FAST National University of Computer and Emerging Sciences

Project Topic: Uber Fare Price Prediction (Cleaning, Visualization, Algorithms (Linear Regression, Improving accuracy (Adaboost, KFold Approach, Holdout Approach)))

Instructor: Dr. Noman Durani

Group Members:

1. Kanwar Muzammil Rohail (20K-0469)
2. Farhan Ahmed Siddique (20I-0439)

Data Cleaning and EDA:

```
In [136]: import pandas as pd
          uber = pd.read_csv('uber.csv')
          uber.head(20)
```

Out[136]:

	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
5	44470845	2011-02-12 02:27:09.0000006	4.9	2011-02-12 02:27:09 UTC	-73.969019	40.755910	-73.969019	40.755910	1
6	48725865	2014-10-12 07:04:00.0000002	24.5	2014-10-12 07:04:00 UTC	-73.961447	40.693965	-73.871195	40.774297	5
7	44195482	2012-12-11 13:52:00.00000029	2.5	2012-12-11 13:52:00 UTC	0.000000	0.000000	0.000000	0.000000	1
8	15822268	2012-02-17 09:32:00.00000043	9.7	2012-02-17 09:32:00 UTC	-73.975187	40.745767	-74.002720	40.743537	1
9	50611056	2012-03-29	12.5	2012-03-29	-74.001065	40.741787	-73.963040	40.775012	1

File contained columns that were not needed for our purpose, such as key and id. Thus, they were removed

```
In [14]: ##key and id not useful in for our purpose
```

```
In [15]: uber_clean = uber.drop(['Unnamed: 0', 'key'],axis = 1)
         uber_clean.head()
```

Out[15]:

	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

Null values were checked in every column and dropped

Data Cleaning and EDA

```
In [13]: #check for null values
         uber.isnull().sum()
```

```
Out[13]: 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 [16]:
```python
uber_clean.dropna(axis =0, inplace = True)
#drop all the null values
```

In [17]:
```python
uber_clean.head()
```

Out[17]:

	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 [18]:
```python
uber_clean.isnull().sum()
#verfy it
```

Out[18]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

"isnull.sum()" was used to verify the presence of null values

In [19]:
```python
# we have used thee haversine distance formula but any other formula can be used
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 [20]:
```python
uber_clean['Distance']= haversine(uber_clean['pickup_longitude'],uber_clean['dropoff_longitude'],
                             uber_clean['pickup_latitude'],uber_clean['dropoff_latitude'])

uber_clean['Distance'] = uber_clean['Distance'].astype(float).round(2)
```

In [21]:
```python
#New column added
uber_clean.head()
```

Out[21]:

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

Haversine formula was defined in a function to calculate the distance from the pickup and dropoff columns to make use of the data and make it meaningful to identify the trends. The new Distance column was added to the table

```
In [133]: #Target Variable (Fare_Amount) greatest coorelation with Distance
          uber_clean.corr(method='pearson', numeric_only=True)
```

Out[133]:

