CAB FARE PRIDICTION

Project report

ABSTRACT
Cab rental prediction->The objective of this Case is to Predication of cab fares on the basis of data collected from pilot project .

Gagan Kohli
Data Scientist

Contents

Chapter 1

Introduction

1.1 Problem Statement

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

1.2 Data

We are given with 2 data sets, a train data set and another test data set .Our task is to build Regression model which will give the fare for cab service based on given parameters. Given below is a sample of the data set that we are using to predict the fare (train data):

Table 1.1: Cab Fare Sample Data (train) (Columns: 1-7)

	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	77	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	40.758092	1.0

Given below is a sample of the data set that we are using test or in simple words for which we need predict the fare (test data):

Table 1.2: Cab Fare Sample Data (test) (Columns: 1-6)

pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count

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

Below are the variables are used to predict the fare prediction for cab

Table 1.3: Cab Fare Predictors

s.no	Variables
1	**pickup_datetime**
2	**pickup_longitude**
3	**pickup_latitude**
4	**dropoff_longitude**
5	**dropoff_latitude**
6	**passenger_count**

The details of data attributes in the test dataset are as follows –

- 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.

The details of data attributes in the train dataset are as follows –

- fare_amount –float for fare amount charged to costumer in dollars $
- 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

2.1 Pre Processing

Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. To start this process we will first try and make initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations look at all the distributions of the Numeric variables. Most analysis like regression, require the data to be normally distributed.
Starting with general analysis the data given to us includes 7 variables and 16067 observations in train data and 6 variables and 9914 observations in test data . Further
Analysing the data types for each column->

```
In [830]: #lets check dtypes for train data
          cab_data.dtypes

Out[830]: fare_amount           object
          pickup_datetime       object
          pickup_longitude     float64
          pickup_latitude      float64
          dropoff_longitude    float64
          dropoff_latitude     float64
          passenger_count      float64
          dtype: object
```

Figure 2.1 data types for variables in train data

```
In [834]: #from above analysis we can draw few insights about data.
             #there are missing values in the data
             #max fare "54343" and max pessenger"5345" sounds absard for a cab
          #continuing further analysis for test data as well.
          cab_data_test.dtypes

Out[834]: pickup_datetime       object
          pickup_longitude     float64
          pickup_latitude      float64
          dropoff_longitude    float64
          dropoff_latitude     float64
          passenger_count        int64
          dtype: object
```

Figure 2.2 data types for variables in test data

From this analysis we found that some variables in the train data set have wrong data type like passenger_count, fare_amount, pikup_datetime. Moreover in test data set also pikup_datetime has wrong data type

Further we also found basic statistical information for our data like median mean and other important information also by observation we can see the train data set is corrupted fare and passenger variables consist of absurd outliers like passenger count 5345 that's not possible for a cab to carry and fare amount 54343 $ that's too high for a cab. Also the data has missing values

```
In [833]: cab_data.describe()
```

Out[833]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16043.000000	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	15.040871	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	430.459997	10.578384	6.826587	10.575062	6.187087	60.844122
min	-3.000000	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	6.000000	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	8.500000	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	12.500000	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	54343.000000	40.766125	401.083332	40.802437	41.366138	5345.000000

Figure 2.3 data analysis for variables in train data

```
In [835]: #data types for test daya seems apt
          #test data looks okay at 1st glance no missing values no visable outliers as of now
          cab_data_test.describe()
```

Out[835]:

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

UNIVARIAT analysis

Figure 2.4 data analysis for variables in test data

Too handle such a corrupt data we have followed regression approach in which we will clean the data first and then move forward in model creation. We will first take variable 1 by 1 for cleaning purpose let's start our analysis by fare amount first.

2.1.1 Feature engineering

Proper data type for each variable is very important in there analysis like pickup date is given a factor but to extract useful information from it we need to convert it into date format. So we will convert fare as a float passenger count as an integer and date time as data type date

```
In [831]: #fare amount has to be converted to float for analysis
          #fare value "430-" is to be cleaned before to convert it in numerical data type
          cab_data["fare_amount"]=cab_data["fare_amount"].str.replace("430-","430")
          cab_data["fare_amount"]=cab_data["fare_amount"].astype(float)
```

```
In [749]: #lets check for test data also
          cab_data["pickup_datetime"]=cab_data["pickup_datetime"].str.replace("UTC","")
          cab_data_test["pickup_datetime"]=cab_data_test["pickup_datetime"].str.replace("UTC","")
          cab_data_test.dtypes

Out[749]: pickup_datetime       object
          pickup_longitude     float64
          pickup_latitude      float64
          dropoff_longitude    float64
          dropoff_latitude     float64
          passenger_count        int64
          dtype: object

In [750]: cab_data["passenger_count"]=cab_data["passenger_count"].astype(int)
          cab_data['pickup_datetime']  = pd.to_datetime(cab_data['pickup_datetime'],format='%Y-%m-%d %H:%M:%S', errors='coerce')
          cab_data_test['pickup_datetime']  = pd.to_datetime(cab_data_test['pickup_datetime'],format='%Y-%m-%d %H:%M:%S', errors='coerce')

In [751]: cab_data.dtypes

Out[751]: dropoff_latitude            float64
          dropoff_longitude           float64
          fare_amount                 float64
          passenger_count               int32
          pickup_datetime      datetime64[ns]
          pickup_latitude             float64
          pickup_longitude            float64
          dtype: object
```

Figure 2.4 data types after changing for variables in test and train data

2.2 Pre-processing Techniques

These include techniques like missing value analysis, outlier treatment, feature engineering, feature selection and feature scaling. These techniques are necessary before model building so the model could be correct and that of a good quality

As discussed before we are treating outliers and other corrupt data variable by variable thus we won't be following the conventional approach. Also seeing number of variables we need to add more variables in both test and train data

Let's start by clean up activity variable by variable clean up involves outlier treatment as it is a run time data we cannot justify statically outliers what we can do is treat logical outliers

Approach followed is that most of the outliers have been converted to NAs and then collectively treated as missing value while missing value analysis

2.2.1 Fare Amount

Fare amount variable in the train data is our target variable it has absurd outliers like 54343$ and other negative fare values so we need to define an boundary for of outliers from research (google) minimum fare in States for a cab is 2.5$ and by estimate I have taken upper boundary as 300 for the fare for a cab in city.

We have converted the outliers into NAs to minimize data loss

```
Out[843]: 5

In [844]: #there are around 11 odd outliers lets convert them to NA
          cab_data["fare_amount"].loc[cab_data["fare_amount"]<2.5]=np.nan
          cab_data["fare_amount"].loc[cab_data["fare_amount"]>300]=np.nan

In [845]: cab_data["fare_amount"].isnull().sum()

Out[845]: 35
```

Figure 2.5 fare amount boundary in train

These NAs along with already present NAs will be treated once with missing value treatment.

2.2.2 Passenger_count

Passenger count is an variable that shows number of passengers in the cab for that trip. Logically a cab can have max 6 persons at a time. these variable consists of absurd values like 0 ,5435 etc . we have taken an boundry of minimum 1 passenger and max 6 passangers and converted rest vlues out of this boundriey as NA.

These NAs along with already present NAs will be treated once with missing value treatment.

```
In [839]: #coverting into NA
          cab_data["passenger_count"].loc[cab_data["passenger_count"]>6]=np.nan
          cab_data["passenger_count"].loc[cab_data["passenger_count"]<1]=np.nan

C:\Users\hp\Anaconda3\lib\site-packages\pandas\core\indexing.py:189: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  self._setitem_with_indexer(indexer, value)

In [840]: cab_data["passenger_count"].describe()

Out[840]: count    15934.000000
          mean         1.649636
          std          1.265896
          min          1.000000
          25%          1.000000
          50%          1.000000
          75%          2.000000
          max          6.000000
          Name: passenger_count, dtype: float64
```

Figure 2.6 passenger amount boundary in train

2.2.3 Latitude and longitude data

Latitude and longitude data for both pikup and dropoff location are given they also contain absurd illogical so called outliers. On analysis it is found that many data point consist of 0 latitude or longitude or both. On research(google) its found that (0.0) lie in the ocean that is physically not possible for a cab to pickup or drop off someone also some of the data points are correct but inverted in latitude longitude. This data can be used if treated first.

The test data can be helpful for us to find a boundry. Using test data we can find max and min latitude longitude for pikup and drop thus forming an boundry.

```
In [848]: #lets find the the min max rnge for latitude and longitude from test data as it is almost free from any outliers
          #max and min longitude from test data
          lon_min=min(cab_data_test.pickup_longitude.min(),cab_data_test.dropoff_longitude.min())
          lon_max=max(cab_data_test.pickup_longitude.max(),cab_data_test.dropoff_longitude.max())
          print(lon_min,',',lon_max)

          -74.263242 , -72.986532

In [849]: #max and min longitude from test data
          lat_min=min(cab_data_test.pickup_latitude.min(),cab_data_test.dropoff_latitude.min())
          lat_max=max(cab_data_test.pickup_latitude.max(),cab_data_test.dropoff_latitude.max())
          print(lat_min,',',lat_max)

          40.568973 , 41.709555
```

Figure 2.7 longitude latitude boundary in train

Now we will separate outliers from the data and extract usefull data rest all have to be droped as the no of outliers are not more than 2 percent of the total observations

```
40.568973 , 41.709555
```

```
In [850]: #let us find outliers on bases of this range
def select_outside_boundingbox(df, BB):
    filter_df = df.loc[(df['pickup_longitude'] < BB[0]) | (df['pickup_longitude'] > BB[1]) | \
           (df['pickup_latitude'] < BB[2]) | (df['pickup_latitude'] > BB[3]) | \
           (df['dropoff_longitude'] < BB[0]) | (df['dropoff_longitude'] > BB[1]) | \
           (df['dropoff_latitude'] < BB[2]) | (df['dropoff_latitude'] > BB[3])]

    return filter_df

BB = (-74.5, -72.8, 40.5, 41.8)
```

```
In [851]: latlon_outliers = select_outside_boundingbox(cab_data, BB)
latlon_outliers.head()
```

Out[851]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
11	5.5	2012-12-24 11:24:00 UTC	0.0	0.0	0.0	0.0	3.0
15	5.0	2013-11-23 12:57:00 UTC	0.0	0.0	0.0	0.0	1.0
26	NaN	2011-02-07 20:01:00 UTC	0.0	0.0	0.0	0.0	1.0
124	8.0	2013-01-17 17:22:00 UTC	0.0	0.0	0.0	0.0	2.0
192	3.7	2010-09-05 17:08:00 UTC	0.0	0.0	0.0	0.0	5.0

```
In [854]: #lets first deal with zeros in lat and long data
#we are deleting all zero as the zero cordinate lies in ocean thats absard in tiself for a cab to travel
def drop_0s(df, verbose=False):
    if verbose:
        print("Dropping all rows with 0s:")
        old_size = len(df)
        print("Old size: {}".format(old_size))

    df = df.loc[~(df == 0).any(axis=1)]


    if verbose:
        new_size = len(df)
        print("New size: {}".format(new_size))
        difference = old_size - new_size
        percent = (difference / old_size) * 100
        print("Dropped {} records, or {:.2f}%".format(difference, percent))

    return df

latlon_outliers = drop_0s(latlon_outliers, True)

latlon_outliers.describe()
```

```
Dropping all rows with 0s:
Old size: 348
New size: 22
Dropped 326 records, or 93.68%
```

Figure 2.8 latitude longitude treatment in train

```
In [855]: #as we can see many rows have values inverted  for latitude and longitude thesr data rows can be usefull if we could
def select_within_boundingbox(df, BB):
    filter_df = df.loc[(df['pickup_longitude'] >= BB[0]) & (df['pickup_longitude'] <= BB[1]) & \
           (df['pickup_latitude'] >= BB[2]) & (df['pickup_latitude'] <= BB[3]) & \
           (df['dropoff_longitude'] >= BB[0]) & (df['dropoff_longitude'] <= BB[1]) & \
           (df['dropoff_latitude'] >= BB[2]) & (df['dropoff_latitude'] <= BB[3])]


    return filter_df

inverted_BB = (40.5, 41.8, -74.5, -72.8)

inverted_outliers = select_within_boundingbox(latlon_outliers, inverted_BB)

inverted_outliers.describe()
```

Out[855]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
mean	16.375000	40.747089	-73.991207	40.752877	-73.973476	1.875000
std	14.882276	0.017046	0.011292	0.024100	0.022082	1.726888
min	5.000000	40.719830	-74.006893	40.723305	-74.006377	1.000000
25%	9.875000	40.734938	-73.996263	40.739497	-73.982855	1.000000
50%	13.000000	40.749922	-73.990153	40.748894	-73.978158	1.000000
75%	15.125000	40.761476	-73.986047	40.759992	-73.958845	2.000000
max	52.000000	40.766125	-73.973047	40.802437	-73.939430	6.000000

Figure 2.9inverted latitude longitude

Swiping the inverted latitude and longitude and concating it to orignal data

```
In [856]: def swap_inverted(df):
    fixed_df = df.rename(columns={'pickup_longitude' : 'pickup_latitude', 'pickup_latitude' : 'pickup_longitude',
                                  'dropoff_longitude' : 'dropoff_latitude', 'dropoff_latitude' : 'dropoff_l
    col_list = fixed_df.columns.tolist()

    col_list[3], col_list[4], col_list[5], col_list[6] = col_list[4], col_list[3], col_list[6], col_list[5]

    fixed_df = fixed_df[col_list]

    return fixed_df

fixed_outliers = swap_inverted(inverted_outliers)

fixed_outliers.describe()
```

Out[856]:

	fare_amount	pickup_latitude	dropoff_latitude	pickup_longitude	passenger_count	dropoff_longitude
count	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
mean	16.375000	40.747089	40.752877	-73.991207	1.875000	-73.973476
std	14.882276	0.017046	0.024100	0.011292	1.726888	0.022082
min	5.000000	40.719830	40.723305	-74.006893	1.000000	-74.006377
25%	9.875000	40.734938	40.739497	-73.996263	1.000000	-73.982855
50%	13.000000	40.749922	40.748894	-73.990153	1.000000	-73.978158
75%	15.125000	40.761476	40.759992	-73.986047	2.000000	-73.958845
max	52.000000	40.766125	40.802437	-73.973047	6.000000	-73.939430

Figure 2.10 corrected latitude longitude

```
In [858]: ## Now we'll remove all rows with a datapoint that doesn't fall within the bounding box for NYC coordinates

print("Old size: {}".format(len(cab_data)))

cab_data =cab_data.loc[(cab_data['pickup_longitude'] >= BB[0]) & (cab_data['pickup_longitude'] <= BB[1]) & \
            (cab_data['pickup_latitude'] >= BB[2]) & (cab_data['pickup_latitude'] <= BB[3]) & \
            (cab_data['dropoff_longitude'] >= BB[0]) & (cab_data['dropoff_longitude'] <= BB[1]) & \
            (cab_data['dropoff_latitude'] >= BB[2]) & (cab_data['dropoff_latitude'] <= BB[3])]

|
print("New size: {}".format(len(cab_data)))

cab_data.describe()

Old size: 16067
New size: 15719
```

Out[858]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	15686.000000	15719.000000	15719.000000	15719.000000	15719.000000	15590.000000
mean	11.300370	-73.974829	40.751332	-73.973838	40.751835	1.650436
std	9.599172	0.041491	0.031504	0.039362	0.033457	1.265927
min	2.500000	-74.438233	40.500046	-74.429332	40.500046	1.000000
25%	6.000000	-73.992401	40.736604	-73.991377	40.736325	1.000000
50%	8.500000	-73.982060	40.753340	-73.980577	40.754255	1.000000
75%	12.500000	-73.968128	40.767805	-73.965399	40.768331	2.000000
max	180.000000	-73.137393	41.366138	-73.137393	41.366138	6.000000

```
In [733]: #concatinating the filtered otliers with the data
cab_data_copy = cab_data # Created a copy so as to avoid the possibility of adding the fixed outliers multiple times

cab_data = pd.concat([cab_data_copy, fixed_outliers],ignore_index=True,sort=True)

cab_data_copy = None  # Doing this to try to be a bit more memory efficient

cab_data.describe()
```

Out[733]:

	dropoff_latitude	dropoff_longitude	fare_amount	passenger_count	pickup_latitude	pickup_longitude
count	15727.000000	15727.000000	15694.000000	15598.000000	15727.000000	15727.000000
mean	40.751836	-73.973838	11.302957	1.650551	40.751329	-73.974837
std	0.033453	0.039354	9.602555	1.266141	0.031499	0.041483
min	40.500046	-74.429332	2.500000	1.000000	40.500046	-74.438233
25%	40.736333	-73.991377	6.000000	1.000000	40.736604	-73.992402
50%	40.754255	-73.980572	8.500000	1.000000	40.753326	-73.982076
75%	40.768322	-73.965396	12.500000	2.000000	40.767803	-73.968134
max	41.366138	-73.137393	180.000000	6.000000	41.366138	-73.137393

Figure 2.11 final data after longitude latitude treatment

Now we are left with 15727 observations

2.2.4 Missing Value Analysis

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models. Also we have created NAs while treating otliers so that we can treat them all at once at this step .

For missing value analysis lets first find number of missing values in the data

```
In [736]: #creating adata frame missing
          missing_val= pd.DataFrame(cab_data.isnull().sum())

In [737]: #reseting the index
          missing_val = missing_val.reset_index()
          #Rename variable
          missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})

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

Out[737]:
```

	Variables	Missing_percentage
0	dropoff_latitude	0.000000
1	dropoff_longitude	0.000000
2	fare_amount	0.209830
3	passenger_count	0.820245
4	pickup_datetime	0.000000
5	pickup_latitude	0.000000
6	pickup_longitude	0.000000

Figure 2.12 missing values

As noted there are missing values less than 1 percent of data thus we will impute the missing values

Creating copy data frame to decide how to impute

```
In [738]: #there are 0.2 percent and 0.8 percent missing values in fare and passenger respectively
          #creating copy of cab data for imputation purpose
          df1= cab_data.copy()
          df2= cab_data.copy()
          df3= cab_data.copy()

In [739]: df1.iloc[6,2]

Out[739]: 7.5

In [740]: df1.iloc[6,2]=np.nan
          df2.iloc[6,2]=np.nan
          df3.iloc[6,2]=np.nan

In [741]: df1.iloc[6,2]

Out[741]: nan

In [742]: #using mean method
          df1['fare_amount'] = df1['fare_amount'].fillna(df1['fare_amount'].mean())
          df1.iloc[6,2]

Out[742]: 11.303199515707629

In [743]: #using median method
          df2['fare_amount'] = df2['fare_amount'].fillna(df2['fare_amount'].median())
          df2.iloc[6,2]

Out[743]: 8.5

In [744]: #using interpolat
          df3['fare_amount'] = df3['fare_amount'].interpolate(method = 'nearest', limit_direction = 'both')
          df3.iloc[6,2]

Out[744]: 12.1
```

Figure 2.13 missing Value process

We try imputing a selected value by different methords like mean, median ,interpolate,knn etc

Median method is giving closest value so will impute the missing values in data by median method

```
In [745]: #usinng median methord for imputation of passenger variable and fare variable
          cab_data.fillna(cab_data.median(), inplace = True)

In [746]: #checking missing values again
          cab_data.isnull().sum()

Out[746]: dropoff_latitude     0
          dropoff_longitude    0
          fare_amount          0
          passenger_count      0
          pickup_datetime      0
          pickup_latitude      0
          pickup_longitude     0
          dtype: int64
```

Figure 2.14 missing value after treatment

Now our data is free of missing values

2.2.5 Creating new variables from existing variables

As we can see the data set has limited number of variables that are not much sufficient for a modal to take as input specially because date column cannot be used in data modle because of its type

We will be creating new varables from date and longitude latitude data namely-> hour,day of week, day of month,week, month,year,H Distance

```
In [755]: def prepare_time_features(df, drop=False):
              df["hour"] = df.pickup_datetime.dt.hour
              df["day_of_week"] = df.pickup_datetime.dt.weekday
              df["day_of_month"] = df.pickup_datetime.dt.day
              df["week"] = df.pickup_datetime.dt.week
              df["month"] = df.pickup_datetime.dt.month
              df["year"] = df.pickup_datetime.dt.year - 2000  # Reducing to 2 digits for less memory usage

              if drop:
                  df.drop(columns=['pickup_datetime'], inplace=True)


              return df

          cab_data= prepare_time_features(cab_data, True)

          cab_data_test = prepare_time_features(cab_data_test, True)

In [756]: cab_data.describe()
```

Figure 2.15 extracting date information

H distance is **Haversine distance**

haversine(θ) = $\sin^2(\theta/2)$

Eventually, the formual boils down to the following where ϕ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371km) to include latitude and longitude coordinates (A and B in this case).

a = $\sin^2((\phi B - \phi A)/2) + \cos \phi A \, . \cos \phi B \, . \sin^2((\lambda B - \lambda A)/2)$

c = $2 * \mathrm{atan2}(\sqrt{a}, \sqrt{(1-a)})$

d = $R \cdot c$

d = Haversine distance

Refer this page for more info and examples on Haversine formula

This distance is the distance in KM and is calculated by pickup latitude longitude and dropoff latitude and longitude

```
In [757]: def haversine_distance(lat1, long1, lat2, long2):
              data = [cab_data, cab_data_test]
              for i in data:
                  R = 6371  #radius of earth in kilometers
                  #R = 3959 #radius of earth in miles
                  phi1 = np.radians(i[lat1])
                  phi2 = np.radians(i[lat2])

                  delta_phi = np.radians(i[lat2]-i[lat1])
                  delta_lambda = np.radians(i[long2]-i[long1])

                  #a = sin²((φB - φA)/2) + cos φA . cos φB . sin²((λB - λA)/2)
                  a = np.sin(delta_phi / 2.0) ** 2 + np.cos(phi1) * np.cos(phi2) * np.sin(delta_lambda / 2.0) ** 2

                  #c = 2 * atan2( √a, √(1-a) )
                  c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))

                  #d = R*c
                  d = (R * c) #in kilometers
                  i['H_Distance'] = d
              return d

In [758]: haversine_distance('pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude')
```

Figure 2.16 creating H_distance

Going for 2nd check of cleaning and analysing we found that there are observations with 0 H distance again an absurd observation because no costumer would pay for a 0 distance ride.

As these data points are not much in number we will drop them from further analysis

```
In [761]: #as we can see there are some observations with zero distance
          #we need to treat them as zero distance is abstract for an costumer to pay fare for
          #depending on no of zero observations we would drop them or impute them with help of missing value analysis.
          cab_data["H_Distance"][cab_data["H_Distance"]==0].count()

Out[761]: 156

In [762]: cab_data_test["H_Distance"][cab_data_test["H_Distance"]==0].count()

Out[762]: 85

In [763]: #0.9 and 0.8 percent of the data set is having Hdistance as zero this value can be droped
          cab_data=cab_data[cab_data["H_Distance"]>0]
          cab_data_test=cab_data_test[cab_data_test["H_Distance"]>0]
```

Figure 2.17 dropping 0 distance observations

- Remember every analysis on train data is also performed on test data

2.2.6 Univariate and bivariate analysis

- Lets start with passanger count

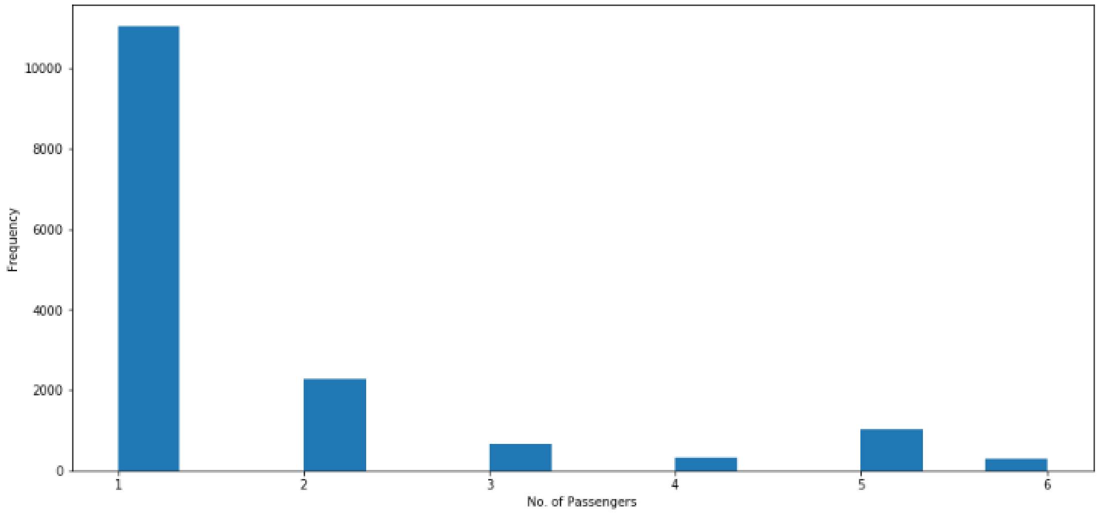


Figure 2.18 passenger amount vs frequency

figure clearly shows that most of the cab rides wer taken by single peronale

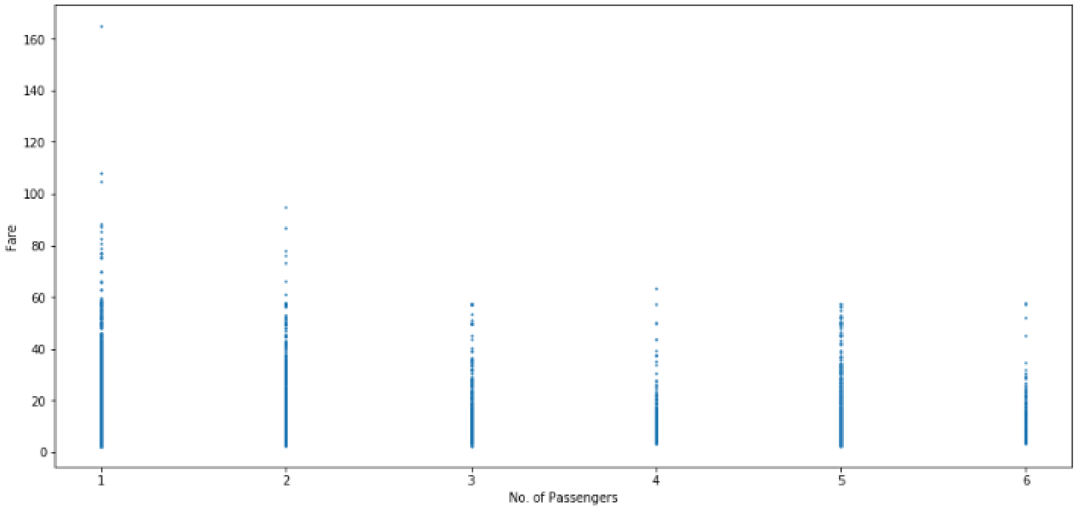


Figure 2.19 passenger amount vs Fare

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

- Day of month

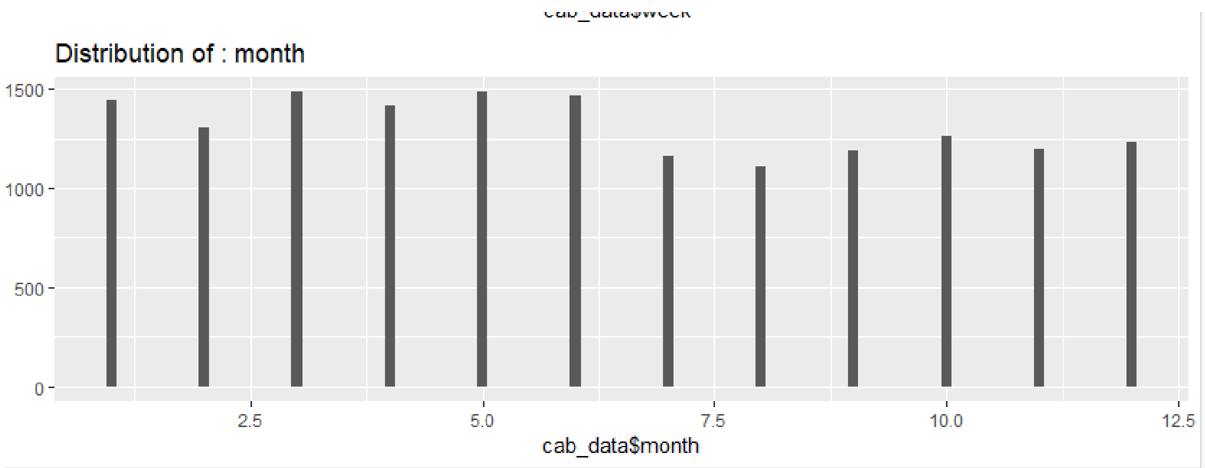


Figure 2.20 day of month vs frequency

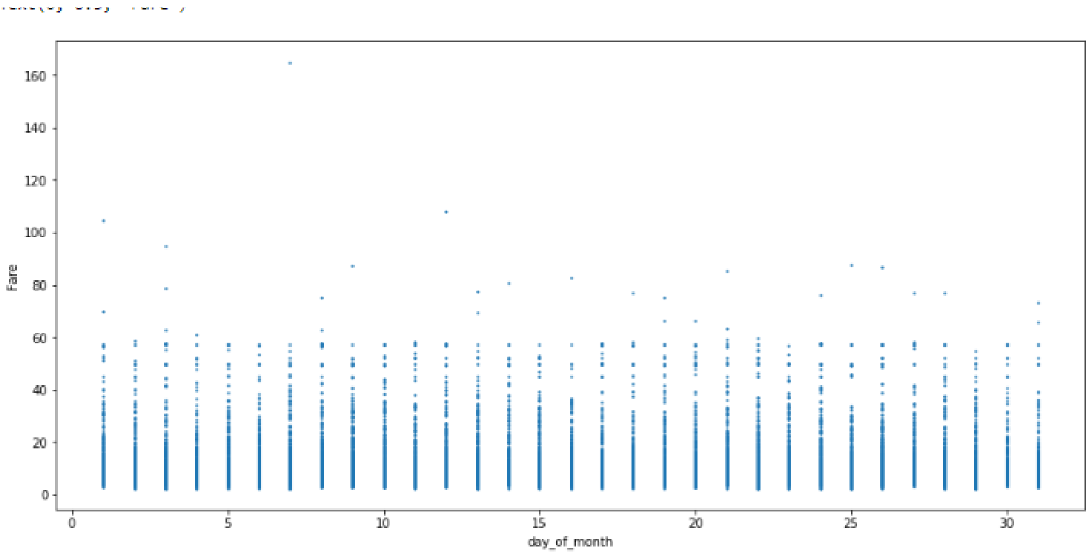


Figure 2.21 day of month vs fare

It shows that mostly the day of month is uniform in terms of no of rides and fare,

- Day of week

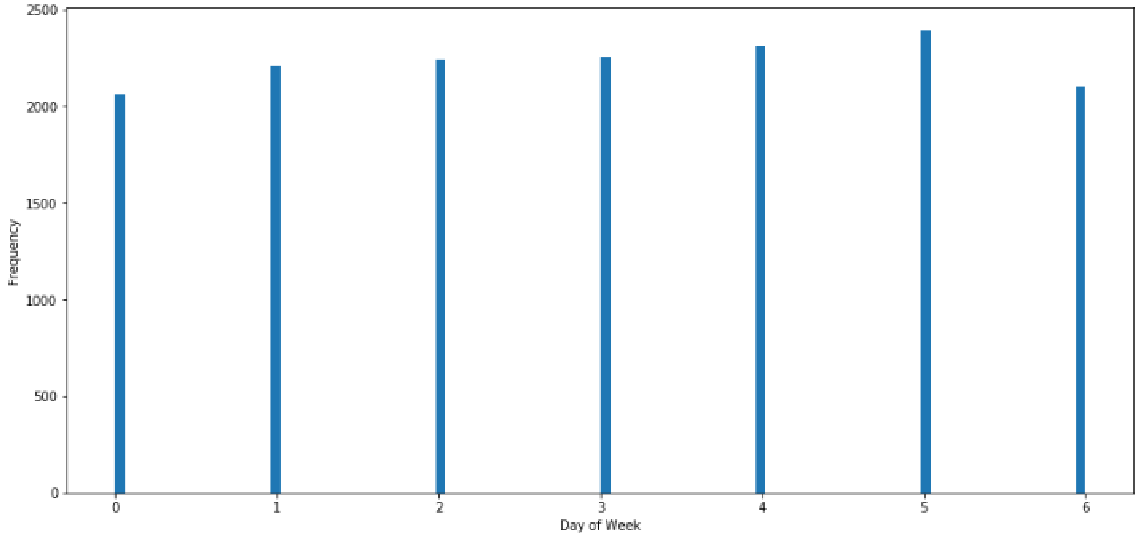


Figure 2.22 day of week vs frequency

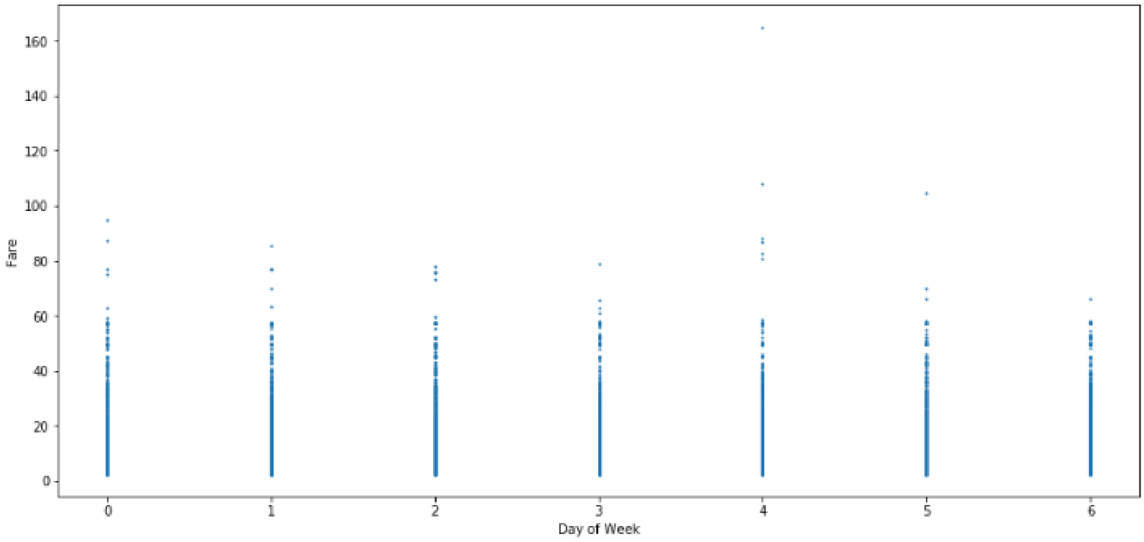


Figure 2.23 day of week vs fare

The highest fares seem to be high weekdays on and the lowest on and sunday almost an uniform distibution

- Hour

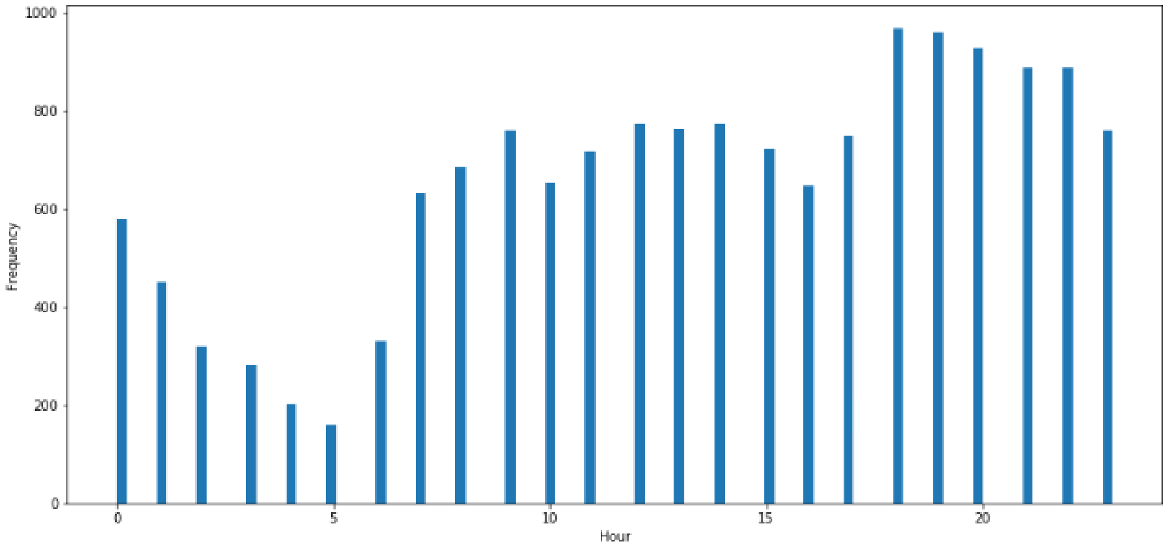


Figure 2.24 hour vs frequency

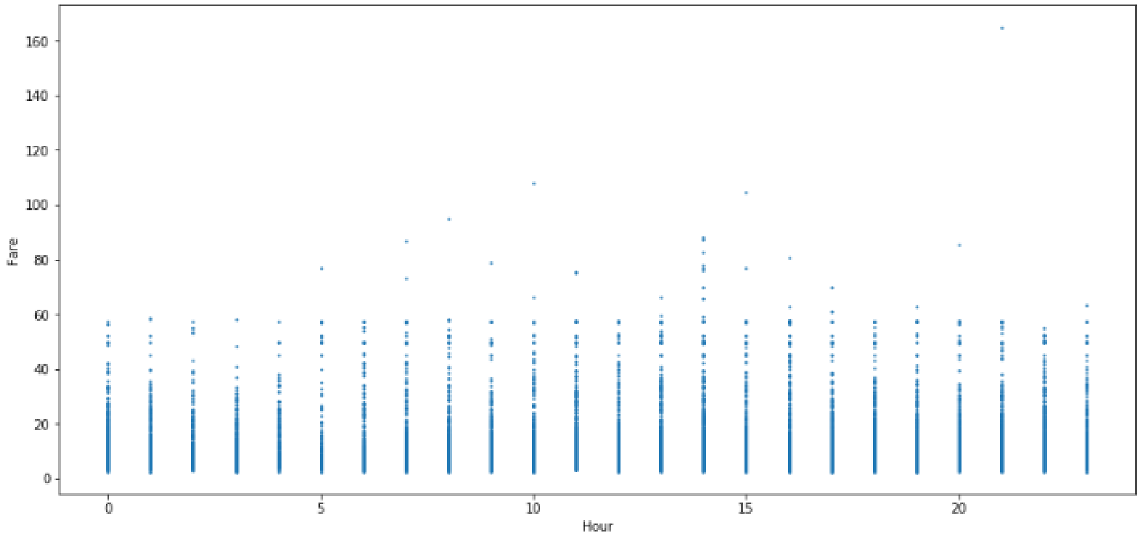


Figure 2.25 hour vs fare

Interesting! The time of day definitely plays an important role. The frequency of cab rides seem to be the lowest at 5AM and the highest at 7PM
The fares, however, seem to be high betweeb 5AM and 10AM, and 2PM to 4PM. Maybe people who live far away prefer to leave earlier to avoid rush hour traffic?

- H distane

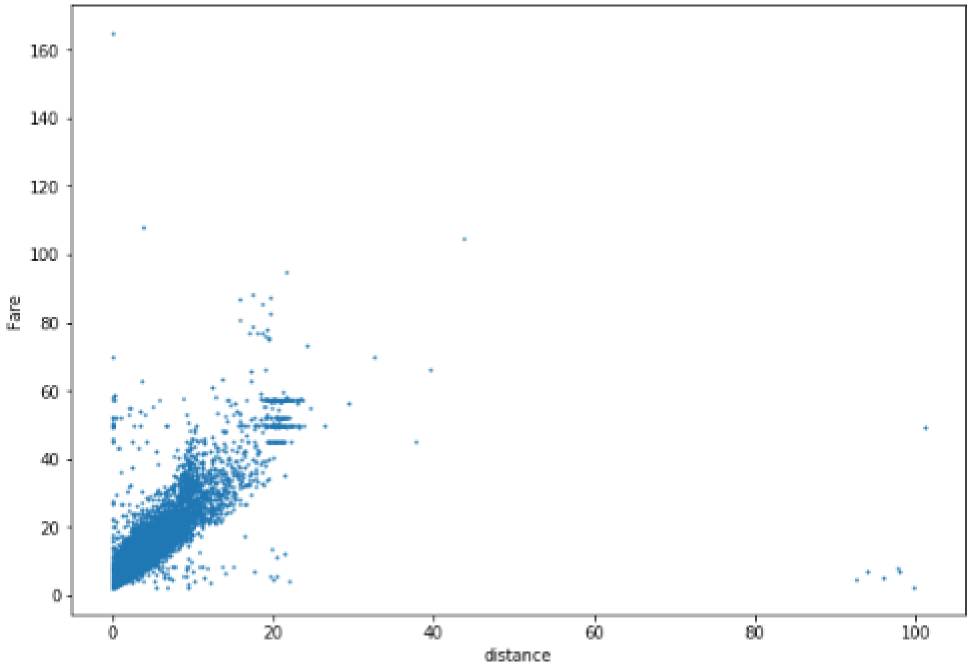


Figure 2.26 h_distance vs fare

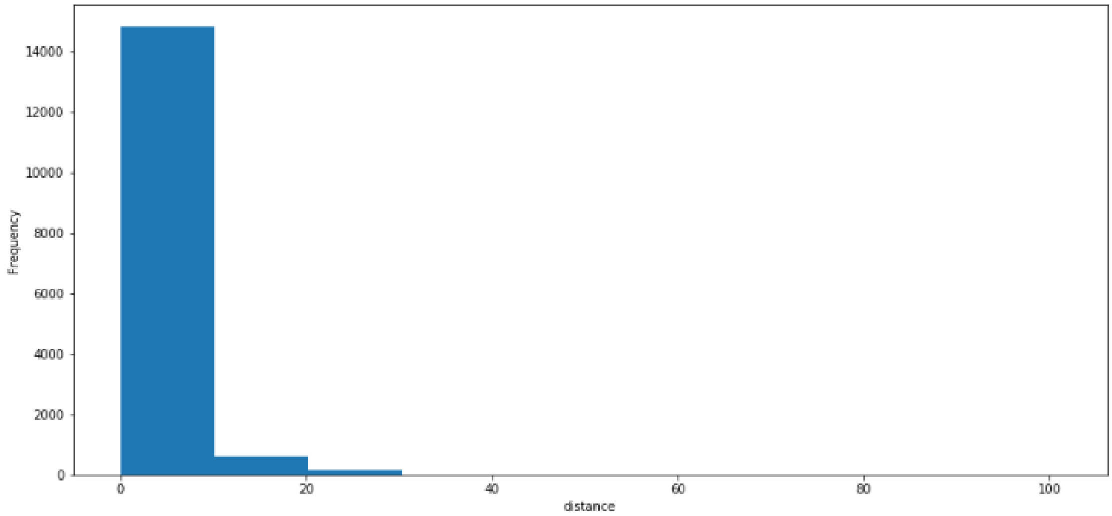


Figure 2.27 H_distance vs frequency

Fare is directly proportional to the distance with few outliers that cannot be removed because we can think of possible situation like in case of late night and unavailability irrespective of low distance cab charged high fare

2.2.7 Features Selections

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

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

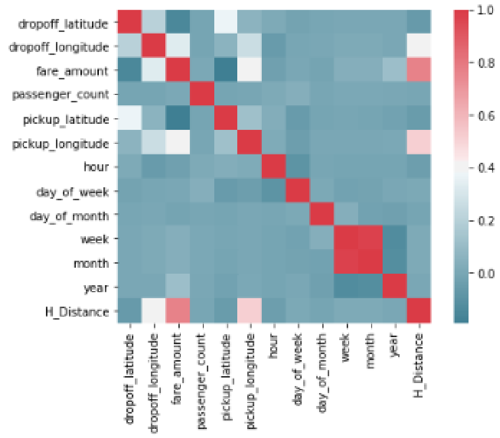
Corrgram : it help us visualize the data in correlation matrices. correlograms are implimented through the **corrgram(x, order = , panel=, lower.panel=, upper.panel=, text.panel=, diag.panel=)**

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

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

Below fig 2.28illustrates that relationship between all numeric variables using Corrgram plot For train data

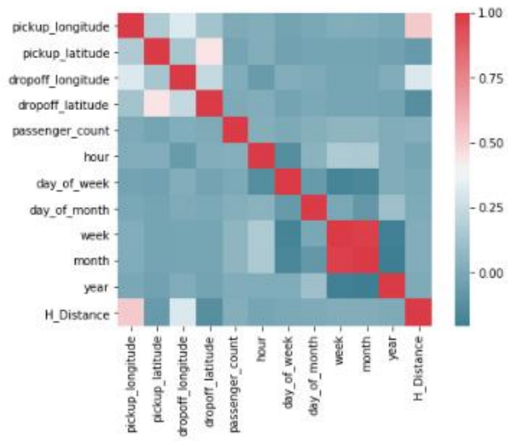
dropoff_longitude	fare_amount	passenger_count	pickup_latitude	pickup_longitude	hour	day_of_week	day_of_month	week	month	year
0.207031	0.144519	0.00377614	0.374562	0.0654771	0.0192099	0.0151573	0.000600593	0.00607006	0.00516605	0.00211637
1	0.33487	0.00426327	0.05999	0.262401	0.0505424	0.00067268	0.00629073	0.0162644	0.017147	0.00556064
0.33487	1	0.00870483	0.188378	0.394142	0.0293913	0.00278293	0.00763756	0.0396973	0.041358	0.115673
0.00426327	0.00870483	1	0.00573307	0.00150718	0.0176572	0.0421869	0.00176315	0.0111648	0.0094056	0.00255079
0.05999	0.188378	0.00573307	1	0.125249	0.0361773	0.049207	0.00551491	0.00263375	0.00271425	0.0191803
0.262401	0.394142	0.00150718	0.125249	1	0.0210253	0.0380527	0.00275952	0.00369939	0.00495907	0.00677318
0.0505424	0.0293913	0.0176572	0.0361773	0.0210253	1	0.0883851	0.000714279	0.00432403	0.00420904	0.00526942
0.00067268	0.00278293	0.0421869	0.049207	0.0380527	0.0883851	1	0.0140327	0.0175755	0.0161068	0.00477613
0.00629073	0.00763756	0.00176315	0.00551491	0.00275952	0.000714279	0.0140327	1	0.0408084	0.0175288	0.0259138
0.0162644	0.0396973	0.0111648	0.00263375	0.00369939	0.00432403	0.0175755	0.0408084	1	0.973625	0.124371
0.017147	0.041358	0.0094056	0.00271425	0.00495907	0.00420904	0.0161068	0.0175288	0.973625	1	0.116454
0.00556064	0.115673	0.00255079	0.0191803	0.00677318	0.00526942	0.00477613	0.0259138	0.124371	0.116454	1
0.394567	0.767307	0.000233448	0.0455561	0.5195	0.0393075	0.0129446	0.0058199	0.0185135	0.0201458	0.00461734



Below fig 2.29 illustrates that relationship between all numeric variables using Corrgram plot for test data

774]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day_of_week	day_of_month	week	m
pickup_longitude	1	0.167505	0.31371	0.130764	0.0105762	0.0189693	0.0268096	0.00625769	0.0176209	0.016
pickup_latitude	0.167505	1	0.1427	0.444648	0.0235318	0.0241912	0.0306391	0.0212845	0.0223248	0.021
dropoff_longitude	0.31371	0.1427	1	0.235937	0.0178654	0.0623241	0.0249899	0.0109758	0.0204272	0.018
dropoff_latitude	0.130764	0.444648	0.235937	1	0.00955993	0.0223828	0.0253852	0.00106414	0.0199351	0.019
passenger_count	0.0105762	0.0235318	0.0178654	0.00955993	1	0.0332818	0.00635001	0.0355267	0.0612535	0.060
hour	0.0189693	0.0241912	0.0623241	0.0223828	0.0332818	1	0.129198	0.0483618	0.166698	0.16
day_of_week	0.0268096	0.0306391	0.0249899	0.0253852	0.00635001	0.129198	1	0.0705047	0.18006	0.16
day_of_month	0.00625769	0.0212845	0.0109758	0.00106414	0.0355267	0.0483618	0.0705047	1	0.00337627	0.077
week	0.0176209	0.0223248	0.0204272	0.0199351	0.0612535	0.166698	0.18006	0.00337627	1	0.97
month	0.0169242	0.0210635	0.0182791	0.0193404	0.0607393	0.161429	0.165385	0.0776814	0.979438	
year	0.00571657	0.0333778	0.0132798	0.029105	0.0129831	0.0126432	0.0147182	0.114427	0.191375	0.20
H_Distance	0.528954	0.0714421	0.309356	0.119786	0.0279968	0.0197777	0.000615338	0.00910044	0.0256275	0.023



From correlation analysis we can drop of day of week and day of month variables as they have very low correlation with target variable fare amount.

Also as during above analysis we have already dropped pickup date while extracting other variables from it

In [775]:
```
# same conclutions can be drawn for test data thus applying same opperations on test data as well
cab_data=cab_data.drop(["day_of_month","day_of_week"],axis=1)
cab_data_test=cab_data_test.drop(["day_of_month","day_of_week"],axis=1)
print(cab_data.shape)
print(cab_data_test.shape)
```

```
(15570, 11)
(9829, 10)
```

Figure 2.30 final count of observations

Chapter 3

Modelling

3.1 Model Selection

In out earlier stage of analysis we have come to understand that few variables like H_Distance, passanger_count ,hour are going to play key role in model development , for model development dependent variable may fall under below categories

i. Nominal
ii. Ordinal
iii. Interval
iv. Ratio

In our case dependent variable is interval so, the predictive analysis that we can perform is Regression Analysis

We will start our model building from Decision Tree and the go on higher and complex model like random forest finally we would select one out of them which would be best fit for our data and predict fare for test data.

3.1.1 Evaluating Regression Model

The main concept of looking at what is called **residuals** or difference between our predictions f(x[I,]) and actual outcomes y[i].

We are using two methods to evaluating performance of model

i. **MAPE**: (Mean Absolute Percent Error) measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.

$$\left(\frac{1}{n}\sum\frac{|Actual - Forecast|}{|Actual|}\right)*100$$

ii. **RMSE:** (Root Mean Square Error) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

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

After doing these pre-processing we need to train our model so that we can predict the outcome in future. Here we need to split our data and then train our model.

Splitting data: we need to divide the data into train(90 percent) and test(10 percent)**.**

Modelling

```
In [777]: #for modling lets first crat test and train split out of train data given to us for acuracy
          X=cab_data.iloc[:,:10]
          Y=cab_data.iloc[:,10].values
          print(X.shape)
          print(Y.shape)

          (15570, 10)
          (15570,)
```

```
In [803]:  from sklearn.model_selection import train_test_split
          train_X, val_X, train_y, val_y =train_test_split(X,Y,test_size=0.1,random_state=0)
```

Model selection: we need to decide which model we need to use for our data. The target variable in our model is a continuous variable. So the models that we choose are Decision Tree and Random Forest, Linear Regression, OLS(python). The error metric chosen for the given problem statement is mean absolute error.

3.2 Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

```
In [804]: #applying discetion tree modle for regression first
          #Load libraries
          from sklearn.model_selection import train_test_split
          from sklearn.tree import DecisionTreeRegressor
          from sklearn import metrics
          #Decision tree for regression
          # Train the model using the training sets
          fit_DT = DecisionTreeRegressor(max_depth=2).fit(train_X, train_y)
          # make the predictions by the model
          predictions_DT = fit_DT.predict(val_X).round(0)
          # data frame for actual and predicted values
          df_dt = pd.DataFrame({'actual': val_y, 'pred': predictions_DT})
          print(df_dt.head())
          #Calculate MAPE
          def MAPE(y_true, y_pred):
              mape = np.mean(np.abs(( y_true - y_pred) / y_true))*100
              return mape
          mape=MAPE(val_y, predictions_DT)
          # Calculate and display accuracy
          accuracy = 100 - np.mean(mape)
          # errors and accuracy
          print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
          print('Accuracy:', round(accuracy, 2), '%.')

             actual  pred
          0     5.0   7.0
          1     8.0   7.0
          2     6.5   7.0
          3     3.7   7.0
          4     5.5   7.0
          MEAN ABSOLUTE ERROR:28.902476909886172%
          Accuracy: 71.1 %.
```

Figure 3.2.1 Decision Tree Algorithm

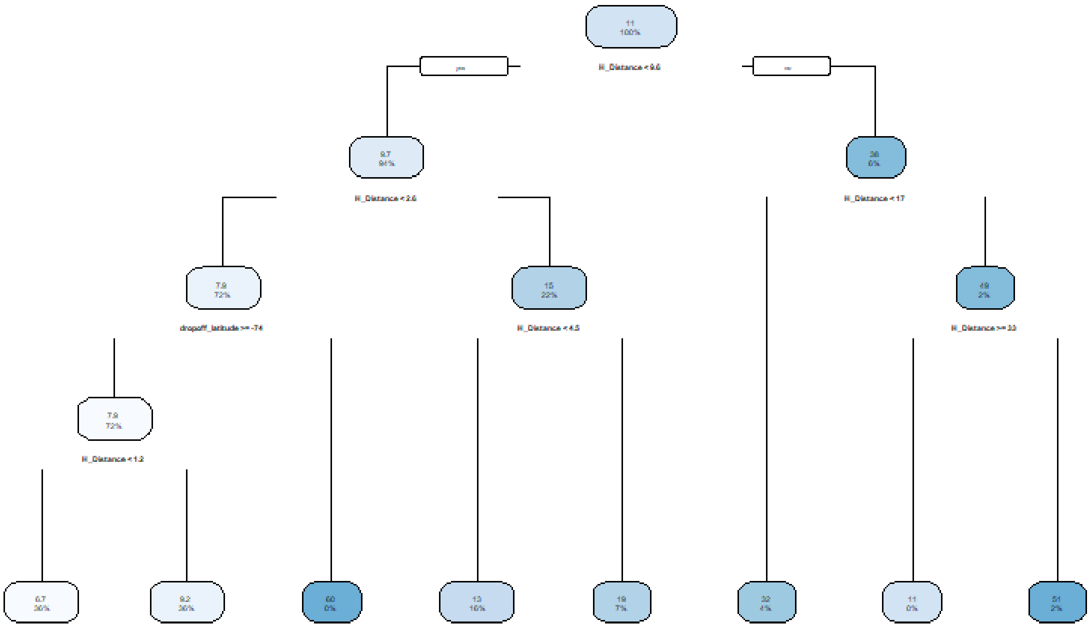


Figure 3.2.2 Graphical Representation of Decision tree

.

3.2.1 Evaluation of Decision Tree Model

```
def MAPE(y_true, y_pred):
    mape = np.mean(np.abs(( y_true - y_pred) / y_true))*100
    return mape
mape=MAPE(val_y, predictions_DT)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
# errors and accuracy
print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
print('Accuracy:', round(accuracy, 2), '%.')
```

```
   actual  pred
0     5.0   7.0
1     8.0   7.0
2     6.5   7.0
3     3.7   7.0
4     5.5   7.0
MEAN ABSOLUTE ERROR:28.902476909886172%
Accuracy: 71.1 %.
```

Figure 3.2.3 Evaluation of Decision Tree using MAPE

Decision tree builds regression is 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.

```
Out[805]: <seaborn.axisgrid.FacetGrid at 0x1e52abda128>
```

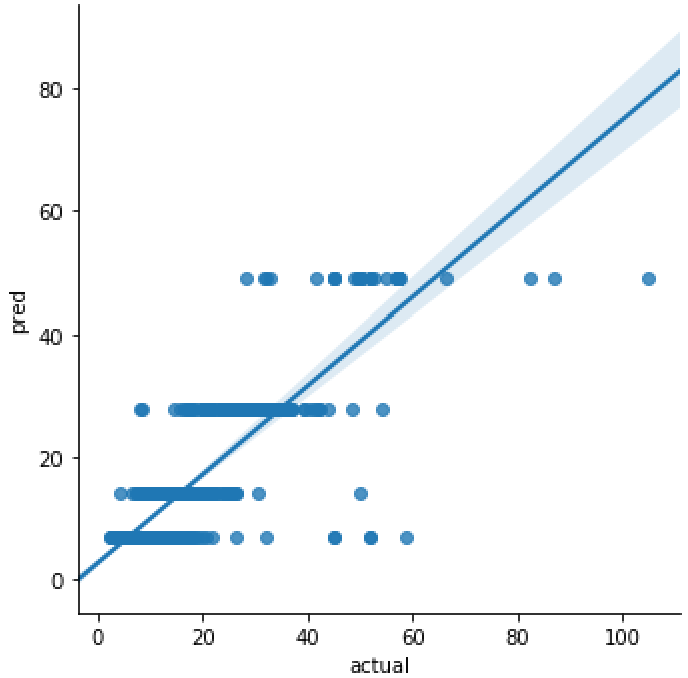


Fig-3.2.4 shows a curve between predicted and actual test data for target variable

	Mae	Accuracy= (1-mae)*100
Python	28.9%	71.1%
R	24.7%	76.77%

3.3 Linear regression

The technique uses statistical calculations to plot a trend line in a set of data points. The trend line could be anything from the number of people diagnosed with skin cancer to the financial performance of a company. Linear regression shows a relationship between an independent variable and a dependent variable being studied.

strong multicollinearity or other numerical problems.

```
In [809]: # make the predictions by the model
          predictions_LR = model.predict(val_X)
          # data frame for actual and predicted values
          df_LR = pd.DataFrame({'actual': val_y, 'pred': predictions_LR})
          print(df_LR.head())
          # Calculate and display accuracy
          mape=MAPE(val_y,predictions_LR)
          accuracy = 100 - np.mean(mape)
          print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
          print('Accuracy:', round(accuracy, 2), '%.')

                actual      pred
          9133     5.0  6.201580
          9697     8.0  9.540742
          6831     6.5  4.492493
          1114     3.7  6.066121
          2854     5.5  9.176965
          MEAN ABSOLUTE ERROR:27.346006302880166%
          Accuracy: 72.65 %.
```

Figure 3.3.1 Linear Regression Model

3.3.2 Evaluation of Linear regression Model

```
# Calculate and display accuracy
mape=MAPE(val_y,predictions_LR)
accuracy = 100 - np.mean(mape)
print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
print('Accuracy:', round(accuracy, 2), '%.')

      actual      pred
9133     5.0  6.201580
9697     8.0  9.540742
6831     6.5  4.492493
1114     3.7  6.066121
2854     5.5  9.176965
MEAN ABSOLUTE ERROR:27.346006302880166%
Accuracy: 72.65 %.
```

Figure 3.3.2 Evaluation of Regression Model

Out[810]: <seaborn.axisgrid.FacetGrid at 0x1e52b0efa58>

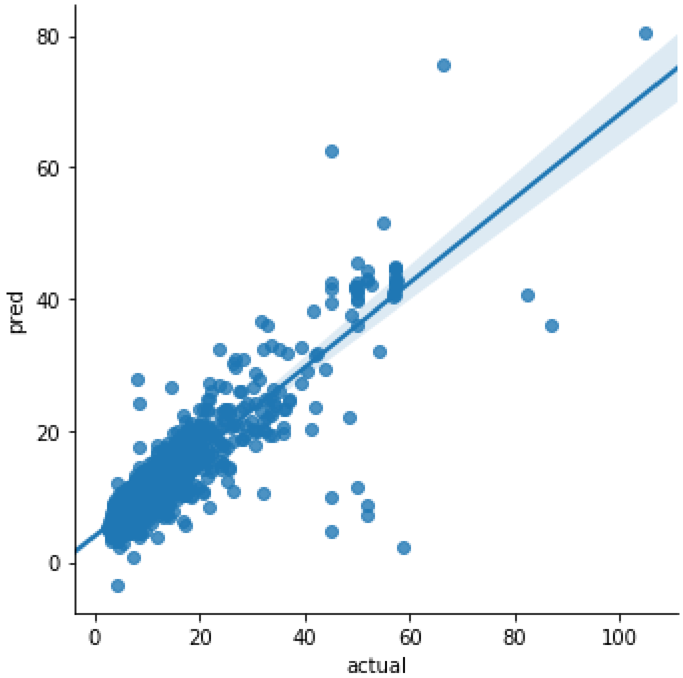


Fig-3.3.3 shows a curve between predicted and actual test data for target variable

	Mae	Accuracy= (1-mae)*100
Python	27.34%	72.65%
R	35.7%	64.3%

3.4 OLS

Multiple linear regression is the most common form of linear regression analysis. As a predictive analysis, the multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical.

VIF (Variance Inflation factor) : It quantifies the multicollinearity between the independent variables.

As Linear regression will work well if multicollinearity between the Independent variables are less.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.837
Model:	OLS	Adj. R-squared:	0.837
Method:	Least Squares	F-statistic:	7174.
Date:	Wed, 10 Jul 2019	Prob (F-statistic):	0.00
Time:	01:28:36	Log-Likelihood:	-44797.
No. Observations:	14013	AIC:	8.961e+04
Df Residuals:	14003	BIC:	8.969e+04
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
pickup_longitude	-16.1390	1.235	-13.065	0.000	-18.560	-13.718
pickup_latitude	-31.6878	1.754	-18.071	0.000	-35.125	-28.251
dropoff_longitude	-4.4253	1.223	-3.618	0.000	-6.823	-2.028
dropoff_latitude	-5.7024	1.639	-3.478	0.001	-8.916	-2.489
passenger_count	0.0428	0.039	1.086	0.278	-0.034	0.120
hour	0.0063	0.008	0.822	0.411	-0.009	0.021
week	0.0171	0.015	1.169	0.242	-0.012	0.046
month	0.0328	0.063	0.518	0.605	-0.091	0.157
year	0.5833	0.027	21.625	0.000	0.530	0.636
H_Distance	1.7899	0.013	141.356	0.000	1.765	1.815

Omnibus:	12183.980	Durbin-Watson:	1.988
Prob(Omnibus):	0.000	Jarque-Bera (JB):	34449751.245
Skew:	-2.752	Prob(JB):	0.00
Kurtosis:	245.841	Cond. No.	5.35e+03

Figure 3.4.1 Multi collinearity between Independent variables

In the above figure it is showing there is strong correlation between no two independent variable, we need to consider all variable

In [816]:
```
#MEAN ABSOLUTE ERROR:28.29157149054831%
#Accuracy: 71.71 %
```

In [807]:
```
# Linear Regression OLS
#Import libraries for LR
import statsmodels.api as sm
# Train the model using the training sets
model = sm.OLS(train_y,train_X.astype(float)).fit()
```

In [808]:
```
#Summary of model
model.summary()
```

```
In [809]: # make the predictions by the model
          predictions_LR = model.predict(val_X)
          # data frame for actual and predicted values
          df_LR = pd.DataFrame({'actual': val_y, 'pred': predictions_LR})
          print(df_LR.head())
          # Calculate and display accuracy
          mape=MAPE(val_y,predictions_LR)
          accuracy = 100 - np.mean(mape)
          print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
          print('Accuracy:', round(accuracy, 2), '%.')

      actual      pred
9133     5.0  6.201580
9697     8.0  9.540742
6831     6.5  4.492493
1114     3.7  6.066121
2854     5.5  9.176965
MEAN ABSOLUTE ERROR:27.346006302880166%
Accuracy: 72.65 %.
```

Figure 3.4.2 Linear Regression Model

3.4.2 Evaluation of Linear regression Model OLS

```
# Calculate and display accuracy
mape=MAPE(val_y,predictions_LR)
accuracy = 100 - np.mean(mape)
print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
print('Accuracy:', round(accuracy, 2), '%.')

      actual      pred
9133     5.0  6.201580
9697     8.0  9.540742
6831     6.5  4.492493
1114     3.7  6.066121
2854     5.5  9.176965
MEAN ABSOLUTE ERROR:27.346006302880166%
Accuracy: 72.65 %.
```

Figure 3.4.3 Evaluation of Regression Model

Out[810]: <seaborn.axisgrid.FacetGrid at 0x1e52b0efa58>

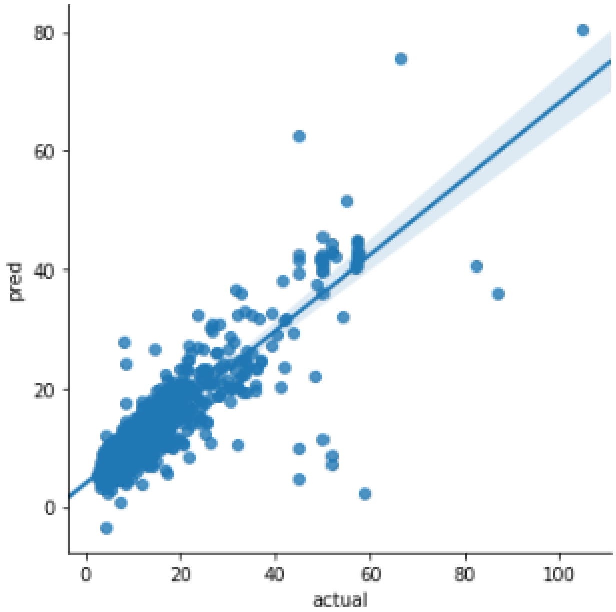


Fig-3.4.4 shows a curve between predicted and actual test data for target variable

	Mae	Accuracy= (1-mae)*100
Python	27.34%	72.65%

3.5 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Figure 3.5.1 Random Forest Implementation

```
In [811]: #Random forest for regression
#Import libraries for RF
from sklearn.ensemble import RandomForestRegressor
# Train the model using the training sets
RFmodel = RandomForestRegressor(n_estimators=200,min_samples_leaf=5,random_state=0).fit(train_X, train_y)
# make the predictions by the model
RF_Predictions = RFmodel.predict(val_X)
# data frame for actual and predicted values
df_RF = pd.DataFrame({'actual': val_y, 'pred': RF_Predictions})
print(df_RF.head())
# Calculate and display accuracy
def MAPE(y_true, y_pred):
    mape = np.mean(np.abs(( y_true - y_pred) / y_true))*100
    return mape

mape=MAPE(val_y,RF_Predictions)
accuracy = 100 - np.mean(mape)
print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
print('Accuracy:', round(accuracy, 2), '%.')
```

```
   actual      pred
0     5.0  6.426116
1     8.0  9.677449
2     6.5  4.902457
3     3.7  4.123403
4     5.5  7.329475
MEAN ABSOLUTE ERROR:18.65273337278679%
Accuracy: 81.35 %.
```

3.3.1 Evaluation of Random Forest

```
# Calculate and display accuracy
def MAPE(y_true, y_pred):
    mape = np.mean(np.abs(( y_true - y_pred) / y_true))*100
    return mape

mape=MAPE(val_y,RF_Predictions)
accuracy = 100 - np.mean(mape)
print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
print('Accuracy:', round(accuracy, 2), '%.')
```

```
   actual      pred
0     5.0  6.426116
1     8.0  9.677449
2     6.5  4.902457
3     3.7  4.123403
4     5.5  7.329475
MEAN ABSOLUTE ERROR:18.65273337278679%
Accuracy: 81.35 %.
```

Figure 3.5.2 Random Forest Evaluation

Out[813]: <seaborn.axisgrid.FacetGrid at 0x1e52b195160>

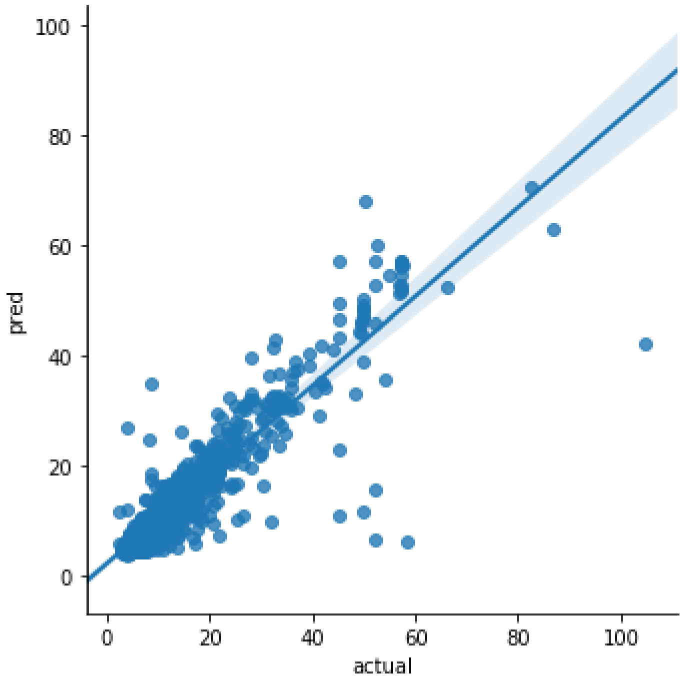


Fig-3.5.3 shows a curve between predicted and actual test data for target variable

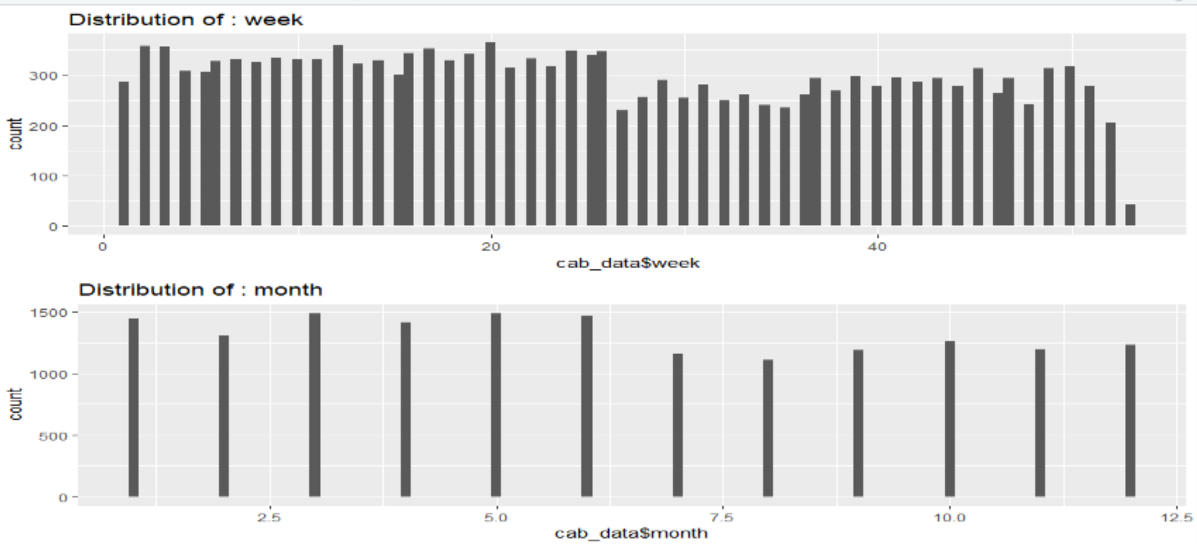
	Mae	Accuracy= (1-mae)*100
Python	18.65%	81.35%
R	19.2%	80.8%

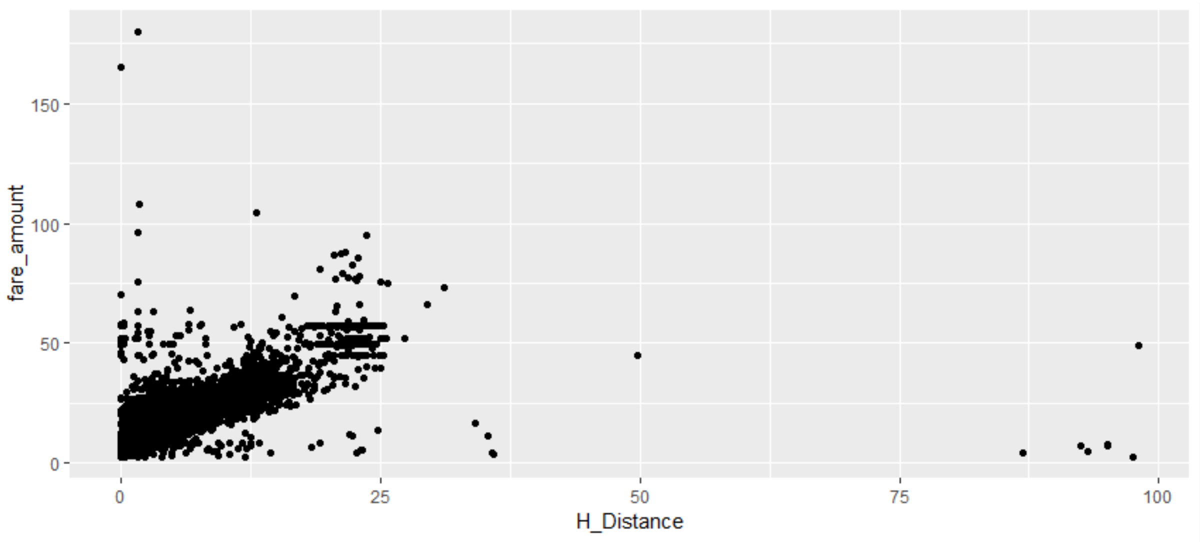
Model Selection

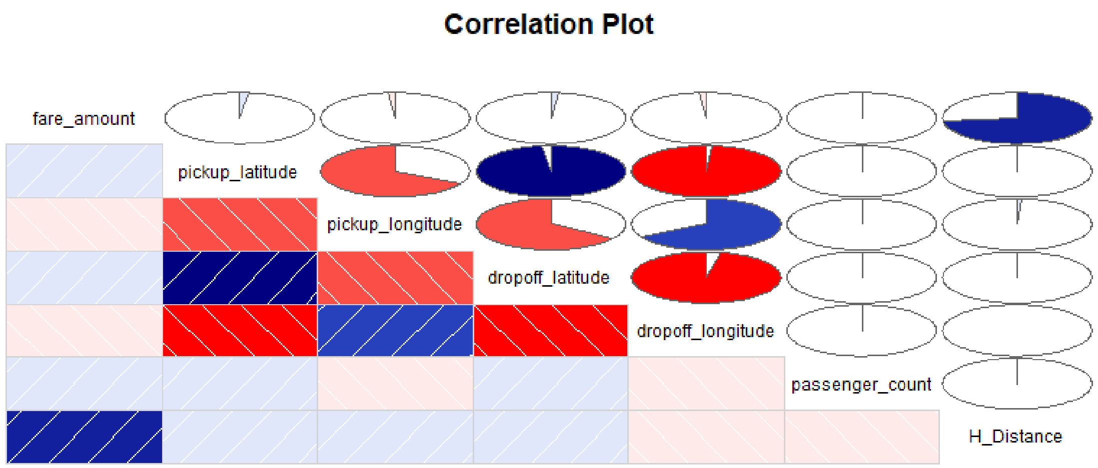
As we predicted counts for Cab fare using three Models Decision Tree, Random Forest and Linear Regression as MAPE is high and RMSE is less for the Linear regression Model so conclusion is

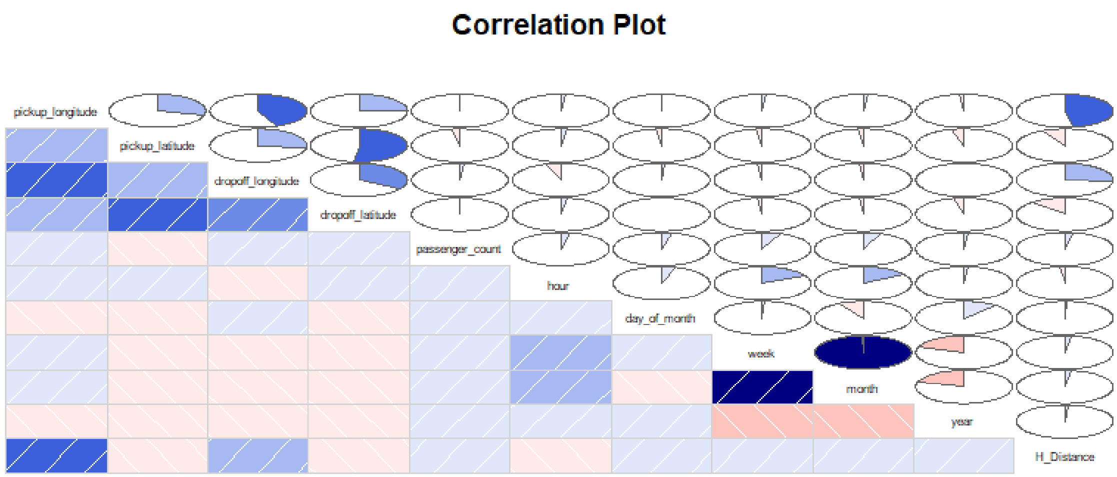
Conclusion: - For the Cab Fare Data Random forest Model is best model to predict the count.

Appendix A- Extra Figures









Appendix B - Python Code

Univariate Analysis (Fig: 2.18->, 2.27)

```
In [ ]: #Let us start aalysis between the variables
        #Does the number of passengers affect the fare?
        plt.figure(figsize=(15,7))
        plt.hist(cab_data['passenger_count'], bins=15)
        plt.xlabel('No. of Passengers')
        plt.ylabel('Frequency')
```

```
In [ ]: plt.figure(figsize=(15,7))
        plt.scatter(x=cab_data['passenger_count'], y=cab_data['fare_amount'], s=1.5)
        plt.xlabel('No. of Passengers')
        plt.ylabel('Fare')
```

#From the above 2 graphs we can see that single passengers are the most frequent travellers, and the highest fare also see just 1 passenger.

```
In [ ]: #Does the date and time of pickup affect the fare?
        plt.figure(figsize=(15,7))
        plt.scatter(x=cab_data['day_of_month'], y=cab_data['fare_amount'], s=1.5)
        plt.xlabel('day_of_month')
        plt.ylabel('Fare')
```

```
In [ ]: #The fares throught the month mostly seem uniform, with the maximum fare received on the 16th
        plt.figure(figsize=(15,7))
        plt.hist(cab_data['hour'], bins=100)
        plt.xlabel('Hour')
        plt.ylabel('Frequency')
```

```
In [ ]: plt.figure(figsize=(15,7))
        plt.scatter(x=cab_data['hour'], y=cab_data['fare_amount'], s=1.5)
        plt.xlabel('Hour')
        plt.ylabel('Fare')
```

```
In [ ]: #Does the day of the week affect the fare?
        plt.figure(figsize=(15,7))
        plt.hist(cab_data['day_of_week'], bins=100)
        plt.xlabel('Day of Week')
        plt.ylabel('Frequency')
```

```
In [ ]: #day of the week doesn't seem to have that much of an influence on the number of cab rides
        plt.figure(figsize=(15,7))
        plt.scatter(x=cab_data['day_of_week'], y=cab_data['fare_amount'], s=1.5)
        plt.xlabel('Day of Week')
        plt.ylabel('Fare')
```

#The highest fares seem to be high weekdays on and the lowest on and sunday almost an uniform distibution

4. Does the distance affect the fare?

This is a no-brainer. I am confident that the distance would affect the fare a great deal. But I will visualise it.

```
In [ ]: #day of the week doesn't seem to have that much of an influence on the number of cab rides
        plt.figure(figsize=(10,7))
        plt.scatter(x=cab_data['H_Distance'], y=cab_data['fare_amount'], s=1.5)
        plt.xlabel('distance')
        plt.ylabel('Fare')
```

```
In [ ]: #Does the day of the week affect the fare?
        plt.figure(figsize=(15,7))
        plt.hist(cab_data['H_Distance'])
        plt.xlabel('distance')
        plt.ylabel('Frequency')
```

Feature selection (Fig: 2.28& 2.29)

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

        #Generate correlation matrix
        corr = cab_data.corr()

        #Plot using seaborn library
        sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
                    square=True, ax=ax)
        # Create correlation matrix
        corr_matrix = cab_data.corr().abs()
        corr_matrix.style.background_gradient(cmap='GnBu_r')
```

#from correlation analysis we can drop of day of week and day of month variables as they hav very low correlation with target variable fare ammount

#also as during above analysis we have already droped pickup date while extracting other variables from it

```
In [ ]: #correlation analysis for numeric variables
        #Let us do the same analysis for test data
        f, ax = plt.subplots(figsize=(7, 5))

        #Generate correlation matrix
        corr = cab_data_test.corr()

        #Plot using seaborn library
        sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
                    square=True, ax=ax)
        # Create correlation matrix
        corr_matrix = cab_data_test.corr().abs()
        corr_matrix.style.background_gradient(cmap='GnBu_r')
```

Decision Tree

```
#Accuracy: 71.1 %

In [ ]: sns.lmplot(x='actual', y='pred', data = df_dt ,fit_reg = True)
```

Linear regression

```
In [ ]: sns.lmplot(x='actual', y='pred', data = df_LR ,fit_reg = True)

In [ ]: #Random forest for regression
```

Random forest

```
In [ ]: #MEAN ABSOLUTE ERROR:18.652733372786792%
        #Accuracy: 81.35 %

In [ ]: # df_RF.plot.scatter(x='actual', y='pred')
        sns.lmplot(x='actual', y='pred', data = df_RF ,fit_reg = True)

In [ ]: #random forrest will be best fit
        #for test data lets predict fare
```

cab_fare_pridicton

```
In [ ]: #loading important libraries
        import os
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [ ]: #changing the directory according to need
        os.chdir("D:\gaggi")
        os.getcwd()
```

```
In [ ]: #loading data both test and train.
        cab_data=pd.read_csv('train_cab.csv')
        cab_data_test =pd.read_csv('test.csv')
```

```
In [ ]: #starting intial analysis on data
        #number of rows and columns
        cab_data.shape
```

```
In [ ]: cab_data_test.shape
```

```
In [ ]: #so we have 16067 records in train data and 9914 records in test data initially
        #viewing first 5 rows of the data
        cab_data.head()
```

```
In [ ]: #lets check dtypes for train data
        cab_data.dtypes
```

```
In [ ]: #fare amount has to be converted to float for analysis
        #fare value "430-" is to be cleaned before to convert it in numerical data type
        cab_data["fare_amount"]=cab_data["fare_amount"].str.replace("430-","430")
        cab_data["fare_amount"]=cab_data["fare_amount"].astype(float)
```

```
In [ ]: #checking again
        cab_data.dtypes
```

```
In [ ]: cab_data.describe()
```

```
In [ ]:  #from above analysis we can draw few insights about data.
             #there are missing values in the data
             #max fare "54343" and max pessenger"5345" sounds absard for a cab
        #continuing further analysis for test data as well.
        cab_data_test.dtypes
```

```
In [ ]: #data types for test daya seems apt
        #test data looks okay at 1st glance no missing values no visable outliers as of now
        cab_data_test.describe()
```

UNIVARIAT analysis

```
In [ ]: #lets start with passenger_count as observed it consists of outliers.
        # from common understanding a cab cannot have max passanger count more than 6.
        #we should consider max passanger count as 6
        #also 0 passanger also sounds absard.
        #pessenger should always be an whole number so all with desimal pessenger are also outliers.
        cab_data["passenger_count"][cab_data["passenger_count"]>6].count()
```

```
In [ ]: cab_data["passenger_count"][cab_data["passenger_count"]<1].count()
```

```
In [ ]: #there ares some odd 78 outliers in passenger_count column .
        #we could treat the outliers in 2 posible ways 11 droping them completly along with whole row
        #second we can convert them into NA values and treat them as missing values. that is populating them baised on data left
        #we are using second way as it results in less loss of data
        #there are already missing values present int passenger_count column thus we will be deaaling with all missing values at once.
        cab_data["passenger_count"].isnull().sum()
```

```
In [ ]: #coverting into NA
        cab_data["passenger_count"].loc[cab_data["passenger_count"]>6]=np.nan
        cab_data["passenger_count"].loc[cab_data["passenger_count"]<1]=np.nan
```

```
In [ ]: cab_data["passenger_count"].describe()
```

```
In [ ]: #Fare_ammount
        cab_data["fare_amount"].describe()
```

```
In [ ]: #initial analysis tells us that even fare ammount variable consists of missing values
        #there are absard outliers also like fare cannot be negative
        #also 54343$ for a cab fare is absard
        # again we would convert the outliers into missing values to avaoid any data loss or atleast minimize it
        # doing some research we are considering the max fare as $300 and min fare to be $2.5
        cab_data["fare_amount"][cab_data["fare_amount"]<2.5].count()
```

```
In [ ]: cab_data["fare_amount"][cab_data["fare_amount"]>300].count()
```

```
In [ ]: #there are around 11 odd outliers lets convert them to NA
        cab_data["fare_amount"].loc[cab_data["fare_amount"]<2.5]=np.nan
        cab_data["fare_amount"].loc[cab_data["fare_amount"]>300]=np.nan
```

```
In [ ]: cab_data["fare_amount"].isnull().sum()
```

```
In [ ]: #we have 35 missing values in fare ammount
        cab_data["fare_amount"].describe()
```

```
In [ ]: #continuing with distance data i.e longitude and longitude
        #intial analysis gives (00)latitude longitude on googling seems absard as it is somewher in the otion thats not fissable for cab
        #0 value lat long would be outliers wich have to be treated
        #also we need to consider gernal otliers regarding min and max lat longitude
        #ther might be an posibility of inversion of latitude and longitude in the data
        cab_data.describe()
```

```
In [ ]: #lets find the the min max rnge for latitude and longitude from test data as it is almost free from any outliers
        #max and min longitude from test data
        lon_min=min(cab_data_test.pickup_longitude.min(),cab_data_test.dropoff_longitude.min())
        lon_max=max(cab_data_test.pickup_longitude.max(),cab_data_test.dropoff_longitude.max())
        print(lon_min,',',lon_max)
```

```
In [ ]: #max and min longitude from test data
        lat_min=min(cab_data_test.pickup_latitude.min(),cab_data_test.dropoff_latitude.min())
        lat_max=max(cab_data_test.pickup_latitude.max(),cab_data_test.dropoff_latitude.max())
        print(lat_min,',',lat_max)
```

```
In [ ]: #let us find outliers on bases of this range
        def select_outside_boundingbox(df, BB):
            filter_df = df.loc[(df['pickup_longitude'] < BB[0]) | (df['pickup_longitude'] > BB[1]) | \
                    (df['pickup_latitude'] < BB[2]) | (df['pickup_latitude'] > BB[3]) | \
                    (df['dropoff_longitude'] < BB[0]) | (df['dropoff_longitude'] > BB[1]) | \
                    (df['dropoff_latitude'] < BB[2]) | (df['dropoff_latitude'] > BB[3])]

            return filter_df


        BB = (-74.5, -72.8, 40.5, 41.8)
```

```
In [ ]: latlon_outliers = select_outside_boundingbox(cab_data, BB)
        latlon_outliers.head()
```

```
In [ ]: latlon_outliers.shape
```

In [ ]:
```
#aroound 348 outliers exist in lat long
#lets further our analysis
latlon_outliers.describe()
```

In [ ]:
```
#lets first deal with zeros in lat and long data
#we are deleting all zero as the zero cordinate lies in ocean thats absard in tiself for a cab to travel
def drop_0s(df, verbose=False):
    if verbose:
        print("Dropping all rows with 0s:")
        old_size = len(df)
        print("Old size: {}".format(old_size))

    df = df.loc[~(df == 0).any(axis=1)]


    if verbose:
        new_size = len(df)
        print("New size: {}".format(new_size))
        difference = old_size - new_size
        percent = (difference / old_size) * 100
        print("Dropped {} records, or {:.2f}%".format(difference, percent))

    return df

latlon_outliers = drop_0s(latlon_outliers, True)

latlon_outliers.describe()
```

In [ ]:
```
#as we can see many rows have values inverted  for latitude and longitude thesr data rows can be usefull if we could fix this
def select_within_boundingbox(df, BB):
    filter_df = df.loc[(df['pickup_longitude'] >= BB[0]) & (df['pickup_longitude'] <= BB[1]) & \
           (df['pickup_latitude'] >= BB[2]) & (df['pickup_latitude'] <= BB[3]) & \
           (df['dropoff_longitude'] >= BB[0]) & (df['dropoff_longitude'] <= BB[1]) & \
           (df['dropoff_latitude'] >= BB[2]) & (df['dropoff_latitude'] <= BB[3])]


    return filter_df

inverted_BB = (40.5, 41.8, -74.5, -72.8)

inverted_outliers = select_within_boundingbox(latlon_outliers, inverted_BB)

inverted_outliers.describe()
```

In [ ]:
```
def swap_inverted(df):
    fixed_df = df.rename(columns={'pickup_longitude' : 'pickup_latitude', 'pickup_latitude' : 'pickup_longitude',
                                  'dropoff_longitude' : 'dropoff_latitude', 'dropoff_latitude' : 'dropoff_l
    col_list = fixed_df.columns.tolist()

    col_list[3], col_list[4], col_list[5], col_list[6] = col_list[4], col_list[3], col_list[6], col_list[5]

    fixed_df = fixed_df[col_list]

    return fixed_df

fixed_outliers = swap_inverted(inverted_outliers)

fixed_outliers.describe()
```

In [ ]:
```
## Now we'll remove all rows with a datapoint that doesn't fall within the bounding box for NYC coordinates

print("Old size: {}".format(len(cab_data)))

cab_data =cab_data.loc[(cab_data['pickup_longitude'] >= BB[0]) & (cab_data['pickup_longitude'] <= BB[1]) & \
             (cab_data['pickup_latitude'] >= BB[2]) & (cab_data['pickup_latitude'] <= BB[3]) & \
             (cab_data['dropoff_longitude'] >= BB[0]) & (cab_data['dropoff_longitude'] <= BB[1]) & \
             (cab_data['dropoff_latitude'] >= BB[2]) & (cab_data['dropoff_latitude'] <= BB[3])]


print("New size: {}".format(len(cab_data)))

cab_data.describe()
```

In [ ]:
```
#concatinating the filtered otliers with the data
cab_data_copy = cab_data # Created a copy so as to avoid the possibility of adding the fixed outliers multiple times

cab_data = pd.concat([cab_data_copy, fixed_outliers],ignore_index=True,sort=True)

cab_data_copy = None  # Doing this to try to be a bit more memory efficient

cab_data.describe()
```

In [ ]:
```
cab_data.describe()
```

In [ ]:
```
#after dealing with latitude longitude data lts first trat missing values in the pessanger and fare variables
cab_data.isnull().sum()
```

In [ ]:
```
#creating adata frame missing
missing_val= pd.DataFrame(cab_data.isnull().sum())
```

In [ ]:
```
#reseting the index
missing_val = missing_val.reset_index()
#Rename variable
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})

#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(cab_data))*100
missing_val
```

In [ ]:
```
#there are 0.2 percent and 0.8 percent missing values in fare and passenger respectively
#creating copy of cab data for imputation purpose
df1= cab_data.copy()
df2= cab_data.copy()
df3= cab_data.copy()
```

In [ ]:
```
df1.iloc[6,2]
```

In [ ]:
```
df1.iloc[6,2]=np.nan
df2.iloc[6,2]=np.nan
df3.iloc[6,2]=np.nan
```

In [ ]:
```
df1.iloc[6,2]
```

In [ ]:
```
#using mean method
df1['fare_amount'] = df1['fare_amount'].fillna(df1['fare_amount'].mean())
df1.iloc[6,2]
```

In [ ]:
```
#using median method
df2['fare_amount'] = df2['fare_amount'].fillna(df2['fare_amount'].median())
df2.iloc[6,2]
```

In [ ]:
```
#using interpolat
df3['fare_amount'] = df3['fare_amount'].interpolate(method = 'nearest', limit_direction = 'both')
df3.iloc[6,2]
```

In [ ]:
```
#usinng median methord for imputation of passenger variable and fare variable
cab_data.fillna(cab_data.median(), inplace = True)
```

In [ ]:
```
#checking missing values again
cab_data.isnull().sum()
```

In [ ]:
```
#this concludes 1st phase of data cleaning
#lets start ourEDA for the data
#lets stars by converting pickup date timestamp to different other usefull columns
#also lets apply feature enginnering to check and convert variables to right data format
cab_data.describe()
```

In [ ]:
```
 cab_data.dtypes
```

In [ ]:
```
#lets check for test data also
cab_data["pickup_datetime"]=cab_data["pickup_datetime"].str.replace("UTC","")
cab_data_test["pickup_datetime"]=cab_data_test["pickup_datetime"].str.replace("UTC","")
cab_data_test.dtypes
```

In [ ]:
```
cab_data["passenger_count"]=cab_data["passenger_count"].astype(int)
cab_data['pickup_datetime']  = pd.to_datetime(cab_data['pickup_datetime'],format='%Y-%m-%d %H:%M:%S', errors='coerce')
cab_data_test['pickup_datetime']  = pd.to_datetime(cab_data_test['pickup_datetime'],format='%Y-%m-%d %H:%M:%S', errors='coerce')
```

In [ ]:
```
cab_data.dtypes
```

In [ ]:
```
cab_data_test.dtypes
```

In [ ]:
```
cab_data.isnull().sum()
```

In [ ]:
```
#drop the missing values
cab_data = cab_data.drop(cab_data[cab_data.isnull().any(1)].index, axis = 0)
```

```
In [ ]: def prepare_time_features(df, drop=False):
            df["hour"] = df.pickup_datetime.dt.hour
            df["day_of_week"] = df.pickup_datetime.dt.weekday
            df["day_of_month"] = df.pickup_datetime.dt.day
            df["week"] = df.pickup_datetime.dt.week
            df["month"] = df.pickup_datetime.dt.month
            df["year"] = df.pickup_datetime.dt.year - 2000  # Reducing to 2 digits for less memory usage

            if drop:
                df.drop(columns=['pickup_datetime'], inplace=True)



            return df

        cab_data= prepare_time_features(cab_data, True)

        cab_data_test = prepare_time_features(cab_data_test, True)
```

```
In [ ]: cab_data.describe()
```

```
In [ ]: def haversine_distance(lat1, long1, lat2, long2):
            data = [cab_data, cab_data_test]
            for i in data:
                R = 6371  #radius of earth in kilometers
                #R = 3959 #radius of earth in miles
                phi1 = np.radians(i[lat1])
                phi2 = np.radians(i[lat2])

                delta_phi = np.radians(i[lat2]-i[lat1])
                delta_lambda = np.radians(i[long2]-i[long1])

                #a = sin²((φB - φA)/2) + cos φA . cos φB . sin²((λB - λA)/2)
                a = np.sin(delta_phi / 2.0) ** 2 + np.cos(phi1) * np.cos(phi2) * np.sin(delta_lambda / 2.0) ** 2

                #c = 2 * atan2( √a, √(1-a) )
                c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))

                #d = R*c
                d = (R * c) #in kilometers
                i['H_Distance'] = d
            return d
```

```
In [ ]: haversine_distance('pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude')
```

```
In [ ]: cab_data.describe()
```

```
In [ ]: cab_data_test.describe()
```

```
In [ ]: #as we can see there are some observations with zero distance
        #we need to treat them as zero distance is abstract for an costumer to pay fare for
        #depending on no of zero observations we would drop them or impute them with help of missing value analysis.
        cab_data["H_Distance"][cab_data["H_Distance"]==0].count()
```

```
In [ ]: cab_data_test["H_Distance"][cab_data_test["H_Distance"]==0].count()
```

```
In [ ]: #0.9 and 0.8 percent of the data set is having Hdistance as zero this value can be droped
        cab_data=cab_data[cab_data["H_Distance"]>0]
        cab_data_test=cab_data_test[cab_data_test["H_Distance"]>0]
```

```
In [ ]: #let us start aalysis between the variables
        #Does the number of passengers affect the fare?
        plt.figure(figsize=(15,7))
        plt.hist(cab_data['passenger_count'], bins=15)
        plt.xlabel('No. of Passengers')
        plt.ylabel('Frequency')
```

```
In [ ]: plt.figure(figsize=(15,7))
        plt.scatter(x=cab_data['passenger_count'], y=cab_data['fare_amount'], s=1.5)
        plt.xlabel('No. of Passengers')
        plt.ylabel('Fare')
```

In [ ]:
```
#Does the date and time of pickup affect the fare?
plt.figure(figsize=(15,7))
plt.scatter(x=cab_data['day_of_month'], y=cab_data['fare_amount'], s=1.5)
plt.xlabel('day_of_month')
plt.ylabel('Fare')
```

In [ ]:
```
#The fares throught the month mostly seem uniform, with the maximum fare received on the 16th
plt.figure(figsize=(15,7))
plt.hist(cab_data['hour'], bins=100)
plt.xlabel('Hour')
plt.ylabel('Frequency')
```

In [ ]:
```
plt.figure(figsize=(15,7))
plt.scatter(x=cab_data['hour'], y=cab_data['fare_amount'], s=1.5)
plt.xlabel('Hour')
plt.ylabel('Fare')
```

In [ ]:
```
#Does the day of the week affect the fare?
plt.figure(figsize=(15,7))
plt.hist(cab_data['day_of_week'], bins=100)
plt.xlabel('Day of Week')
plt.ylabel('Frequency')
```

In [ ]:
```
#day of the week doesn't seem to have that much of an influence on the number of cab rides
plt.figure(figsize=(15,7))
plt.scatter(x=cab_data['day_of_week'], y=cab_data['fare_amount'], s=1.5)
plt.xlabel('Day of Week')
plt.ylabel('Fare')
```

In [ ]:
```
#day of the week doesn't seem to have that much of an influence on the number of cab rides
plt.figure(figsize=(10,7))
plt.scatter(x=cab_data['H_Distance'], y=cab_data['fare_amount'], s=1.5)
plt.xlabel('distance')
plt.ylabel('Fare')
```

In [ ]:
```
#Does the day of the week affect the fare?
plt.figure(figsize=(15,7))
plt.hist(cab_data['H_Distance'])
plt.xlabel('distance')
plt.ylabel('Frequency')
```

Feature selection

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

#Generate correlation matrix
corr = cab_data.corr()

#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
# Create correlation matrix
corr_matrix = cab_data.corr().abs()
corr_matrix.style.background_gradient(cmap='GnBu_r')
```

In [ ]:
```
#correlation analysis for numeric variables
#let us do the same analysis for test data
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
corr = cab_data_test.corr()

#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
# Create correlation matrix
corr_matrix = cab_data_test.corr().abs()
corr_matrix.style.background_gradient(cmap='GnBu_r')
```

In [ ]:
```
# same conclutions can be drawn for test data thus applying same opperations on test data as well
cab_data=cab_data.drop(["day_of_month","day_of_week"],axis=1)
cab_data_test=cab_data_test.drop(["day_of_month","day_of_week"],axis=1)
print(cab_data.shape)
print(cab_data_test.shape)
```

In [ ]:
```
#rearranging the colums for test data
cab_data = cab_data.reindex(columns=["pickup_longitude","pickup_latitude","dropoff_longitude","dropoff_latitude","passenger_coun
```

Modelling

In [ ]:
```
#for modling lets first crat test and train split out of train data given to us for acuracy
X=cab_data.iloc[:,:10]
Y=cab_data.iloc[:,10].values
print(X.shape)
print(Y.shape)
```

In [ ]:
```
 from sklearn.model_selection import train_test_split
train_X, val_X, train_y, val_y =train_test_split(X,Y,test_size=0.1,random_state=0)
```

```
In [ ]: #applying discetion tree modle for regression first
        #Load libraries
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeRegressor
        from sklearn import metrics
        #Decision tree for regression
        # Train the model using the training sets
        fit_DT = DecisionTreeRegressor(max_depth=2).fit(train_X, train_y)
        # make the predictions by the model
        predictions_DT = fit_DT.predict(val_X).round(0)
        # data frame for actual and predicted values
        df_dt = pd.DataFrame({'actual': val_y, 'pred': predictions_DT})
        print(df_dt.head())
        #Calculate MAPE
        def MAPE(y_true, y_pred):
            mape = np.mean(np.abs(( y_true - y_pred) / y_true))*100
            return mape
        mape=MAPE(val_y, predictions_DT)
        # Calculate and display accuracy
        accuracy = 100 - np.mean(mape)
        # errors and accuracy
        print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
        print('Accuracy:', round(accuracy, 2), '%.')
```

```
In [ ]: sns.lmplot(x='actual', y='pred', data = df_dt ,fit_reg = True)
```

```
In [ ]: # linear
        from sklearn.linear_model import LinearRegression
        lr_model = LinearRegression().fit(train_X, train_y)
        #predict
        lr_prediction = lr_model.predict(val_X)
        df_lr = pd.DataFrame({'actual':val_y, 'prediction':lr_prediction})
        # Calculate and display accuracy
        mape=MAPE(val_y, lr_prediction)
        # Calculate and display accuracy
        accuracy = 100 - np.mean(mape)
        # errors and accuracy
        print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
        print('Accuracy:', round(accuracy, 2), '%.')
```

```
In [ ]: # Linear Regression OLS
        #Import libraries for LR
        import statsmodels.api as sm
        # Train the model using the training sets
        model = sm.OLS(train_y,train_X.astype(float)).fit()
```

```
In [ ]: #Summary of model
        model.summary()
```

```
In [ ]: # make the predictions by the model
        predictions_LR = model.predict(val_X)
        # data frame for actual and predicted values
        df_LR = pd.DataFrame({'actual': val_y, 'pred': predictions_LR})
        print(df_LR.head())
        # Calculate and display accuracy
        mape=MAPE(val_y,predictions_LR)
        accuracy = 100 - np.mean(mape)
        print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
        print('Accuracy:', round(accuracy, 2), '%.')
```

In [ ]:
```
sns.lmplot(x='actual', y='pred', data = df_LR ,fit_reg = True)
```

In [ ]:
```
#Random forest for regression
#Import libraries for RF
from sklearn.ensemble import RandomForestRegressor
# Train the model using the training sets
RFmodel = RandomForestRegressor(n_estimators=200,min_samples_leaf=5,random_state=0).fit(train_X, train_y)
# make the predictions by the model
RF_Predictions = RFmodel.predict(val_X)
# data frame for actual and predicted values
df_RF = pd.DataFrame({'actual': val_y, 'pred': RF_Predictions})
print(df_RF.head())
# Calculate and display accuracy
def MAPE(y_true, y_pred):
    mape = np.mean(np.abs(( y_true - y_pred) / y_true))*100
    return mape

mape=MAPE(val_y,RF_Predictions)
accuracy = 100 - np.mean(mape)
print("MEAN ABSOLUTE ERROR:"+str(mape)+"%")
print('Accuracy:', round(accuracy, 2), '%.')
```

In [ ]:
```
# df_RF.plot.scatter(x='actual', y='pred')
sns.lmplot(x='actual', y='pred', data = df_RF ,fit_reg = True)
```

In [ ]:
```
#random forrest will be best fit
#for test data lets predict fare
X_test=cab_data_test
print(X_test.columns)
print(X_test.shape)
print(X_test.dtypes)
```

In [ ]:
```
Regression =RandomForestRegressor(n_estimators=70,min_samples_leaf=5,random_state=0)
Regression.fit(X,Y)
# make the predictions by the model
y_test_pred=Regression.predict(X_test)
print(y_test_pred)
# data frame for predicted values
cab_data_test["predicted_fare"]=y_test_pred
cab_data_test.head()
```

In [ ]:
```
#output
cab_data_test.to_csv("py_output.csv", index= False)
```

Complete R File

```
#Clean the environment

rm(list = ls())

#Setting the  working directory
setwd("D:/gaggi")


#get Working directory
getwd()

#loading the libraries which would be needed
libraries = c("dummies","caret","rpart.plot","plyr","dplyr",
"ggplot2","rpart","dplyr","DMwR","randomForest","usdm","DataCombine")
lapply(X = libraries,require, character.only = TRUE)


rm(libraries)

#read the csv file
cab_data = read.csv ("train_cab.csv", header = T)
cab_data_test = read.csv ("test.csv", header = T)

# let’s start our general analysis
str(cab_data)
summary(cab_data)
head(cab_data)

#from above analysis we can draw few insights about data.
#there are missing values in the data
#max fare "54343" and max pessenger"5345" sounds absurd for a cab
#continuing further analysis for test data as well.
```

```
str(cab_data_test)
summary(cab_data_test)

#data types for test days seems apt
#test data looks okay at 1st glance no missing values no visable outliers as of now
#feature enginnering
#let us first covert all variables in the train data in right data type

cab_data$pickup_datetime=gsub("UTC","", cab_data$pickup_datetime)
cab_data_test$pickup_datetime=gsub("UTC","", cab_data_test$pickup_datetime)
new_date=cab_data$pickup_datetime
new_date_test=cab_data_test$pickup_datetime
new_date=strptime(new_date,"%Y-%m-%d%H:%M:%S")
new_date_test=strptime(new_date_test,"%Y-%m-%d%H:%M:%S")
cab_data$pickup_datetime=new_date
cab_data_test$pickup_datetime=new_date_test
cab_data$fare_amount=as.numeric(as.character(cab_data$fare_amount))
cab_data$passenger_count=as.integer(cab_data$passenger_count)
str(cab_data)
str(cab_data_test)
summary(cab_data)

#UNIVARIAT analysis
#lets start with passenger_count as observed it consists of outliers.
# from common understanding a cab cannot have max passanger count more than 6.
#we should consider max passanger count as 6
#also 0 passanger also sounds absurd.

cab_data$passenger_count[cab_data$passenger_count < 0 | cab_data$passenger_count >6]=NaN
cab_data$fare_amount[cab_data$fare_amount < 2.5 | cab_data$fare_amount >300]=NaN

# doing some research we are considering the max fare as $300 and min fare to be $2.5
```

```
#we could treat the outliers in 2 posible ways 11 droping them completly along with whole row

#second we can convert them into NA values and treat them as missing values. that is populating them baised on data left

#we are using second way as it results in less loss of data

#there are already missing values present int passenger_count column thus we will be deaaling with all missing values at once.


summary(cab_data)


#let us move our focus on latitude and longitude data

#data seems curropt as it has many 00 cordinates that actualy lie in sea not posible for cab to drop or pickup

#also it seems that lat and long data for few observations are interchanged

#lets find the the min max rnge for latitude and longitude from test data as it is almost free from any outliers

#max and min longitude from test data


lon_min=min(min(cab_data_test$pickup_longitude),min(cab_data_test$dropoff_longitude))

lon_max=max(max(cab_data_test$pickup_longitude),max(cab_data_test$dropoff_longitude))

print(lon_min)

print(lon_max)


#max and min longitude from test data


lat_min=min(min(cab_data_test$pickup_latitude),min(cab_data_test$dropoff_latitude))

lat_max=max(max(cab_data_test$pickup_latitude),max(cab_data_test$dropoff_latitude))

print(lat_min)

print(lat_max)


#let us find outliers on bases of this range


BB = c(-74.5, -72.8, 40.5, 41.8)

latlon_outliers = cab_data[which ((cab_data$pickup_longitude < -74.5)|(cab_data$pickup_longitude > -72.8)|(cab_data$pickup_latitude < 40.5)|(cab_data$pickup_latitude > BB[4])|(cab_data$dropoff_longitude < BB[1])|(cab_data$dropoff_longitude > BB[2])|(cab_data$dropoff_latitude < BB[3])|(cab_data$dropoff_latitude > BB[4])),]

summary(latlon_outliers)
```

```
str(latlon_outliers)

head(latlon_outliers)


#lets first deal with zeros in lat and long data

#we are deleting all zero as the zero cordinate lies in ocean thats absard in tiself for a cab to travel


latlon_outliers=latlon_outliers[!(latlon_outliers$pickup_longitude==0 | latlon_outliers$dropoff_latitude==0 | latlon_outliers$dropoff_longitude==0 | latlon_outliers$dropoff_latitude==0),]

str(latlon_outliers)


#as we can see many rows have values inverted  for latitude and longitude thesr data rows can be usefull if we could fix this


latlon_outliers = latlon_outliers[which((latlon_outliers$pickup_longitude >=40.5)&(latlon_outliers$pickup_longitude <=41.8)&(latlon_outliers$pickup_latitude >=-74.5)&(latlon_outliers$pickup_latitude <=-72.8)&(latlon_outliers$dropoff_longitude >=40.5)&(latlon_outliers$dropoff_longitude <=41.8)&(latlon_outliers$dropoff_latitude >=-74.5)&(latlon_outliers$dropoff_latitude <=-72.8)),]

str(latlon_outliers)

head(latlon_outliers)

setnames(latlon_outliers, old=c("pickup_longitude","pickup_latitude","dropoff_longitude","dropoff_latitude"), new=c("pickup_latitude", "pickup_longitude","dropoff_latitude","dropoff_longitude"))

head(latlon_outliers)

colnames(latlon_outliers)

df=latlon_outliers[,c(1,2,4,3,6,5,7)]

head(df)


## Now we'll remove all rows with a datapoint that doesn't fall within the bounding box


cab_data = cab_data[!((cab_data$pickup_longitude >=BB[1])&(cab_data$pickup_longitude <=BB[2])&(cab_data$pickup_latitude >= BB[3])&(cab_data$pickup_latitude <= BB[4])&(cab_data$dropoff_longitude >= BB[1])&(cab_data$dropoff_longitude <= BB[2])&(cab_data$dropoff_latitude >= BB[3])&(cab_data$dropoff_latitude <= BB[4])),]

str(cab_data)

cab_data=cab_data[!(cab_data$pickup_longitude==0 | cab_data$dropoff_latitude==0 | cab_data$dropoff_longitude==0 | cab_data$dropoff_latitude==0),]
```

```
str(cab_data)

#no we will concate the filterd data with this data

cab_data_copy=copy(cab_data)
cab_data=rbind(cab_data,df)
rm(cab_data_copy)
summary(cab_data)

#after dealing with latitude longitude data lts first trat missing values in the pessanger and fare variables

missing_val = data.frame(apply(cab_data,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(cab_data)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val
cab_data=subset(cab_data,!is.na(cab_data$pickup_datetime))

#as we can see there are about 0.4  0.2 percent missing values in fare and passanger count respectively
##Create missing value and impute using mean, median and knn

df1 =cab_data
df2= cab_data
df1[10,1]
# here the value we have choosen to remove is 8.9
df1[10,1]=NA
df2[10,1]=NA
# checking for different values
#mean
df1[10,1] = mean(df1$fare_amount, na.rm = T)
```

```
df1[10,1] #the value we got is 11. 3

#median

df2[10,1] = median(df2$fare_amount, na.rm = T)

df2[10,1] #the value we got is 8.5

#thus imputing with median for whole data

#Median Method

cab_data$fare_amount[is.na(cab_data$fare_amount)] = median(cab_data$fare_amount, na.rm = T)

cab_data$passenger_count[is.na(cab_data$passenger_count)] = median(cab_data$passenger_count, na.rm =
T)

#Check if any missing values

sum(is.na(cab_data))

#this concludes 1st phase of data cleaning

#lets start ourEDA for the data

#lets stars by converting pickup date timestamp to different other usefull columns

#also distance column using latitude longitude data


str(cab_data_test)

library(lubridate)

new_date=cab_data$pickup_datetime

new_date=strptime(new_date,"%Y-%m-%d%H:%M:%S")

cab_data$pickup_datetime=new_date

cab_data$hour = hour(cab_data$pickup_datetime)

cab_data$day_of_week = weekdays (cab_data$pickup_datetime)

cab_data$day_of_month = day(cab_data$pickup_datetime)

cab_data$week = week(cab_data$pickup_datetime)

cab_data$month= month(cab_data$pickup_datetime)

cab_data$year = year(cab_data$pickup_datetime)-2000


#doing the same for test data


cab_data_test$hour = hour(cab_data_test$pickup_datetime)

cab_data_test$day_of_week = weekdays (cab_data_test$pickup_datetime)

cab_data_test$day_of_month = day(cab_data_test$pickup_datetime)
```

```
cab_data_test$week = week(cab_data_test$pickup_datetime)

cab_data_test$month= month(cab_data_test$pickup_datetime)

cab_data_test$year = year(cab_data_test$pickup_datetime)-2000


#creating distance column with latitude and longitude data


library(NISTunits)

  R = 6371  #radius of earth in kilometers

#R = 3959 #radius of earth in miles

phi1 =NISTdegTOradian(cab_data$pickup_latitude)

phi2 = NISTdegTOradian(cab_data$dropoff_latitude)

phi3=NISTdegTOradian(cab_data$pickup_longitude)

phi4= NISTdegTOradian(cab_data$dropoff_longitude)

delta_phi = phi2-phi1

delta_lambda = phi4-phi3

#a = sin²((ϕB - ϕA)/2) + cos ϕA . cos ϕB . sin²((λB - λA)/2)

a = sin(delta_phi / 2.0) ** 2 + cos(phi1) * cos(phi2) * sin(delta_lambda / 2.0) ** 2


#c = 2 * atan2( √a, √(1−a) )

c = 2 * atan2(sqrt(a), sqrt(1-a))


#d = R*c

d = (R * c) #in kilometers

cab_data$H_Distance=d


#doing the same for test data


Phi1 =NISTdegTOradian(cab_data_test$pickup_latitude)

Phi2 = NISTdegTOradian(cab_data_test$dropoff_latitude)

Phi3=NISTdegTOradian(cab_data_test$pickup_longitude)

Phi4= NISTdegTOradian(cab_data_test$dropoff_longitude)

Delta_phi = Phi2-Phi1

Delta_lambda = Phi4-Phi3
```

```
#a = sin²((ϕB - ϕA)/2) + cos ϕA . cos ϕB . sin²((λB - λA)/2)

A = sin(Delta_phi / 2.0) ** 2 + cos(Phi1) * cos(Phi2) * sin(Delta_lambda / 2.0) ** 2


#c = 2 * atan2( √a, √(1−a) )

C = 2 * atan2(sqrt(A), sqrt(1-A))


#d = R*c

D = (R * C) #in kilometers

cab_data_test$H_Distance=D

summary(cab_data)

summary(cab_data_test)


#as we can see there are observations that have zero distance

#as we donot know direct relationship of fare with distance we need to clean aur data of these 0 distances

cab_data$H_Distance[cab_data$H_Distance <= 0 | cab_data$H_Distance >100]=NaN

sum(is.na(cab_data))

cab_data$H_Distance[is.na(cab_data$H_Distance)] = median(cab_data$H_Distance, na.rm = T)


#let us start aalysis between the variables

#Does the number of passengers affect the fare?

#Check the distribution of numerical data using histogram

hist1 = ggplot(data = cab_data, aes(x =cab_data$fare_amount)) + ggtitle("Distribution of : fare") +
geom_histogram(bins = 100)

hist2 = ggplot(data = cab_data, aes(x =cab_data$passenger_count)) + ggtitle("Distribution of: passenger
count") + geom_histogram(bins = 100)

hist3 = ggplot(data = cab_data, aes(x =cab_data$hour)) + ggtitle("Distribution of: hour") +
geom_histogram(bins = 100)

#hist4 = ggplot(data = cab_data, aes(x =cab_data$day_of_week)) + ggtitle("Distribution of :day of week") +
geom_histogram(bins = 25)

hist4 = ggplot(data = cab_data, aes(x =cab_data$day_of_month)) + ggtitle("Distribution of :day of month") +
geom_histogram(bins = 100)

hist5 = ggplot(data = cab_data, aes(x =cab_data$week)) + ggtitle("Distribution of : week") +
geom_histogram(bins = 100)

hist6 = ggplot(data = cab_data, aes(x =cab_data$month)) + ggtitle("Distribution of : month") +
geom_histogram(bins = 100)
```

```
hist7 = ggplot(data = cab_data, aes(x =cab_data$year)) + ggtitle("Distribution of : year") +
geom_histogram(bins = 100)

hist8 = ggplot(data = cab_data, aes(x =cab_data$H_Distance)) + ggtitle("Distribution of : distance") +
geom_histogram(bins = 100)

bar1 = ggplot(data = cab_data, aes(x = cab_data$day_of_week)) + geom_bar() + ggtitle("day of week") +
theme_dark()

#making a grid


gridExtra::grid.arrange(hist1,hist2,ncol=1)

gridExtra::grid.arrange(hist3,hist4,ncol=1)

gridExtra::grid.arrange(hist5,hist6,ncol=1)

gridExtra::grid.arrange(hist7,hist8,ncol=1)

gridExtra::grid.arrange(bar1,ncol=1)

#fare and pessanger

ggplot(cab_data, aes(x= passenger_count,y=fare_amount)) +

  geom_point()

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

ggplot(cab_data, aes(x= day_of_month,y=fare_amount)) +

  geom_point()

#The fares throught the month mostly seem uniform, with the maximum fare received on the 16th

ggplot(cab_data, aes(x= hour,y=fare_amount)) +

  geom_point()


#Interesting! The time of day definitely plays an important role. The frequency of cab rides seem to be the
lowest at 5AM and the highest at 7PM

#The fares, however, seem to be high betweeb 5AM and 10AM, and 2PM to 4PM. Maybe people who live far
away prefer to leave earlier to avoid rush hour traffic?


ggplot(cab_data, aes(x= day_of_week,y=fare_amount)) +

  geom_point()


#The highest fares seem to be high weekdays  on  and the lowest on  and sunday almost an uniform distibution

#distace should have a direct relationship with fare

ggplot(cab_data, aes(x= H_Distance,y=fare_amount)) +
```

```
 geom_point()

summary(cab_data)


#feature selection

## Correlation Plot


numeric_index = sapply(cab_data,is.numeric) #selecting only numeric

numeric_index_test = sapply(cab_data_test,is.numeric) #selecting only numeri


## Correlation Plot


corrgram(cab_data[,numeric_index], order = F,

        upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

## Correlation Plot

corrgram(cab_data_test[,numeric_index_test], order = F,

        upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")


#as seen day of month anad day of week have very less co relation with targit variable thus we will drop both

#also date is being droped as all its features have been extracted already

#  dimensional  reduction


cab_data = subset(cab_data,select=-c(pickup_datetime,day_of_month,day_of_week))

cab_data_test = subset(cab_data_test,select=-c(pickup_datetime,day_of_month,day_of_week))

####################################Model Development######################################

head(cab_data)

########DECISION TREE



#Splitting the  data (90-10 percent)

set.seed(1)

train_index = sample(1:nrow(cab_data), 0.9*nrow(cab_data))

train = cab_data[train_index,]
```

```
test = cab_data[-train_index,]
#Build decsion tree using rpart
dt_model = rpart(fare_amount ~ ., data = train, method = "anova")
# here we can try any method other than anova ,
#one of "anova", "poisson", "class" or "exp".
#If method is missing then the routine tries to make an intelligent guess.
#Ploting the tree
rpart.plot(dt_model)
#Perdict for test cases
dt_predictions = predict(dt_model, test[,-1])
df3= data.frame((dt_predictions))
#Create data frame for actual and predicted values
df_pred = data.frame("actual"= test[,1], "dt_pred"=dt_predictions)
head(df_pred)
# analyse relationship between actual and predicted count
ggplot(df_pred, aes(x= actual ,y=dt_predictions)) +
  geom_point()+
  geom_smooth()

############# Evaluate  Decision tree ###################


#MAPE
#calculate MAPE
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y))
}

MAPE(test[,1], dt_predictions)

#Error Rate: 0.24789
#Accuracy: 76.77%
```

```
#Evaluate  Model using RMSE

RMSE <- function(y_test,y_predict) {

  difference = y_test - y_predict
  root_mean_square = sqrt(mean(difference^2))
  return(root_mean_square)

}

RMSE(test[,1], dt_predictions)

#RMSE = 4.36

########RANDOM FOREST
#Training the model using training data
rf_model = randomForest(fare_amount~., data = train, ntree = 200)

#Predict the test cases
rf_predictions = predict(rf_model, test[,-1])

#Create dataframe for actual and predicted values
df_pred = cbind(df_pred,rf_predictions)
head(df_pred)
# analyse relationship between actual and predicted count
ggplot(df_pred, aes(x= actual ,y=rf_predictions)) +
  geom_point()+
  geom_smooth()
############# Evaluate  ranom forest ###################
#MAPE
#calculate MAPE
MAPE(test[,1], rf_predictions)
```

```
#0.1925

#81.911% acuracy

RMSE(test[,1], rf_predictions)


#RMSE = 293.857


################### Develop  Linear Regression Model ##########################


#check multicollearity

library(usdm)


vif(cab_data[,-1])


vifcor(cab_data[,-1], th = 0.9)


# develop Linear Regression  model

#dividind data into test and train

train_index = sample(1:nrow(cab_data), 0.9 * nrow(cab_data))

train_lr = cab_data[train_index,]

test_lr = cab_data[-train_index,]

train_lr = subset(train_lr,select=-c(pickup_longitude ,pickup_latitude ,dropoff_latitude ,week))


#run regression model

lm_model = lm(fare_amount ~., data = train_lr)


#Summary of the model

summary(lm_model)



# observe the  residuals and   coefficients  of the linear regression model

# Predict  the Test data
```

```
#Predict
lm_predictions = predict(lm_model, test_lr[,-1])
#Creating a new dataframe for actual and predicted values
df_pred = cbind(df_pred,lm_predictions)
head(df_pred)

# analyse relationship between actual and predicted count
ggplot(df_pred, aes(x= actual ,y=lm_predictions)) +
  geom_point()+
  geom_smooth()

# Evaluate Linear Regression Model




MAPE(test_lr[,1], lm_predictions)

#Error Rate: 0.03757554
#Accuracy: 96.3%

RMSE(test_lr[,1], lm_predictions)

#RMSE = 2.327632e-12
#random forrest will be best fit
#for test data lets predict fare
X_test=cab_data_test
RF_fare_ammount=predict(rf_model,X_test)
cab_data_test$fare_predicted=RF_fare_ammount
#writinf the file in csv
write.csv(cab_data_test, file = 'output_cab_R .csv', row.names = FALSE, quote=FALSE)
```

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 6. Springer.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer Science & Business Media

https://stackoverflow.com

https://towardsdatascience.com/

https://en.wikipedia.org

http://mathforum.org/

https://www.analyticsvidhya.com/