1. Importing Necessary Libraries and uploading Datasets and saving it to a DataFrame

In [1]:

```
import pandas as pd                          # for analying Data
import numpy as np                           # for working with arrays
import seaborn as sns                        # for visualisation
import matplotlib.pyplot as plt              #for visualisation
%matplotlib inline
sns.set(color_codes=True)
```

Reading the dataset

In [2]:

```
uber_data=pd.read_csv("uber.csv")
```

2. Checking head of the datasets

In [3]:

```
uber_data.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

3. Checking the types of data and size of each column along with the shape of the dataset

In [4]:

```
uber_data.info()
```

```
<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
```

4. Describing the Data

In [5]:

```
uber_data.describe()
```

Out[5]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	40.768001	2.000000
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

5. Shape of the Data

In [6]:

```
uber_data.shape
```

Out[6]:

```
(200000, 9)
```

6. Dropping irrelevant columns

In [7]:

```
uber_data = uber_data.drop(['key'], axis=1)
```

7. Checking for Missing Data

In [8]:

```
uber_data.isna().sum()
```

Out[8]:

```
Unnamed: 0           0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

8. Dropping Missing Values

In [9]:

```
uber_data = uber_data.dropna(axis = 0, how ='any')
```

In [10]:

```
uber_data
```

Out[10]:

	Unnamed: 0	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	27835199	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	44984355	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	25894730	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5
...	...	...	...	...	...	...	...	...
199995	42598914	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73.986525	40.740297	1
199996	16382965	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74.006672	40.739620	1
199997	27804658	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73.858957	40.692588	2
199998	20259894	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-73.983215	40.695415	1
199999	11951496	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-73.985508	40.768793	1

199999 rows × 8 columns

In [11]:

```
uber_data.isna().sum()                          #Number of null values in the each column of the dataset
```

Out[11]:

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

9. Feature Engineering

In [12]:

```
def haversine (lon_1, lon_2, lat_1, lat_2):

    lon_1, lon_2, lat_1, lat_2 = map(np.radians, [lon_1, lon_2, lat_1, lat_2])  #Degrees to Radians


    diff_lon = lon_2 - lon_1
    diff_lat = lat_2 - lat_1


    km = 2 * 6371 * np.arcsin(np.sqrt(np.sin(diff_lat/2.0)**2 +
                                  np.cos(lat_1) * np.cos(lat_2) * np.sin(diff_lon/2.0)**2))

    return km
```

In [13]:

```
#Adding new column named distance into the dataset using the pickup and drop off locations

uber_data['distance']= haversine(uber_data['pickup_longitude'],uber_data['dropoff_longitude'],
                             uber_data['pickup_latitude'],uber_data['dropoff_latitude'])
```

In [14]:

```
uber_data['distance'] = uber_data['distance'].astype(float).round(2)     # Round-off Optional
```

In [15]:

```
uber_data.head()
```

Out[15]:

	Unnamed: 0	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	distance
0	24238194	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1	1.68
1	27835199	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1	2.46
2	44984355	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1	5.04
3	25894730	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3	1.66
4	17610152	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5	4.48

In [16]:

```
#Dropping pickup and drop locations in the dataset because they are combined and formed as distance feature

uber_data = uber_data.drop(['pickup_latitude', 'pickup_longitude','dropoff_longitude','dropoff_latitude'], axis=1)
```

In [17]:

```
uber_data.head()
```

Out[17]:

	Unnamed: 0	fare_amount	pickup_datetime	passenger_count	distance
0	24238194	7.5	2015-05-07 19:52:06 UTC	1	1.68
1	27835199	7.7	2009-07-17 20:04:56 UTC	1	2.46
2	44984355	12.9	2009-08-24 21:45:00 UTC	1	5.04
3	25894730	5.3	2009-06-26 08:22:21 UTC	3	1.66
4	17610152	16.0	2014-08-28 17:47:00 UTC	5	4.48

In [18]:

```
uber_data.pickup_datetime=pd.to_datetime(uber_data.pickup_datetime)
```

In [19]:

```
#Converting the date time and days into features for better utilization of the dataset

uber_data['year'] = uber_data.pickup_datetime.dt.year
uber_data['month'] = uber_data.pickup_datetime.dt.month
uber_data['weekday'] = uber_data.pickup_datetime.dt.weekday
uber_data['hour'] = uber_data.pickup_datetime.dt.hour
```

In [20]:

```
#Converting the month and hours into segements

uber_data['Monthly_Quarter'] = uber_data.month.map({1:'Q1',2:'Q1',3:'Q1',4:'Q2',5:'Q2',6:'Q2',7:'Q3',
                                    8:'Q3',9:'Q3',10:'Q4',11:'Q4',12:'Q4'})
uber_data['Hourly_Segments'] = uber_data.hour.map({0:'H1',1:'H1',2:'H1',3:'H1',4:'H2',5:'H2',6:'H2',7:'H2',8:'H3',
                                   9:'H3',10:'H3',11:'H3',12:'H4',13:'H4',14:'H4',15:'H4',16:'H5',
                                   17:'H5',18:'H5',19:'H5',20:'H6',21:'H6',22:'H6',23:'H6'})
```

In [21]:

```
#Dropping pickup date,month and hour in the dataset because they are converted into usable features
uber_data.drop(['pickup_datetime','month', 'hour',], axis=1, inplace=True)
```

In [22]:

```
uber_data.head()
```

Out[22]:

	Unnamed: 0	fare_amount	passenger_count	distance	year	weekday	Monthly_Quarter	Hourly_Segments
0	24238194	7.5	1	1.68	2015	3	Q2	H5
1	27835199	7.7	1	2.46	2009	4	Q3	H6
2	44984355	12.9	1	5.04	2009	0	Q3	H6
3	25894730	5.3	3	1.66	2009	4	Q2	H3
4	17610152	16.0	5	4.48	2014	3	Q3	H5

2. Visualization

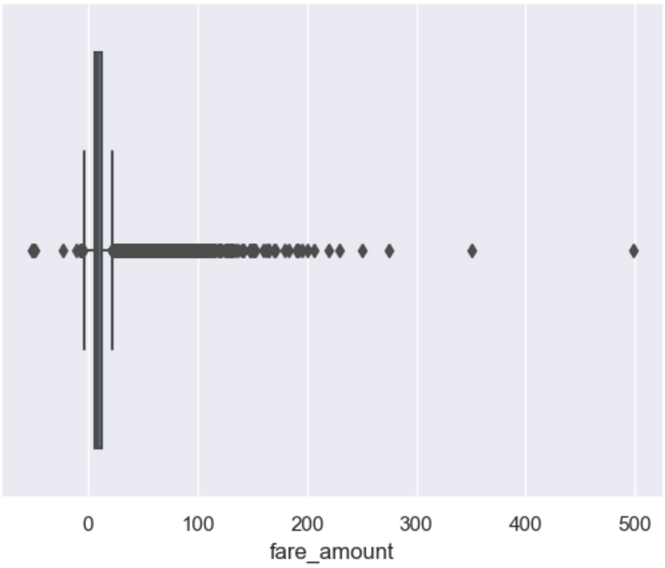
1. Box Plot to detect Outliers

In [23]:

```
sns.boxplot(x=uber_data['fare_amount'])
```

Out[23]:

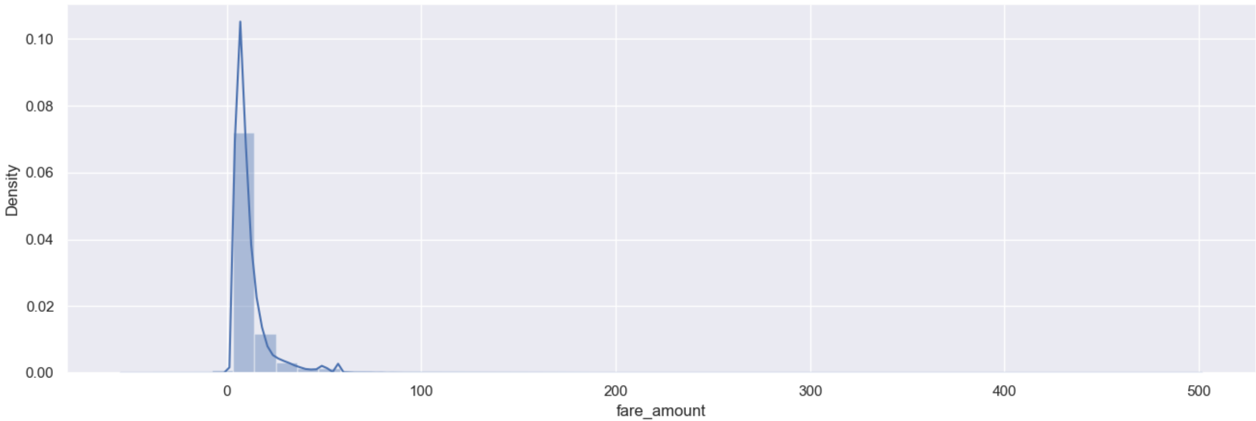
```
<AxesSubplot: xlabel='fare_amount'>
```



In [24]:

```
import warnings

warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,1,1)
sns.distplot(uber_data['fare_amount'])
plt.show()
```



In [25]:

```
print("Highest allowed",uber_data['fare_amount'].mean() + 3*uber_data['fare_amount'].std())
print("Lowest allowed",uber_data['fare_amount'].mean() - 3*uber_data['fare_amount'].std())
```

```
Highest allowed 41.06517154773827
Lowest allowed -18.345388448822774
```

In [26]:

```
uber_data[(uber_data['fare_amount'] > 39.81) | (uber_data['fare_amount'] < -17.1)]
```

Out[26]:

	Unnamed: 0	fare_amount	passenger_count	distance	year	weekday	Monthly_Quarter	Hourly_Segments
48	22405517	56.80	1	0.00	2013	3	Q1	H6
84	25485719	49.57	1	0.00	2009	4	Q3	H3
104	46435788	43.00	2	11.88	2015	4	Q2	H5
204	6403066	45.00	1	20.07	2010	5	Q4	H2
226	24085207	49.80	1	18.21	2012	6	Q3	H5
...	...	...	...	...	...	...	...	...
199914	17686068	57.33	5	21.56	2014	4	Q4	H2
199972	31236221	45.00	1	20.28	2010	4	Q3	H4
199976	1780041	49.70	1	24.90	2011	1	Q4	H6
199977	21117828	43.50	1	20.85	2012	1	Q4	H6
199982	13096190	57.33	1	19.48	2014	2	Q3	H3

5770 rows × 8 columns

In [27]:

```
new_df = uber_data[(uber_data['fare_amount'] > 39.81) & (uber_data['fare_amount']> -17.1)]
new_df
```

Out[27]:

	Unnamed: 0	fare_amount	passenger_count	distance	year	weekday	Monthly_Quarter	Hourly_Segments
48	22405517	56.80	1	0.00	2013	3	Q1	H6
84	25485719	49.57	1	0.00	2009	4	Q3	H3
104	46435788	43.00	2	11.88	2015	4	Q2	H5
204	6403066	45.00	1	20.07	2010	5	Q4	H2
226	24085207	49.80	1	18.21	2012	6	Q3	H5
...	...	...	...	...	...	...	...	...
199914	17686068	57.33	5	21.56	2014	4	Q4	H2
199972	31236221	45.00	1	20.28	2010	4	Q3	H4
199976	1780041	49.70	1	24.90	2011	1	Q4	H6
199977	21117828	43.50	1	20.85	2012	1	Q4	H6
199982	13096190	57.33	1	19.48	2014	2	Q3	H3

5765 rows × 8 columns

In [28]:

```
upper_limit = uber_data['fare_amount'] .mean() + 3*uber_data['fare_amount'] .std()
lower_limit = uber_data['fare_amount'] .mean() - 3*uber_data['fare_amount'] .std()
```

In [29]:

```
uber_data['fare_amount'] = np.where(
    uber_data['fare_amount']>upper_limit,
    upper_limit,
    np.where(
        uber_data['fare_amount']<lower_limit,
        lower_limit,
        uber_data['fare_amount']
    )
)
```

In [30]:

```
uber_data['fare_amount'].describe()
```

Out[30]:

```
count    199999.000000
mean         11.008919
std           8.088040
min         -18.345388
25%           6.000000
50%           8.500000
75%          12.500000
max          41.065172
Name: fare_amount, dtype: float64
```

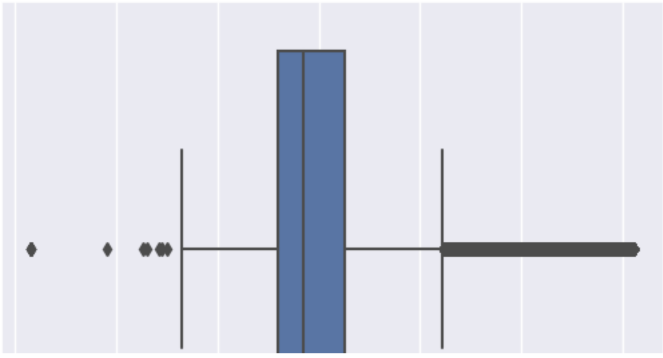
Box Plot After Removing Outliers

In [31]:

```
sns.boxplot(x=uber_data['fare_amount'])
```

Out[31]:

```
<AxesSubplot: xlabel='fare_amount'>
```



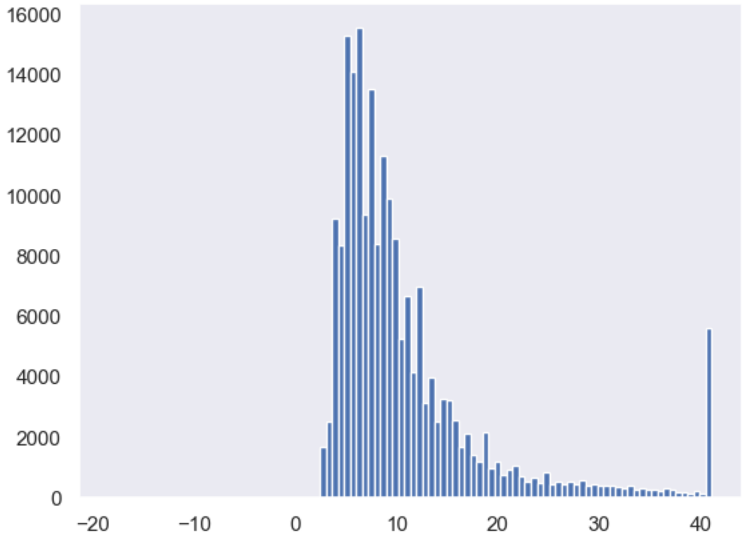
2. Histogram for Distribution Analyis

In [32]:

```
uber_data['fare_amount'].hist(bins=100,grid=False)
```

Out[32]:

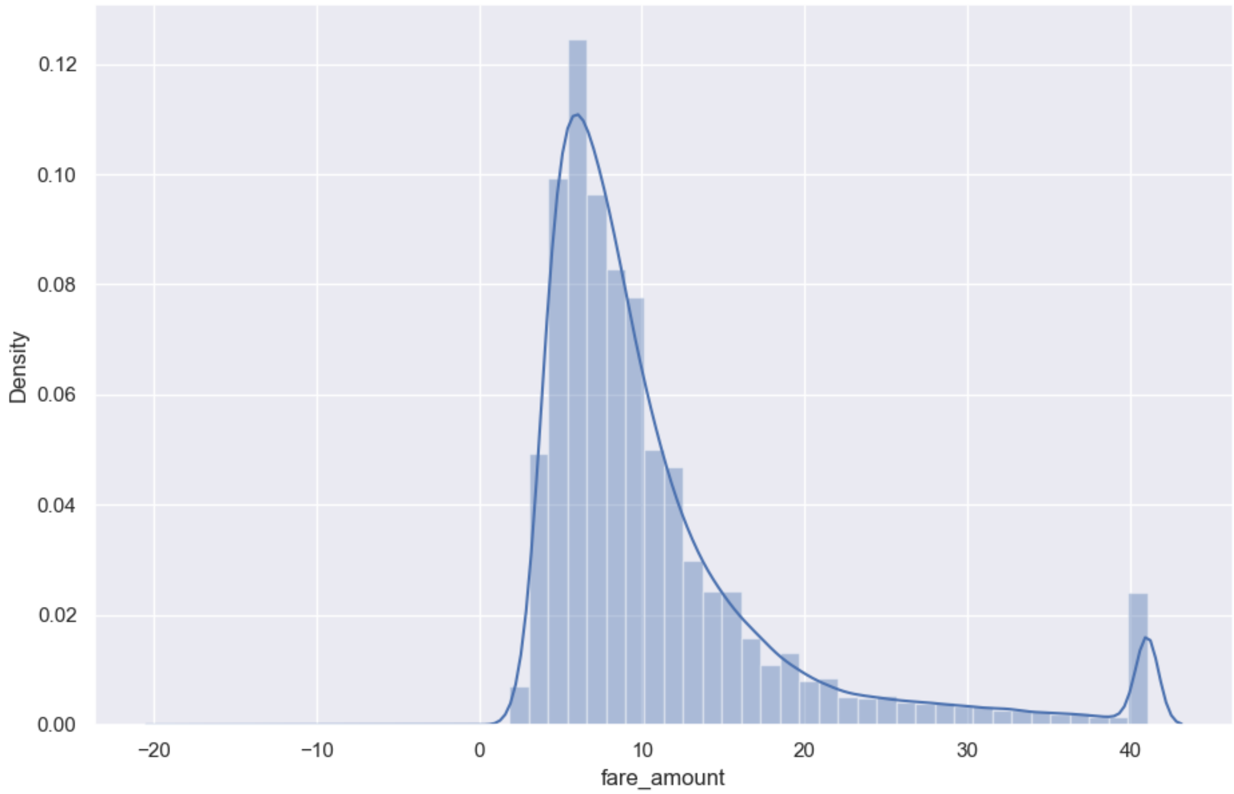
```
<AxesSubplot: >
```



3. Distribution of the data

In [33]:

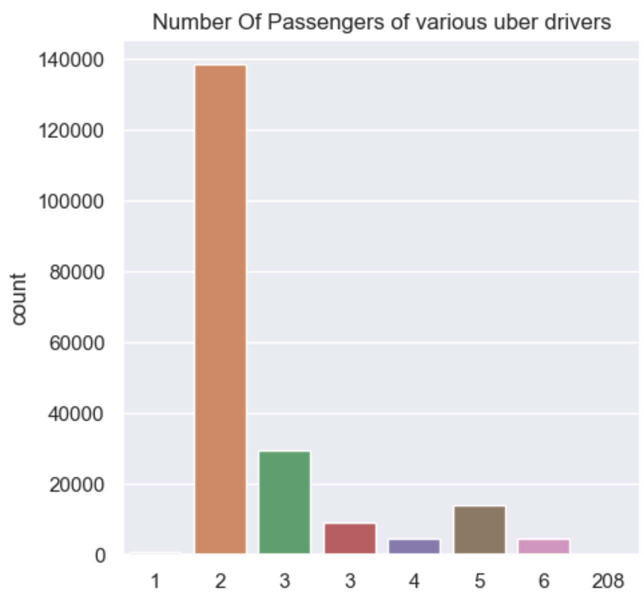
```
#DistPlot
plt.figure(figsize=(11,7))
sns.distplot(uber_data["fare_amount"])
plt.show()
```



Distribution of data to see the distribution and as we can see due to outliers.

In [34]:

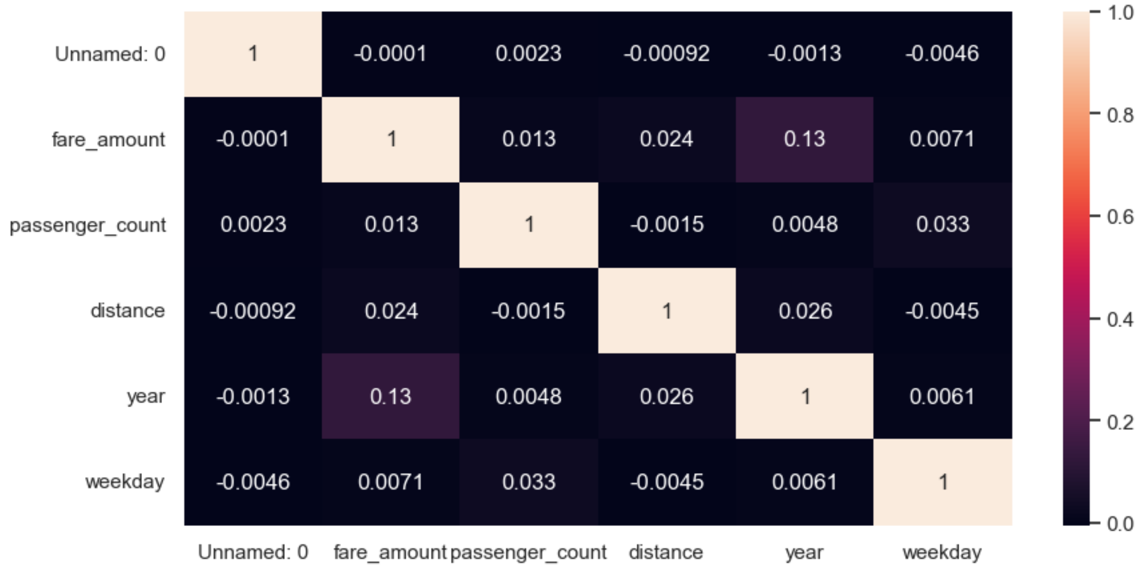
```
fig, ax = plt.subplots(figsize = (5, 5))
sns.countplot(x = uber_data.passenger_count.values, data=uber_data)
labels = [item.get_text() for item in ax.get_xticklabels()]
labels[0] = '1'
labels[1] = '2'
labels[2] = '3'
ax.set_xticklabels(labels)
ax.set_title("Number Of Passengers of various uber drivers")
plt.show()
```



4. Heat Map

In [35]:

```
#heat map
plt.figure(figsize=(10,5))
hm = sns.heatmap(uber_data.corr(),annot=True)
plt.show()
```



Creating a Heatmap to check the correlation of dataset and we can see that Passenger_count and year has more correlation with respect to fare_amount.

Scatter Plot

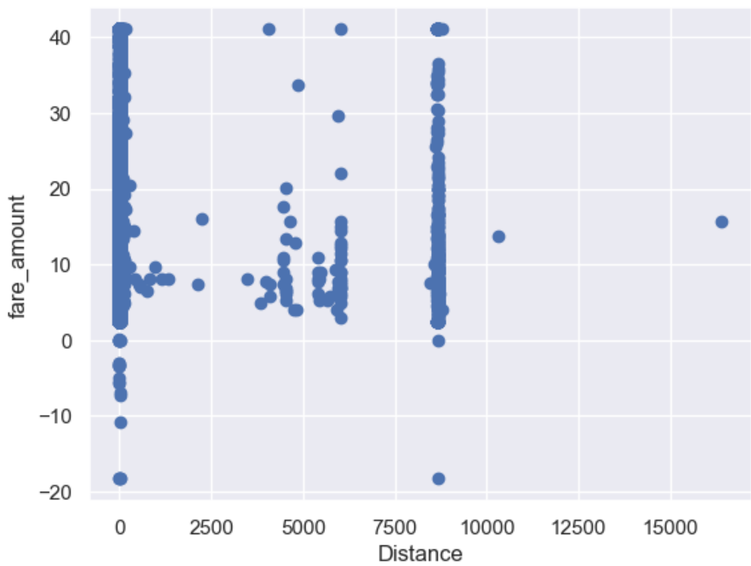
Distance vs Fare Amount

In [36]:

```
plt.scatter(uber_data['distance'], uber_data['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

Out[36]:

```
Text(0, 0.5, 'fare_amount')
```



Pie Chart

In [37]:

```
grouped=uber_data.groupby('passenger_count')
print(grouped['fare_amount'].mean())
```

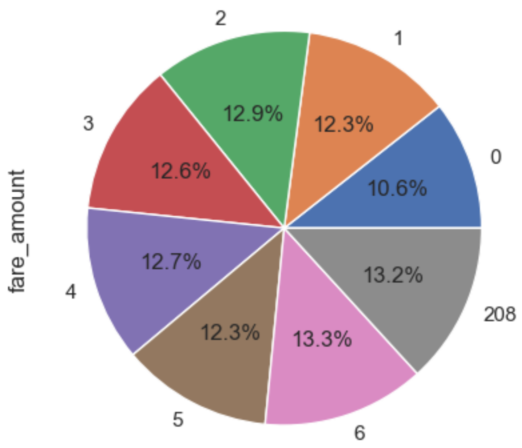
```
passenger_count
0       9.388547
1      10.908245
2      11.378192
3      11.119097
4      11.273607
5      10.922754
6      11.784432
208    11.700000
Name: fare_amount, dtype: float64
```

In [38]:

```
grouped['fare_amount'].mean().plot(kind='pie',y='fare_amount',autopct='%1.1f%%')
```

Out[38]:

```
<AxesSubplot: ylabel='fare_amount'>
```



In [39]:

```
uber_data.drop(uber_data[uber_data['distance'] > 60].index, inplace = True)
uber_data.drop(uber_data[uber_data['distance'] == 0].index, inplace = True)
uber_data.drop(uber_data[uber_data['fare_amount'] == 0].index, inplace = True)
uber_data.drop(uber_data[uber_data['fare_amount'] < 0].index, inplace = True)
```

In [40]:

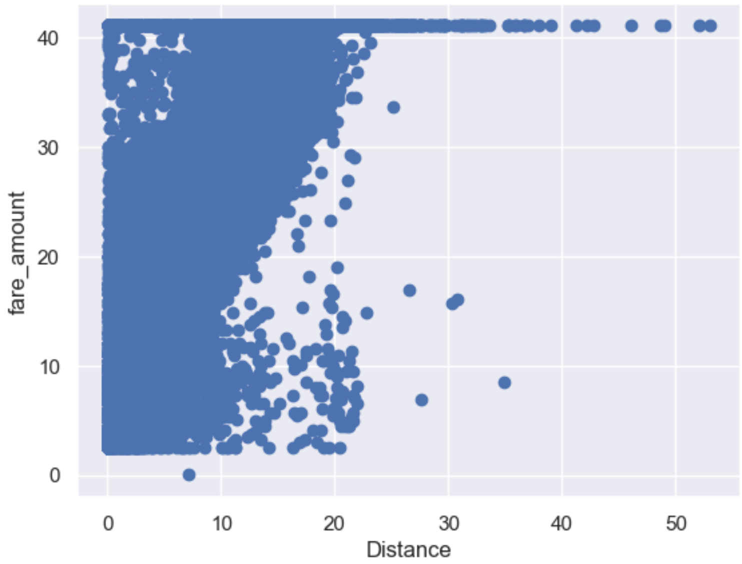
```
uber_data.drop(uber_data[(uber_data['fare_amount']>100) & (uber_data['distance']<1)].index, inplace = True )
uber_data.drop(uber_data[(uber_data['fare_amount']<100) & (uber_data['distance']>100)].index, inplace = True )
```

In [41]:

```
plt.scatter(uber_data['distance'], uber_data['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

Out[41]:

```
Text(0, 0.5, 'fare_amount')
```



In [42]:

```
uber_data.head()
```

Out[42]:

	Unnamed: 0	fare_amount	passenger_count	distance	year	weekday	Monthly_Quarter	Hourly_Segments
0	24238194	7.5	1	1.68	2015	3	Q2	H5
1	27835199	7.7	1	2.46	2009	4	Q3	H6
2	44984355	12.9	1	5.04	2009	0	Q3	H6
3	25894730	5.3	3	1.66	2009	4	Q2	H3
4	17610152	16.0	5	4.48	2014	3	Q3	H5

In [43]:

```
X = uber_data.drop(columns=['fare_amount','year','Monthly_Quarter','Hourly_Segments'])
y = uber_data['fare_amount']
```

Splitting the Dataset

Training and Test Set

In [44]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

In [45]:

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

Out[45]:

```
0
```

Simple Linear Regression

Training the simple linear regression model on the training set

In [46]:

```
from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

```
Training set score: 0.79
Test set score: 0.7864422
```

Actual vs Predicted Values

In [47]:

```
y_pred = l_reg.predict(X_test)
df = {'Actual': y_test, 'Predicted': y_pred}

from tabulate import tabulate
print(tabulate(df, headers = 'keys', tablefmt = 'psql'))
```

```
|    7     |     7.05229 |
|    7.7   |     8.42661 |
|    5.3   |     6.01908 |
|    8     |     8.81026 |
|    4     |     5.86363 |
|    5.3   |     6.09901 |
|   41.0652 |   29.6998  |
|    8.5   |     6.54519 |
|   11.7   |    16.46    |
|    9     |     7.32847 |
|   16.1   |     8.42902 |
|   10.9   |    11.8248  |
|   22     |    14.4625  |
|    8.5   |    10.1424  |
|   10.5   |     9.4284  |
|    9     |     7.35697 |
|    5.5   |     6.78705 |
|    6.9   |     4.68774 |
|    4.1   |     5.64013 |
|   10.5   |    13.5933  |
```

Accuracy Checking

Finding the MSE,MAE, RMSE, etc.

In [48]:

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
#print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 2.2618411903054576
Mean Squared Error: 13.877840600737255
Root Mean Squared Error: 3.7252973842013275
```

Statistical Analysis

1. One sample T- test

In [49]:

```
import scipy
```

In [50]:

```
uber_data1=uber_data
```

In [57]:

```
uber_data1
```

Out[57]:

	Unnamed: 0	fare_amount	passenger_count	distance	year	weekday	Monthly_Quarter	Hourly_Segments
0	24238194	7.5	1	1.68	2015	3	Q2	H5
1	27835199	7.7	1	2.46	2009	4	Q3	H6
2	44984355	12.9	1	5.04	2009	0	Q3	H6
3	25894730	5.3	3	1.66	2009	4	Q2	H3
4	17610152	16.0	5	4.48	2014	3	Q3	H5
...	...	...	...	...	...	...	...	...
199995	42598914	3.0	1	0.11	2012	6	Q4	H3
199996	16382965	7.5	1	1.88	2014	4	Q1	H1
199997	27804658	30.9	2	12.85	2009	0	Q2	H1
199998	20259894	14.5	1	3.54	2015	2	Q2	H4
199999	11951496	14.1	1	5.42	2010	5	Q2	H2

193490 rows × 8 columns

In [51]:

```
uber_data1['fare_amount'].mean()
```

Out[51]:

```
10.999986181927715
```

In [52]:

```
scipy.stats.ttest_1samp(uber_data1['fare_amount'],popmean=11)
```

Out[52]:

```
Ttest_1sampResult(statistic=-0.0007581454529918717, pvalue=0.9993950882877489)
```

2. Two Sample Paired T- Test

In [53]:

```
uber_data1
```

Out[53]:

	Unnamed: 0	fare_amount	passenger_count	distance	year	weekday	Monthly_Quarter	Hourly_Segments
0	24238194	7.5	1	1.68	2015	3	Q2	H5
1	27835199	7.7	1	2.46	2009	4	Q3	H6
2	44984355	12.9	1	5.04	2009	0	Q3	H6
3	25894730	5.3	3	1.66	2009	4	Q2	H3
4	17610152	16.0	5	4.48	2014	3	Q3	H5
...	...	...	...	...	...	...	...	...
199995	42598914	3.0	1	0.11	2012	6	Q4	H3
199996	16382965	7.5	1	1.88	2014	4	Q1	H1
199997	27804658	30.9	2	12.85	2009	0	Q2	H1
199998	20259894	14.5	1	3.54	2015	2	Q2	H4
199999	11951496	14.1	1	5.42	2010	5	Q2	H2

193490 rows × 8 columns

In [58]:

```
uber_data2=uber_data1['fare_amount'][:96745]
```

In [59]:

```
uber_data3=uber_data1['fare_amount'][96745:]
```

In [60]:

```
scipy.stats.ttest_rel(uber_data2,uber_data3)
```

Out[60]:

```
Ttest_relResult(statistic=0.4252172814795518, pvalue=0.6706792331557414)
```

In [ ]: