ASSIGNEMNT-1

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

Importing the Basic Libraries

In [1]:

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn import metrics
import math
from statsmodels.tools.eval_measures import rmse

from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
```

Loading the Dataset

First we load the dataset and find out the number of columns, rows, NULL values, etc.

1. Data is from 2009 to 2015
2. 200,000 Entries

In [2]:

```
uber = pd.read_csv('uber.csv')

uber.head()
```

Out[2]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.99951
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.99471

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.96256
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.96531
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.97308

In [3]:

```
uber.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
```

Dropping the NULL values

We will check for NULL values in Dataset

In [4]:

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

Out[4]:

```
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
```

Getting rid of first and second column, since key and ID are not useful in predictions.

In [5]:

```
uber_2 = uber.drop(['Unnamed: 0','key'],axis=1)
uber_2.dropna(axis=0,inplace=True)
```

We have gotten rid of the first two columns and the NULL values

In [6]:

```
uber_2.isnull().sum()
#uber_2.describe()
```

Out[6]:

```
fare_amount          0
pickup_datetime      0
```

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

Haversine Formula

Calculatin the distance between the pickup and drop co-ordinates using the Haversine formual for accuracy.

In [7]:

```
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 t


    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)

    return km
```

Defining the ride distance dataframe.

In [8]:

```
uber_2['Distance']= haversine(uber_2['pickup_longitude'],uber_2['dropoff_longitude'],
                             uber_2['pickup_latitude'],uber_2['dropoff_latitude'])

uber_2['Distance'] = uber_2['Distance'].astype(float).round(2)    # Round-off Optional
```

In [9]:

```
uber_2.head()
```

Out[9]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
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	

Scatter Plot

Distance vs Fare Amount

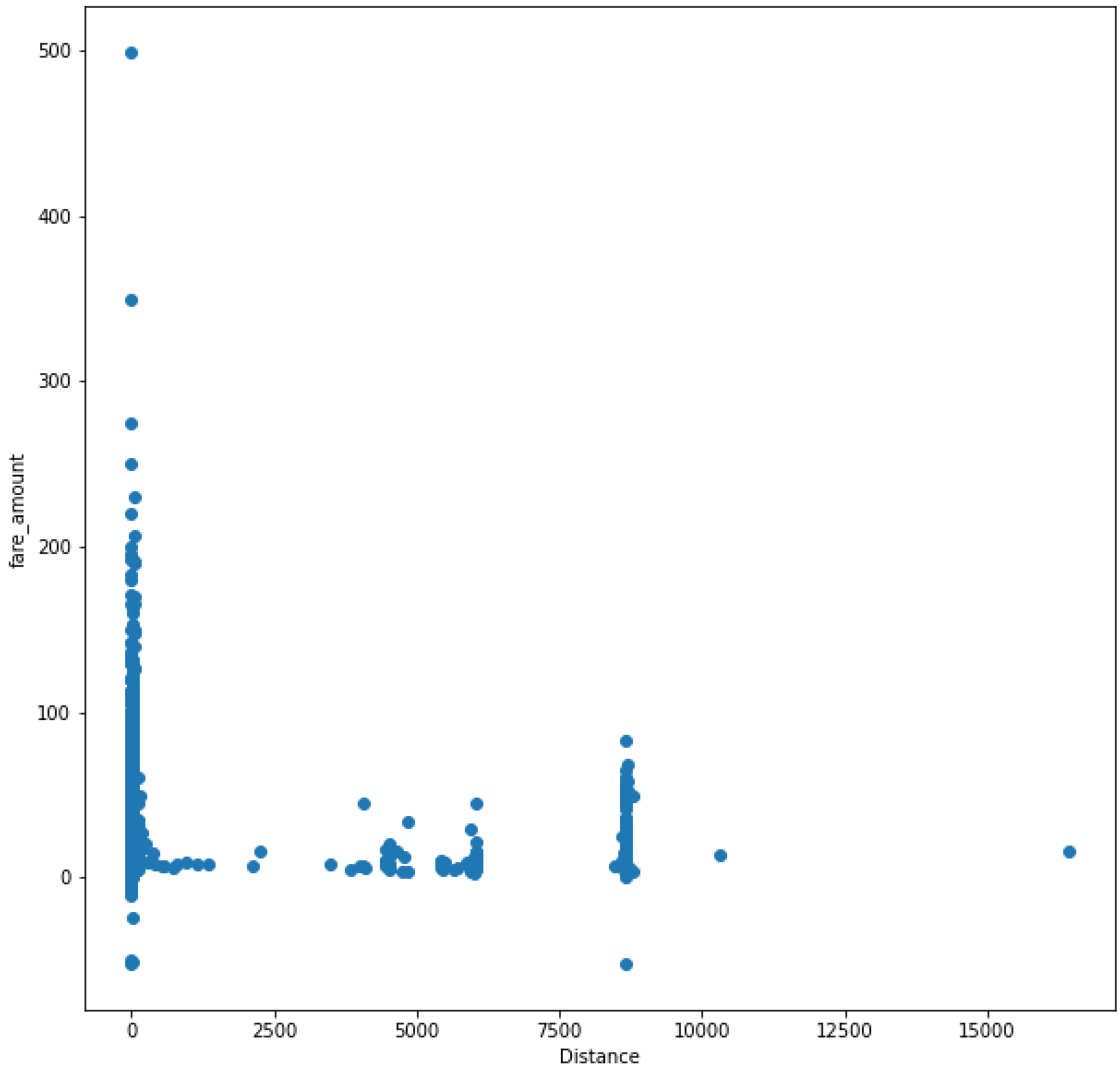
In [10]:

```
plt.figure(figsize=[10,10])
plt.scatter(uber_2['Distance'], uber_2['fare_amount'])
```

```
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

Out[10]:

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



Outliers

We can get rid of the trips with very large distances that are outliers as well as trips with 0 distance.

In [11]:

```
uber_2.drop(uber_2[uber_2['Distance'] > 60].index, inplace = True)
uber_2.drop(uber_2[uber_2['Distance'] == 0].index, inplace = True)
uber_2.drop(uber_2[uber_2['Distance'] < 0].index, inplace = True)

uber_2.drop(uber_2[uber_2['fare_amount'] == 0].index, inplace = True)
uber_2.drop(uber_2[uber_2['fare_amount'] < 0].index, inplace = True)
```

In [12]:

```
uber_2.drop(uber_2[uber_2['Distance'] > 100].index, inplace = True)
uber_2.drop(uber_2[uber_2['fare_amount'] > 100].index, inplace = True)
```

Also removing rows with non-plausible fare amounts and distance travelled

In [13]:

```
uber_2.drop(uber_2[(uber_2['fare_amount']>100) & (uber_2['Distance']<1)].index, inplace =
```

```
uber_2.drop(uber_2[(uber_2['fare_amount']<100) & (uber_2['Distance']>100)].index, inplace
```

In [14]:

```
uber_2.info()
```

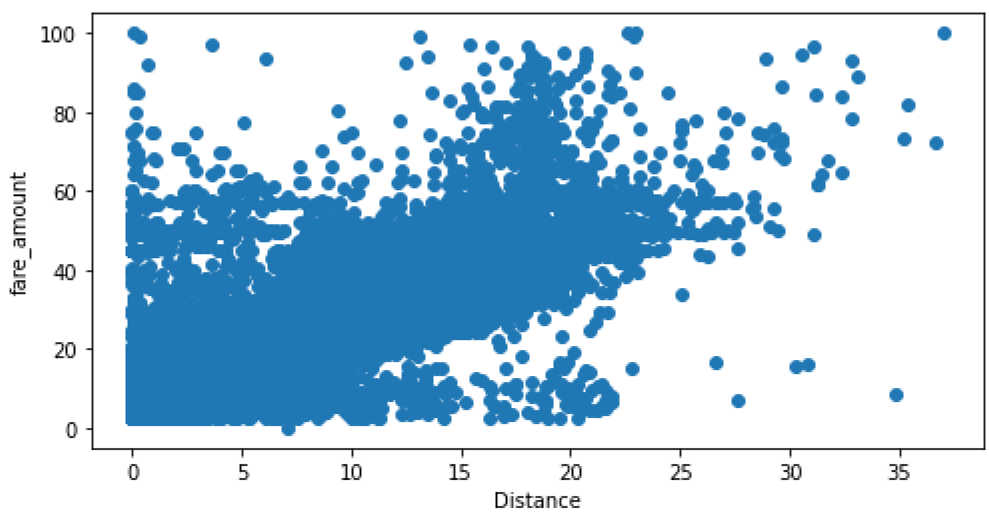
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 193436 entries, 0 to 199999
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        193436 non-null  float64
 1   pickup_datetime    193436 non-null  object 
 2   pickup_longitude   193436 non-null  float64
 3   pickup_latitude    193436 non-null  float64
 4   dropoff_longitude  193436 non-null  float64
 5   dropoff_latitude   193436 non-null  float64
 6   passenger_count    193436 non-null  int64  
 7   Distance           193436 non-null  float64
dtypes: float64(6), int64(1), object(1)
memory usage: 17.3+ MB
```

In [15]:

```
plt.figure(figsize=[8,4])
plt.scatter(uber_2['Distance'], uber_2['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

Out[15]:

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



Now the scatter plot is looking more suitable.

Date and Time

Separating the date and time into separate columns for more usability.

In [16]:

```
uber_2['pickup_datetime'] = pd.to_datetime(uber_2['pickup_datetime'])

uber_2['Year'] = uber_2['pickup_datetime'].apply(lambda time: time.year)
uber_2['Month'] = uber_2['pickup_datetime'].apply(lambda time: time.month)
uber_2['Day'] = uber_2['pickup_datetime'].apply(lambda time: time.day)
uber_2['Day of Week'] = uber_2['pickup_datetime'].apply(lambda time: time.dayofweek)
uber_2['Day of Week_num'] = uber_2['pickup_datetime'].apply(lambda time: time.dayofweek)
uber_2['Hour'] = uber_2['pickup_datetime'].apply(lambda time: time.hour)

day_map = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

```
uber_2['Day of Week'] = uber_2['Day of Week'].map(day_map)

uber_2['counter'] = 1
```

Pickup and Dropoff Columns

Creating separate coumns for pickup and droppoff coordinates for more usability.

In [17]:
```
uber_2['pickup'] = uber_2['pickup_latitude'].astype(str) + "," + uber_2['pickup_longitude'
uber_2['drop off'] = uber_2['dropoff_latitude'].astype(str) + "," + uber_2['dropoff_longit
```

In [18]:
```
uber_2.head()
```

Out[18]:

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

Thus, we have increased the usability of the dataset.

Data Visualizations

Finding the trends in the data variables

Average Yearly Trips

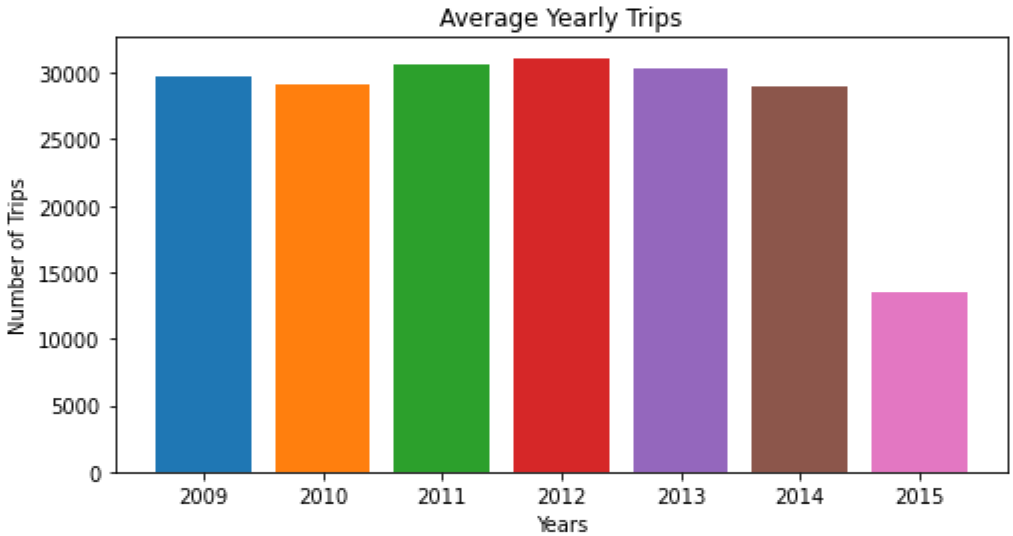
In [19]:
```
no_of_trips = []
year = [2009, 2010, 2011, 2012, 2013, 2014, 2015]

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
          '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

for i in range(2009, 2016):
    x = uber_2.loc[uber_2['Year'] == i, 'counter'].sum()
    no_of_trips.append(x)

print("Average trips a year: ")
print(year, no_of_trips)
plt.figure(figsize=[8,4])
plt.title("Average Yearly Trips")
plt.xlabel("Years")
plt.ylabel("Number of Trips")
```

```
plt.bar(year, no_of_trips, color=colors)
```

```
Average trips a year:
[2009, 2010, 2011, 2012, 2013, 2014, 2015] [29672, 29092, 30710, 31131, 30354, 29048, 1342
9]
```

Out[19]:

```
<BarContainer object of 7 artists>
```



Average Monthly Trips

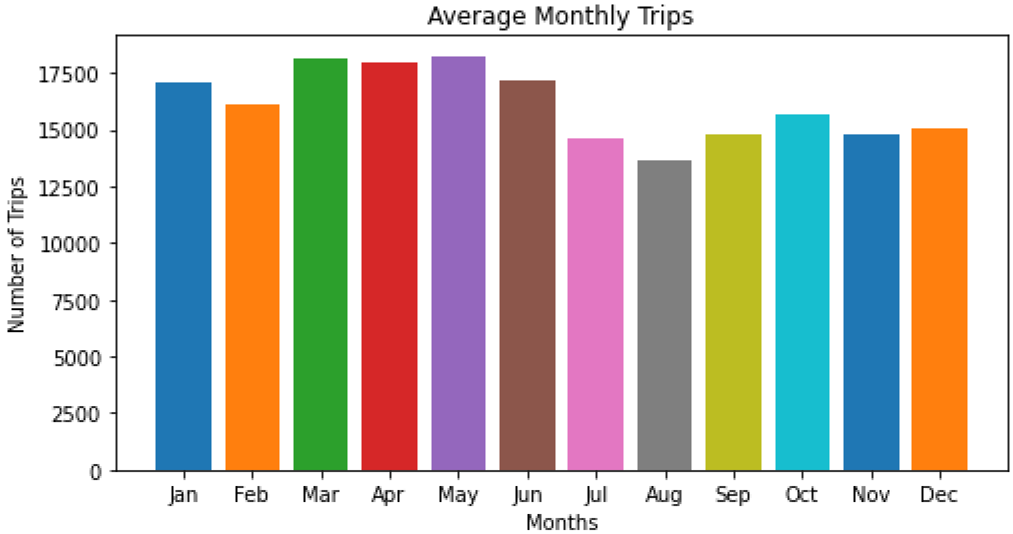
In [20]:

```
no_of_trips = []
month = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
          '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

for i in range(1, 13):
    x = uber_2.loc[uber_2['Month'] == i, 'counter'].sum()
    no_of_trips.append(x)

print("Average trips a Month: ")
print(month, no_of_trips)
plt.figure(figsize=[8,4])
plt.title("Average Monthly Trips")
plt.xlabel("Months")
plt.ylabel("Number of Trips")

plt.bar(month, no_of_trips, color=colors)
```

```
Average trips a Month:
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'] [1712
3, 16137, 18160, 18001, 18256, 17202, 14582, 13659, 14768, 15688, 14818, 15042]
```

Out[20]:

```
<BarContainer object of 12 artists>
```



Average Daily Trips

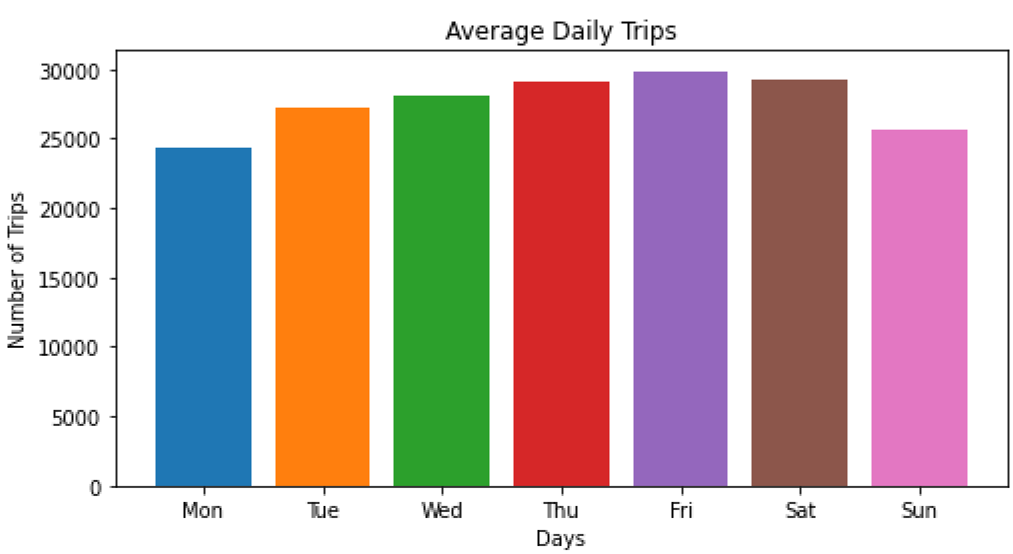
In [21]:

```
no_of_trips = []
day = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
          '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']

for i in range(0, 7):
    x = uber_2.loc[uber_2['Day of Week_num'] == i, 'counter'].sum()
    no_of_trips.append(x)

print("Average trips by Days: ")
print(day, no_of_trips)
plt.figure(figsize=[8,4])
plt.title("Average Daily Trips")
plt.xlabel("Days")
plt.ylabel("Number of Trips")

plt.bar(day, no_of_trips, color=colors)
```

```
Average trips by Days: 
['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'] [24373, 27231, 28075, 29035, 29857, 2930
2, 25563]
```

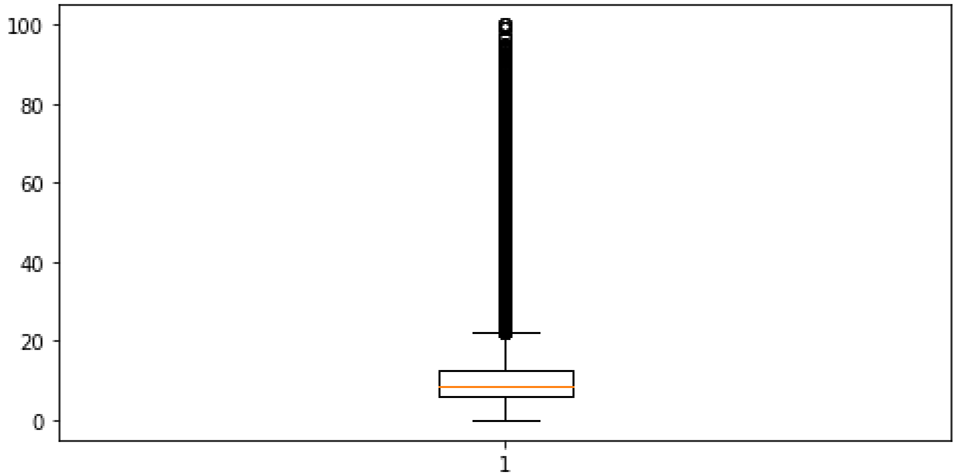
Out[21]:

```
<BarContainer object of 7 artists>
```



In [22]:
```
plt.figure(figsize=[8,4])
fig, ax = plt.subplots(figsize=[8,4])
fare_amount = uber_2['fare_amount']
ax.boxplot(fare_amount,)
plt.show()
```

```
<Figure size 576x288 with 0 Axes>
```

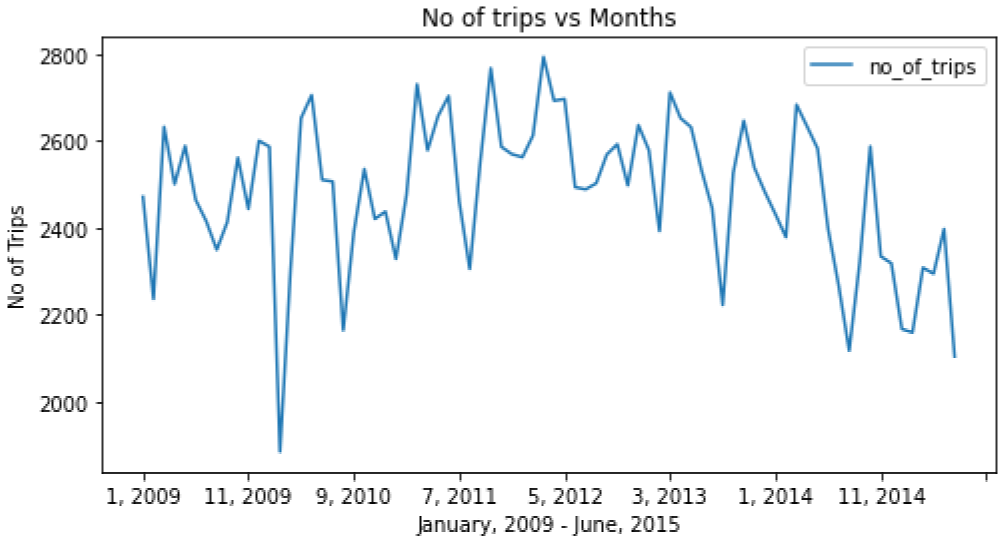


Rides vs Time

Relation between average number of rides over a period of time.

In [23]:
```
year_vs_trips = uber_2.groupby(['Year','Month']).agg(
    no_of_trips = ('counter','count'),
    Average_fair = ('fare_amount','mean'),
    Total_fair = ('fare_amount','sum'),
    Avg_distance = ( 'Distance', 'mean')).reset_index()

year_vs_trips['avg_no_of_trips'] = year_vs_trips['no_of_trips']/30
year_vs_trips['month_year'] = year_vs_trips['Month'].astype(str) +", "+ year_vs_trips['Yea


year_vs_trips = year_vs_trips.reset_index()

year_vs_trips.head()


year_vs_trips.plot(kind='line',x='month_year',y='no_of_trips', xlabel='January, 2009 - Jur
    ylabel='No of Trips', title='No of trips vs Months',figsize=[8,4])
```

Out[23]:
```
<AxesSubplot:title={'center':'No of trips vs Months'}, xlabel='January, 2009 - June, 201
5', ylabel='No of Trips'>
```



Heat-Map

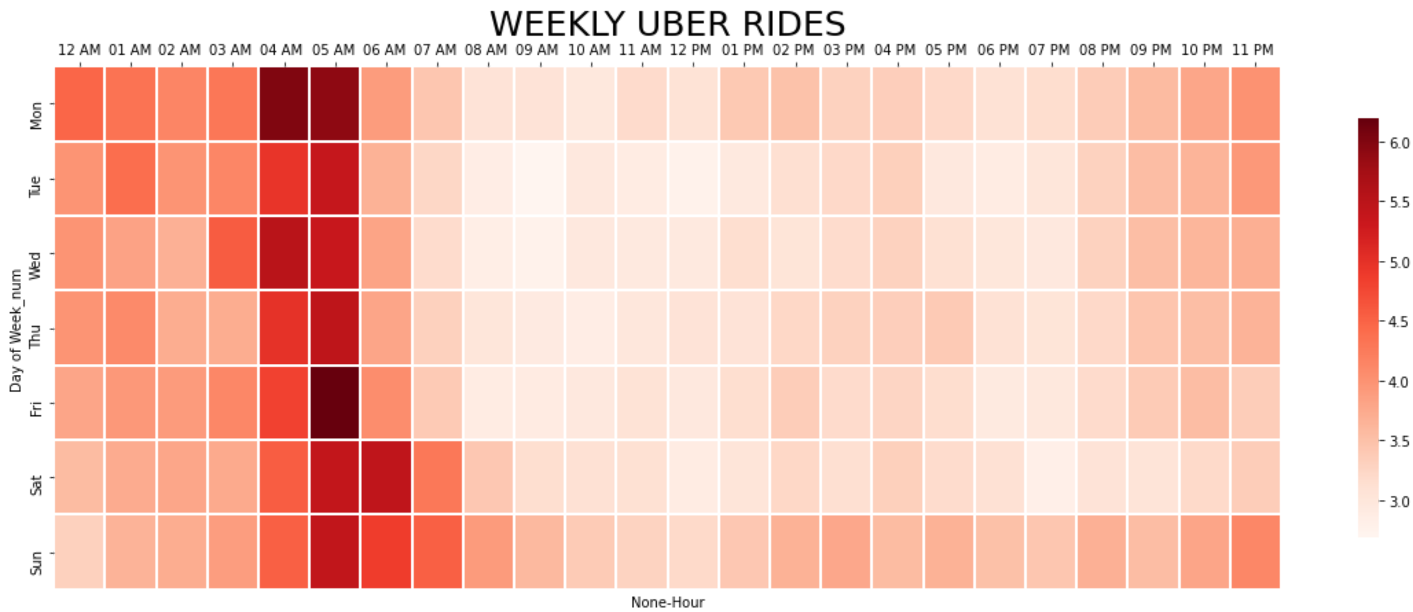
A heat map to illustrate at what time of day and week, people are using Uber the most.

In [24]:

```
import seaborn as sns

df_1 = uber_2[['Distance', 'Day of Week_num', 'Hour']].copy()

df_h = df_1.copy()

df_h = df_h.groupby(['Hour', 'Day of Week_num']).mean()
df_h = df_h.unstack(level=0)
```

In [25]:

```
fig, ax = plt.subplots(figsize=(20, 7))
sns.heatmap(df_h, cmap="Reds",
           linewidth=0.3, cbar_kws={"shrink": .8})

xticks_labels = ['12 AM', '01 AM', '02 AM ', '03 AM ', '04 AM ', '05 AM ', '06 AM ', '07 A
                 '08 AM ', '09 AM ', '10 AM ', '11 AM ', '12 PM ', '01 PM ', '02 PM ', '03
                 '04 PM ', '05 PM ', '06 PM ', '07 PM ', '08 PM ', '09 PM ', '10 PM ', '11

yticks_labels = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']

plt.xticks(np.arange(24) + .5, labels=xticks_labels)
plt.yticks(np.arange(7) + .5, labels=yticks_labels)

ax.xaxis.tick_top()

title = 'Weekly Uber Rides'.upper()
plt.title(title, fontdict={'fontsize': 25})

plt.show()
```



Coorelation Matrix

To find the two variables that have the most inter-dependence

In [26]:
```
corr = uber_2.corr()

corr.style.background_gradient(cmap='BuGn')
```

```
c:\users\orionoriginal\appdata\local\programs\python\python39\lib\site-packages\pandas\io
\formats\style.py:1264: RuntimeWarning: All-NaN slice encountered
  smin = np.nanmin(s.to_numpy()) if vmin is None else vmin
c:\users\orionoriginal\appdata\local\programs\python\python39\lib\site-packages\pandas\io
\formats\style.py:1265: RuntimeWarning: All-NaN slice encountered
  smax = np.nanmax(s.to_numpy()) if vmax is None else vmax
```

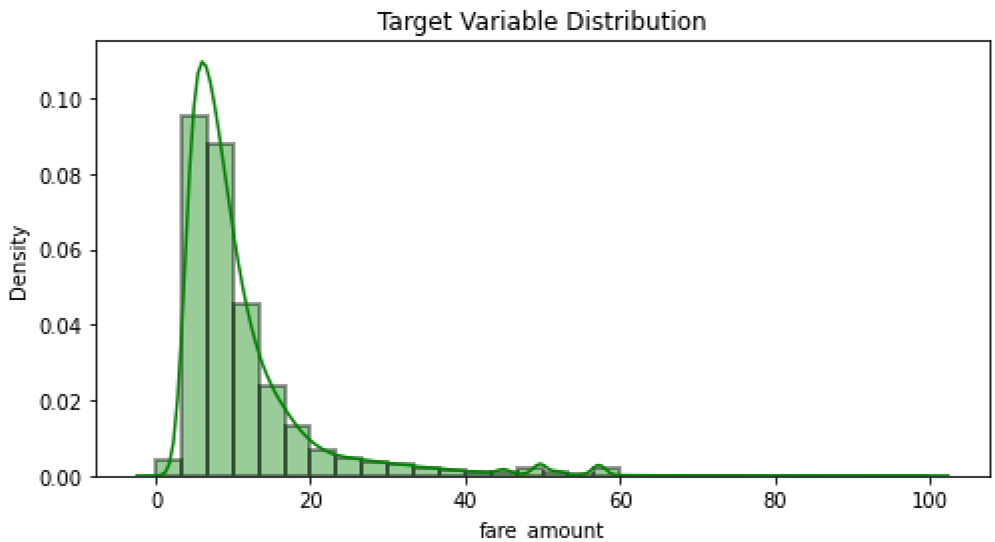
Out[26]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_co
fare_amount	1.000000	0.012292	-0.008891	0.010831	-0.009044	0.014
pickup_longitude	0.012292	1.000000	-0.949099	0.999885	-0.993976	0.009
pickup_latitude	-0.008891	-0.949099	1.000000	-0.949096	0.954760	-0.009
dropoff_longitude	0.010831	0.999885	-0.949096	1.000000	-0.993964	0.009
dropoff_latitude	-0.009044	-0.993976	0.954760	-0.993964	1.000000	-0.009
passenger_count	0.014409	0.009176	-0.009219	0.009164	-0.009263	1.000
Distance	0.895513	0.005356	0.003243	0.004464	-0.002255	0.007
Year	0.124050	0.013480	-0.013693	0.013373	-0.014365	0.005
Month	0.024850	-0.007497	0.007602	-0.007452	0.007982	0.010
Day	0.000277	0.019531	-0.019393	0.019555	-0.020119	0.003
Day of Week_num	0.004881	0.008243	-0.008924	0.008543	-0.008916	0.033
Hour	-0.020270	0.001835	-0.001821	0.000937	-0.001016	0.013
counter	nan	nan	nan	nan	nan	

In [27]:

```
plt.figure(figsize=[8,4])
sns.distplot(uber_2['fare_amount'], color='g',hist_kws=dict(edgecolor="black", linewidth=2
plt.title('Target Variable Distribution')
plt.show()
```

```
c:\users\orionoriginal\appdata\local\programs\python\python39\lib\site-packages\seaborn\di
stributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot` (a figure-level func
tion with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Statistics

Some general statistical information about the data

Fare Amount

There is some coorelation between the distance and fare amount. So we will implement our simple linear regression model using these two varaibles

Standardization

For more accurate results on our linear regression model

In [28]:

```
import statistics as st

print("Mean of fare prices is % s "
          % (st.mean(uber_2['fare_amount'])))

print("Median of fare prices is % s "
          % (st.median(uber_2['fare_amount'])))

print("Standard Deviation of Fare Prices is % s "
                 % (st.stdev(uber_2['fare_amount'])))
```

```
Mean of fare prices is 11.282935803056308
Median of fare prices is 8.5
Standard Deviation of Fare Prices is 9.308836645207887
```

Assigning the dependent and independent variable

Distance

In [29]:
```
X = uber_2['Distance'].values.reshape(-1, 1)        #Independent Variable
y = uber_2['fare_amount'].values.reshape(-1, 1)     #Dependent Variable
```

In [30]:
```
import statistics as st

print("Mean of Distance is % s "
         % (st.mean(uber_2['Distance'])))

print("Median of Distance is % s "
         % (st.median(uber_2['Distance'])))

print("Standard Deviation of Distance is % s "
                % (st.stdev(uber_2['Distance'])))
```

```
Mean of Distance is 3.3513455096259226
Median of Distance is 2.18
Standard Deviation of Distance is 3.573555973014084
```

In [31]:
```
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(y)
print(y_std)

x_std = std.fit_transform(X)
print(x_std)
```

```
[[-0.40638221]
 [-0.38489719]
 [ 0.17371326]
 ...
 [ 2.10736482]
 [ 0.3455934 ]
 [ 0.30262337]]
[[-0.46769936]
 [-0.24942881]
 [ 0.472543  ]
 ...
 [ 2.65804681]
 [ 0.05279195]
 [ 0.57887993]]
```

Splitting the Dataset

Training and Test Set

In [32]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2, random_st
```

Simple Linear Regression

Training the simple linear regression model on the training set

In [33]:
```
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: 0.80
Test set score: 0.8033899
```

Actual vs Predicted Values

In [36]:
```
y_pred = l_reg.predict(X_test)
df = {'Actual': y_test, 'Predicted': y_pred}
```

Accuracy Checking

Finding the MSE,MAE, RMSE, etc.

In [37]:
```
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: 0.2446126190350342
Mean Squared Error: 0.1954007431280604
Root Mean Squared Error: 0.4420415626703675
```

Plotting the Graph

Intercept and Co-efficient

In [38]:
```
print(l_reg.intercept_)
print(l_reg.coef_)
```

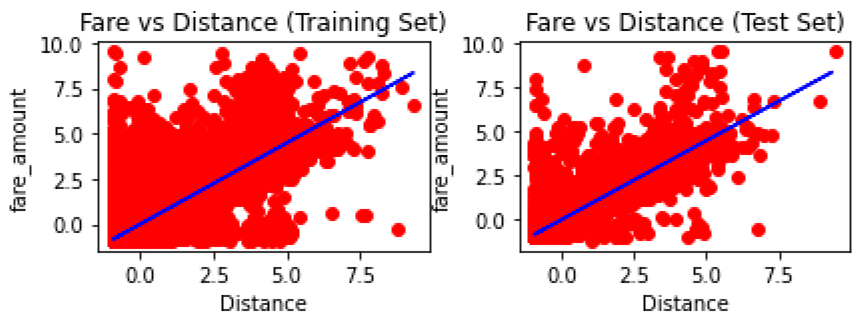
```
[0.0002494]
[[0.89611325]]
```

Final Graph

Plotting the linear regression line against the training and test set side by side.

In [39]:
```
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")


plt.tight_layout()
```

```
plt.rcParams["figure.figsize"] = (32,22)
plt.show()
```



RandomForestRegressor

In [40]:

```
# import library for random forest regressor
from sklearn.ensemble import RandomForestRegressor
```

In [41]:

```
#intantiate the regressor
rf_reg = RandomForestRegressor(n_estimators=100, random_state=10)

# fit the regressor with training dataset
rf_reg.fit(X_train, y_train)
```

```
<ipython-input-41-114d8cc06ea3>:5: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples,), for example using
ravel().
  rf_reg.fit(X_train, y_train)
```

Out[41]:

```
▾        RandomForestRegressor
RandomForestRegressor(random_state=10)
```

In [42]:

```
# predict the values on test dataset using predict()
y_pred_RF = rf_reg.predict(X_test)
```

In [43]:

```
y_pred_RF
```

Out[43]:

```
array([-0.54871913,  0.1470156 , -0.44811847, ...,  0.10513227,
       -0.35566696, -0.35648786])
```

In [44]:

```
from sklearn.metrics import mean_squared_error
r_squared_RF = rf_reg.score(X_test,y_test)
# Number of observation or sample size
n = 159999

# No of independent variables
p = 11

#Compute Adj-R-Squared
Adj_r_squared_RF = 1 - (1-r_squared_RF)*(n-1)/(n-p-1)
Adj_r_squared_RF
# Compute RMSE
rmse_RF = math.sqrt(mean_squared_error(y_test, y_pred_RF))
```

In [45]:

```python
# Calculate MAE
rf_reg_MAE = metrics.mean_absolute_error(y_test, y_pred_RF)
print('Mean Absolute Error (MAE):', rf_reg_MAE)

# Calculate MSE
rf_reg_MSE = metrics.mean_squared_error(y_test, y_pred_RF)
print('Mean Squared Error (MSE):', rf_reg_MSE)

# Calculate RMSE
rf_reg_RMSE = np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF))
print('Root Mean Squared Error (RMSE):', rf_reg_RMSE)
```

```
Mean Absolute Error (MAE): 0.24633853538180536
Mean Squared Error (MSE): 0.19949470601350427
Root Mean Squared Error (RMSE): 0.44664830237391956
```

In [ ]: