. Our **target variable has a integer** value which is not referring to a classification or discrete class variable.

So for our dataset we are choosing **regression models** like follows:

- 1) **Linear Regression Model**
- 2) **Decision Tree Regression Model**
- 3) **Random Forest Regression Model**

Step 1: First we split data in Train and Test data.

1. Linear Regression Model
2. Decision Tree regression Model
3. Random Forest Regression Model

```
In [80]: # Lets Divide the data into train and test set
```

```
x= train_cab.drop(['fare_amount'],axis=1)
y= train_cab['fare_amount']
```

```
In [81]: # Now Split the data into train and test using train_test_split function
```

```
X train, X test, y train, y test = train_test_split(x, y, test_size=0.2, random_state=101)
```

R code:

```
library(DataCombine)
library(caret)
rmExcept("Train_Cab")
# Split the data set into train and test
set.seed(1234)
train.index = createDataPartition(Train_Cab$fare_amount, p = .80, list =
FALSE)
train_data = Train_Cab[train.index,]
test_data = Train_Cab[-train.index,]
dim(train_data)
dim(test_data)
```

We here splitting data by considering test_size=20% i.e we are applying training model on 80% train data.

Step 2: What we choose to check accuracy

As we have observed our data follows regression modelling, that's why for finding accuracy we go with **regression mtrices**.

There are 3 types of regression matrix we have seen:

1. **MAE (Mean Absolute Error)**
2. **MAPE (Mean Absolute Percentage Error)**
3. **RSME (Root squered Mean Error)**

As RSME we use when we have time series type data. and MAPE we use when we just want percentage of deifference between actual and predicted outcome.

So for our data we choose to go with MAPE matrix to find accuracy of model.

```
In [83]: def MAPE(y_true,y_prediction):  
         mape= np.mean(np.abs(y_true-y_prediction)/y_true)*100  
         return mape
```

Step 3: Applying Regression Models one by one

4.1 Linear Regression Model

***Python Library Use: numpy, sklearn.linear_model ,
sklearn.model_selection***

R libraray Use: DataCombine, caret

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation $Y = a * X + b$.

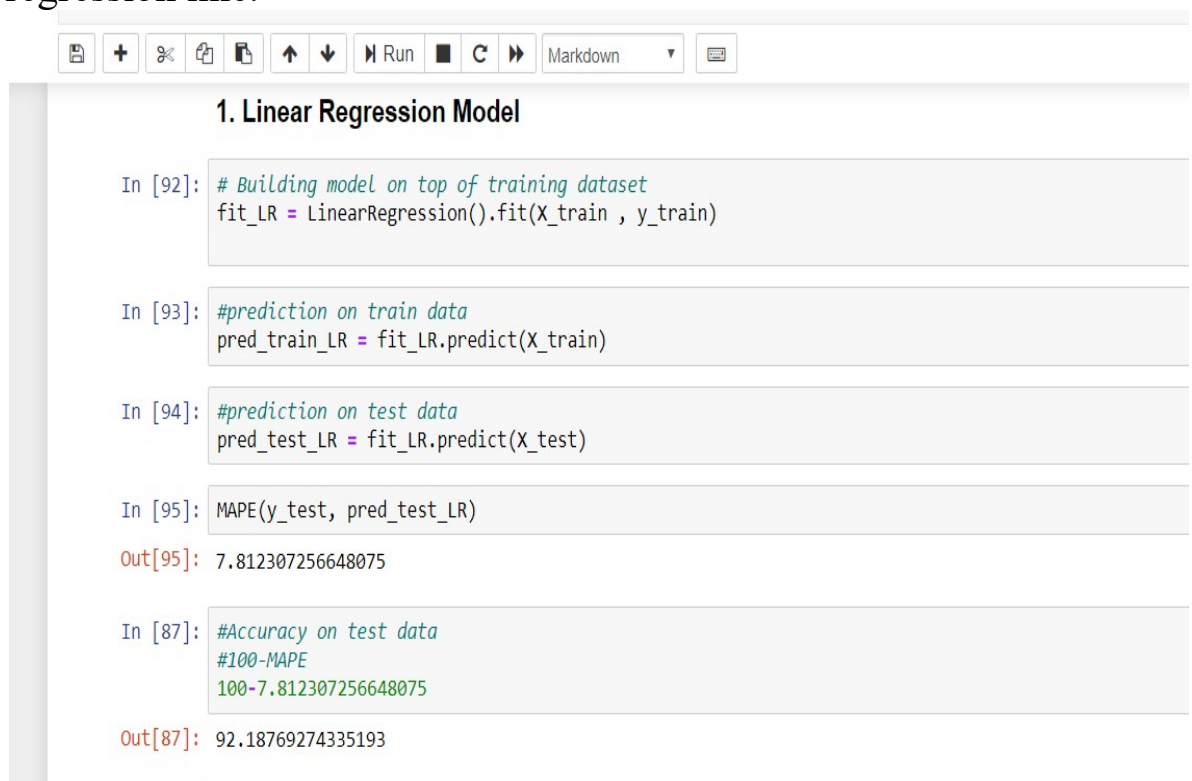
The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade

to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.



The screenshot shows a Jupyter Notebook with a toolbar at the top containing icons for file operations, navigation, and execution. The notebook content is titled "1. Linear Regression Model" and contains five input cells with Python code and their corresponding outputs.

```
In [92]: # Building model on top of training dataset
fit_LR = LinearRegression().fit(X_train , y_train)

In [93]: #prediction on train data
pred_train_LR = fit_LR.predict(X_train)

In [94]: #prediction on test data
pred_test_LR = fit_LR.predict(X_test)

In [95]: MAPE(y_test, pred_test_LR)

Out[95]: 7.812307256648075

In [87]: #Accuracy on test data
#100-MAPE
100-7.812307256648075

Out[87]: 92.18769274335193
```

R code:


```

# fit linear regression model
# we will use the lm() function in the stats package
lm_model = lm(fare_amount ~.,data = train_data)
summary(lm_model)

#Residual standard error: 0.1226 on 12297 degrees of freedom
#Multiple R-squared:  0.7757,    Adjusted R-squared:  0.775
#F-statistic: 1013 on 42 and 12297 DF,  p-value: < 2.2e-16
#lets predict for train and test data

Predictions_LR_train = predict(lm_model,train_data)
Predictions_LR_test = predict(lm_model,test_data)
#let us check performance of our model

#mape calculation
LR_train_mape = MAPE(Predictions_LR_train,train_data[,1])
LR_test_mape = MAPE(test_data[,1],Predictions_LR_test)
print(LR_train_mape)#0.089
print(LR_test_mape)#0.10

```

4.2 Decision Tree Regression

Python Library Use: numpy, sklearn.tree

R Libraray Use: rpart

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor

called root node. Decision trees can handle both categorical and numerical data.



2. Decision Tree regression

```
In [99]: fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)
```

```
In [100]: #prediction on train data
pred_train_DT = fit_DT.predict(X_train)

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

```
In [101]: MAPE(y_train, pred_train_DT)
```

```
Out[101]: 9.445229174410636
```

```
In [102]: #Accuracy on train data=
100-9.445229174410755
```

```
Out[102]: 90.55477082558924
```

R code:

```
library(rpart)
DT_model = rpart(fare_amount ~ ., data = train_data, method =
"anova")
DT_model
```

```
# Lets predict for train and test data
predictions_DT_train= predict(DT_model,train_data)
```

```
predictions_DT_test= predict(DT_model,test_data)

# MAPE calculation
DT_train_mape = MAPE(train_data[,1],predictions_DT_train)
DT_test_mape = MAPE(test_data[,1],predictions_DT_test)
print(DT_train_mape)#0.09
print(DT_test_mape)#0.11
```

4.3 Random Forest Regression Model

Python Library Use: numpy, sklearn.ensemble

R Library Use: randomForest, e1071, dplyr

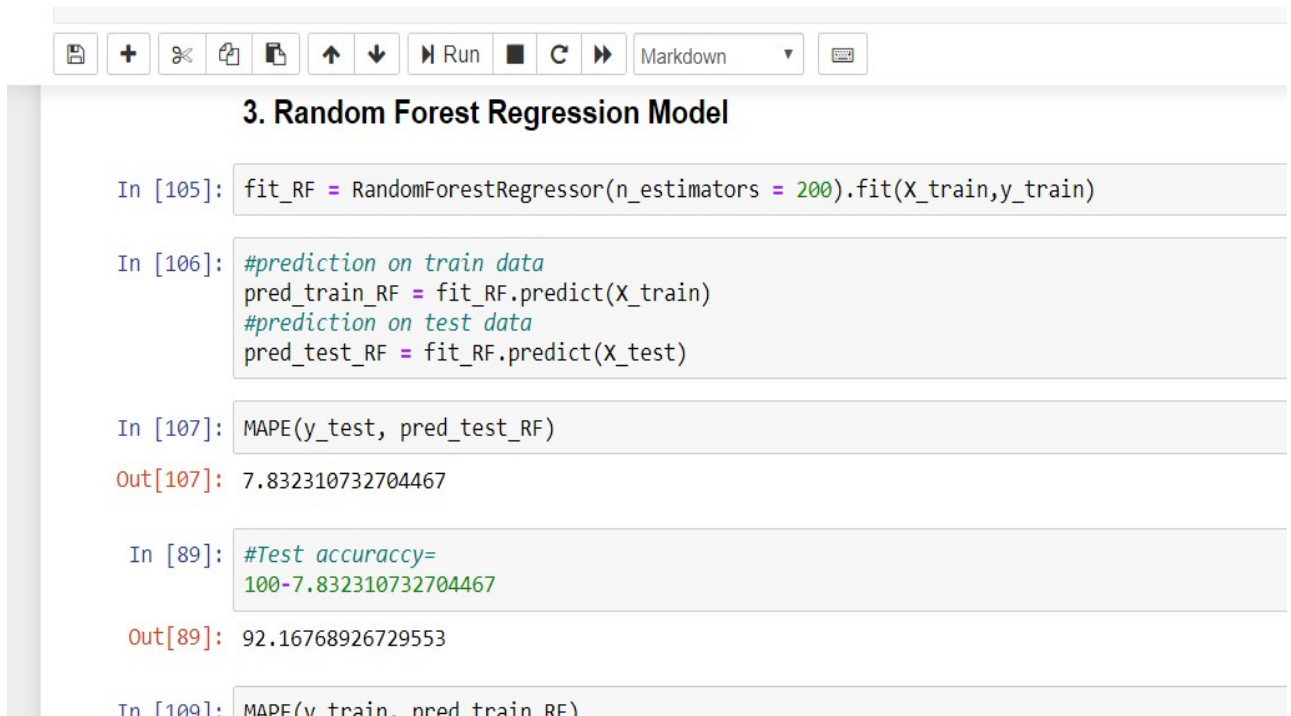
Random forest is a bagging technique and not a boosting technique. The trees in random forests are run in parallel. There is no interaction between these trees while building the trees.

It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees, with some helpful modifications:

1. The number of features that can be split on at each node is limited to some percentage of the total (which is known as the **hyperparameter**). This ensures that the ensemble model **does not rely too heavily on any individual feature**, and makes **fair use of all potentially predictive features**.

2. Each tree draws a random sample from the original data set when generating its splits, adding a further element of randomness that prevents **overfitting**.



The screenshot shows a Jupyter Notebook interface with a toolbar at the top containing icons for saving, adding, deleting, copying, pasting, and running code. Below the toolbar, the notebook is titled "3. Random Forest Regression Model". The code is as follows:

```
In [105]: fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)

In [106]: #prediction on train data
pred_train_RF = fit_RF.predict(X_train)
#prediction on test data
pred_test_RF = fit_RF.predict(X_test)

In [107]: MAPE(y_test, pred_test_RF)

Out[107]: 7.832310732704467

In [89]: #Test accuracy=
100-7.832310732704467

Out[89]: 92.16768926729553

In [109]: MAPE(v train. pred train RF)
```

R code:

```
library(randomForest)
```

```
library(e1071)
```

```
library(dplyr)
```

#While considering character variable in random forest modelling it will give an error "foreign function call"

#so we have to convert that character variable into numeric or factor type

```
train_data=train_data %>% mutate_if(is.character, as.factor)
```

```
str(train_data)
```

```
test_data=test_data %>% mutate_if(is.character, as.factor)

str(test_data)

#lets build the random forest model

RF_model = randomForest(fare_amount~.,data = train_data, n.trees =
500)

print(RF_model)

# randomForest(formula = fare_amount ~ ., data = train_data, n.trees =
500)

#Type of random forest: regression

#Number of trees: 500

#No. of variables tried at each split: 1

#Mean of squared residuals: 0.02147973 Var explained: 67.82

#lets predict for both train and test data

predictions_RF_train = predict(RF_model,train_data)

predictions_RF_test = predict(RF_model,test_data)

#MAPE calculation

RF_train_mape = MAPE(predictions_RF_train,train_data[,1])

RF_test_mape = MAPE(predictions_RF_test,test_data[,1])

print(RF_train_mape)#0.098

print(RF_test_mape)#0.11
```

4.4 Evaluation of model

As per observing above models we can conclude that:

Sr.no	Modeling Technique	MAPE	Accuracy
1]	Linear Regression	7.8123	92.1876
2]	Decision Tree Regression	9.7239	90.2761
3]	Random Forest Regression	2.8136	97.1864

Step 4: Observation on modeling

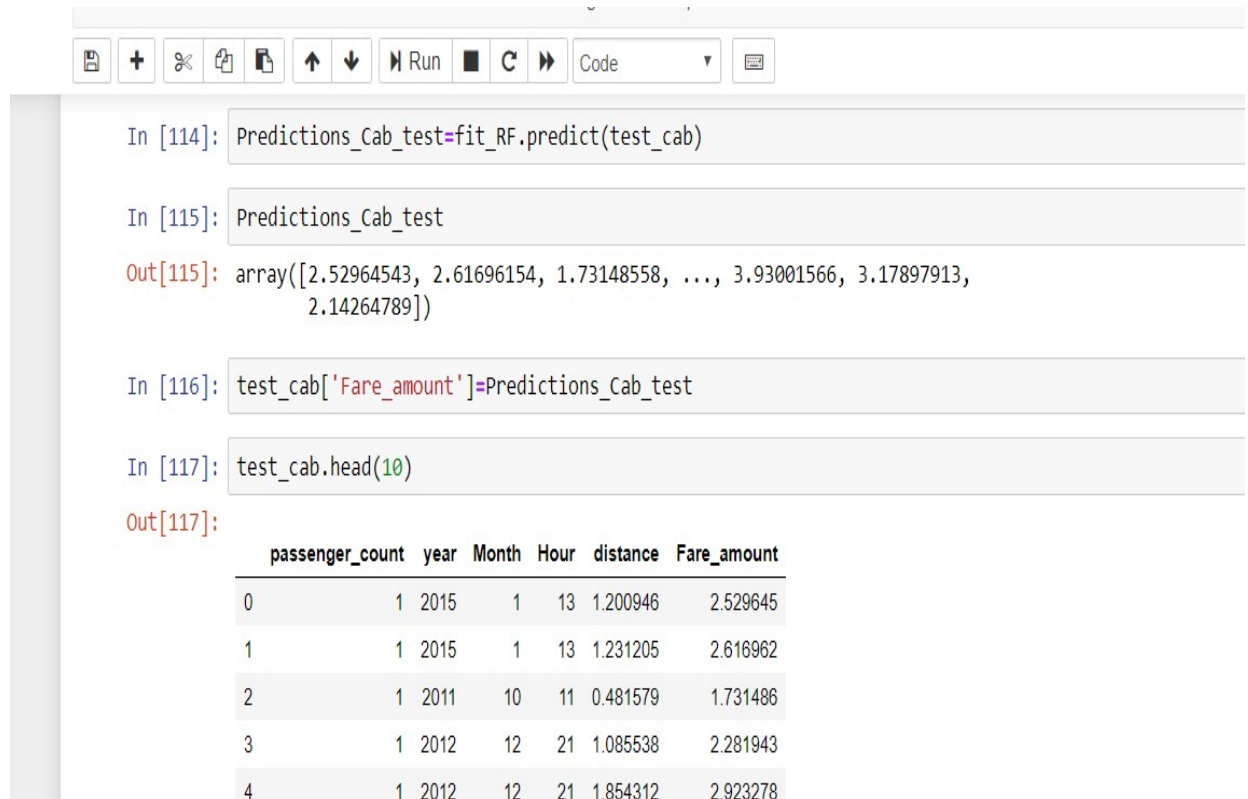
So here we can say that **Random forest regressor** model gives us **best mach** accuracy than other two model, therefore we choose random forest regressor model for getting **predictable fare amount** for test dataset.

Step 5: Create predicatable fare amount for test dataset

4.5 Create predicted variable on test data

In python we will apply all preprocessing techniques simultaneously on test dataset , so here we directly apply best match i.e random forest regression model on test dataset.

Python code:



The screenshot shows a Jupyter Notebook interface with a toolbar at the top containing icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. Below the toolbar, the notebook contains several input and output cells. The first input cell (In [114]) contains the code `Predictions_Cab_test=fit_RF.predict(test_cab)`. The second input cell (In [115]) contains `Predictions_Cab_test`. The output of the second cell (Out[115]) is an array of predicted fare amounts: `array([2.52964543, 2.61696154, 1.73148558, ..., 3.93001566, 3.17897913, 2.14264789])`. The third input cell (In [116]) contains `test_cab['Fare_amount']=Predictions_Cab_test`. The fourth input cell (In [117]) contains `test_cab.head(10)`. The output of the fourth cell (Out[117]) is a table showing the first 10 rows of the `test_cab` dataset, with columns: `passenger_count`, `year`, `Month`, `Hour`, `distance`, and `Fare_amount`.

```
In [114]: Predictions_Cab_test=fit_RF.predict(test_cab)
```

```
In [115]: Predictions_Cab_test
```

```
Out[115]: array([2.52964543, 2.61696154, 1.73148558, ..., 3.93001566, 3.17897913, 2.14264789])
```

```
In [116]: test_cab['Fare_amount']=Predictions_Cab_test
```

```
In [117]: test_cab.head(10)
```

```
Out[117]:
```

	passenger_count	year	Month	Hour	distance	Fare_amount
0	1	2015	1	13	1.200946	2.529645
1	1	2015	1	13	1.231205	2.616962
2	1	2011	10	11	0.481579	1.731486
3	1	2012	12	21	1.085538	2.281943
4	1	2012	12	21	1.854312	2.923278

R code:

```
RFTest_Cab = predict(RF_model, Test_Cab)
```

```
Test_Cab$Fare_amount<-RFTest_Cab
```

```
print(head(Test_Cab))
```

CONCLUSION

We can conclude that before finding any future predictable output we have to first apply all necessary data mining techniques to make optimized data then only we can form the best match model development algorithm for predicting future output. Our predictable fare amount got from final considering fact of passengers count, distance, hours, year and month of traveling.

REFERENCES

1. <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>
2. https://en.wikipedia.org/wiki/Haversine_formula
3. https://het.as.utexas.edu/HET/Software/Numpy/reference/generate_d/numpy.log1p.html
4. https://knowledge.domo.com/Visualize/Adding_Cards_to_Domo/KPI_Cards/Building_Each_Chart_Type/Latitude-Longitude_Map