72017912C
COMP8EB1326

GROUP B : ASSIGNMENT No. 1

Page:
Date: / /

Title: Uber ride price prediction.

Objective: To predict the price of the Uber ride from a given pickup point to the agreed drop location.

Problem Statement:

Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

Perform following tasks:

1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression & random forest regression models.
5. Evaluate the models & compare their respective scores like R2, RMSE, etc.

Software & Hardware Requirements:

1. Desktop / Laptop
2. Any Operating System
3. Python & Required Libraries
4. Jupyter Notebook

Theory:

Machine Learning:

Machine learning is the field of study that gives computers the capability to learn without being externally programmed.

Page No.
Date

Types of Machine Learning Algorithms:

1) Supervised Machine Learning:

In this technique we train the machines using the "labelled" dataset & based on the training, the machine predicts the output. The main goal of the supervised learning technique is ~~the~~ to map the input variable (x) with the output variable (y).
e.g. Classification, Regression

2) Unsupervised Machine Learning:

In unsupervised machine learning, the machine is trained using the unlabelled dataset & the machine predicts the output without any supervision. The main aim of the unsupervised learning algorithm is to group or categories the unsorted dataset according to the similarities, patterns & differences.
e.g. Clustering, Association

3) Semi-Supervised Learning:

It is a type of Machine Learning algorithm that lies between Supervised & Unsupervised machine learning. It uses the combination of labelled & unlabelled datasets during the training period. To overcome the drawbacks of supervised learning & unsupervised learning algorithms, the concept of semi supervised learning is introduced.

4) Reinforcement Learning:

Reinforcement learning works on a feedback based process, in which an AI agent (A software component) automatically explore its surrounding by hitting & trail, taking action, learning from experiences & improving its performance.

Regression:

Regression is a technique for investigating the relationship between independant variables or features & a dependant variable or outcome. It's used as a method for predictive modelling in machine learning, in which an algorithm is used to predict continuous outcomes.

Linear Regression:

Linear regression is a machine learning algorithm based on supervised learning. It performs a regression task. Linear regression performs a task to predict a dependant variable value (y) based on a given independant variable (x). So this regression technique finds out a linear relationship between x (input) & y (output.) Hence, the name is Linear Regression.

Hypothesis Function for Linear regression:

$$y = \theta_1 + \theta_2 \cdot X$$

Random Forest Regression:

Random Forest is an ensemble technique capable of performing both regression & classification tasks with the use of multiple decision trees & a technique called Bootstrap & Aggregation, commonly known as bagging.

Page No.			
Date			

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Random forest has multiple decision trees as base learning models.

Libraries Used:

1) Numpy: It is used for working with arrays & offers comprehensive mathematical functions, random no. generator, linear algebra routines, fourier transforms, etc.

2) Pandas: It is fast, powerful, flexible & easy to use open-source data analysis & manipulation tool.

3) Seaborn & Matplotlib: These are data visualization libraries.

4) Scikit Learn: Machine learning library. It provides efficient tools for ML & statistical modelling including classification, regression, clustering & dimensionality reduction.

Conclusion: Successfully predicted the price of Uber ride for given pickup & drop location.

At

30/8/22

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import chi2_contingency
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from random import randrange, uniform
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
Train_Data = pd.read_csv(r'trainn.csv')
Train_Data.head(1)
```

Out[2]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude
0	24238194	2015-05-07 19:52:06	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512

In [3]:
```
Train_Data.drop(labels='Unnamed: 0',axis=1,inplace=True)
```

In [4]:
```
Train_Data.drop(labels='key',axis=1,inplace=True)
```

In [5]:
```
Train_Data.shape
```

Out[5]: (24019, 7)

In [6]:
```
test = pd.read_csv(r'testt.csv')
test.head(1)
```

Out[6]:

	Unnamed: 0.2	Unnamed: 0	Unnamed: 0.1	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_l
0	0	37338	31401407	2011-02-10 19:06:00	2011-02-10 19:06:00 UTC	-73.951662	40.79071	-

In [7]:
```
test.shape,Train_Data.shape
```

Out[7]: ((5489, 10), (24019, 7))

In [8]:
```
Train_Data.head()
```

Out[8]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	

In [9]:
```
test.head()
```

Out[9]:

	Unnamed: 0.2	Unnamed: 0	Unnamed: 0.1	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_
0	0	37338	31401407	2011-02-10 19:06:00	2011-02-10 19:06:00 UTC	-73.951662	40.790710	-7
1	1	160901	33158465	2011-06-23 9:24:00	2011-06-23 09:24:00 UTC	-73.951007	40.771508	-7
2	2	40428	10638355	2012-07-14 10:37:00	2012-07-14 10:37:00 UTC	-73.996473	40.747930	-7
3	3	63353	3836845	2014-10-19 22:27:05	2014-10-19 22:27:05 UTC	-73.997934	40.716890	-7
4	4	165491	27114503	2015-05-25 22:54:43	2015-05-25 22:54:43 UTC	-73.952583	40.714039	-7

#As this is Taxi fare data and we know there are many factors which affect the price of taxi like

1. Travelled distance
2. Time of Travel
3. Demand and Availability of Taxi
4. Some special places are more costlier like Airport or other places where there might be toll

In [10]:
```
print(Train_Data.info())
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24019 entries, 0 to 24018
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   fare_amount        24019 non-null  float64
 1   pickup_datetime    24019 non-null  object 
 2   pickup_longitude   24019 non-null  float64
 3   pickup_latitude    24019 non-null  float64
 4   dropoff_longitude  24019 non-null  float64
 5   dropoff_latitude   24019 non-null  float64
 6   passenger_count    24019 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 1.3+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5489 entries, 0 to 5488
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0.2       5489 non-null   int64  
 1   Unnamed: 0         5489 non-null   int64  
 2   Unnamed: 0.1       5489 non-null   int64  
 3   key                5489 non-null   object 
 4   pickup_datetime    5489 non-null   object 
 5   pickup_longitude   5489 non-null   float64
 6   pickup_latitude    5489 non-null   float64
 7   dropoff_longitude  5489 non-null   float64
 8   dropoff_latitude   5489 non-null   float64
 9   passenger_count    5489 non-null   int64  
dtypes: float64(4), int64(4), object(2)
memory usage: 429.0+ KB
None
```

#here we can see there are 8columns in which 6 numerics and 2 are object. #Lets change the type of pickup_datetime from object to DateTime

In [11]:
```
Train_Data["pickup_datetime"] = pd.to_datetime(Train_Data["pickup_datetime"])
```

In [12]:
```
print(Train_Data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24019 entries, 0 to 24018
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype              
---  ------             --------------  -----              
 0   fare_amount        24019 non-null  float64            
 1   pickup_datetime    24019 non-null  datetime64[ns, UTC]
 2   pickup_longitude   24019 non-null  float64            
 3   pickup_latitude    24019 non-null  float64            
 4   dropoff_longitude  24019 non-null  float64            
 5   dropoff_latitude   24019 non-null  float64            
 6   passenger_count    24019 non-null  int64              
dtypes: datetime64[ns, UTC](1), float64(5), int64(1)
memory usage: 1.3 MB
None
```

In [13]:
```
Train_Data.describe()
```

Out[13]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_coun
count	24019.000000	24019.000000	24019.000000	24019.000000	24019.000000	24019.00000(
mean	11.407995	-72.586070	39.952671	-72.575570	39.952896	1.67729
std	10.233290	11.150753	6.041139	10.199019	6.041087	1.29838
min	0.000000	-748.016667	-74.015515	-75.350437	-74.008745	0.00000(
25%	6.000000	-73.992114	40.734940	-73.991557	40.733700	1.00000(
50%	8.500000	-73.981852	40.752399	-73.980173	40.752846	1.00000(
75%	12.500000	-73.967328	40.767165	-73.963594	40.768161	2.00000(
max	350.000000	40.770667	45.031653	40.828377	45.031598	6.00000(

1.Here first thing which we can see is minimum value of fare is negative which is -52 which is not the valid value, so we need to remove the fare which are negative values.

2.Secondly, passenger_count minimum value is 0 and maximum value is 208 which impossible, so we need to remove them as well, for safer side we can think that a taxi can have maximum 7 people.

```
# Lets check if there is any null value
```

In [14]:
```
Train_Data.isnull().sum()
```

Out[14]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [15]:
```
Train_Data.dropna(axis = 0, inplace= True)
```

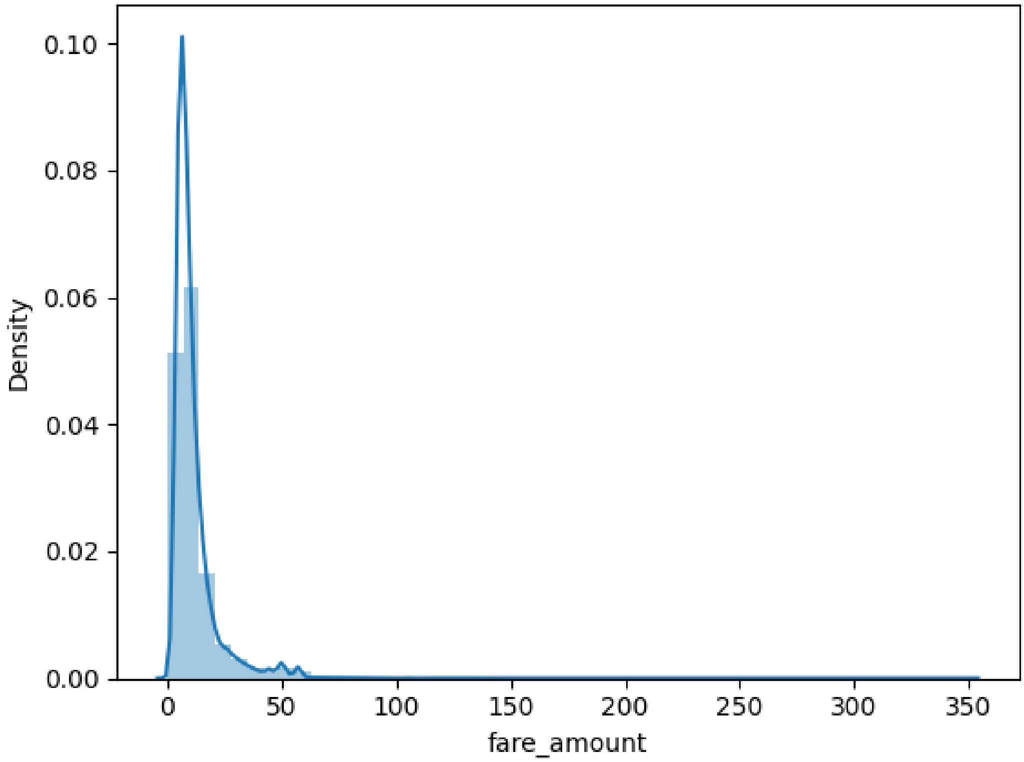
In [16]:
```
print(Train_Data.isnull().sum())
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

#Lets see the statistics of our data

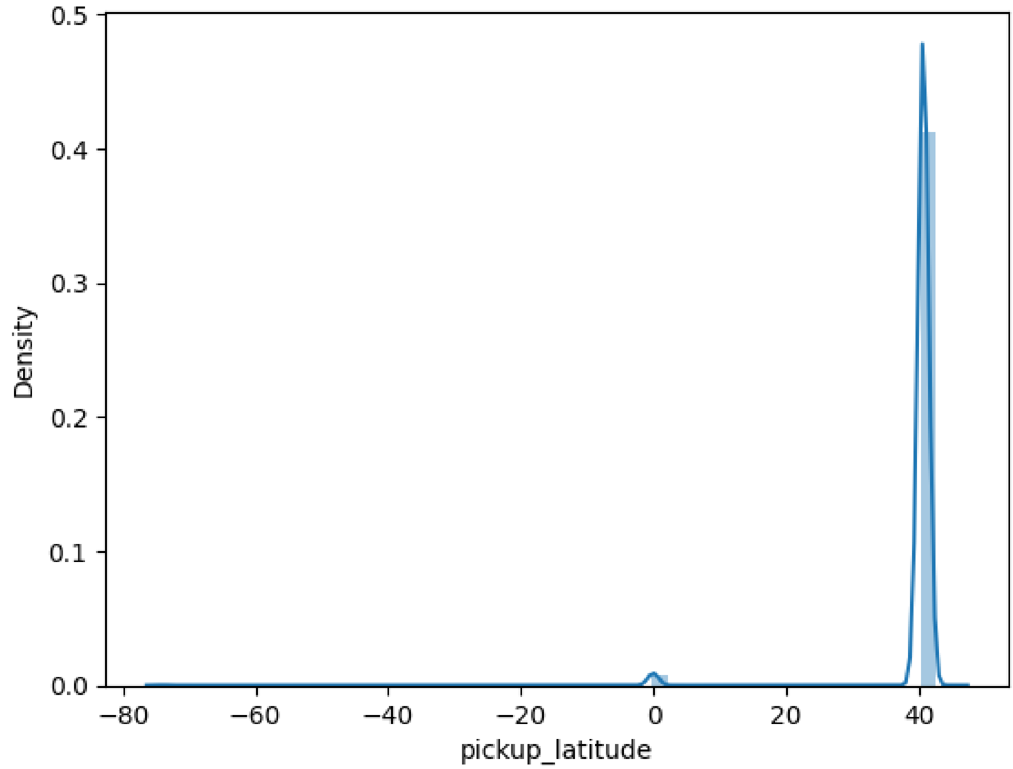
In [17]:
```
sns.distplot(Train_Data['fare_amount'])
```

Out[17]: <AxesSubplot:xlabel='fare_amount', ylabel='Density'>



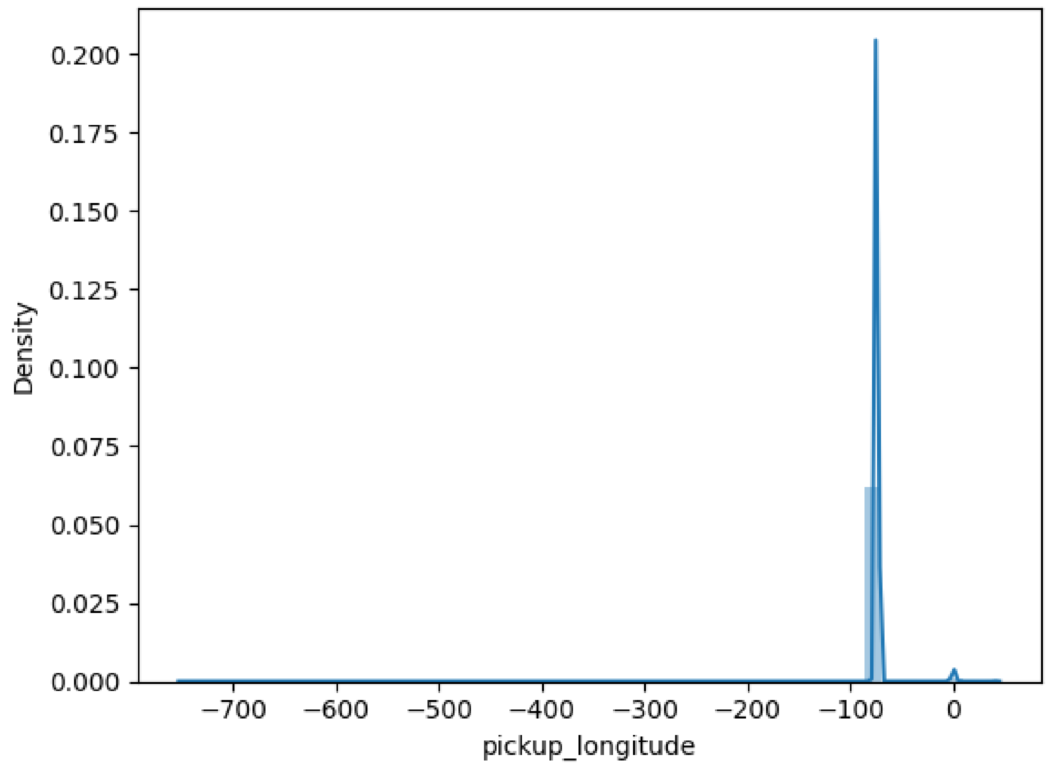
In [18]:
```
sns.distplot(Train_Data['pickup_latitude'])
```

Out[18]: <AxesSubplot:xlabel='pickup_latitude', ylabel='Density'>



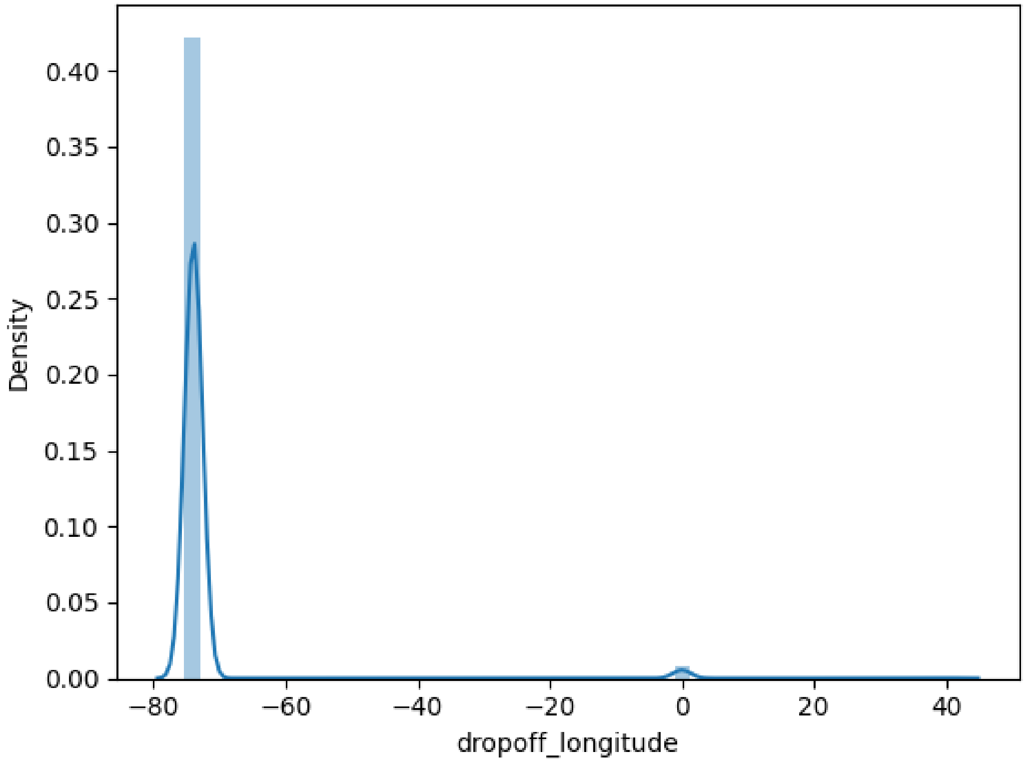
In [19]:
```
sns.distplot(Train_Data['pickup_longitude'])
```

Out[19]: <AxesSubplot:xlabel='pickup_longitude', ylabel='Density'>



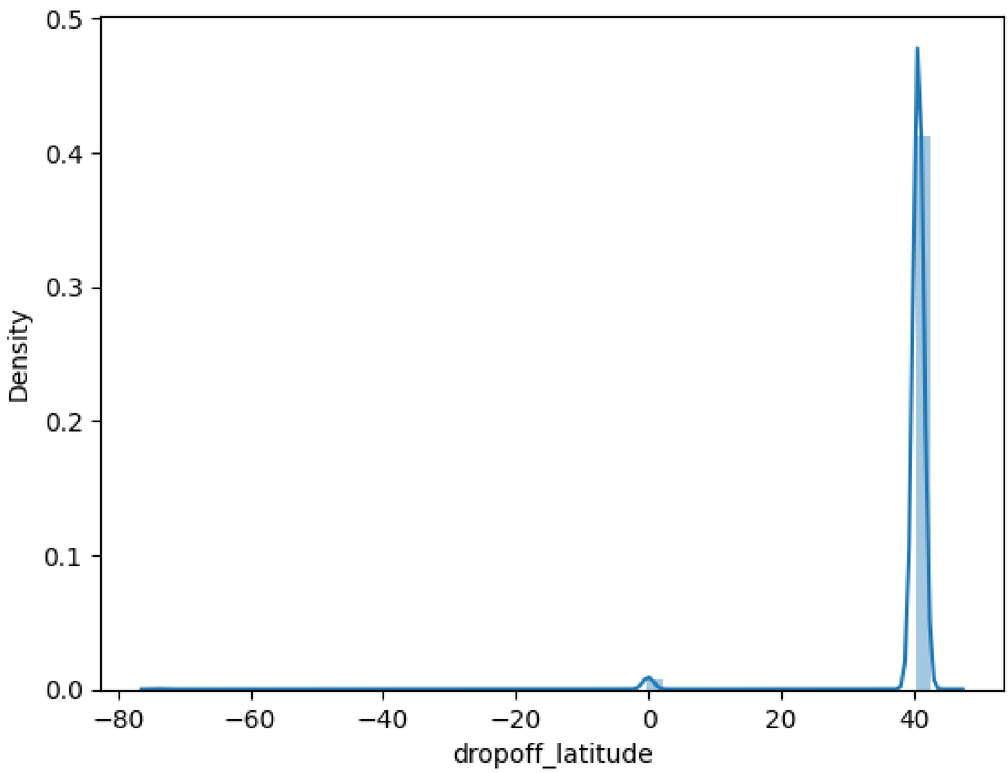
In [20]:
```
sns.distplot(Train_Data['dropoff_longitude'])
```

Out[20]: <AxesSubplot:xlabel='dropoff_longitude', ylabel='Density'>



In [21]:
```
sns.distplot(Train_Data['dropoff_latitude'])
```

Out[21]: <AxesSubplot:xlabel='dropoff_latitude', ylabel='Density'>



In [22]:
```
print("drop_off latitude min value",Train_Data["dropoff_latitude"].min())
print("drop_off latitude max value",Train_Data["dropoff_latitude"].max())
print("drop_off longitude min value", Train_Data["dropoff_longitude"].min())
print("drop_off longitude max value",Train_Data["dropoff_longitude"].max())
print("pickup latitude min value",Train_Data["pickup_latitude"].min())
print("pickup latitude max value",Train_Data["pickup_latitude"].max())
print("pickup longitude min value",Train_Data["pickup_longitude"].min())
print("pickup longitude max value",Train_Data["pickup_longitude"].max())
```

```
drop_off latitude min value -74.008745
drop_off latitude max value 45.031598
drop_off longitude min value -75.35043709
drop_off longitude max value 40.828377
pickup latitude min value -74.015515
pickup latitude max value 45.031653
pickup longitude min value -748.016667
pickup longitude max value 40.770667
```

In [23]:
```
print("drop_off latitude min value",test["dropoff_latitude"].min())
print("drop_off latitude max value",test["dropoff_latitude"].max())
print("drop_off longitude min value", test["dropoff_longitude"].min())
print("drop_off longitude max value",test["dropoff_longitude"].max())
print("pickup latitude min value",test["pickup_latitude"].min())
print("pickup latitude max value",test["pickup_latitude"].max())
print("pickup longitude min value",test["pickup_longitude"].min())
print("pickup longitude max value",test["pickup_longitude"].max())
```

```
drop_off latitude min value -73.98548
drop_off latitude max value 41.366138
drop_off longitude min value -74.68983078
drop_off longitude max value 40.796262
pickup latitude min value -73.988292
pickup latitude max value 41.366138
pickup longitude min value -80.734728
pickup longitude max value 40.812005
```

In [24]:
```
min_longitude=-1491.194073,
min_latitude=-74.001047,
max_longitude=40.812005,
max_latitude=41.709555
```

In [25]:
```
min_longitude=-1491.194073,
min_latitude=-74.001047,
max_longitude=40.812005,
max_latitude=41.709555
```

In [26]:
```
tempdf=Train_Data[(Train_Data["dropoff_latitude"]<min_latitude) |
                (Train_Data["pickup_latitude"]<min_latitude) |
                (Train_Data["dropoff_longitude"]<min_longitude) |
                (Train_Data["pickup_longitude"]<min_longitude) |
                (Train_Data["dropoff_latitude"]>max_latitude) |
                (Train_Data["pickup_latitude"]>max_latitude) |
                (Train_Data["dropoff_longitude"]>max_longitude) |
                (Train_Data["pickup_longitude"]>max_longitude) ]
print("before droping",Train_Data.shape)
Train_Data.drop(tempdf.index,inplace=True)
print("after droping",Train_Data.shape)
```

```
before droping (24019, 7)
after droping (24013, 7)
```

In [27]:
```
import calendar
Train_Data['day']=Train_Data['pickup_datetime'].apply(lambda x:x.day)
Train_Data['hour']=Train_Data['pickup_datetime'].apply(lambda x:x.hour)
Train_Data['month']=Train_Data['pickup_datetime'].apply(lambda x:x.month)
Train_Data['year']=Train_Data['pickup_datetime'].apply(lambda x:x.year)
Train_Data['weekday']=Train_Data['pickup_datetime'].apply(lambda x: calendar.day_name[x
```

In [28]:
```
Train_Data.weekday = Train_Data.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesd
```

In [29]:
```
Train_Data.drop(labels = 'pickup_datetime',axis=1,inplace=True)
```

In [30]:
```
Train_Data.head(1)
Train_Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24013 entries, 0 to 24018
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   fare_amount        24013 non-null  float64
 1   pickup_longitude   24013 non-null  float64
 2   pickup_latitude    24013 non-null  float64
 3   dropoff_longitude  24013 non-null  float64
 4   dropoff_latitude   24013 non-null  float64
 5   passenger_count    24013 non-null  int64  
 6   day                24013 non-null  int64  
 7   hour               24013 non-null  int64  
 8   month              24013 non-null  int64  
 9   year               24013 non-null  int64  
 10  weekday            24013 non-null  int64  
dtypes: float64(5), int64(6)
memory usage: 2.2 MB
```

Model Building

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
x=Train_Data.drop("fare_amount", axis=1)
x
```

Out[32]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour
0	-73.999817	40.738354	-73.999512	40.723217	1	7	19
1	-73.994355	40.728225	-73.994710	40.750325	1	17	20
2	-74.005043	40.740770	-73.962565	40.772647	1	24	21
3	-73.976124	40.790844	-73.965316	40.803349	3	26	8
4	-73.925023	40.744085	-73.973082	40.761247	5	28	17
...	...	...	...	...	...	...	...
24014	-74.006287	40.733092	-73.994787	40.723552	1	7	19
24015	-73.962636	40.767135	-73.989525	40.738478	1	29	9
24016	-73.990382	40.756092	-73.971990	40.753315	2	25	11
24017	-73.988453	40.721255	-73.963474	40.757762	1	21	21
24018	-73.988573	40.736953	-73.978623	40.740888	1	9	15

24013 rows × 10 columns

In [33]:
```
y=Train_Data["fare_amount"]
```

In [34]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=101)
```

In [35]:
```
x_train.head()
```

Out[35]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour
21532	-73.953862	40.810725	-73.980255	40.783567	1	2	18
1431	-73.981990	40.776260	-73.960503	40.775747	3	31	19
12369	-73.974873	40.783367	-73.953148	40.770052	1	13	15
771	-73.977881	40.754195	-73.985108	40.753559	1	13	9
325	-73.982832	40.770589	-73.980315	40.754798	1	12	7

In [36]:
```
x_test.head()
```

Out[36]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour
19466	-73.984755	40.774573	-73.971953	40.763194	2	31	13
12905	-74.014142	40.715860	-73.998223	40.754208	1	24	16
9941	-73.999395	40.761177	-73.995162	40.734937	1	19	8
19451	-73.969682	40.753943	-73.968873	40.788022	5	1	20
1849	-73.982217	40.743170	-73.986019	40.735378	2	26	21

In [37]:
```
y_train.head()
```

Out[37]:
```
21532    11.0
1431      7.7
12369    14.9
771       6.1
325      10.5
Name: fare_amount, dtype: float64
```

In [38]:
```
y_test.head()
```

Out[38]:
```
19466     7.3
12905    10.1
9941     11.5
19451     9.3
1849      4.1
Name: fare_amount, dtype: float64
```

In [39]:
```
print(x_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_train.shape)
```

```
(19210, 10)
(4803, 10)
(4803,)
(19210,)
```

Linear Regression

In [40]:
```
from sklearn.linear_model import LinearRegression
```

In [41]:
```
lrmodel=LinearRegression()
lrmodel.fit(x_train, y_train)
```

Out[41]:
```
▾ LinearRegression
LinearRegression()
```

In [42]:
```
predictedvalues = lrmodel.predict(x_test)
```

In [43]:
```
from sklearn.metrics import mean_squared_error
```

In [44]:
```
lrmodelrmse = np.sqrt(mean_squared_error(predictedvalues, y_test))
print("RMSE value for Linear regression is", lrmodelrmse)
```

```
RMSE value for Linear regression is 9.988890905732967
```

Random Forest

In [45]:
```
from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators=100, random_state=101)
```

In [46]:
```
rfrmodel.fit(x_train,y_train)
rfrmodel_pred= rfrmodel.predict(x_test)
```

In [47]:
```
rfrmodel_rmse=np.sqrt(mean_squared_error(rfrmodel_pred, y_test))
print("RMSE value for Random forest regression is ",rfrmodel_rmse)
```

```
RMSE value for Random forest regression is  5.3606877991468735
```

In [48]:
```
rfrmodel_pred.shape
```

Out[48]:
```
(4803,)
```

Working on Test Data

In [49]:
```
test = pd.read_csv(r'testt.csv')
```

In [50]:
```
test.drop(test[['Unnamed: 0','Unnamed: 0.1','Unnamed: 0.2','key']],axis=1,inplace=True)
```

In [51]:
```
test.isnull().sum()
```

Out[51]:
```
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [52]:
```
test["pickup_datetime"] = pd.to_datetime(test["pickup_datetime"])
```

In [53]:
```
test['day']=test['pickup_datetime'].apply(lambda x:x.day)
test['hour']=test['pickup_datetime'].apply(lambda x:x.hour)
test['month']=test['pickup_datetime'].apply(lambda x:x.month)
test['year']=test['pickup_datetime'].apply(lambda x:x.year)
test['weekday']=test['pickup_datetime'].apply(lambda x: calendar.day_name[x.weekday()])
```

In [54]:
```
test.head(5)
```

Out[54]:

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2011-02-10 19:06:00+00:00	-73.951662	40.790710	-73.947570	40.756220	1
1	2011-06-23 09:24:00+00:00	-73.951007	40.771508	-73.974075	40.763553	1
2	2012-07-14 10:37:00+00:00	-73.996473	40.747930	-73.990298	40.756152	6
3	2014-10-19 22:27:05+00:00	-73.997934	40.716890	-73.952617	40.727149	1
4	2015-05-25 22:54:43+00:00	-73.952583	40.714039	-73.906128	40.711281	1

In [55]:
```
test.drop(['pickup_datetime'], axis = 1, inplace = True)
```

In [56]:
```
test.weekday = test.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thurs
```

In [57]:
```
rfrmodel_pred= rfrmodel.predict(test)
```

In [58]:
```
df = pd.DataFrame(rfrmodel_pred)
df
```

Out[58]:

	0
0	8.3030
1	8.9870
2	7.1210
3	9.8780
4	9.6230
...	...
5484	32.4796
5485	11.0750
5486	36.0938
5487	7.2800
5488	13.7110

5489 rows × 1 columns

In [59]:
```
df.to_csv('pred.csv')
```