In [3]:
```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
#We do not want to see warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
#import data
data = pd.read_csv("7431_uber.csv")
```

In [5]:
```
#Create a data copy
df = data.copy()
```

In [6]:
```
#Print data
df.head()
```

Out[6]:

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

In [7]:
```
#Get Info
df.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
```

In [8]:
```
#pickup_datetime is not in required data format
```

```
df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"])
```

In [9]:
```
df.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  datetime64[ns, UTC]
 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: datetime64[ns, UTC](1), float64(5), int64(2), object(1)
memory usage: 13.7+ MB
```

In [10]:
```
#Statistics of data
df.describe()
```

Out[10]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longituc
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.00000
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.52529
std	1.601382e+07	9.901776	11.437787	7.720539	13.11740
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.66630
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.99140
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.98009
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.96365
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.57260

In [11]:
```
#Number of missing values
df.isnull().sum()
```

Out[11]:
```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [12]:
```
#Correlation - only numeric columns
df.select_dtypes(include=[np.number]).corr()
```

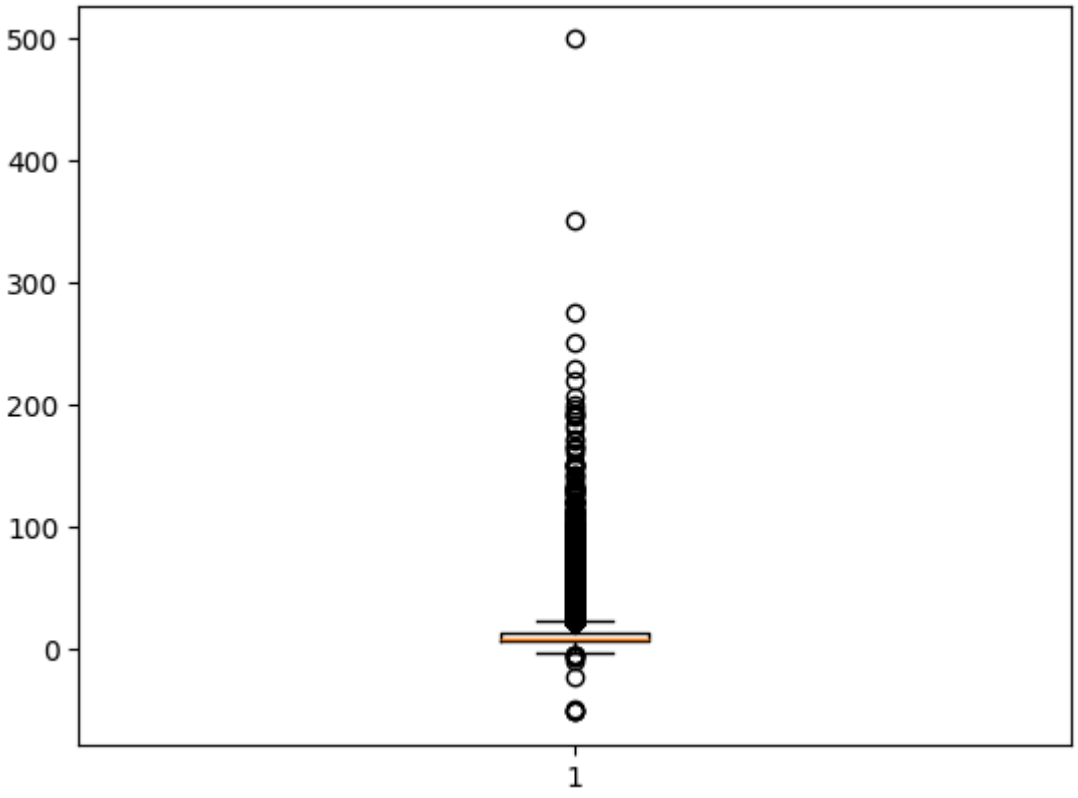
Out[12]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_l
Unnamed: 0	1.000000	0.000589	0.000230	-0.000341	
fare_amount	0.000589	1.000000	0.010457	-0.008481	
pickup_longitude	0.000230	0.010457	1.000000	-0.816461	
pickup_latitude	-0.000341	-0.008481	-0.816461	1.000000	
dropoff_longitude	0.000270	0.008986	0.833026	-0.774787	
dropoff_latitude	0.000271	-0.011014	-0.846324	0.702367	
passenger_count	0.002257	0.010150	-0.000414	-0.001560	

In [13]:
```
#Drop the rows with missing values
df.dropna(inplace=True)
```

In [14]:
```
plt.boxplot(df['fare_amount'])
```

Out[14]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x255a28602d0>,
  <matplotlib.lines.Line2D at 0x255a2862ad0>],
 'caps': [<matplotlib.lines.Line2D at 0x255a2862c10>,
  <matplotlib.lines.Line2D at 0x255a2862d50>],
 'boxes': [<matplotlib.lines.Line2D at 0x2559d596850>],
 'medians': [<matplotlib.lines.Line2D at 0x255a2862e90>],
 'fliers': [<matplotlib.lines.Line2D at 0x255a2862fd0>],
 'means': []}
```



In [15]:
```
#Remove Outliers
q_low = df["fare_amount"].quantile(0.01)
q_hi  = df["fare_amount"].quantile(0.99)
```

```
df = df[(df["fare_amount"] < q_hi) & (df["fare_amount"] > q_low)]
```

In [16]:
```
#Check the missing values now
df.isnull().sum()
```

Out[16]:
```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [17]:
```
#Time to apply learning models
from sklearn.model_selection import train_test_split
```

In [18]:
```
#Take x as predictor variable
x = df.drop("fare_amount", axis = 1)
#And y as target variable
y = df['fare_amount']
```

In [19]:
```
#Necessary to apply model
x['pickup_datetime'] = pd.to_numeric(pd.to_datetime(x['pickup_datetime']))
x = x.loc[:, x.columns.str.contains('^Unnamed')]
```

In [20]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, randc
```

In [21]:
```
from sklearn.linear_model import LinearRegression
```

In [22]:
```
lrmodel = LinearRegression()
lrmodel.fit(x_train, y_train)
```

Out[22]:

LinearRegression

▼ Parameters

fit_intercept	True
copy_X	True
tol	1e-06
n_jobs	None
positive	False

In [23]:
```
#Prediction
predict = lrmodel.predict(x_test)
```

In [24]:
```
#Check Error
from sklearn.metrics import mean_squared_error
```

```
lrmodelrmse = np.sqrt(mean_squared_error(predict, y_test))
print("RMSE error for the model is ", lrmodelrmse)
```

```
RMSE error for the model is  8.063863046328837
```

In [25]:
```
#Let's Apply Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators = 100, random_state = 101)
```

In [26]:
```
#Fit the Forest
rfrmodel.fit(x_train, y_train)
rfrmodel_pred = rfrmodel.predict(x_test)
```

In [27]:
```
#Errors for the forest
rfrmodel_rmse = np.sqrt(mean_squared_error(rfrmodel_pred, y_test))
print("RMSE value for Random Forest is:",rfrmodel_rmse)
```

```
RMSE value for Random Forest is: 9.757713738069647
```