## 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 and random forest regression models.

## 5. 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("C:\\Users\\Owner\\Desktop\\Machine Learning BE\\Practical\\Practi
df.head()
```

Out[2]:

|  | Unnamed: 0 | key | fare_amount | pickup_datetime | pickup_longitude | pickup_latitude |
|---|---|---|---|---|---|---|
| **0** | 24238194 | 2015-05-07 19:52:06.0000003 | 7.5 | 2015-05-07 19:52:06 UTC | -73.999817 | 40.738354 |
| **1** | 27835199 | 2009-07-17 20:04:56.0000002 | 7.7 | 2009-07-17 20:04:56 UTC | -73.994355 | 40.728225 |
| **2** | 44984355 | 2009-08-24 21:45:00.00000061 | 12.9 | 2009-08-24 21:45:00 UTC | -74.005043 | 40.740770 |
| **3** | 25894730 | 2009-06-26 08:22:21.0000001 | 5.3 | 2009-06-26 08:22:21 UTC | -73.976124 | 40.790844 |
| **4** | 17610152 | 2014-08-28 17:47:00.000000188 | 16.0 | 2014-08-28 17:47:00 UTC | -73.925023 | 40.744085 |

# 1)Pre-process the dataset

In [3]:

```
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):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [4]:

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

Out[4]:

```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [5]:

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

In [6]:

```
df.head()
```

Out[6]:

| | fare_amount | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_lat |
|---|---|---|---|---|---|---|
| **0** | 7.5 | 2015-05-07 19:52:06 UTC | -73.999817 | 40.738354 | -73.999512 | 40.7 |
| **1** | 7.7 | 2009-07-17 20:04:56 UTC | -73.994355 | 40.728225 | -73.994710 | 40.7 |
| **2** | 12.9 | 2009-08-24 21:45:00 UTC | -74.005043 | 40.740770 | -73.962565 | 40.7 |
| **3** | 5.3 | 2009-06-26 08:22:21 UTC | -73.976124 | 40.790844 | -73.965316 | 40.8 |
| **4** | 16.0 | 2014-08-28 17:47:00 UTC | -73.925023 | 40.744085 | -73.973082 | 40.7 |

In [7]:
```
df.shape #To get the total (Rows,Columns)
```

Out[7]:
```
(200000, 7)
```

In [8]:
```
df.dtypes #To get the type of each column
```

Out[8]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        200000 non-null  float64
 1   pickup_datetime    200000 non-null  object 
 2   pickup_longitude   200000 non-null  float64
 3   pickup_latitude    200000 non-null  float64
 4   dropoff_longitude  199999 non-null  float64
 5   dropoff_latitude   199999 non-null  float64
 6   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
```

In [10]:
```
df.describe() #To get statistics of each columns
```

Out[10]:

| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passe |
|---|---|---|---|---|---|---|
| **count** | 200000.000000 | 200000.000000 | 200000.000000 | 199999.000000 | 199999.000000 | 20 |
| **mean** | 11.359955 | -72.527638 | 39.935885 | -72.525292 | 39.923890 | |
| **std** | 9.901776 | 11.437787 | 7.720539 | 13.117408 | 6.794829 | |
| **min** | -52.000000 | -1340.648410 | -74.015515 | -3356.666300 | -881.985513 | |
| **25%** | 6.000000 | -73.992065 | 40.734796 | -73.991407 | 40.733823 | |
| **50%** | 8.500000 | -73.981823 | 40.752592 | -73.980093 | 40.753042 | |
| **75%** | 12.500000 | -73.967154 | 40.767158 | -73.963658 | 40.768001 | |
| **max** | 499.000000 | 57.418457 | 1644.421482 | 1153.572603 | 872.697628 | |

In [11]:

```
df.isnull()
```

Out[11]:

| | fare_amount | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | drop |
|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | |
| **1** | False | False | False | False | False | |
| **2** | False | False | False | False | False | |
| **3** | False | False | False | False | False | |
| **4** | False | False | False | False | False | |
| **...** | ... | ... | ... | ... | ... | |
| **199995** | False | False | False | False | False | |
| **199996** | False | False | False | False | False | |
| **199997** | False | False | False | False | False | |
| **199998** | False | False | False | False | False | |
| **199999** | False | False | False | False | False | |

200000 rows × 7 columns

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.isnull().sum().sum()
```

Out[13]:

```
2
```

# 2)Filling Missing values

In [14]:

```
df.isnull().sum()
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = Tru
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
```

# 3) Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format

In [15]:

```
df.dtypes
```

Out[15]:

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [16]:

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce',utc=True)

#df.pickup_datetime = pd.to_datetime(df.pickup_datetime)
```

In [17]:

```
df.dtypes
```

Out[17]:

```
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
```

# 4)To segregate each time of date and time

In [18]:

```
df.head()
```

Out[18]:

| | fare_amount | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_lat |
|---|---|---|---|---|---|---|
| **0** | 7.5 | 2015-05-07 19:52:06+00:00 | -73.999817 | 40.738354 | -73.999512 | 40.7 |
| **1** | 7.7 | 2009-07-17 20:04:56+00:00 | -73.994355 | 40.728225 | -73.994710 | 40.7 |
| **2** | 12.9 | 2009-08-24 21:45:00+00:00 | -74.005043 | 40.740770 | -73.962565 | 40.7 |
| **3** | 5.3 | 2009-06-26 08:22:21+00:00 | -73.976124 | 40.790844 | -73.965316 | 40.8 |
| **4** | 16.0 | 2014-08-28 17:47:00+00:00 | -73.925023 | 40.744085 | -73.973082 | 40.7 |

In [19]:

```
df= df.assign(hour = df.pickup_datetime.dt.hour,day= df.pickup_datetime.dt.day,mont
df.head()
```

Out[19]:

| | fare_amount | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_lat |
|---|---|---|---|---|---|---|
| **0** | 7.5 | 2015-05-07 19:52:06+00:00 | -73.999817 | 40.738354 | -73.999512 | 40.7 |
| **1** | 7.7 | 2009-07-17 20:04:56+00:00 | -73.994355 | 40.728225 | -73.994710 | 40.7 |
| **2** | 12.9 | 2009-08-24 21:45:00+00:00 | -74.005043 | 40.740770 | -73.962565 | 40.7 |
| **3** | 5.3 | 2009-06-26 08:22:21+00:00 | -73.976124 | 40.790844 | -73.965316 | 40.8 |
| **4** | 16.0 | 2014-08-28 17:47:00+00:00 | -73.925023 | 40.744085 | -73.973082 | 40.7 |

In [20]:

```
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column
df = df.drop(['pickup_datetime'], axis=1)
```

In [21]:

```
df.head()
```

Out[21]:

| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_ |
|---|---|---|---|---|---|---|
| **0** | 7.5 | -73.999817 | 40.738354 | -73.999512 | 40.723217 | |
| **1** | 7.7 | -73.994355 | 40.728225 | -73.994710 | 40.750325 | |
| **2** | 12.9 | -74.005043 | 40.740770 | -73.962565 | 40.772647 | |
| **3** | 5.3 | -73.976124 | 40.790844 | -73.965316 | 40.803349 | |
| **4** | 16.0 | -73.925023 | 40.744085 | -73.973082 | 40.761247 | |

In [22]:

```
df.describe()
```

Out[22]:

|  | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passe |
|---|---|---|---|---|---|---|
| **count** | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 20 |
| **mean** | 11.359955 | -72.527638 | 39.935885 | -72.525299 | 39.923890 |  |
| **std** | 9.901776 | 11.437787 | 7.720539 | 13.117375 | 6.794812 |  |
| **min** | -52.000000 | -1340.648410 | -74.015515 | -3356.666300 | -881.985513 |  |
| **25%** | 6.000000 | -73.992065 | 40.734796 | -73.991407 | 40.733823 |  |
| **50%** | 8.500000 | -73.981823 | 40.752592 | -73.980093 | 40.753042 |  |
| **75%** | 12.500000 | -73.967154 | 40.767158 | -73.963659 | 40.768001 |  |
| **max** | 499.000000 | 57.418457 | 1644.421482 | 1153.572603 | 872.697628 |  |

In [23]:

```
number_of_columns = len(df.columns)
```

In [24]:

```
number_of_columns
```

Out[24]:

```
11
```

In [25]:

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

def distance_formula(longitude1, latitude1, longitude2, latitude2):
    travel_dist = []

    for pos in range (len(longitude1)):
        lon1, lan1, lon2, lan2 = map(radians, [longitude1[pos], latitude1[pos], lor
        dist_lon = lon2 - lon1
        dist_lan = lan2 - lan1

        a = sin(dist_lan/2)**2 + cos(lan1) * cos(lan2) * sin(dist_lon/2)**2

        #radius of earth = 6371
        c = 2 * asin(sqrt(a)) * 6371
        travel_dist.append(c)

    return  travel_dist
```

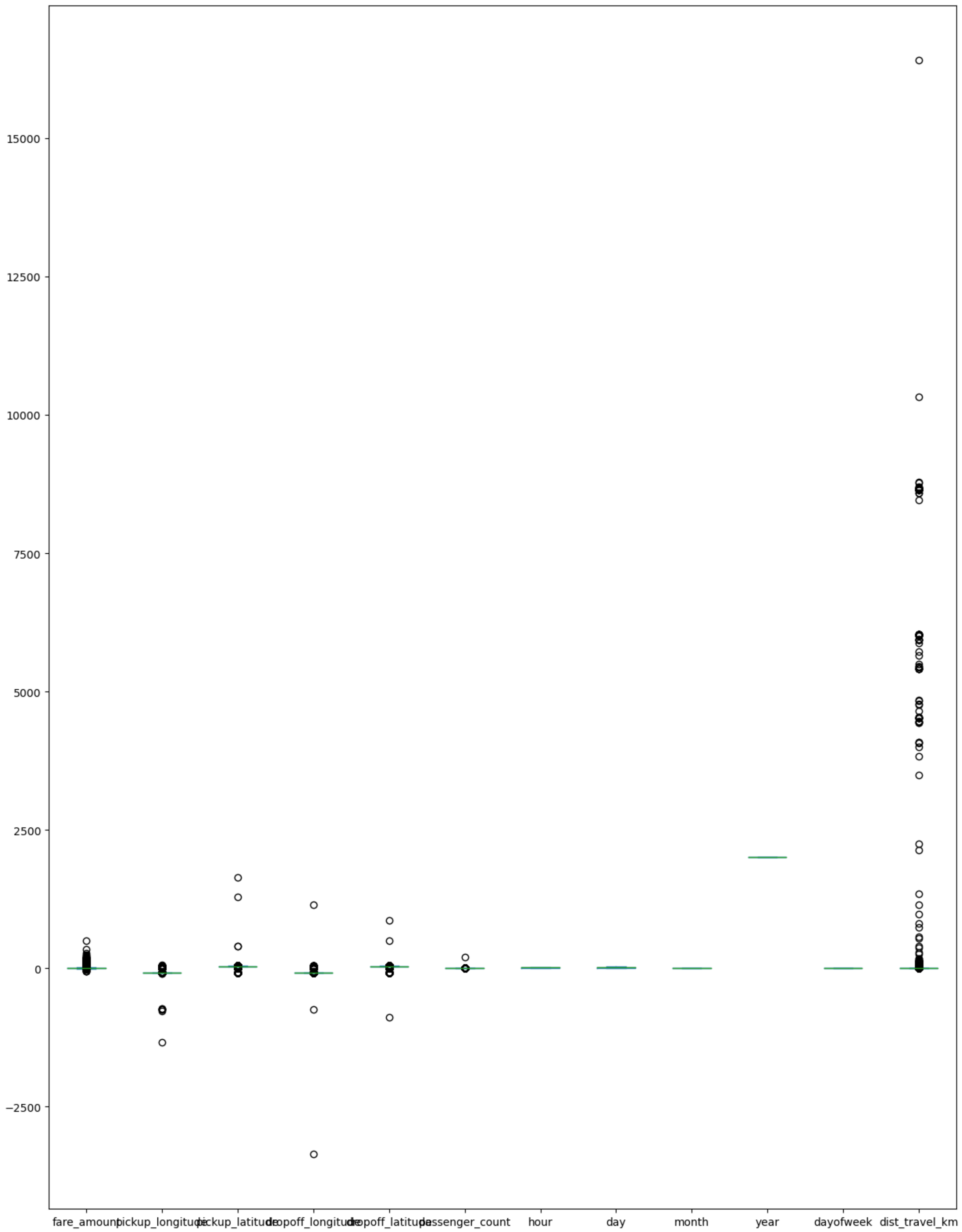
In [26]:

```
df['dist_travel_km'] = distance_formula(df.pickup_longitude.to_numpy(), df.pickup_l
```

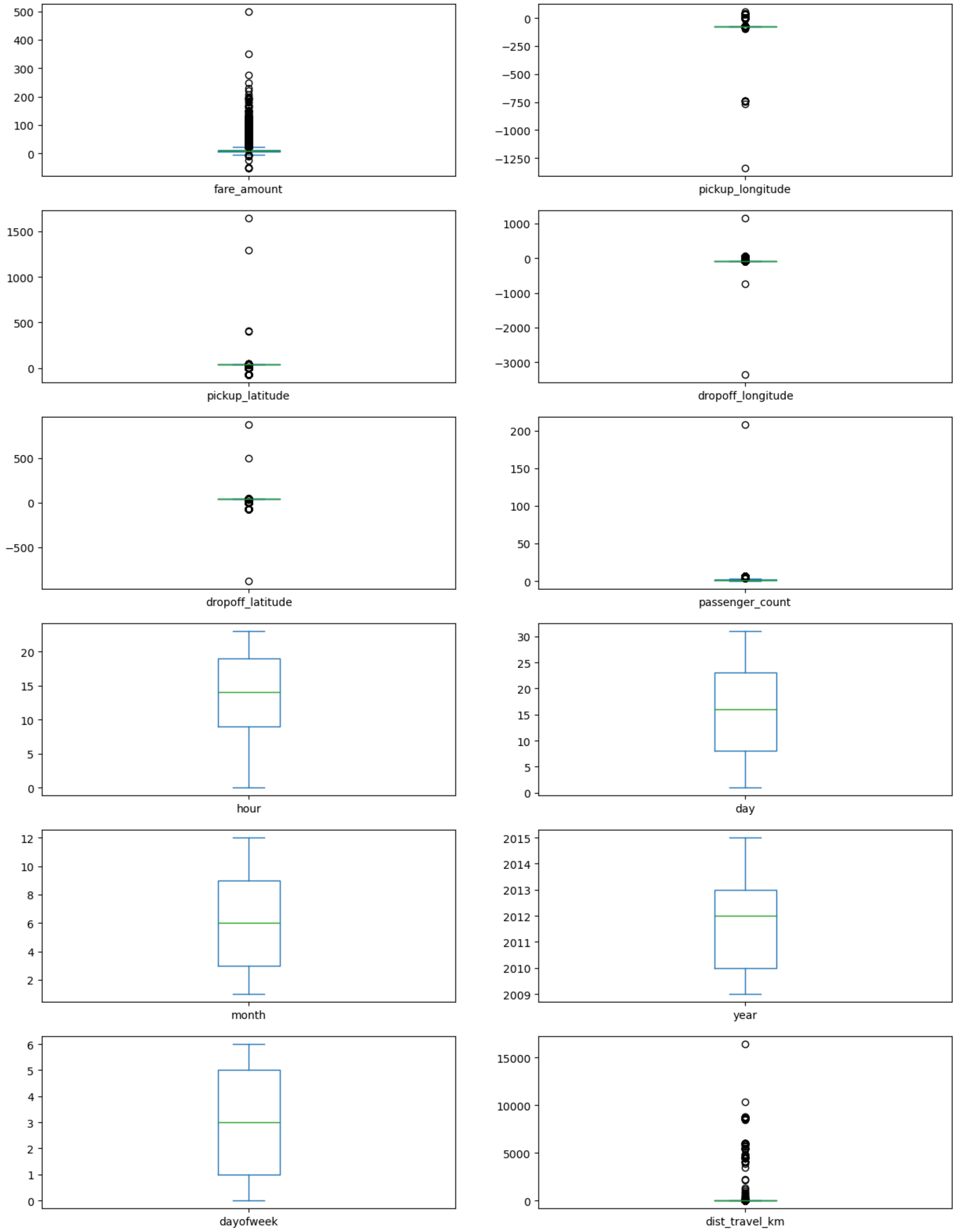
# 5) Checking outliers and filling them

In [27]:

```
df.plot(kind = "box",subplots = False,layout = (6,2),figsize=(15,20)) #Boxplot to c
plt.show()
```



```
In [28]: df.plot(kind = "box",subplots = True,layout = (6,2),figsize=(15,20)) #Boxplot to ch
         plt.show()
```



In [ ]:

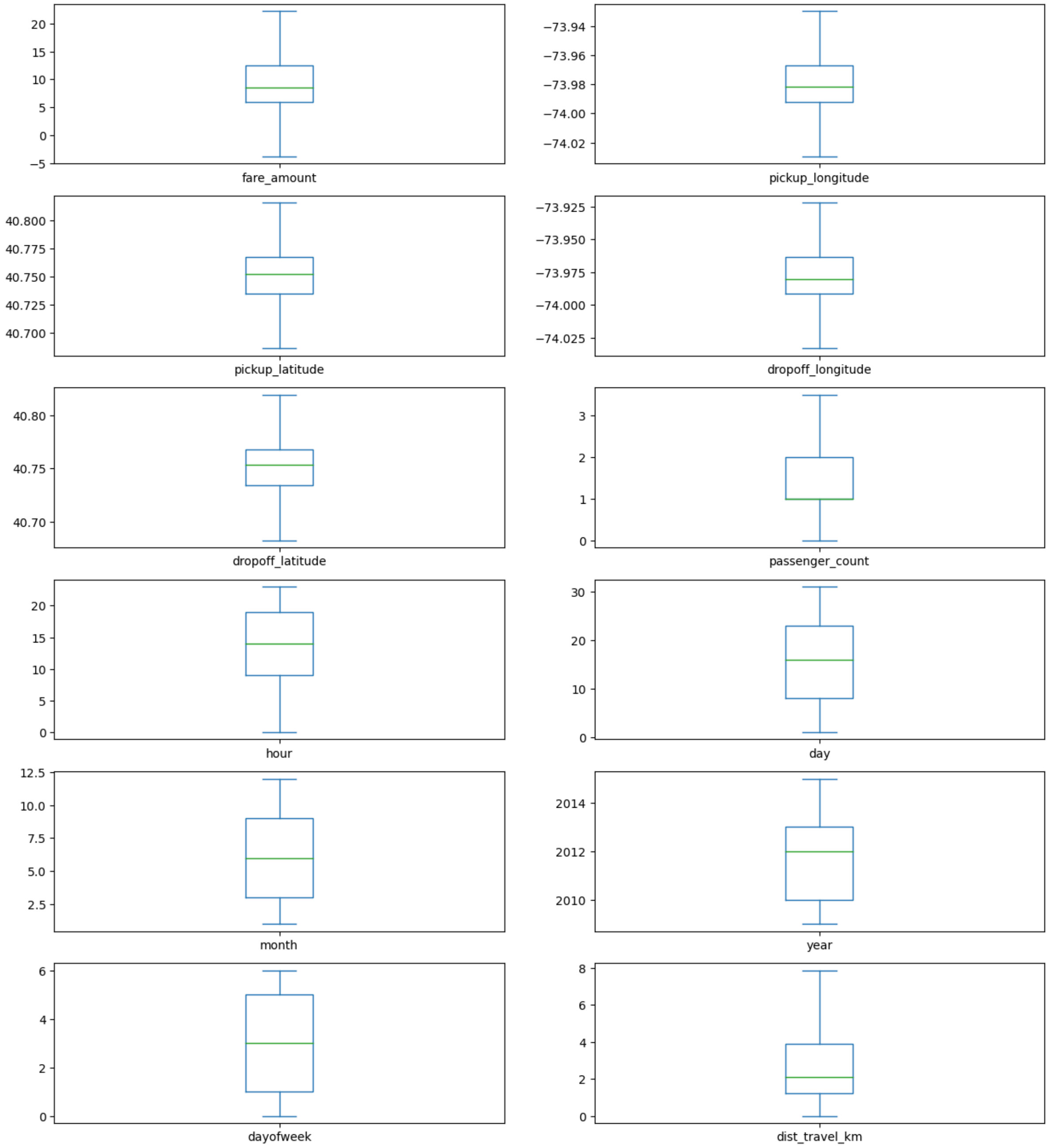
In [29]:

```
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
```

```
df = treat_outliers_all(df , df.columns)
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot shows
```

Out[29]:

```
fare_amount             Axes(0.125,0.786098;0.352273x0.0939024)
pickup_longitude     Axes(0.547727,0.786098;0.352273x0.0939024)
pickup_latitude         Axes(0.125,0.673415;0.352273x0.0939024)
dropoff_longitude    Axes(0.547727,0.673415;0.352273x0.0939024)
dropoff_latitude        Axes(0.125,0.560732;0.352273x0.0939024)
passenger_count      Axes(0.547727,0.560732;0.352273x0.0939024)
hour                    Axes(0.125,0.448049;0.352273x0.0939024)
day                  Axes(0.547727,0.448049;0.352273x0.0939024)
month                   Axes(0.125,0.335366;0.352273x0.0939024)
year                 Axes(0.547727,0.335366;0.352273x0.0939024)
dayofweek               Axes(0.125,0.222683;0.352273x0.0939024)
dist_travel_km       Axes(0.547727,0.222683;0.352273x0.0939024)
dtype: object
```

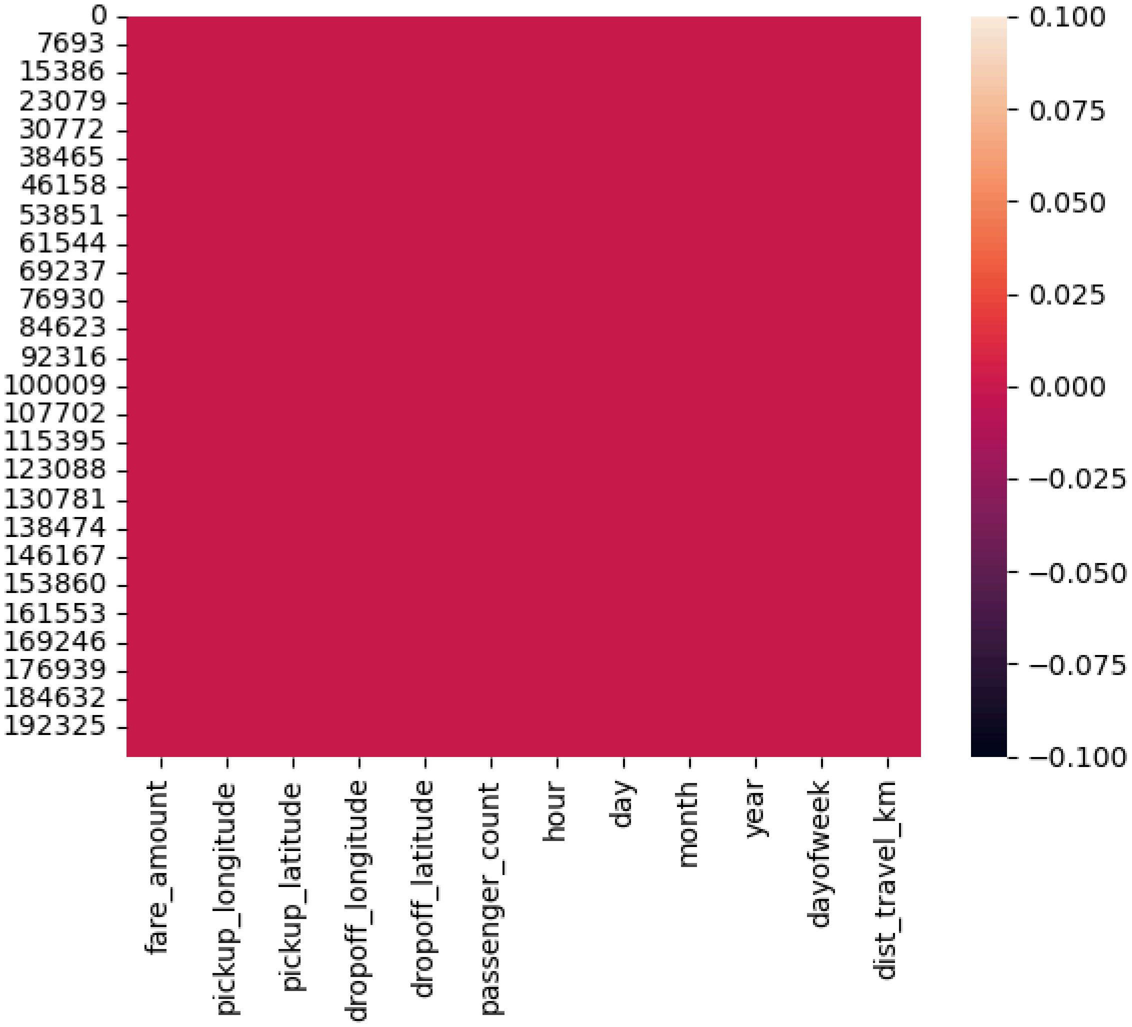


In [30]:

```
#Finding inccorect latitude (Less than or greater than 90) and Longitude (greater t
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latitude < -9
 (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
 (df.pickup_longitude > 180) |(df.pickup_longitude <-180) |
 (df.dropoff_longitude > 90) |(df.dropoff_longitude <-90) ]
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
df.head()
df.isnull().sum()
sns.heatmap(df.isnull()) #Free for null values
```

Out[30]:

```
<Axes: >
```



In [31]:

```
incorrect_coordinates
```

Out[31]:

| fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_c |
|---|---|---|---|---|---|

In [32]:

```
#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)
```

# 7)Check the correlation.

In [33]:

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

Out[33]:

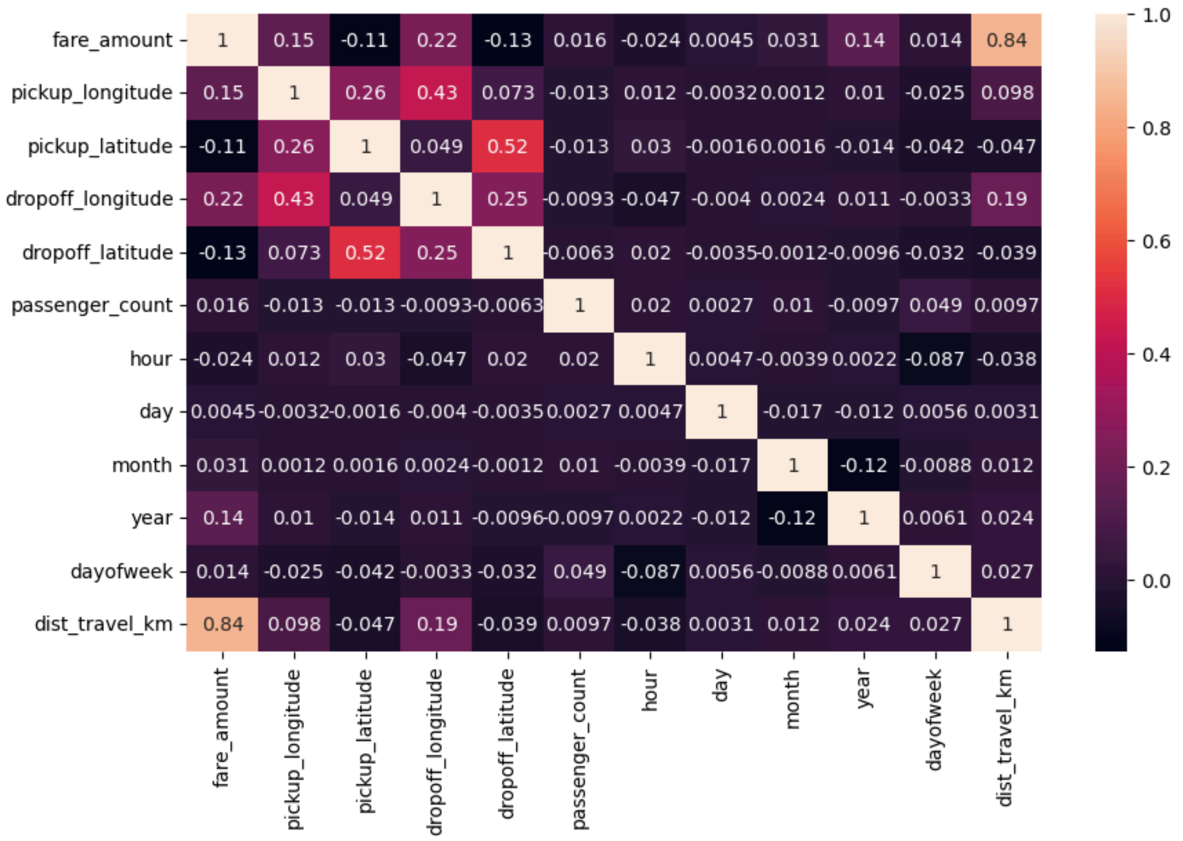
| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latit |
|---|---|---|---|---|---|
| **fare_amount** | 1.000000 | 0.154069 | -0.110842 | 0.218675 | -0.125 |
| **pickup_longitude** | 0.154069 | 1.000000 | 0.259497 | 0.425619 | 0.073 |
| **pickup_latitude** | -0.110842 | 0.259497 | 1.000000 | 0.048889 | 0.515 |
| **dropoff_longitude** | 0.218675 | 0.425619 | 0.048889 | 1.000000 | 0.245 |
| **dropoff_latitude** | -0.125898 | 0.073290 | 0.515714 | 0.245667 | 1.000 |
| **passenger_count** | 0.015778 | -0.013213 | -0.012889 | -0.009303 | -0.006 |
| **hour** | -0.023623 | 0.011579 | 0.029681 | -0.046558 | 0.019 |
| **day** | 0.004534 | -0.003204 | -0.001553 | -0.004007 | -0.003 |
| **month** | 0.030817 | 0.001169 | 0.001562 | 0.002391 | -0.001 |
| **year** | 0.141277 | 0.010198 | -0.014243 | 0.011346 | -0.009 |
| **dayofweek** | 0.013652 | -0.024652 | -0.042310 | -0.003336 | -0.031 |
| **dist_travel_km** | 0.844374 | 0.098094 | -0.046812 | 0.186531 | -0.038 |

In [34]:

```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means highly
```

Out[34]:

```
<Axes: >
```



# 6)Dividing the dataset into feature and target value

```
In [35]: df.columns
```

```
Out[35]: Index(['fare_amount', 'pickup_longitude', 'pickup_latitude',
                'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'hour',
                'day', 'month', 'year', 'dayofweek', 'dist_travel_km'],
               dtype='object')
```

```
In [36]: df.dtypes
```

```
Out[36]: fare_amount          float64
         pickup_longitude     float64
         pickup_latitude      float64
         dropoff_longitude    float64
         dropoff_latitude     float64
         passenger_count      float64
         hour                   int64
         day                    int64
         month                  int64
         year                   int64
         dayofweek              int64
         dist_travel_km       float64
         dtype: object
```

```
In [37]: x = df[['hour','day','month','year','dayofweek','dist_travel_km']]
         y = df['fare_amount']
```

# 7) Scaling and Dividing the dataset into training and testing dataset

```
In [38]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         X_scaled = scaler.fit_transform(x)
```

```
In [39]: from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test = train_test_split(X_scaled,y,test_size = 0.25)
```

```
In [40]: X_train
```

```
Out[40]: array([[ 0.69198905, -1.69264582,  1.66279263,  0.13874222,  1.00238004,
                  0.81391428],
                [ 1.45938811, -0.88687998, -0.08194296,  0.13874222,  0.48875385,
                  1.22698408],
                [-2.07064754,  0.26421407,  0.2088463 ,  0.13874222, -0.53849854,
                 -1.03599597],
                ...,
                [-0.68932925,  0.9548705 , -1.24510003, -1.4772954 , -1.05212474,
                  2.25341213],
                [ 0.38502943,  0.26421407, -1.24510003, -0.9386162 , -0.02487235,
                 -0.86046649],
                [ 1.30590829, -1.11709879,  1.37200337, -0.9386162 ,  1.00238004,
                 -0.9189509 ]])
```

```
In [41]: y_train
```

Out[41]:

```
32942     20.50
175963    14.10
15734      3.30
88189      5.70
151034    17.30
          ...  
77880      8.90
120656     5.30
105690    22.25
56691      3.70
21554     10.50
Name: fare_amount, Length: 150000, dtype: float64
```

In [42]:

```
len(y)
```

Out[42]:

```
200000
```

In [43]:

```
len(y_train)
```

Out[43]:

```
150000
```

In [44]:

```
len(y_test)
```

Out[44]:

```
50000
```

# 8)Linear Regression

In [45]:

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [46]:

```
regression.fit(X_train,y_train)
```

Out[46]:

```
▾ LinearRegression
LinearRegression()
```

In [47]:

```
regression.coef_ #To find the linear coeeficient
```

Out[47]:

```
array([ 0.03649165,  0.02062   ,  0.19146925,  0.68237769, -0.04565736,
        4.57557551])
```

In [48]:

```
regression.intercept_ #To find the linear intercept
```

Out[48]:

```
10.076692713222002
```

In [49]:

```
for i in range(0,len(regression.coef_)):
    print("Theta",i,"=",regression.coef_[i])
```

```
Theta 0 = 0.036491647419307924
Theta 1 = 0.02062000382452531
Theta 2 = 0.19146924507828567
Theta 3 = 0.6823776928080336
Theta 4 = -0.04565735884232406
Theta 5 = 4.575575509222712
```

In [50]:

```
y_prediction = regression.predict(X_test) #To predict the target values
print(y_prediction)
```

```
[ 8.35461279  8.30441313  7.68471507 ...  7.56411998 20.54416486
  5.65656427]
```

In [51]:

```
y_test
```

Out[51]:

```
110440     7.00
170545     6.50
115711    10.50
158343     8.90
115299     9.00
          ...  
147278    13.50
89654     22.25
58881      6.90
183886    22.25
49177      5.30
Name: fare_amount, Length: 50000, dtype: float64
```

In [52]:

```
comparison=pd.DataFrame({"Actual Value":y_test, "Predicted Value":y_prediction })
```

In [53]:

```
comparison
```

Out[53]:

| | Actual Value | Predicted Value |
|---|---|---|
| **110440** | 7.00 | 8.354613 |
| **170545** | 6.50 | 8.304413 |
| **115711** | 10.50 | 7.684715 |
| **158343** | 8.90 | 9.254395 |
| **115299** | 9.00 | 6.520236 |
| **...** | ... | ... |
| **147278** | 13.50 | 14.636705 |
| **89654** | 22.25 | 20.855361 |
| **58881** | 6.90 | 7.564120 |
| **183886** | 22.25 | 20.544165 |
| **49177** | 5.30 | 5.656564 |

50000 rows × 2 columns

In [54]:

```
comparison.reset_index()
```

Out[54]:

| | index | Actual Value | Predicted Value |
|---|---|---|---|
| **0** | 110440 | 7.00 | 8.354613 |
| **1** | 170545 | 6.50 | 8.304413 |
| **2** | 115711 | 10.50 | 7.684715 |
| **3** | 158343 | 8.90 | 9.254395 |
| **4** | 115299 | 9.00 | 6.520236 |
| **...** | ... | ... | ... |
| **49995** | 147278 | 13.50 | 14.636705 |
| **49996** | 89654 | 22.25 | 20.855361 |
| **49997** | 58881 | 6.90 | 7.564120 |
| **49998** | 183886 | 22.25 | 20.544165 |
| **49999** | 49177 | 5.30 | 5.656564 |

50000 rows × 3 columns

In [55]:

```
comparison.reset_index().drop(["index"],axis=1)
```

Out[55]:

| | Actual Value | Predicted Value |
|---|---|---|
| **0** | 7.00 | 8.354613 |
| **1** | 6.50 | 8.304413 |
| **2** | 10.50 | 7.684715 |
| **3** | 8.90 | 9.254395 |
| **4** | 9.00 | 6.520236 |
| **...** | ... | ... |
| **49995** | 13.50 | 14.636705 |
| **49996** | 22.25 | 20.855361 |
| **49997** | 6.90 | 7.564120 |
| **49998** | 22.25 | 20.544165 |
| **49999** | 5.30 | 5.656564 |

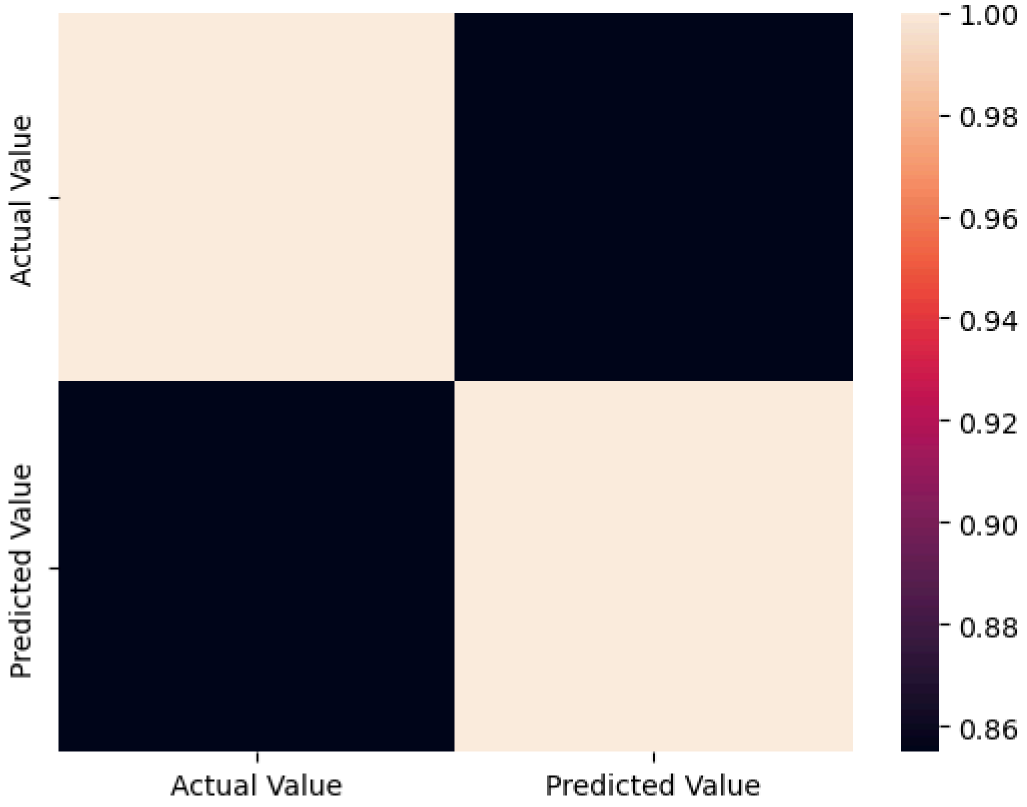
50000 rows × 2 columns

In [56]:

```
sns.heatmap(comparison.corr())
```

Out[56]:

```
<Axes: >
```



## 9)Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

```
In [57]: from sklearn.metrics import r2_score
         r2_score(y_test,y_prediction)

Out[57]: 0.730770273098684
```

```
In [58]: from sklearn.metrics import mean_squared_error
         MSE = mean_squared_error(y_test,y_prediction)
         MSE

Out[58]: 7.976966086868074
```

```
In [59]: RMSE = np.sqrt(MSE)
         RMSE

Out[59]: 2.8243523305119127
```

## 10)Random Forest Regression

```
In [65]: from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of tre
         rf.fit(X_train,y_train)
         y_pred = rf.predict(X_test)
         y_pred

Out[65]: array([ 8.1  ,  8.064,  8.4  , ...,  7.196, 17.36 ,  5.559])
```

In [66]:

```
from sklearn.metrics import r2_score
r2_score(y_test,y_prediction)
```

Out[66]:

```
0.730770273098684
```

In [67]:

```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test,y_prediction)
MSE
```

Out[67]:

```
7.976966086868074
```

In [68]:

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

Out[68]:

```
2.8243523305119127
```

In [ ]:

In [ ]: