Name :

#Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

Assignment No. 1 - Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

## Pre-process the dataset.

1. Identify outliers.

## Check the correlation.

1. Implement linear regression and random forest regression models.
2. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: <https://www.kaggle.com/datasets/yasserh/uber-fares-dataset>

In [1]:

*#Importing the required libraries*

**import** pandas **as** pd **import** numpy **as** np **import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

In [2]:

*#importing the dataset*

df **=** pd**.**read\_csv("uber.csv")

# 1. Pre-process the dataset.

In [3]: df**.**head()

Out[3]: **Unnamed: 0 key fare\_amount pickup\_datetime pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count 0** 24238194 2015-05-07 19:52:06.0000003 7.5 2015-05-07 19:52:06 UTC -73.999817 40.738354 -73.999512 40.723217 1

**1** 27835199 2009-07-17 20:04:56.0000002 7.7 2009-07-17 20:04:56 UTC -73.994355 40.728225 -73.994710 40.750325

1

**2** 44984355 2009-08-24 21:45:00.00000061 12.9 2009-08-24 21:45:00 UTC -74.005043 40.740770 -73.962565 40.772647 1

**3** 25894730 2009-06-26 08:22:21.0000001 5.3 2009-06-26 08:22:21 UTC -73.976124 40.790844 -73.965316 40.803349

3

**4** 17610152 2014-08-28 17:47:00.000000188 16.0 2014-08-28 17:47:00 UTC -73.925023 40.744085 -73.973082 40.761247 5

In [4]:

df**.**info() *#To get the required information of the dataset*

<class 'pandas.core.frame.DataFrame'> RangeIndex: 200000 entries, 0 to 199999 Data columns (total 9 columns):

Unnamed: 0 200000 non-null int64

key 200000 non-null object

fare\_amount 200000 non-null float64 pickup\_datetime 200000 non-null object pickup\_longitude 200000 non-null float64 pickup\_latitude 200000 non-null float64 dropoff\_longitude 199999 non-null float64 dropoff\_latitude 199999 non-null float64 passenger\_count 200000 non-null int64 dtypes: float64(5), int64(2), object(2) memory usage: 13.7+ MB

In [5]:

df**.**columns *#TO get number of columns in the dataset*

Out[5]: Index(['Unnamed: 0', 'key', 'fare\_amount', 'pickup\_datetime', 'pickup\_longitude', 'pickup\_latitude', 'dropoff\_longitude',

'dropoff\_latitude', 'passenger\_count'], dtype='object')

In [6]:

df **=** df**.**drop(['Unnamed: 0', 'key'], axis**=** 1) *#To drop unnamed column as it isn't required*

df**.**head()

In [7]:

Out[7]: **fare\_amount pickup\_datetime pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count 0** 7.5 2015-05-07 19:52:06 UTC -73.999817 40.738354 -73.999512 40.723217 1

**1** 7.7 2009-07-17 20:04:56 UTC -73.994355 40.728225 -73.994710 40.750325

1

**2** 12.9 2009-08-24 21:45:00 UTC -74.005043 40.740770 -73.962565 40.772647 1

**3** 5.3 2009-06-26 08:22:21 UTC -73.976124 40.790844 -73.965316 40.803349

3

**4** 16.0 2014-08-28 17:47:00 UTC -73.925023 40.744085 -73.973082 40.761247 5

In [8]:

df**.**shape *#To get the total (Rows,Columns)*

Out[8]: (200000, 7)

In [9]:

df**.**dtypes *#To get the type of each column*

Out[9]: fare\_amount float64

pickup\_datetime object

pickup\_longitude float64 pickup\_latitude float64 dropoff\_longitude float64 dropoff\_latitude float64 passenger\_count int64 dtype: object

Column pickup\_datetime is in wrong format (Object). Convert it to DateTime Format

In [10]:

df**.**pickup\_datetime **=** pd**.**to\_datetime(df**.**pickup\_datetime)

df**.**dtypes

In [11]:

Out[11]: fare\_amount float64 pickup\_datetime datetime64[ns, UTC]

pickup\_longitude float64

pickup\_latitude float64

dropoff\_longitude float64

dropoff\_latitude float64

passenger\_count int64 dtype: object

Filling Missing values

In [12]: df**.**isnull()**.**sum()

Out[12]: fare\_amount 0

pickup\_datetime 0

pickup\_longitude 0

pickup\_latitude 0

dropoff\_longitude 1

dropoff\_latitude 1

passenger\_count 0

dtype: int64

In [13]:

df['dropoff\_latitude']**.**fillna(value**=**df['dropoff\_latitude']**.**mean(),inplace **= True**) df['dropoff\_longitude']**.**fillna(value**=**df['dropoff\_longitude']**.**median(),inplace **= True**)

In [14]:

df**.**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

To segregate each time of date and time

In [15]:

df**=** df**.**assign(hour **=** df**.**pickup\_datetime**.**dt**.**hour, day**=** df**.**pickup\_datetime**.**dt**.**day, month **=** df**.**pickup\_datetime**.**dt**.**month, year **=** df**.**pickup\_datetime**.**dt**.**year,

dayofweek **=** df**.**pickup\_datetime**.**dt**.**dayofweek)

In [16]:

df**.**head()

Out[16]: **fare\_amount pickup\_datetime pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count hour day month year dayofweek 0** 7.5 2015-05-07 19:52:06+00:00 -73.999817 40.738354 -73.999512 40.723217 1 19 7 5 2015 3

**1** 7.7 2009-07-17 20:04:56+00:00 -73.994355 40.728225 -73.994710 40.750325

1 20 17 7 2009 4

**2** 12.9 2009-08-24 21:45:00+00:00 -74.005043 40.740770 -73.962565 40.772647 1 21 24 8 2009 0

**3** 5.3 2009-06-26 08:22:21+00:00 -73.976124 40.790844 -73.965316 40.803349

3 8 26 6 2009 4

**4** 16.0 2014-08-28 17:47:00+00:00 -73.925023 40.744085 -73.973082 40.761247 5 17 28 8 2014 3

Here we are going to use Heversine formula to calculate the distance between two points and journey, using the longitude and latitude values.

## Heversine formula hav(θ) = sin\*\*2(θ/2).

In [19]:

**from** math **import \***

*# function to calculate the travel distance from the longitudes and latitudes*

**def** distance\_transform(longitude1, latitude1, longitude2, latitude2): travel\_dist **=** []

**for** pos **in** range(len(longitude1)):

long1,lati1,long2,lati2 **=** map(radians,[longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]]) dist\_long **=** long2 **-** long1

dist\_lati **=** lati2 **-** lati1

a **=** sin(dist\_lati**/**2)**\*\***2 **+** cos(lati1) **\*** cos(lati2) **\*** sin(dist\_long**/**2)**\*\***2 c **=** 2 **\*** asin(sqrt(a))**\***6371

travel\_dist**.**append(c)

**return** travel\_dist

In [20]:

df['dist\_travel\_km'] **=** distance\_transform(df['pickup\_longitude']**.**to\_numpy(),

df['pickup\_latitude']**.**to\_numpy(), df['dropoff\_longitude']**.**to\_numpy(), df['dropoff\_latitude']**.**to\_numpy()

)

In [21]:

df**.**head()

Out[21]: **fare\_amount pickup\_datetime pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count hour day month year dayofweek dist\_travel\_km 0** 7.5 2015-05-07 19:52:06+00:00 -73.999817 40.738354 -73.999512 40.723217 1 19 7 5 2015 3 1.683323

**1** 7.7 2009-07-17 20:04:56+00:00 -73.994355 40.728225 -73.994710 40.750325 1 20 17 7 2009 4 2.457590

**2** 12.9 2009-08-24 21:45:00+00:00 -74.005043 40.740770 -73.962565 40.772647 1 21 24 8 2009 0 5.036377

**3** 5.3 2009-06-26 08:22:21+00:00 -73.976124 40.790844 -73.965316 40.803349 3 8 26 6 2009 4 1.661683

**4** 16.0 2014-08-28 17:47:00+00:00 -73.925023 40.744085 -73.973082 40.761247 5 17 28 8 2014 3 4.475450

In [22]:

*# drop the column 'pickup\_daetime' using drop() # 'axis = 1' drops the specified column*

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

In [23]:

df**.**head()

Out[23]: **fare\_amount pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count hour day month year dayofweek dist\_travel\_km 0** 7.5 -73.999817 40.738354 -73.999512 40.723217 1 19 7 5 2015 3 1.683323

**1** 7.7 -73.994355 40.728225 -73.994710 40.750325

1 20 17 7 2009 4 2.457590

**2** 12.9 -74.005043 40.740770 -73.962565 40.772647 1 21 24 8 2009 0 5.036377

**3** 5.3 -73.976124 40.790844 -73.965316 40.803349 3 8 26 6 2009 4 1.661683

**4** 16.0 -73.925023 40.744085 -73.973082 40.761247 5 17 28 8 2014 3 4.475450

Checking outliers and filling them

In

[24]: df**.**plot(kind **=** "box",subplots **= True**,layout **=** (7,2),figsize**=**(15,20)) *#Boxplot to check the outliers*

Out[24]: fare\_amount AxesSubplot(0.125,0.787927;0.352273x0.0920732) pickup\_longitude AxesSubplot(0.547727,0.787927;0.352273x0.0920732)

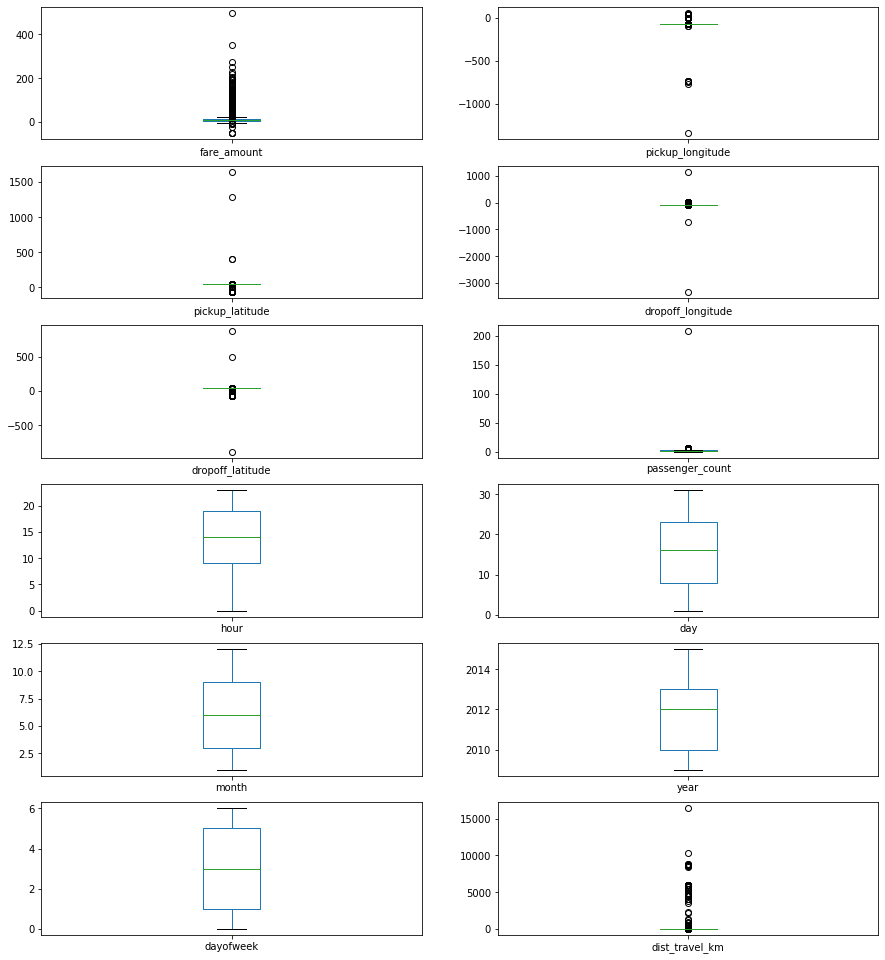
pickup\_latitude AxesSubplot(0.125,0.677439;0.352273x0.0920732) dropoff\_longitude AxesSubplot(0.547727,0.677439;0.352273x0.0920732) dropoff\_latitude AxesSubplot(0.125,0.566951;0.352273x0.0920732) passenger\_count AxesSubplot(0.547727,0.566951;0.352273x0.0920732) hour AxesSubplot(0.125,0.456463;0.352273x0.0920732)

day AxesSubplot(0.547727,0.456463;0.352273x0.0920732)

month AxesSubplot(0.125,0.345976;0.352273x0.0920732)

year AxesSubplot(0.547727,0.345976;0.352273x0.0920732)

dayofweek AxesSubplot(0.125,0.235488;0.352273x0.0920732) dist\_travel\_km AxesSubplot(0.547727,0.235488;0.352273x0.0920732) dtype: object



In [25]:

*#Using the InterQuartile Range to fill the values*

**def** remove\_outlier(df1 , col): Q1 **=** df1[col]**.**quantile(0.25) Q3 **=** df1[col]**.**quantile(0.75)

IQR **=** Q3 **-** Q1

lower\_whisker **=** Q1**-**1.5**\***IQR upper\_whisker **=** Q3**+**1.5**\***IQR

df[col] **=** np**.**clip(df1[col] , lower\_whisker , upper\_whisker)

**return** df1

**def** treat\_outliers\_all(df1 , col\_list):

**for** c **in** col\_list:

df1 **=** remove\_outlier(df , c)

**return** df1

In [26]:

df **=** treat\_outliers\_all(df , df**.**iloc[: , 0::])

df**.**plot(kind **=** "box",subplots **= True**,layout **=** (7,2),figsize**=**(15,20)) *#Boxplot shows that dataset is free from outliers*

In [27]:

Out[27]: fare\_amount AxesSubplot(0.125,0.787927;0.352273x0.0920732) pickup\_longitude AxesSubplot(0.547727,0.787927;0.352273x0.0920732)

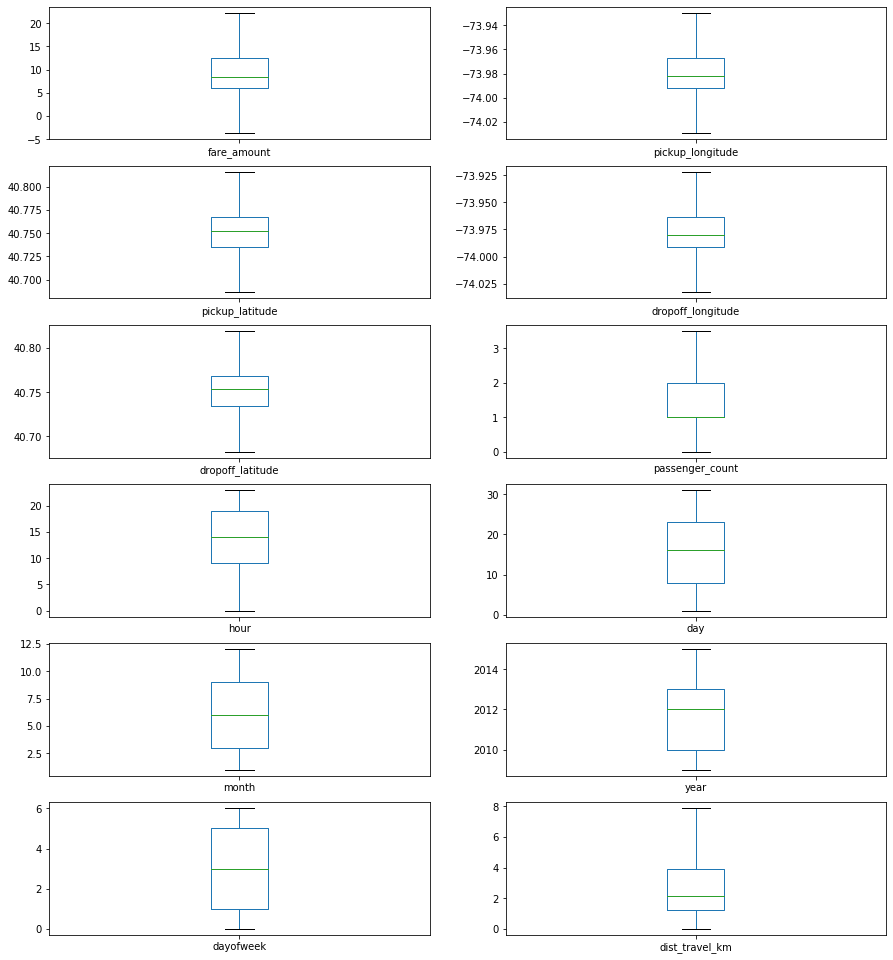
pickup\_latitude AxesSubplot(0.125,0.677439;0.352273x0.0920732) dropoff\_longitude AxesSubplot(0.547727,0.677439;0.352273x0.0920732) dropoff\_latitude AxesSubplot(0.125,0.566951;0.352273x0.0920732) passenger\_count AxesSubplot(0.547727,0.566951;0.352273x0.0920732) hour AxesSubplot(0.125,0.456463;0.352273x0.0920732)

day AxesSubplot(0.547727,0.456463;0.352273x0.0920732)

month AxesSubplot(0.125,0.345976;0.352273x0.0920732)

year AxesSubplot(0.547727,0.345976;0.352273x0.0920732)

dayofweek AxesSubplot(0.125,0.235488;0.352273x0.0920732) dist\_travel\_km AxesSubplot(0.547727,0.235488;0.352273x0.0920732) dtype: object



In [28]:

*#Uber doesn't travel over 130 kms so minimize the distance*

df**=** df**.**loc[(df**.**dist\_travel\_km **>=** 1) **|** (df**.**dist\_travel\_km **<=** 130)] print("Remaining observastions in the dataset:", df**.**shape)

Remaining observastions in the dataset: (200000, 12)

In [29]:

*#Finding inccorect latitude (Less than or greater than 90) and longitude (greater than or less than 180)*

incorrect\_coordinates **=** df**.**loc[(df**.**pickup\_latitude **>** 90) **|**(df**.**pickup\_latitude **< -**90) **|**

(df**.**dropoff\_latitude **>** 90) **|**(df**.**dropoff\_latitude **< -**90) **|** (df**.**pickup\_longitude **>** 180) **|**(df**.**pickup\_longitude **< -**180) **|** (df**.**dropoff\_longitude **>** 90) **|**(df**.**dropoff\_longitude **< -**90)

]

In [30]:

df**.**drop(incorrect\_coordinates, inplace **= True**, errors **=** 'ignore')

df**.**head()

In [31]:

Out[31]: **fare\_amount pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count hour day month year dayofweek dist\_travel\_km 0** 7.5 -73.999817 40.738354 -73.999512 40.723217 1.0 19 7 5 2015 3 1.683323

**1** 7.7 -73.994355 40.728225 -73.994710 40.750325

1.0 20 17 7 2009 4 2.457590

**2** 12.9 -74.005043 40.740770 -73.962565 40.772647 1.0 21 24 8 2009 0 5.036377

**3** 5.3 -73.976124 40.790844 -73.965316 40.803349

3.0 8 26 6 2009 4 1.661683

**4** 16.0 -73.929786 40.744085 -73.973082 40.761247 3.5 17 28 8 2014 3 4.475450

In [32]:

df**.**isnull()**.**sum()

Out[32]: fare\_amount 0

pickup\_longitude 0

pickup\_latitude 0

dropoff\_longitude 0

dropoff\_latitude 0

passenger\_count 0

hour 0

day 0

month 0

year 0

dayofweek 0

dist\_travel\_km 0

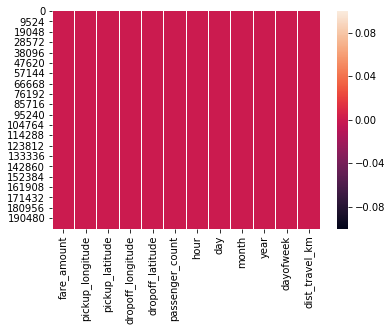
dtype: int64

In [33]:

sns**.**heatmap(df**.**isnull()) *#Free for null values*

Out[33]: <matplotlib.axes.\_subplots.AxesSubplot at 0x8d8af2a080>

In [34]:



corr **=** df**.**corr() *#Function to find the correlation*

corr

In [35]:

Out[35]:

**fare\_amount pickup\_longitude pickup\_latitude dropoff\_longitude dropoff\_latitude passenger\_count hour day month year dayofweek dist\_travel\_km fare\_amount** 1.000000 0.154069 -0.110842 0.218675 -0.125898 0.015778 -0.023623 0.004534 0.030817 0.141277 0.013652 0.844374

**pickup\_latitude** -0.110842 0.259497 1.000000 0.048889 0.515714 -0.012889 0.029681 -0.001553 0.001562 -0.014243 -0.042310 -0.046812

**pickup\_longitude** 0.154069 1.000000 0.259497 0.425619 0.073290 -0.013213 0.011579 -0.003204 0.001169 0.010198 -0.024652 0.098094

**dropoff\_longitude** 0.218675 0.425619 0.048889 1.000000 0.245667 -0.009303 -0.046558 -0.004007 0.002391 0.011346 -0.003336 0.186531

**dropoff\_latitude** -0.125898 0.073290 0.515714 0.245667 1.000000 -0.006308 0.019783 -0.003479 -0.001193 -0.009603 -0.031919 -0.038900

**passenger\_count** 0.015778 -0.013213 -0.012889 -0.009303 -0.006308 1.000000 0.020274 0.002712 0.010351 -0.009749 0.048550 0.009709

**hour** -0.023623 0.011579 0.029681 -0.046558 0.019783 0.020274 1.000000 0.004677 -0.003926 0.002156 -0.086947 -0.038366

**day** 0.004534 -0.003204 -0.001553 -0.004007 -0.003479 0.002712 0.004677 1.000000 -0.017360 -0.012170 0.005617 0.003062

**month** 0.030817 0.001169 0.001562 0.002391 -0.001193 0.010351 -0.003926 -0.017360 1.000000 -0.115859 -0.008786 0.011628

**year** 0.141277 0.010198 -0.014243 0.011346 -0.009603 -0.009749 0.002156 -0.012170 -0.115859 1.000000 0.006113 0.024278

**dayofweek** 0.013652 -0.024652 -0.042310 -0.003336 -0.031919 0.048550 -0.086947 0.005617 -0.008786 0.006113 1.000000 0.027053

**dist\_travel\_km** 0.844374 0.098094 -0.046812 0.186531 -0.038900 0.009709 -0.038366 0.003062 0.011628 0.024278 0.027053 1.000000

In [36]:

fig,axis **=** plt**.**subplots(figsize **=** (10,6))

sns**.**heatmap(df**.**corr(),annot **= True**) *#Correlation Heatmap (Light values means highly correlated)*

Out[36]: <matplotlib.axes.\_subplots.AxesSubplot at 0x8d8affc588>

In [37]:



Dividing the dataset into feature and target values

x **=** df[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude','passenger\_count','hour','day','month','year','dayofweek','dist\_travel\_km']]

y **=** df['fare\_amount']

Dividing the dataset into training and testing dataset

In [38]:

In [39]:

**from** sklearn.model\_selection **import** train\_test\_split X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(x,y,test\_size **=** 0.33)

# Linear Regression

In [40]:

**from** sklearn.linear\_model **import** LinearRegression regression **=** LinearRegression()

In [41]:

regression**.**fit(X\_train,y\_train)

Out[41]: LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

In [42]:

regression**.**intercept\_ *#To find the linear intercept*

Out[42]: 2809.192377415925

In [43]:

regression**.**coef\_ *#To find the linear coeeficient*

Out[43]: array([ 1.75328304e+01, -9.83172673e+00, 1.54611809e+01, -1.69707270e+01, 5.40456388e-02, 9.46950748e-03, 1.66720620e-03, 5.40917698e-02,

3.61743634e-01, -3.69474342e-02, 2.00077959e+00])

In [44]:

prediction **=** regression**.**predict(X\_test) *#To predict the target values*

print(prediction)

In [45]:

[10.80422002 4.74707896 9.95283165 ... 5.89597937 17.00144322

5.38487972]

In [46]:

y\_test

Out[46]: 16850 8.50

181076 4.10

70798 9.30

87421 12.90

169443 22.25

18976 11.00

50921 13.70

199564 14.50

125215 5.30

67510 8.50

85217 22.25

156903 21.50

116795 4.10

112179 16.90

124459 3.70

173299 22.25

51448 19.70

99502 22.25

174467 10.90

78880 20.50

26798 22.25

38501 4.50

63091 12.90

171207 22.25

142238 8.50

101106 7.30

120177 4.50

154585 14.50

75840 5.50

85918 14.00

...

104227 10.10

14172 19.70

49985 3.70

183045 6.50

11927 12.90

93684 4.50

101795 13.70

21444 6.10

85147 8.50

81311 8.00

157686 11.70

194074 6.50

132558 10.50

132616 11.70

188536 5.70

179629 8.90

11277 3.70

147880 7.30

116553 5.70

157394 6.50

103519 13.30

41348 12.90

12608 4.50

6820 5.50

84612 5.00

168836 3.70

39719 21.00

124536 4.90

90432 22.10

12543 4.90

Name: fare\_amount, Length: 66000, dtype: float64

In [47]:

Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

**from** sklearn.metrics **import** r2\_score

r2\_score(y\_test,prediction)

In [48]:

Out[48]: 0.7471032194200018

In [49]:

**from** sklearn.metrics **import** mean\_squared\_error

MSE **=** mean\_squared\_error(y\_test,prediction)

MSE

In [50]:

In [51]:

Out[51]: 7.464818887848474

In [52]:

RMSE **=** np**.**sqrt(MSE)

RMSE

In [53]:

Out[53]: 2.7321820744321696

# Random Forest Regression

In [54]: **from** sklearn.ensemble **import** RandomForestRegressor

In [55]: rf **=** RandomForestRegressor(n\_estimators**=**100) *#Here n\_estimators means number of trees you want to build before making the prediction*

In [56]: rf**.**fit(X\_train,y\_train)

Out[56]: RandomForestRegressor(bootstrap=True, criterion='mse', max\_depth=None,

max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False, random\_state=None, verbose=0, warm\_start=False)

In [57]:

y\_pred **=** rf**.**predict(X\_test)

y\_pred

In [58]:

Out[58]: array([ 9.7025, 4.744 , 9.202 , ..., 6.468 , 16.2802, 4.47 ])

In [59]:

Metrics evaluatin for Random Forest

R2\_Random **=** r2\_score(y\_test,y\_pred)

R2\_Random

In [60]:

Out[60]: 0.8024361566950065

In [64]:

MSE\_Random **=** mean\_squared\_error(y\_test,y\_pred) MSE\_Random

Out[64]: 5.831542440662031

In [65]:

RMSE\_Random **=** np**.**sqrt(MSE\_Random) RMSE\_Random

Out[65]: 2.4148586792319815