Cab Fare Prediction

# Project statement:

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

# Aim:

Design a system that predicts the fare amount for a cab ride in the city.

# Data Set:

train\_cab (training data)

test (test data )

# Error metric used:

MAPE (Mean Absolute Percentage Error )

I like using the mean absolute percentage error (MAPE) because it's often more interpretable The MAPE, as a percentage makes more sense for values where divisions and ratios make sense. Ex let’s consider 2 fare 2$ and 20$ and our estimate is off by 20%(MAPE=20%) then then predicted value is 2.40$ or 1.40$ and 24$ or 16$ .

# How to run and deploy code ?

I am sharing jupyter notebook for python code and rscript for R code.One more thing I like to mention that i am implementing basic model in R and little more advance model in python

# Features provided: ·

. 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.

# Importing libraries

## In Python

import os

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import StratifiedKFold

from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import hdbscan

IN R

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

library(tidyverse)

library(lubridate)

library(geosphere)

library("dplyr")

lapply(x, require, character.only = TRUE)

rm(x)

# Importing files

## In Python

train=pd.read\_csv("train\_cab.csv")

## IN R

train=read.csv("train\_cab.csv",stringsAsFactors = F)

# Exploratory Data Analysis(EDA)

## Describing files

### In Python

Input:train.shape

Output:(16067, 7)

train.info()

Output:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 16067 entries, 0 to 16066

Data columns (total 7 columns):

fare\_amount 16043 non-null object

pickup\_datetime 16067 non-null object

pickup\_longitude 16067 non-null float64

pickup\_latitude 16067 non-null float64

dropoff\_longitude 16067 non-null float64

dropoff\_latitude 16067 non-null float64

passenger\_count 16012 non-null float64

dtypes: float64(5), object(2)

memory usage: 878.7+ KB

### IN R

glimpse(train)

Observations: 16,067

Variables: 7

$ fare\_amount <chr> "4.5", "16.9", "5.7", "7.7", "5.3", "12.1", "7.5", "16.5", "", "8.9", "5.3", "5.5", "4.1", "7...

$ pickup\_datetime <chr> "2009-06-15 17:26:21 UTC", "2010-01-05 16:52:16 UTC", "2011-08-18 00:35:00 UTC", "2012-04-21 ...

$ pickup\_longitude <dbl> -73.84431, -74.01605, -73.98274, -73.98713, -73.96810, -74.00096, -73.98000, -73.95130, -74.0...

$ pickup\_latitude <dbl> 40.72132, 40.71130, 40.76127, 40.73314, 40.76801, 40.73163, 40.75166, 40.77414, 40.72671, 40....

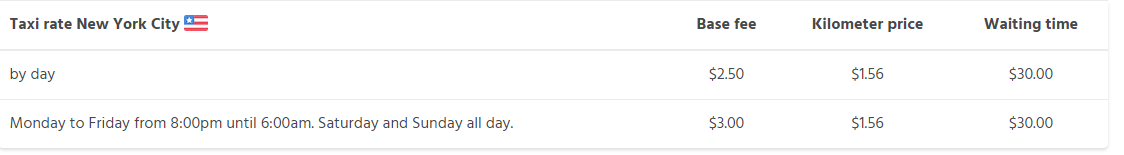
$ dropoff\_longitude <dbl> -73.84161, -73.97927, -73.99124, -73.99157, -73.95665, -73.97289, -73.97380, -73.99009, -73.9...

$ dropoff\_latitude <dbl> 40.71228, 40.78200, 40.75056, 40.75809, 40.78376, 40.75823, 40.76484, 40.75105, 40.73163, 40....

$ passenger\_count <dbl> 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 3, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 2, 1, ...

By looking at the output we can say that there are some missing value in fare\_amount,passenger\_count and we have to change the data type of fare\_amount (to float) and pickup\_datetime(to Date\_time )

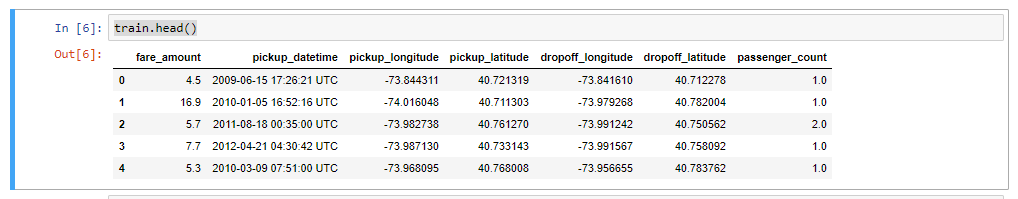
By using simple google search on coordinate provided in the data and we came to find that is New York coordinate where :



And there is point to noted that we were not provided any explicit feature (like drop off timing) to explain waiting time amount in fare amount .So we make an assumption the Waiting time increase with farther the distance travel.

Let’s look at the data

train.head()



### Removing outliner and missing values from target variable

#### In Python

#### train['fare\_amount']=pd.to\_numeric(train['fare\_amount'],errors='coercion')

#### print(f"There are {len(train[train['fare\_amount'] < 2.5])} fares less than Base fare.")

#### print(f"There are {len(train[train['fare\_amount'] > 100])} fares more than 100.")

output:

There are 6 fares less than Base fare.

There are 9 fares more than 100.

train=train[(train['fare\_amount'] >= 2.5)]

train=train[train['fare\_amount']<100]

### IN R

train$fare\_amount=as.numeric(train$fare\_amount,na.rm=T)

train<-train[!is.na(train$fare\_amount) ,]

train<-train[train$fare\_amount>=2.5 & train$fare\_amount< 100,]

## Feature Generation and Feature preparation

**Now we will try to generate as many as feature that will try to explain the target variable in better way**

By using Pickup longitude,latitude and dropoff longitude,latitude:

### Removing outliers by visualization

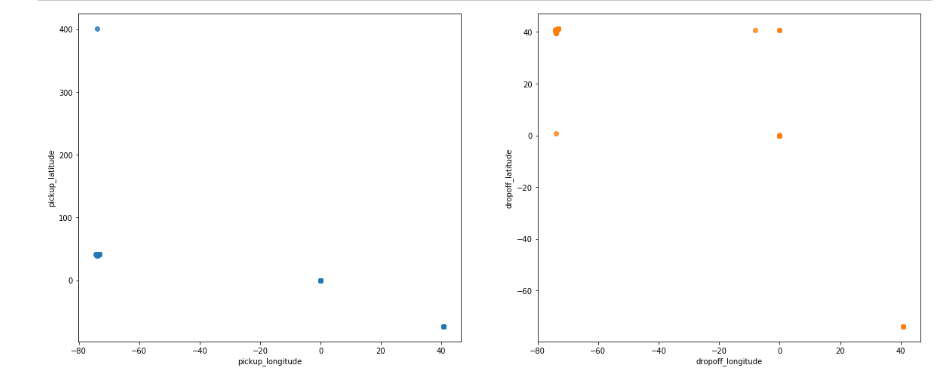
#### In Python

fig, axes = plt.subplots(1, 2, figsize = (20, 8))

axes = axes.flatten()

sns.regplot('pickup\_longitude', 'pickup\_latitude',data=train, ax = axes[0],fit\_reg = False);

sns.regplot('dropoff\_longitude', 'dropoff\_latitude',data = train, ax = axes[1],fit\_reg = False);



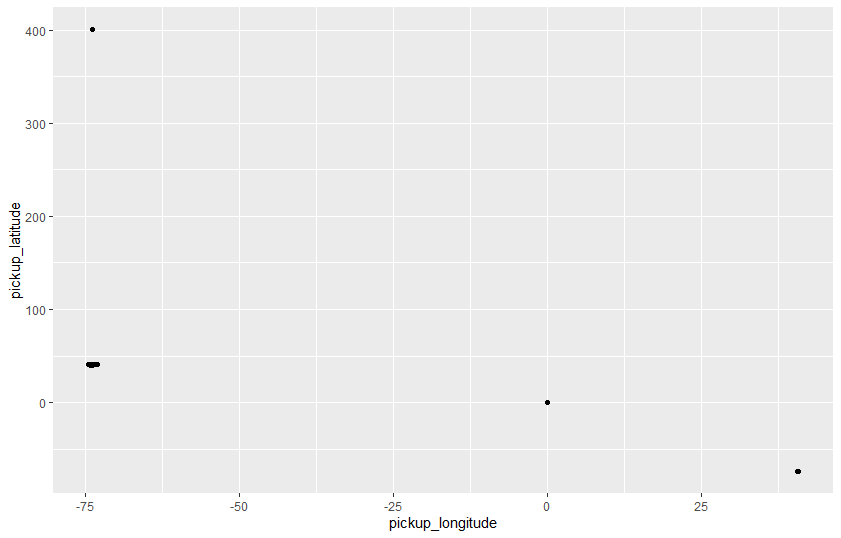
Removing outliers

train=train[(train['pickup\_latitude']<100) & (train['dropoff\_latitude'] >20) & (train['pickup\_longitude'] < -60) & (train['dropoff\_longitude'] < -60) ]

### IN R

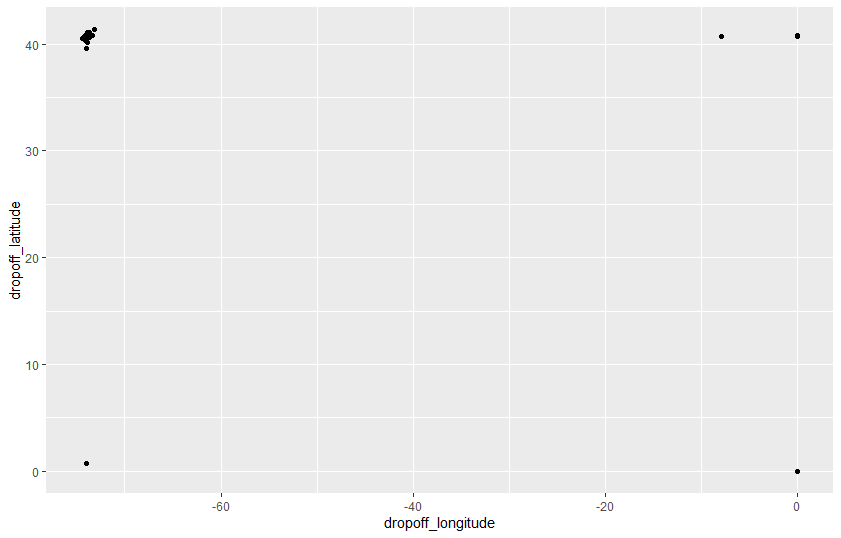
ggplot(train, aes(y = pickup\_latitude, x =pickup\_longitude)) +

geom\_point()

train<-train[train$pickup\_latitude<100 & train$pickup\_longitude< -50,]

ggplot(train, aes(x = dropoff\_longitude, y =dropoff\_latitude)) +

geom\_point()



train<-train[train$dropoff\_latitude>25 & train$dropoff\_longitude< -50,]

IN Python

**#Let’s create a temp data frame for better analysis**

temp1=pd.Series(train['dropoff\_latitude']-train['pickup\_latitude']) **#latitude difference**

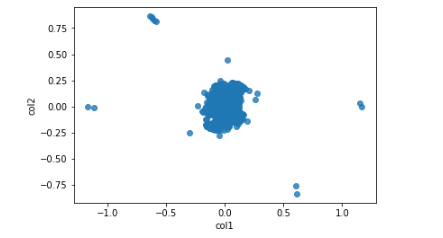
temp2=pd.Series(train['pickup\_longitude']-train['dropoff\_longitude']) **#longitude difference**

fare\_amount=train['fare\_amount']

dicts ={'col1':temp1,'col2':temp2,'fare\_amount':fare\_amount} #creating directory

df = pd.DataFrame(dicts)

sns.regplot('col1', 'col2',data=df,fit\_reg = False);

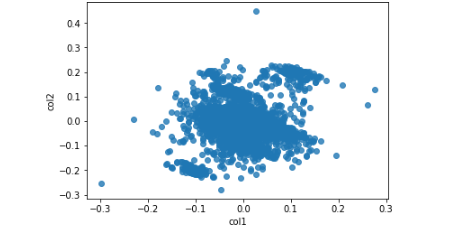


**#Removing variable with very high difference in longitude and latitudes**

index=((np.power(df['col1'],2)+np.power(df['col2'],2)) < 0.5)

df=df[index]

sns.regplot('col1', 'col2',data=df,fit\_reg = False);



train=train[index**] #updating the train values**

distance\_0=(df['col1']==0) & (df['col2']==0)# **when total distance travel is zero**

train[distance\_0]['fare\_amount'].mean()

10.01

train[distance\_0]['fare\_amount'].median()

6.5

**#when distance is 0 then fare is base fare = 2.5 plus some extra charge for waiting time**

train.loc[distance\_0,'fare\_amount']=train[distance\_0]['fare\_amount'].median()

### IN R

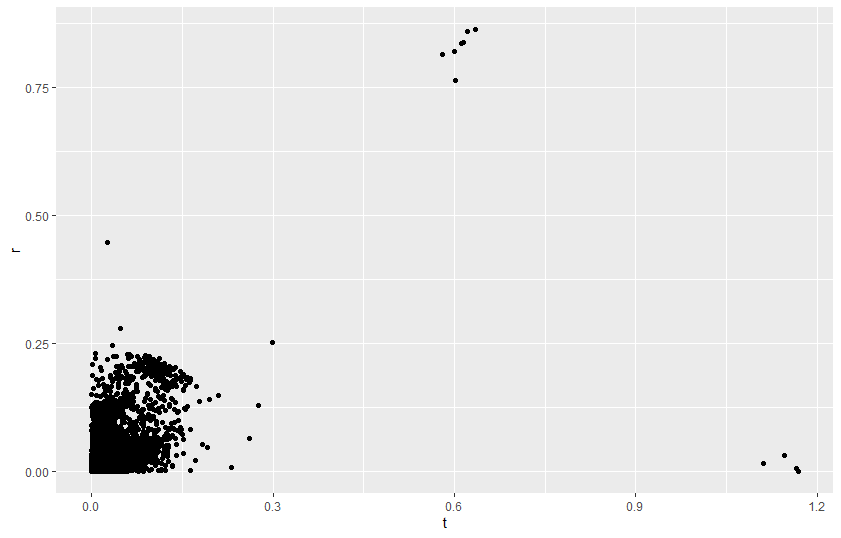
t=abs(train$pickup\_latitude-train$dropoff\_latitude)

r=abs(train$pickup\_longitude-train$dropoff\_longitude)

tz=as.data.frame(list(t=t,r=r))

ggplot(tz, aes(x = t, y =r)) +

geom\_point()

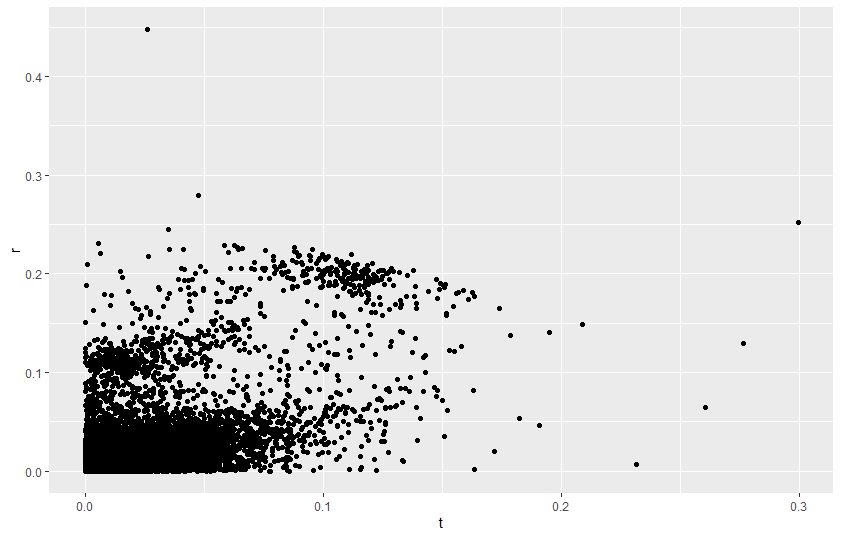


train<-train[(tz$t<0.5) & (tz$r<0.5),]

tz<-tz[tz$t<0.5 & tz$r<0.5,]

ggplot(tz, aes(x = t, y =r)) +

geom\_point()



index\_dist=(tz$t==0 & tz$r==0)

train[index\_dist,"fare\_amount"]=median(train[index\_dist,"fare\_amount"])

### feature generation by some visualization and analysis

#### In Python

lets create bins A/q to fare amount for better analysis

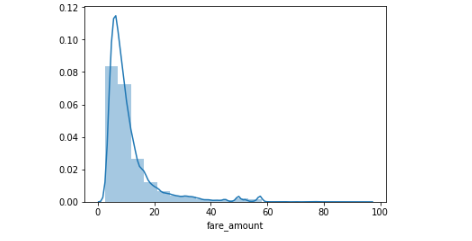
bins = [0, 5, 10, 15, 25,100] **#bins range**

labels = [1,2,3,4,5] **#bin labels**

train['fare\_amt'] = pd.cut(train['fare\_amount'], bins=bins, labels=labels)

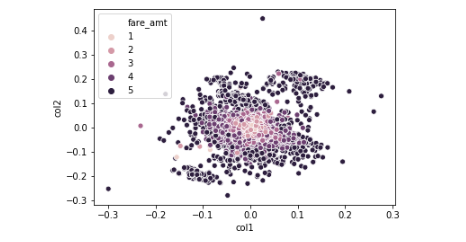
**#new column pointing to the labels**

df['fare\_amt'] = pd.cut(train['fare\_amount'], bins=bins, labels=labels)

sns.distplot(train.loc[train['fare\_amount']<100,'fare\_amount'], bins=20) 

**Visulization**

sns.scatterplot('col1', 'col2',data=df,hue="fare\_amt",legend="full");



As we can see that fare amount increase as we move away from origin So

We Can say there is change in fare amount a/q change in latitude and longitude and euclidean distance from origin for this scatter plot

**#Three new features**

def euc\_distance(x1, x2, y1, y2,n):

return ((abs(x2 - x1) \*\* n) + (abs(y2 - y1)) \*\* n) \*\* (1/n)

train['euc\_distance']=euc\_distance(train['pickup\_longitude'],train['dropoff\_longitude'],train['pickup\_latitude'],train['dropoff\_latitude'],2)

train['long\_diff']=abs(train['pickup\_longitude']-train['dropoff\_longitude'])

train['lat\_diff']=abs(train['pickup\_latitude']-train['dropoff\_latitude'])

### IN R

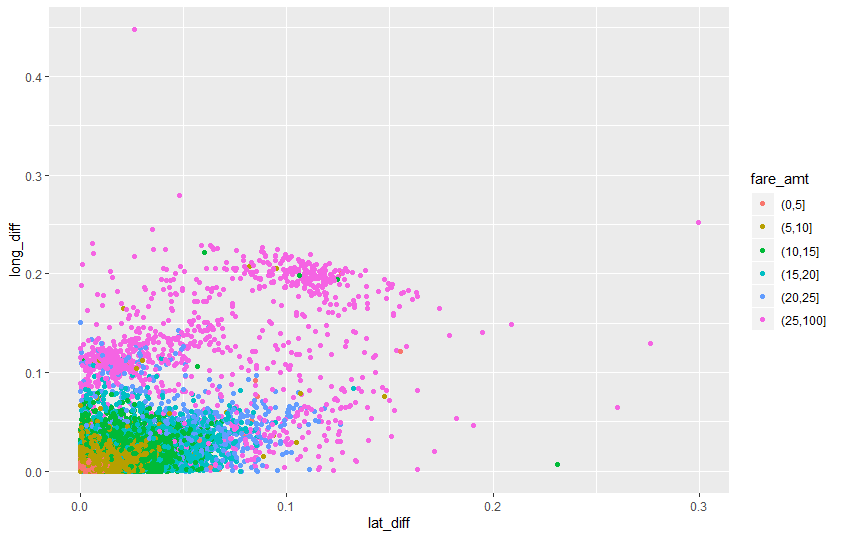
train$fare\_amt <- cut(train$fare\_amount, breaks=c(0,5,10,15,20,25,100))

train$lat\_diff=abs(train$pickup\_latitude-train$dropoff\_latitude)

train$long\_diff=abs(train$pickup\_longitude-train$dropoff\_longitude)

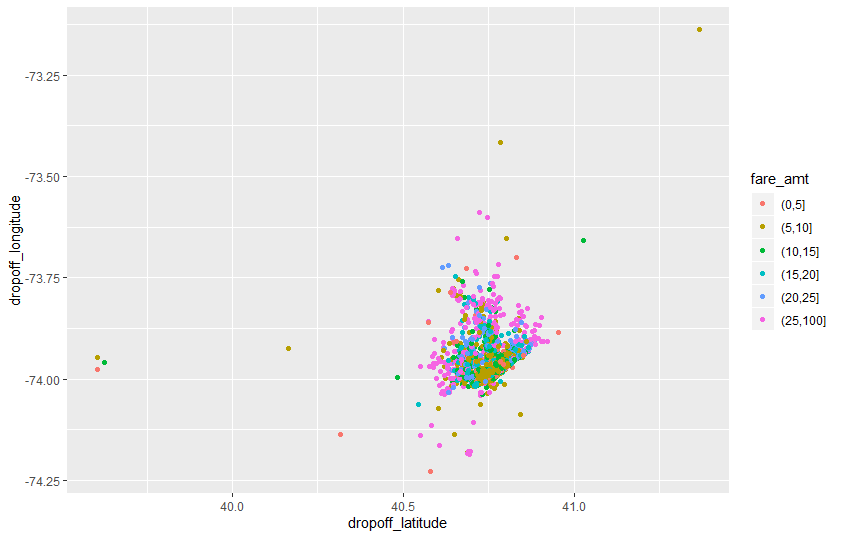
ggplot(train, aes(lat\_diff, long\_diff, fill = fare\_amt, group = fare\_amt)) +

geom\_point(aes(group = fare\_amt,color=fare\_amt))

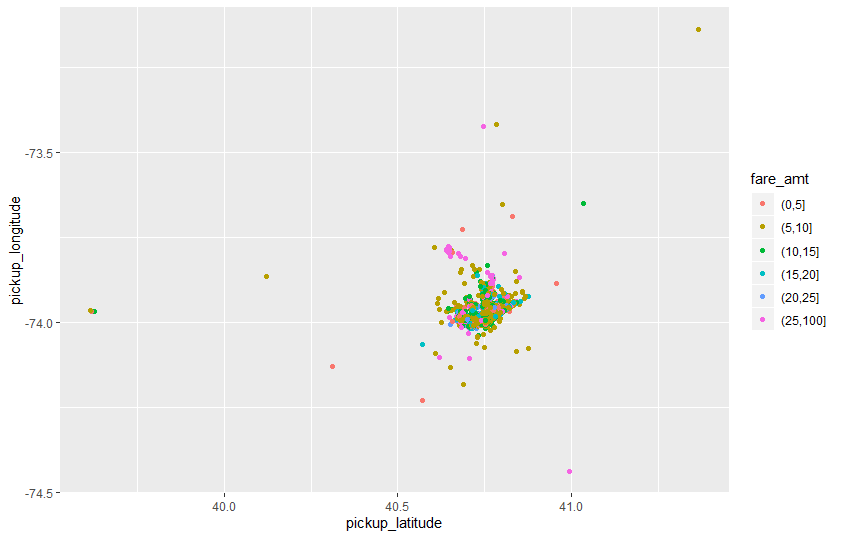


ggplot(train, aes(pickup\_latitude ,pickup\_longitude, fill = fare\_amt, group = fare\_amt)) +

geom\_point(aes(group = fare\_amt,color=fare\_amt))



ggplot(train, aes(dropoff\_latitude ,dropoff\_longitude, fill = fare\_amt, group = fare\_amt)) +

geom\_point(aes(group = fare\_amt,color=fare\_amt)) 

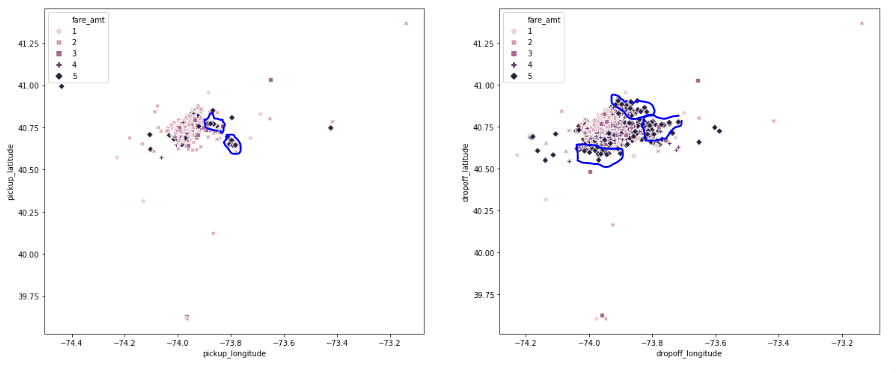
Generating another feature in python

fig, axes = plt.subplots(1, 2, figsize = (20, 8))

axes = axes.flatten()

sns.scatterplot(data=train,x='pickup\_longitude',y='pickup\_latitude',hue="fare\_amt",ax=axes[0],legend="full",style="fare\_amt");

sns.scatterplot('dropoff\_longitude', 'dropoff\_latitude',data = train, ax = axes[1],hue="fare\_amt",legend="full",style="fare\_amt");



I have tried to classify high fare amount location(which can be airport or tourist location)by using KNN ,random forest but classification is not providing a good result So I have drop that method rather than I will use some density based clustering method DBScan or HDBScan

clusterer=hdbscan.HDBSCAN(min\_cluster\_size=20).fit\_predict(train[['pickup\_latitude', 'pickup\_longitude']])

clusterer1=hdbscan.HDBSCAN(min\_cluster\_size=20).fit\_predict(train[['dropoff\_latitude', 'dropoff\_longitude']])

train['pickup\_area'] = clusterer

train['dropoff\_area'] =clusterer1

clusterer = hdbscan.HDBSCAN(min\_cluster\_size=20, prediction\_data=True).fit(train[['pickup\_latitude', 'pickup\_longitude']]) #**fitting the data point into clusters**

test\_labels, strengths = hdbscan.approximate\_predict(clusterer, test[['pickup\_latitude', 'pickup\_longitude']] ) #**predicting clusters for new data point**

test['pickup\_area']=test\_labels

clusterer = hdbscan.HDBSCAN(min\_cluster\_size=20, prediction\_data=True).fit(train[['dropoff\_latitude', 'dropoff\_longitude'**]]) #fitting the data point into clusters**

test\_labels, strengths = hdbscan.approximate\_predict(clusterer, test[['dropoff\_latitude', 'dropoff\_longitude']])

test['dropoff\_area']=test\_labels

**These cluster might not be that good but it can explain something extra about fare\_amount that will not be explained by above generated features**

IN R

There is no library in R to perform these action So I am dropping this feature in R

## Calculating Distance As a new feature

### In Python

R = 6371 #radius of earth in kilometers

**#R = 3959 #radius of earth in miles**

phi1 = np.radians(train['pickup\_latitude'])

phi2 = np.radians(train['dropoff\_latitude'])

delta\_phi = np.radians(train['dropoff\_latitude']-train['pickup\_latitude'])

delta\_lambda = np.radians(train['pickup\_longitude']-train['dropoff\_longitude'])

**#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) )**

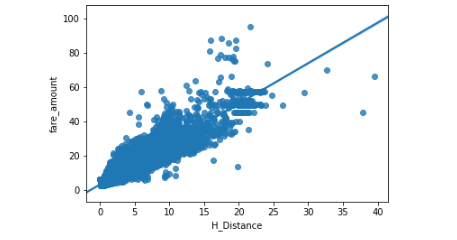
c = 2 \* np.arctan2(np.sqrt(a), np.sqrt(1-a))

**#d = R\*c**

d = (R \* c) #in kilometers

train['H\_Distance'] = d

sns.regplot('H\_Distance', 'fare\_amount',data=train);



### IN R

R = 6371 **#radius of earth in kilometers**

phi1 = train$pickup\_latitude\*(pi/180)

phi2 = train$dropoff\_latitude\*(pi/180)

delta\_phi =(train$dropoff\_latitude-train$pickup\_latitude)\*(pi/180)

delta\_lambda = (train$dropoff\_longitude-train$pickup\_longitude)\*(pi/180)

a = sin(delta\_phi / 2.0) \*\* 2 + cos(phi1) \* cos(phi2) \* sin(delta\_lambda / 2.0) \*\* 2

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

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

**#d = R\*c**

d = (R \* c) **#in kilometers**

train$H\_Distance = d

Dropping few rows on basis of very high fare per km >5x or <.20x of fare per km

**#removing abusrd fare**

**#there are some high fair at low distance this might be bcz of longer waiting time(the time driver is waiting for passenger)**

**#but we are not provided with any features to calculate waiting time So**

train['fare\_km'] = train.fare\_amount.sub(2.5).div(train.H\_Distance)

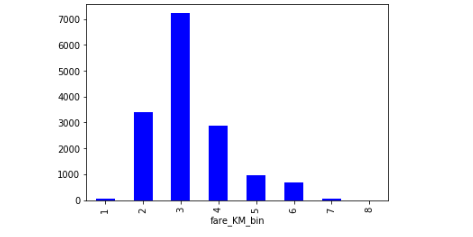
bins = [0 ,1, 2, 3, 4,5,8,15,100]

labels = [1,2,3,4,5,6,7,8]

train['fare\_KM\_bin'] = pd.cut(train['fare\_km'], bins=bins, labels=labels)

df['fare\_KM\_bin'] = pd.cut(train['fare\_km'], bins=bins, labels=labels)

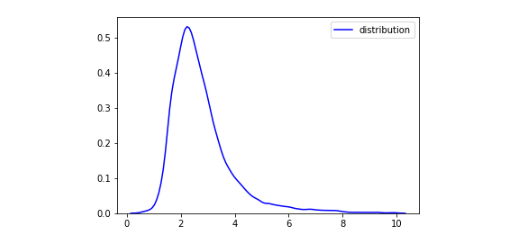
train[train['H\_Distance'] !=0].groupby('fare\_KM\_bin')['fare\_km'].count().plot.bar(color = 'b');



**#removing values with absurd fare\_km value**

train=train[(train["fare\_km"]<10) & (train["fare\_km"]>0.5)]

sns.kdeplot(train['fare\_km'], color="b", label = 'distribution')



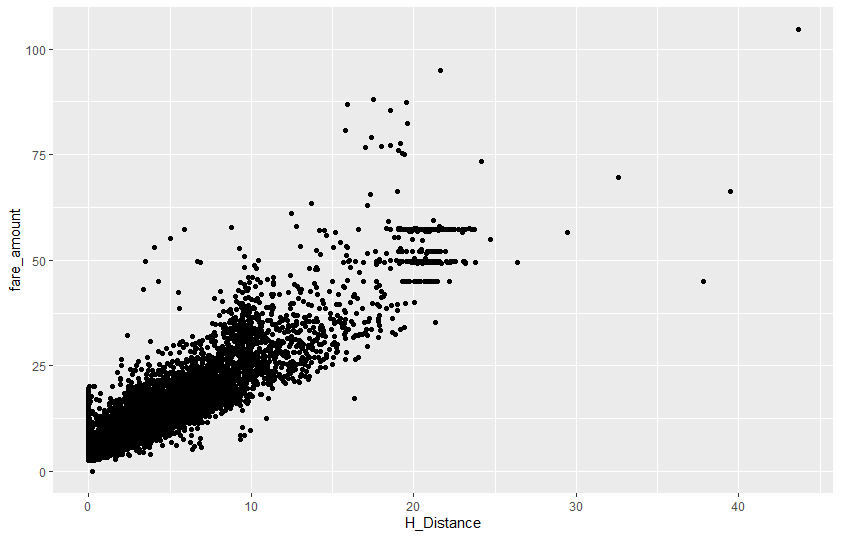
**#removing unnecessary columns**

train=train.drop(["fare\_km","fare\_KM\_bin","fare\_amt","fare\_amt1"],axis=1)

### IN R

ggplot(train, aes(x = H\_Distance, y = fare\_amount)) +

geom\_point()



train$fare\_km=(train$fare\_amount-2.5)/(train$H\_Distance)

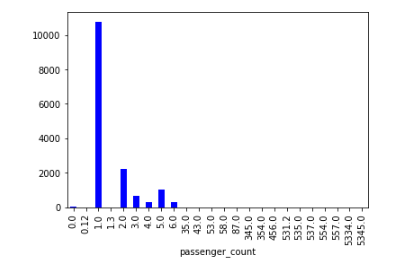
train[is.infinite(train$fare\_km),"fare\_km"]=2

train=train[train$fare\_km>0.4 & train$fare\_km<10,]

### Passenger\_count Variable

### In python

train.groupby('passenger\_count')['fare\_amount'].count().plot.bar(color = 'b');

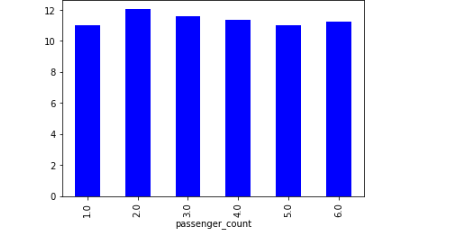


**#removing any variable with fraction ,0 or more than 6 passanger**

train=train[train['passenger\_count'].isin([1,2,3,4,5,6])]

train.groupby('passenger\_count')['fare\_amount'].mean().plot.bar(color = 'b');

**#checking avg fare with respect to passanger count**



### IN R

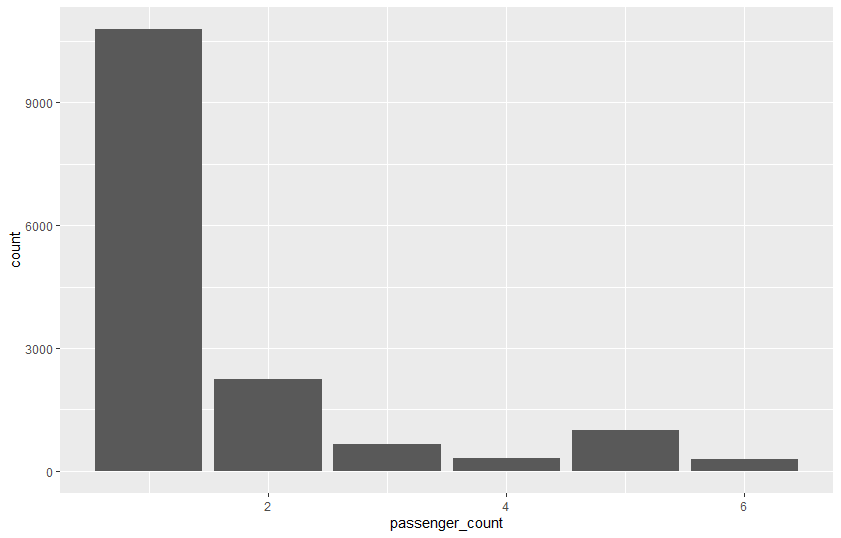
table(train$passenger\_count)

**#removing the columns with 0,fraction and more than 6 passenger**

train=train[!(train$passenger\_count>6 )& !(train$passenger\_count %in% c(0,0.12,1.3)),]

ggplot(data = train) +

geom\_bar(mapping = aes(x = passenger\_count))



**#imputing missing value with Mode value**

train[is.na(train$passenger\_count),"passenger\_count"]=1

train[is.na(train$passenger\_count),"passenger\_count"]=1

sum(is.na(train$passenger\_count))

**#average fares depanding of no of passanger**

round(tapply(train$fare\_amount,train$passenger\_count,mean,na.rm=T))

func\_plot(train$fare\_amount,train$passenger\_count)

**#fare slightly decrease with increase in no of passanger**

## Pickup\_datetime Variable

First Converting datetime into proper data type and then spilting in into year,month,day,dayofweek and hour

### In Python

train['pickup\_datetime'] = pd.to\_datetime(train['pickup\_datetime'], errors='coerce')

train['year']=train['pickup\_datetime'].dt.year #**year**

train['month']=train['pickup\_datetime'].dt.month #**month**

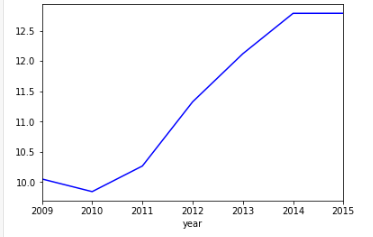
train['day']=train['pickup\_datetime'].dt.day #**day**

train['dayofweek']=train['pickup\_datetime'].dt.dayofweek #**perticular day of week**

train['hour']=train['pickup\_datetime'].dt.hour

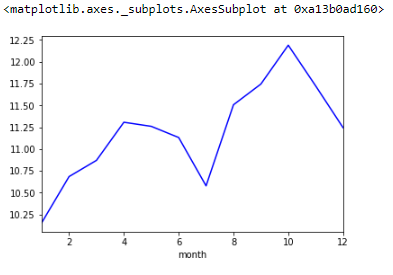
train=train.drop('pickup\_datetime',axis=1)

train.groupby("year")['fare\_amount'].mean().plot.line(color = 'b')

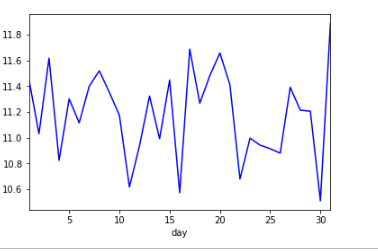


So average fare increases with every passing year

train.groupby("month")['fare\_amount'].mean().plot.line(color = 'b')

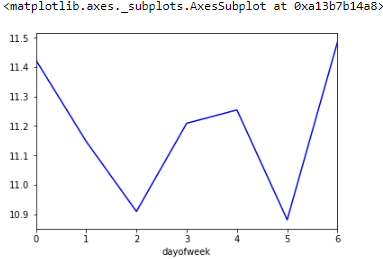


There is upword trend in first 4 month for avg fare then downward till 7 again there is upword till 11 then again downward trend for last month



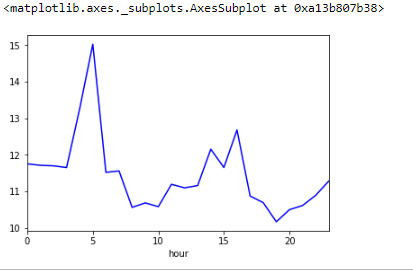
**There is not any particular pattern for avg fare change with respect to a perticular day**

train.groupby("dayofweek")['fare\_amount'].mean().plot.line(color = 'b')



In this Monday=0 and Sunday=6 so avg fare is slightly higher on Monday,Sunday and Friday

train.groupby("hour")['fare\_amount'].mean().plot.line(color = 'b')



Avg fare is high for night time and slightly higher in afternoon time

### IN R

train$pickup\_datetime= ymd\_hms(train$pickup\_datetime)

train= train %>%

mutate(year=year(pickup\_datetime),

month=month(pickup\_datetime),

week=weekdays(pickup\_datetime),

hour=hour(pickup\_datetime),

date=day(pickup\_datetime)

)

train$pickup\_datetime=NULL

### IN R

**#function for plotting tapply output**

func\_plot=function(values,index){

xnames <- names(tapply(values,index,mean))

plot(tapply(values,index,mean),xaxt="n",col="blue",type = "o")

axis(1, at=1:length(xnames), labels=xnames)

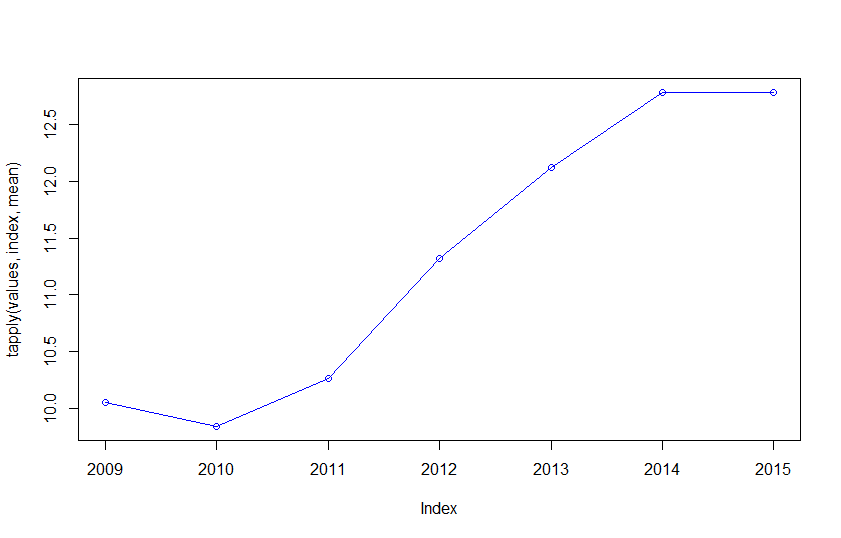
}

**#average fare on the basis of year**

round(tapply(train$fare\_amount,train$year,mean,na.rm=T))

**#there is increment in fare with every passing year**

func\_plot(train$fare\_amount,train$year)



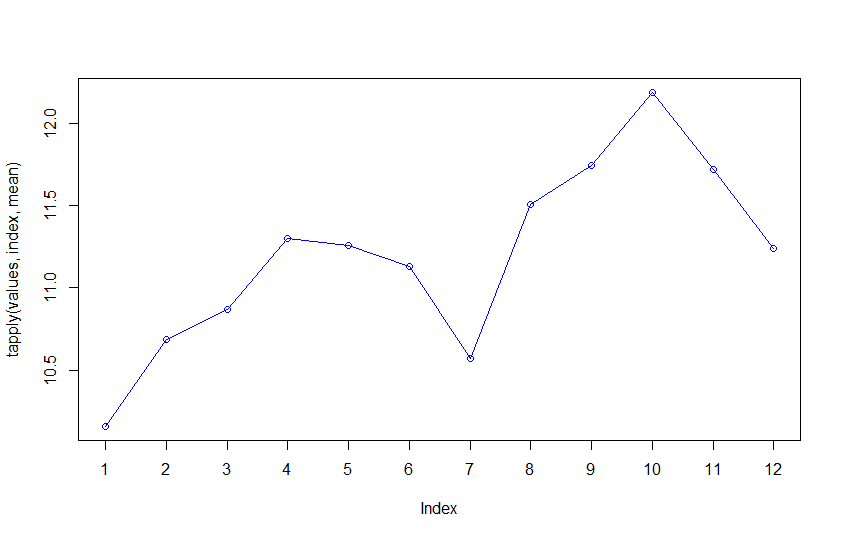
#average fare changes wrt month

table(train$month)

round(tapply(train$fare\_amount,train$month,mean,na.rm=T))

func\_plot(train$fare\_amount,train$month)

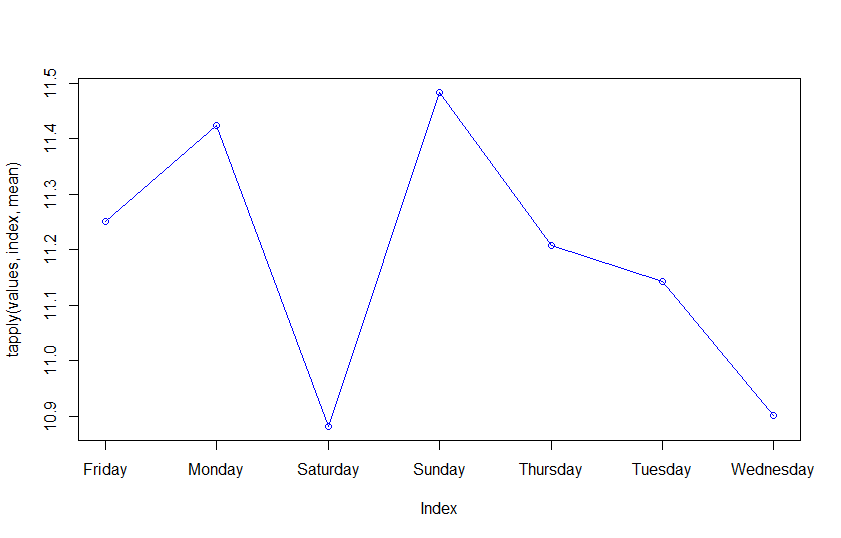
**#average fare with by month can be divided into 3 groups**



#average fare relation day features

tapply(train$fare\_amount,train$week,mean,na.rm=T)

func\_plot(train$fare\_amount,train$week)

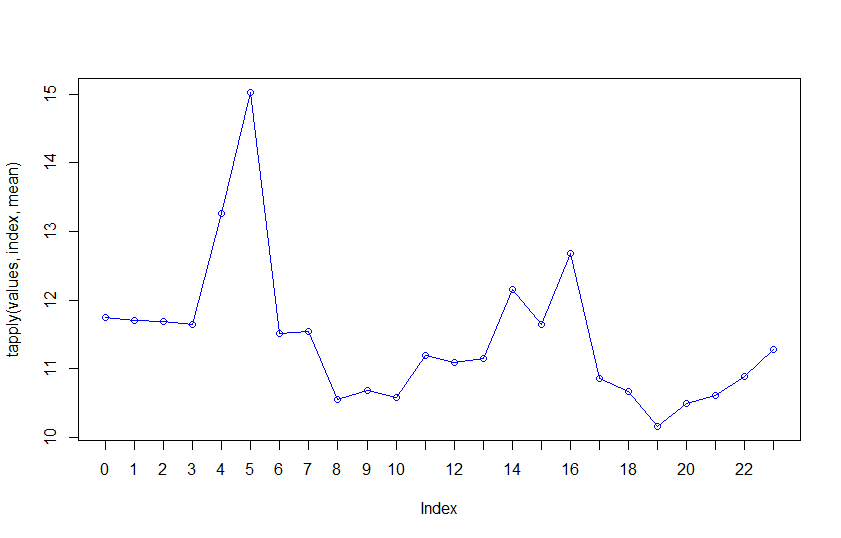


**#Friday,sunday and monday having slightly higher fare than rest of the days**

table(train$hour)

round(tapply(train$fare\_amount,train$hour,mean,na.rm=T))

func\_plot(train$fare\_amount,train$hour)



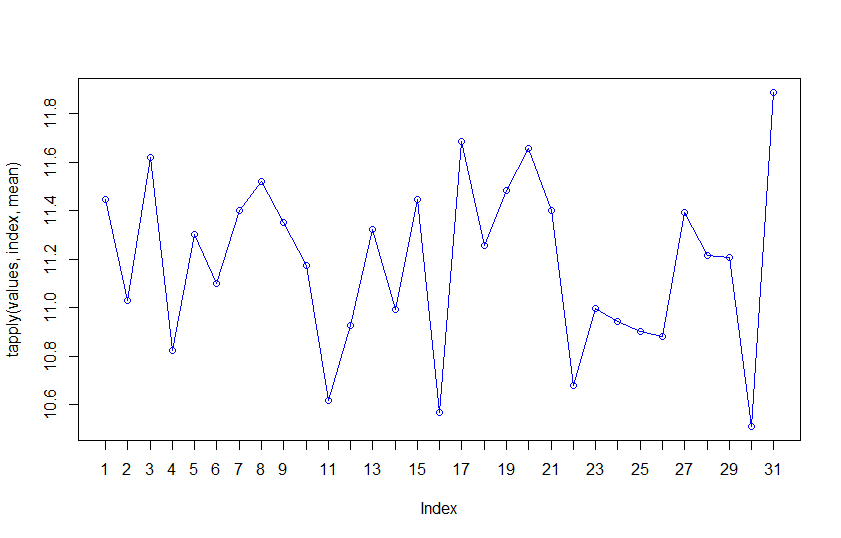
**# there is a price jump from 4 to 7Pm(In afternoon) and 12pm to 5AM(night time)**

**#average fair depanding on date**

table(train$date)

round(tapply(train$fare\_amount,train$date,mean,na.rm=T))

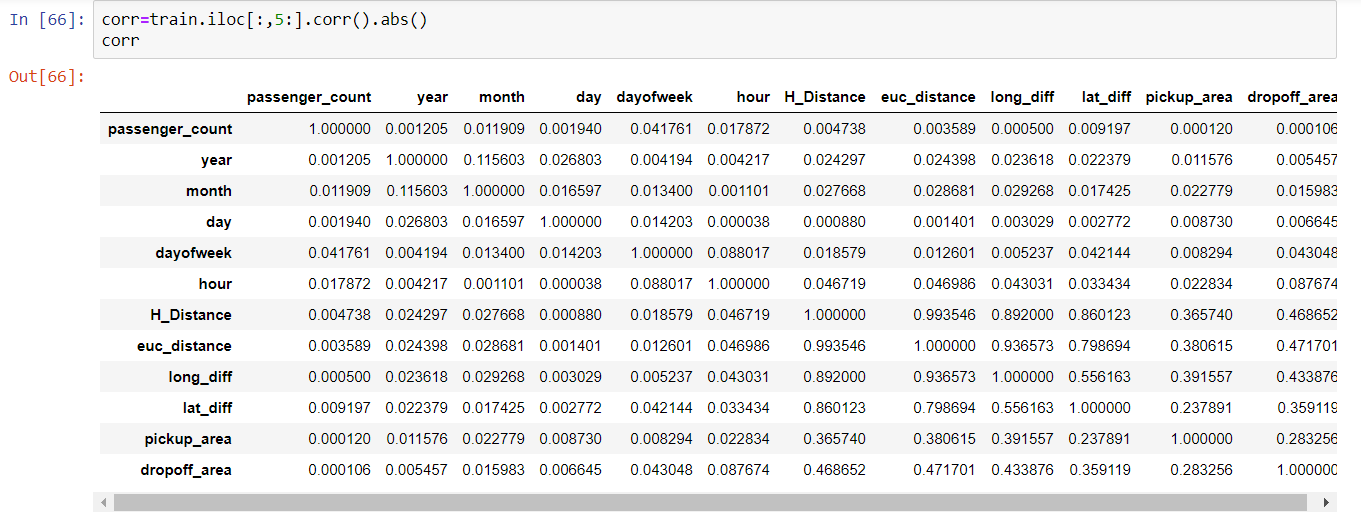
func\_plot(train$fare\_amount,train$date)



**#for date wise group there is no substantial pattern or trend So we will make dummies on the round of value only**

### Multi Collinearity

#### In Python

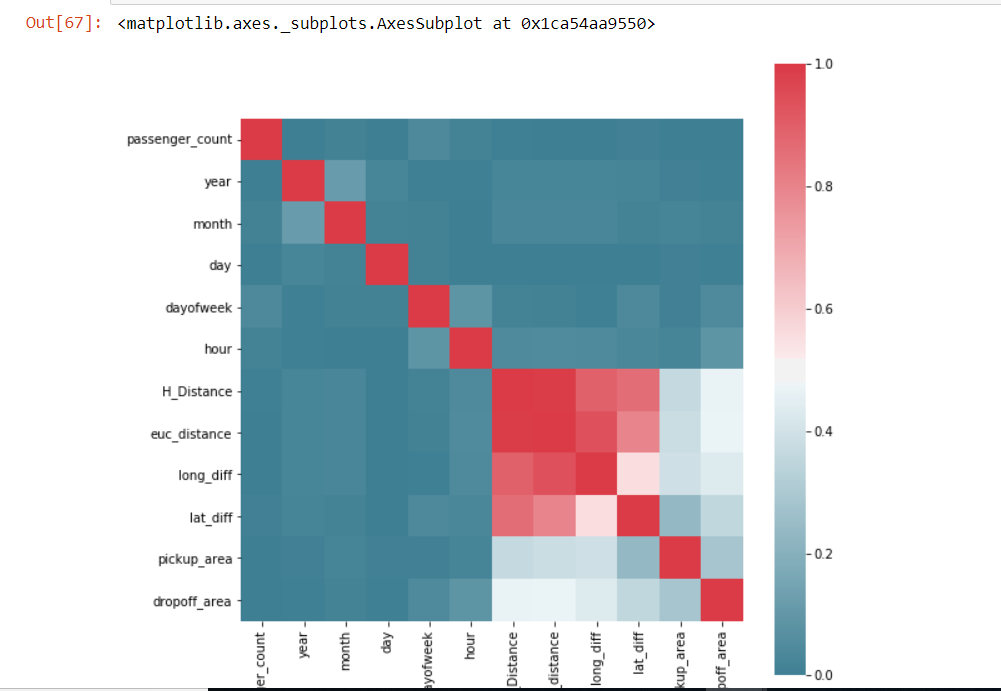


f, ax = plt.subplots(figsize=(9, 9))

**#Plot using seaborn library**

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True),

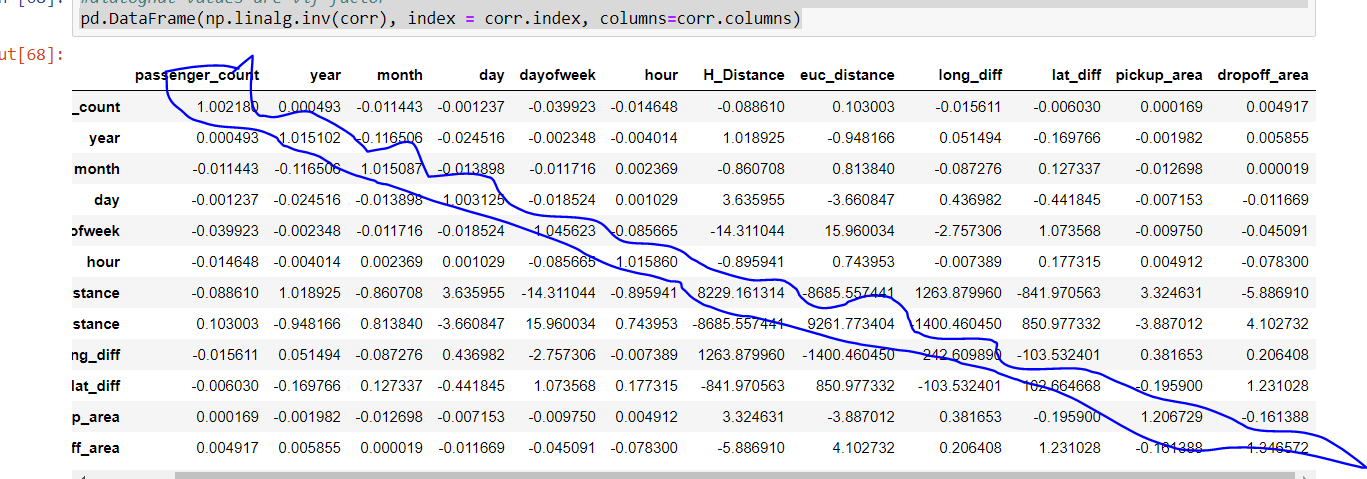
square=True, ax=ax)



**#dialognal values are vif factor**

pd.DataFrame(np.linalg.inv(corr), index = corr.index, columns=corr.columns)

Note:-

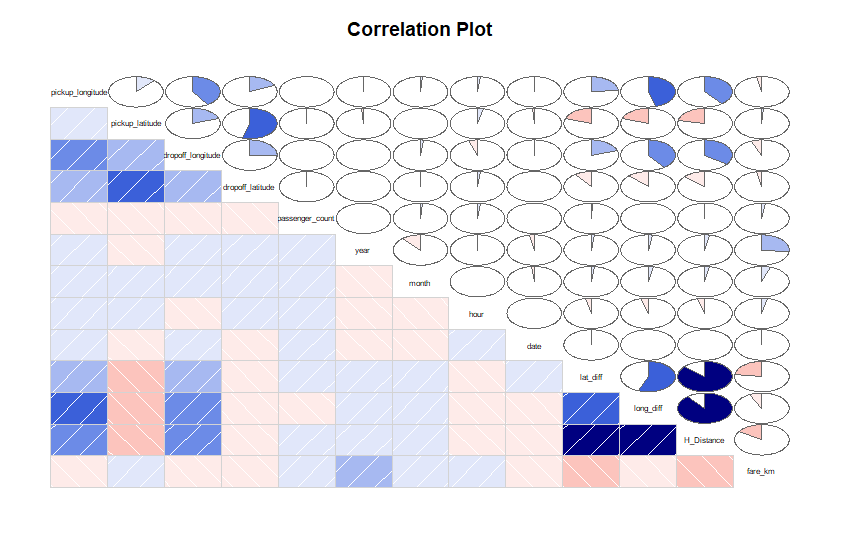


I am submitting the basic model code in R(R Script) and little more advance model code in python(jupyter notebook)

### IN R

corrgram(train[,-1], order = F,

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

**#let’s treat some variable as categorical in R**

new\_train =train %>%

transmute(ll\_y1=as.numeric(year %in% c("2009","2010","2011") ),

ll\_y2=as.numeric(year %in% c("2014","2015")),

ll\_m1=as.numeric(month %in% c(9:11) ),

ll\_m2=as.numeric(month==1),

ll\_h1=as.numeric(hour %in% c(0,1,2,3,4,5,14,15,16)),

ll\_d1=as.numeric(date %in% c(1,3,17,20,31)),

ll\_w1=as.numeric(week %in% c("Sunday","Monday","Friday")),

ll\_pc1=as.numeric(passenger\_count %in% c(2,3))

)

new\_train$fare\_amount=train$fare\_amount

new\_train$H\_Distance=train$H\_Distance

new\_train$lat\_diff=train$lat\_diff

new\_train$long\_diff=train$long\_diff

#**dropped all useless variable**

## For basic model

### let's drop euc\_distance,lat\_diff,long\_diff,dropoff\_area and pickup\_area

### IN Python

x=train **#so we don't lose our generated data Frame**

train=train.drop(["euc\_distance","long\_diff","lat\_diff","pickup\_area","dropoff\_area"],axis=1)

Splitting the data into into train/test in 75/25 set

X = train.iloc[:,1:]

Y = train['fare\_amount']

train\_X, val\_X, train\_y, val\_y = train\_test\_split(X, Y,test\_size=.25,random\_state=42)

train\_X=train\_X.iloc[:,4]

# **droping pickup\_longitude and latitude ,dropoff\_longitude and latitude**

val\_X=val\_X.iloc[:,4:]

Using stats model for better analysis

import statsmodels.api as sm

from scipy import stats

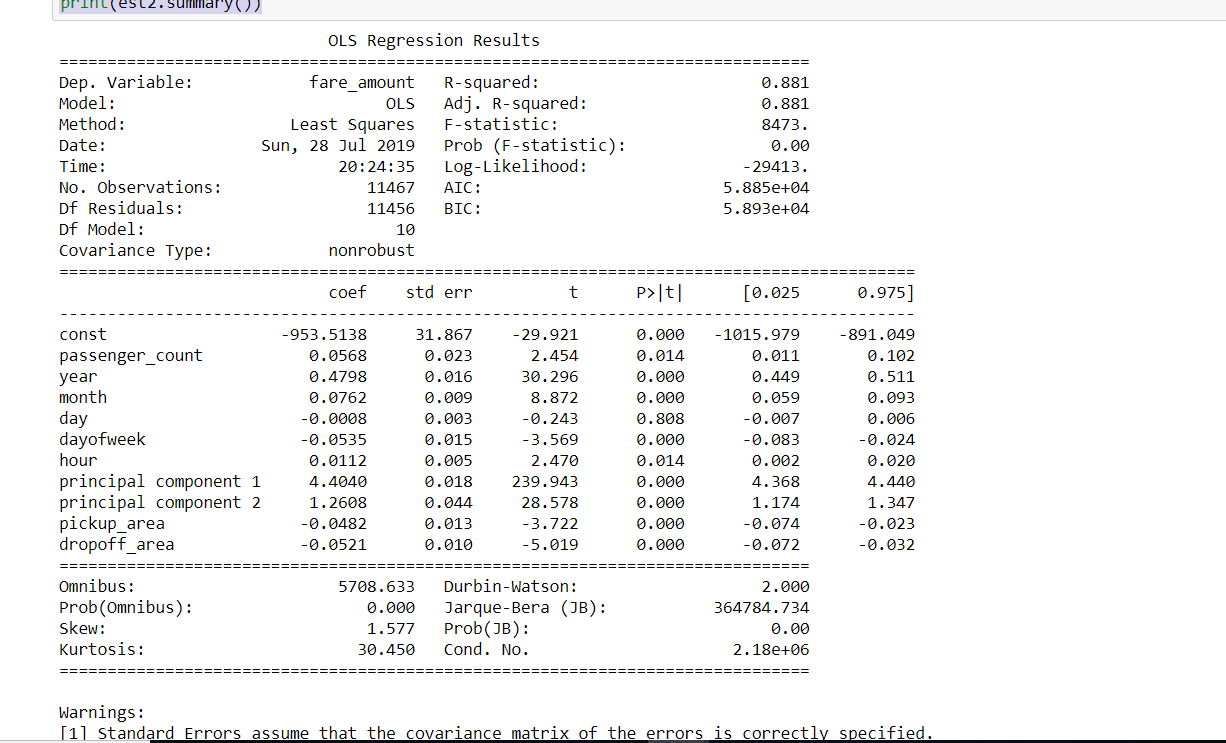
X2 = sm.add\_constant(train\_X)

est = sm.OLS(train\_y, X2)

est2 = est.fit()

print(est2.summary())

d



dropping the features on the basis of P value

train\_X=train\_X.drop(["day"],axis=1)

val\_X=val\_X.drop(["day"],axis=1)

#### Linear Regression

lm=LinearRegression()

lm.fit(train\_X,train\_y) #**fitting the model in our training Data**

pred\_fare=lm.predict(val\_X) **# making prediction on validation set**

valid\_pred=abs((val\_y-pred\_fare)/val\_y)

valid\_mape = 100 \* np.mean(valid\_pred) **#MAPE SCORE**

valid\_mape

#**output=18.680812790154366**

#### Random Forest

random\_forest = RandomForestRegressor(n\_estimators = 100, max\_depth = 20,

max\_features = None, oob\_score = True,

bootstrap = True, verbose = 1, n\_jobs = -1)

random\_forest.fit(train\_X,train\_y) #fitting into tree

pred\_fare=random\_forest.predict(val\_X)

valid\_pred=abs((val\_y-pred\_fare)/val\_y)

valid\_mape\_random\_forest = 100 \* np.mean(valid\_pred)

valid\_mape\_random\_forest

output=17.319331197010907

LightGBM

import lightgbm as lgbm

params = {

'boosting\_type':'gbdt',

'objective': 'regression',

'nthread': 4,

'num\_leaves': 63,

'learning\_rate': 0.05,

'max\_depth': -1,

'bagging\_fraction' : 1,

'bagging\_freq': 15,

'metric': 'mape',

'min\_split\_gain': 0.8,

'min\_child\_weight': 1,

'min\_child\_samples': 15,

'scale\_pos\_weight':1,

'num\_rounds':50000

}

train\_set = lgbm.Dataset(train\_X, train\_y, silent=False)

valid\_set = lgbm.Dataset(val\_X,val\_y, silent=False)

model = lgbm.train(params, train\_set = train\_set, num\_boost\_round=100000,early\_stopping\_rounds=5000,verbose\_eval=1000, valid\_sets=[valid\_set,train\_set])

output=

[1000] training's mape: 0.140547 valid\_0's mape: 0.161231

[2000] training's mape: 0.140547 valid\_0's mape: 0.161231

[3000] training's mape: 0.140547 valid\_0's mape: 0.161231

[4000] training's mape: 0.140547 valid\_0's mape: 0.161231

[5000] training's mape: 0.140547 valid\_0's mape: 0.161231

Early stopping, best iteration is:

[154] training's mape: 0.14487 valid\_0's mape: 0.160615

### IN R

#model development

s=sample(1:nrow(new\_train),0.75\*nrow(new\_train))

ld\_train=new\_train[s,]

ld\_train2=new\_train[-s,]

fit=lm(fare\_amount~.,data=ld\_train)

library(car)

# we'll take vif cutoff as 5

sort(vif(fit))

ll\_d1 ll\_pc1 ll\_w1 ll\_h1 ll\_m2 l l\_m1 ll\_y1 ll\_y2 lat\_diff 1.000995 1.002789 1.003903 1.007806 1.033372 1.036431 1.315421 1.318660 23.765726

long\_diff H\_Distance

29.672622 79.237469

summary(fit)

Residuals:

Min 1Q Median 3Q Max

-69.366 -1.415 -0.331 0.968 41.307

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 3.94509 0.07035 56.081 < 2e-16 \*\*\*

ll\_y1 -1.30587 0.06693 -19.510 < 2e-16 \*\*\*

ll\_y2 0.97227 0.08095 12.011 < 2e-16 \*\*\*

ll\_m1 0.51932 0.07051 7.366 1.88e-13 \*\*\*

ll\_m2 -0.24972 0.10191 -2.450 0.01428 \*

ll\_h1 -0.21010 0.06626 -3.171 0.00152 \*\*

ll\_d1 0.04270 0.08282 0.516 0.60619

ll\_w1 -0.24095 0.05918 -4.071 4.71e-05 \*\*\*

ll\_pc1 0.31227 0.07484 4.172 3.04e-05 \*\*\*

H\_Distance 3.67482 0.07208 50.985 < 2e-16 \*\*\*

lat\_diff -165.62837 6.20934 -26.674 < 2e-16 \*\*\*

long\_diff -50.61672 4.65788 -10.867 < 2e-16 \*\*\*

#**removing feature based on AIC Score**

fit=step(fit)

formula(fit)

fit=lm(fare\_amount ~ ll\_y1 + ll\_y2 + ll\_m1 + ll\_m2 + ll\_h1 + ll\_w1 +

ll\_pc1 + H\_Distance,data=ld\_train) #**lat\_diff and long\_diff is generating the H\_distance if we remove vif value will be controlled**

summary(fit)

val.pred=predict(fit,newdata=ld\_train2)

error=ld\_train2$fare\_amount-val.pred

RMSE=sqrt(mean(error\*\*2,na.rm=T))

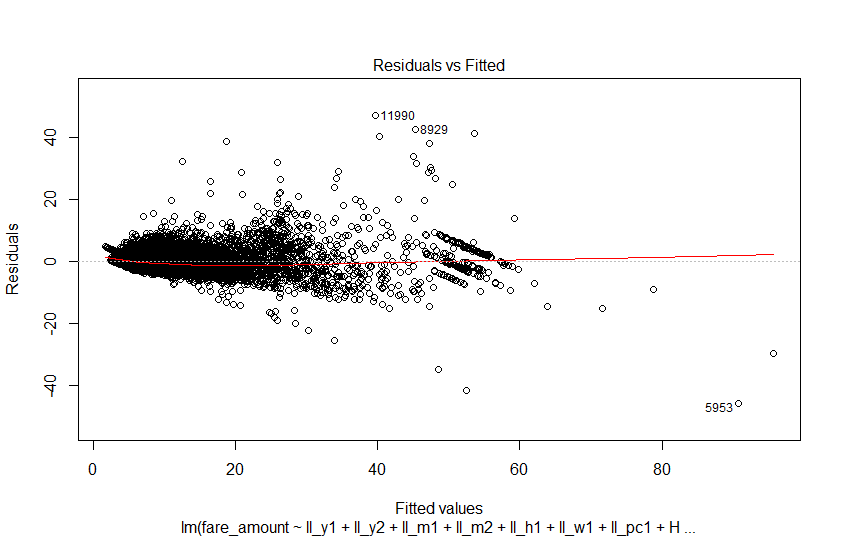
#**3.61**

mape\_score=mean(abs((ld\_train2$fare\_amount-val.pred)/ld\_train2$fare\_amount)\*100,na.rm=T)

#**19.28**

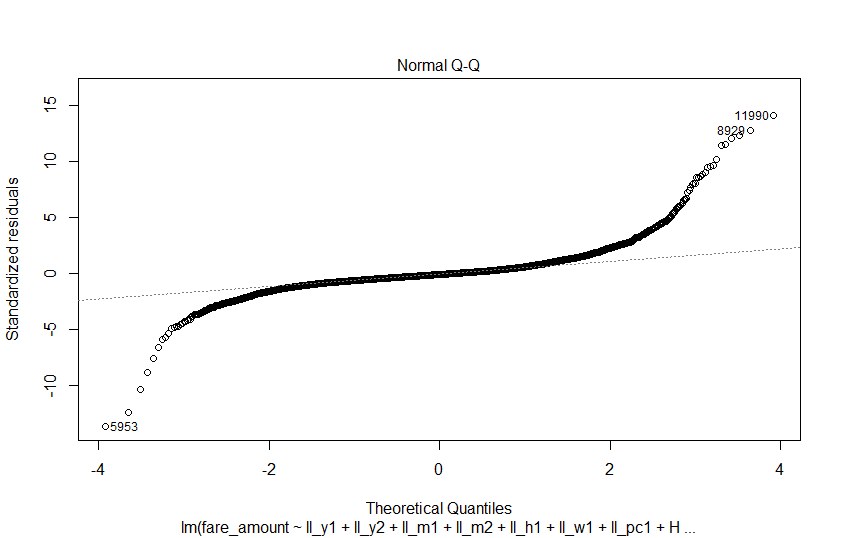
plot(fit,1)

**Residuals vs Fitted**



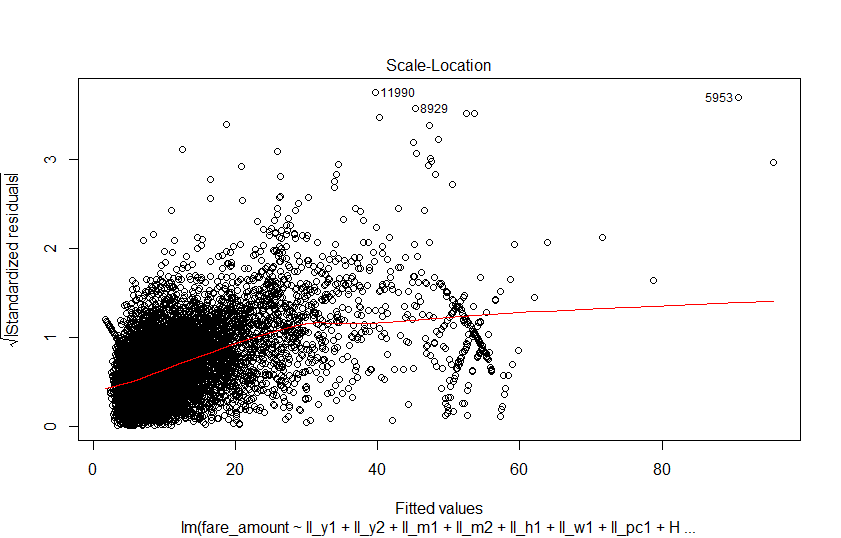
plot(fit,2)

**Normal Q-Q**



Errors are not distributed normally distributed

plot(fit,3)



#Decision Tree

require(rpart)

fit <- rpart(fare\_amount ~ ll\_y1 + ll\_y2 + ll\_m1 + ll\_m2 + ll\_h1 + ll\_w1 +

ll\_pc1 + H\_Distance, method = 'anova',parms = list(split = "information"),data=ld\_train)

val.pred=predict(fit,newdata=ld\_train2)

error=ld\_train2$fare\_amount-val.pred

RMSE=sqrt(mean(error\*\*2,na.rm=T))

#**3.73**

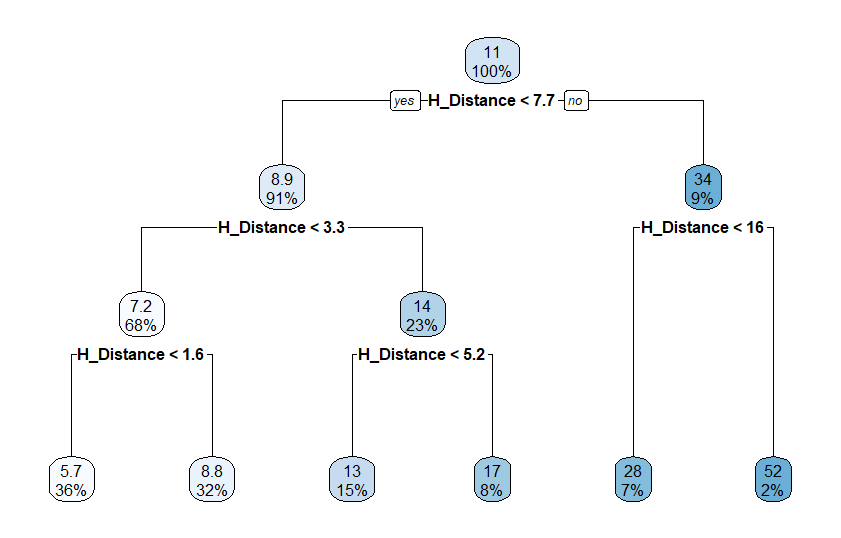
Mape\_score=mean(abs((ld\_train2$fare\_amount-val.pred)/ld\_train2$fare\_amount)\*100,na.rm=T)

#**23.93**

library(rpart.plot)

rpart.plot(fit)

rpart.rules(fit,cover = TRUE)



NEW MODEL

Add two more features pickup\_area and drop\_off area and run the model again

Then result

Linear regression:

Mape=18.721412438556726

random Forest:

mape=16.90586674061424

LightGBM:

Mape=15.6189

There is some improvement in lightGBM and randomforest score and then there is a minor decrement in linear regression score(some more non linearity is introduced in the model)

### Final Model

there is very high multi collinerity between "H\_Distance","long\_diff","lat\_diff","euc\_distance"

let's use PCA and try to explain maximum variance and remove multicollinearity

**#scaling the variable**

from sklearn.preprocessing import StandardScaler

features = ["H\_Distance","long\_diff","lat\_diff","euc\_distance"]

**# Separating out the features**

temp= train.loc[:, features].values

temp= StandardScaler().fit\_transform(temp)

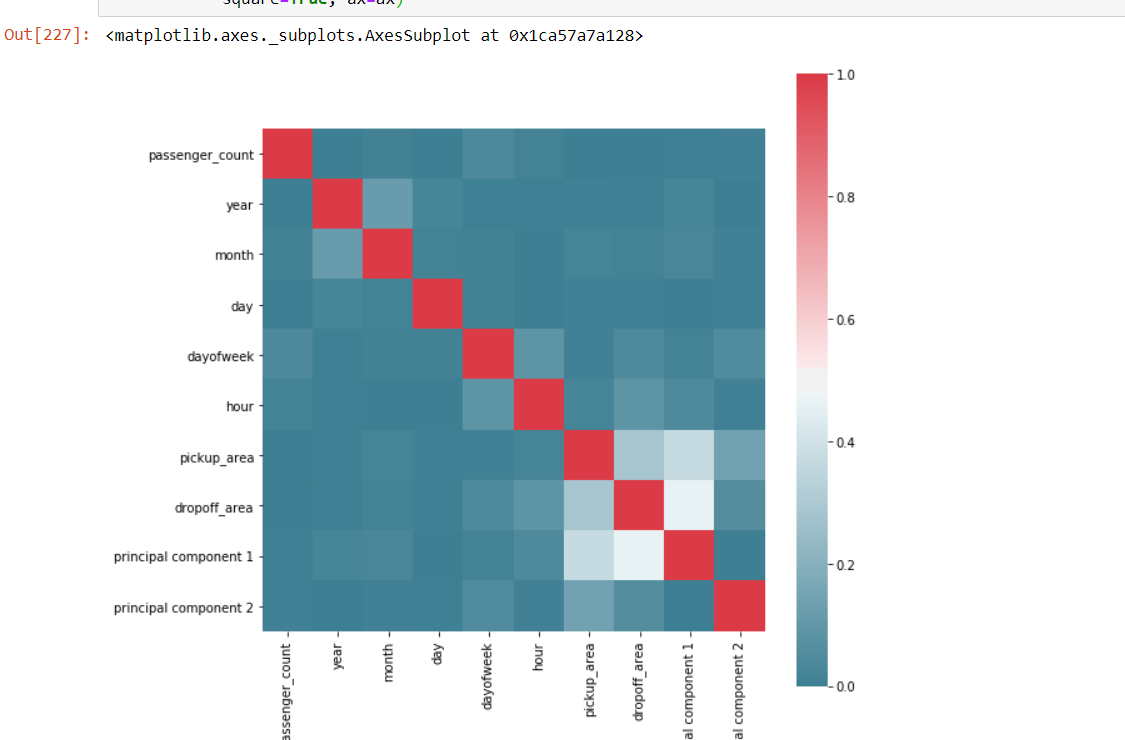
**#applying PCA**

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

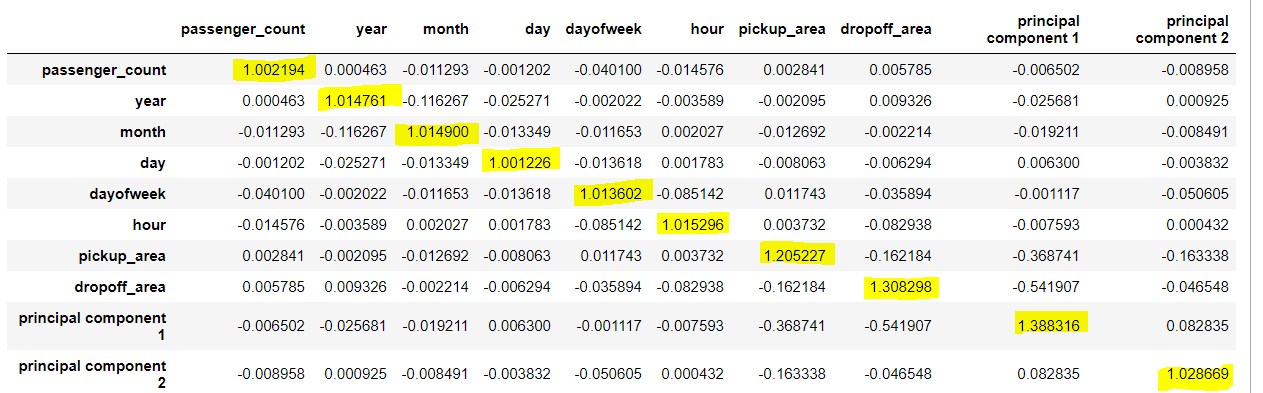
principalComponents = pca.fit\_transform(temp)

principalDf = pd.DataFrame(data = principalComponents,columns = ['principal component 1', 'principal component 2'])



Now there is no problem in multi collinearity

Lets check for VIF factor



There is No problem in VIF also

#**concating the dataframes**

x1=pd.concat([train.reset\_index(drop=True),principalDf.reset\_index(drop=True)], axis=1)

train=x1

**#droping the useless variable**

train=train.drop(["euc\_distance","long\_diff","lat\_diff","H\_Distance"],axis=1)

then applying the all model

linear regression

MAPE=19.216100234024562

Random Forest

MAPE=15.971699924806565

Light GBM

MAPE=14.7292

Making the final prediction by using model giving lowest validation Score

Generate the feature in test data using above process and make prediction using best model

test["fare\_amount"]=model.predict(test,num\_iteration=model.best\_iteration) #**LIghtGBM**