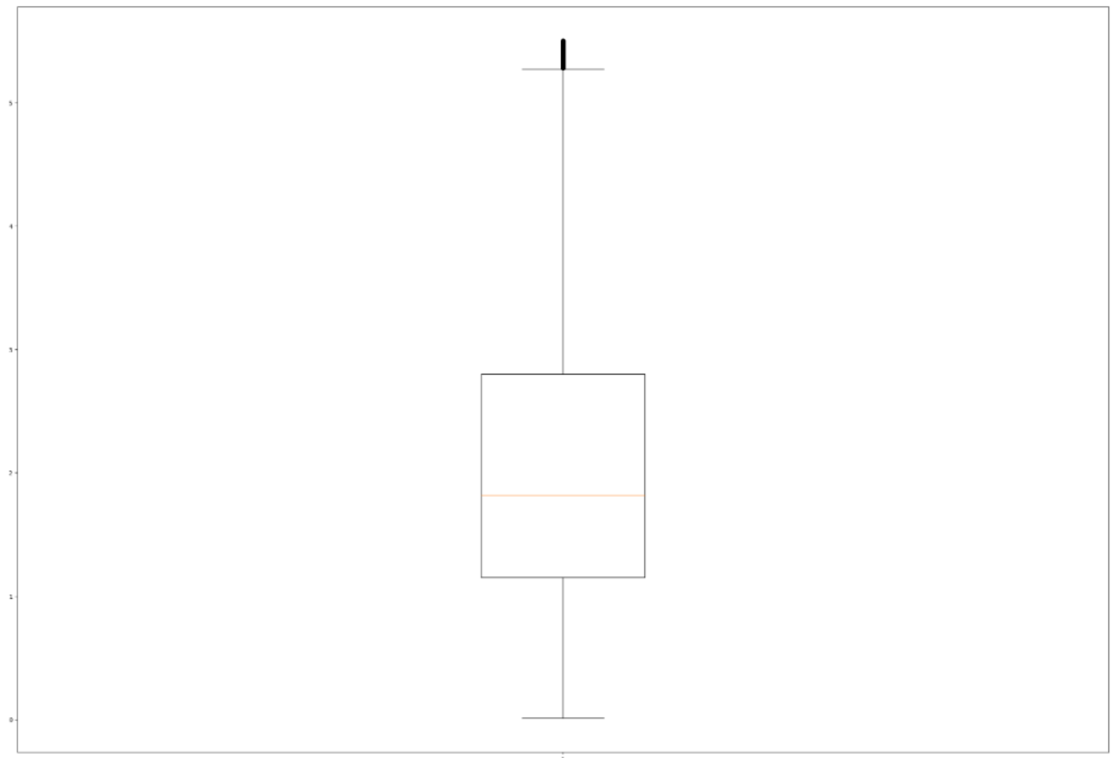
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	Distance	Year	Month	Da
fare_amount	1.000000	0.003157	-0.003688	0.003395	-0.003853	0.017873	0.774574	0.156694	0.030277	0.00245
pickup_longitude	0.003157	1.000000	-0.993037	0.999980	-0.993041	0.008973	-0.015033	0.012772	-0.006889	0.01853
pickup_latitude	-0.003688	-0.993037	1.000000	-0.993047	0.999975	-0.009221	0.013521	-0.013975	0.007355	-0.01911
dropoff_longitude	0.003395	0.999980	-0.993047	1.000000	-0.993030	0.008964	-0.014584	0.012812	-0.006900	0.01850
dropoff_latitude	-0.003853	-0.993041	0.999975	-0.993030	1.000000	-0.009226	0.013647	-0.013935	0.007343	-0.01916
passenger_count	0.017873	0.008973	-0.009221	0.008964	-0.009226	1.000000	0.002287	0.002280	0.008280	0.00414
Distance	0.774574	-0.015033	0.013521	-0.014584	0.013647	0.002287	1.000000	-0.030644	0.002107	0.00556
Year	0.156694	0.012772	-0.013975	0.012812	-0.013935	0.002280	-0.030644	1.000000	-0.115349	-0.01157
Month	0.030277	-0.006889	0.007355	-0.006900	0.007343	0.008280	0.002107	-0.115349	1.000000	-0.01618
Day	0.002459	0.018534	-0.019113	0.018501	-0.019160	0.004140	0.005561	-0.011577	-0.016187	1.00000
Day of Week_num	0.007567	0.007446	-0.008084	0.007459	-0.008043	0.034405	0.033940	0.007452	-0.009767	0.00583
Hour	0.017843	0.003483	-0.002886	0.003400	-0.002894	0.014507	-0.002722	0.002342	-0.003051	0.00380
counter	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Pearson correlation was calculated for each variable against every other variable. **It was observed that Fare amount had the strongest relation with the Distance variable (0.774574)**

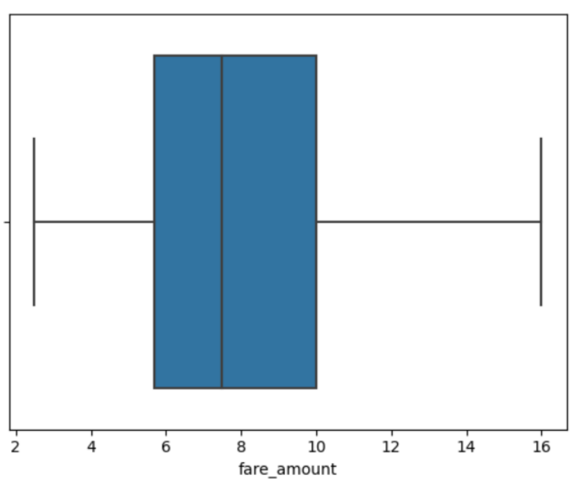
Data Visualization

Box Plots:

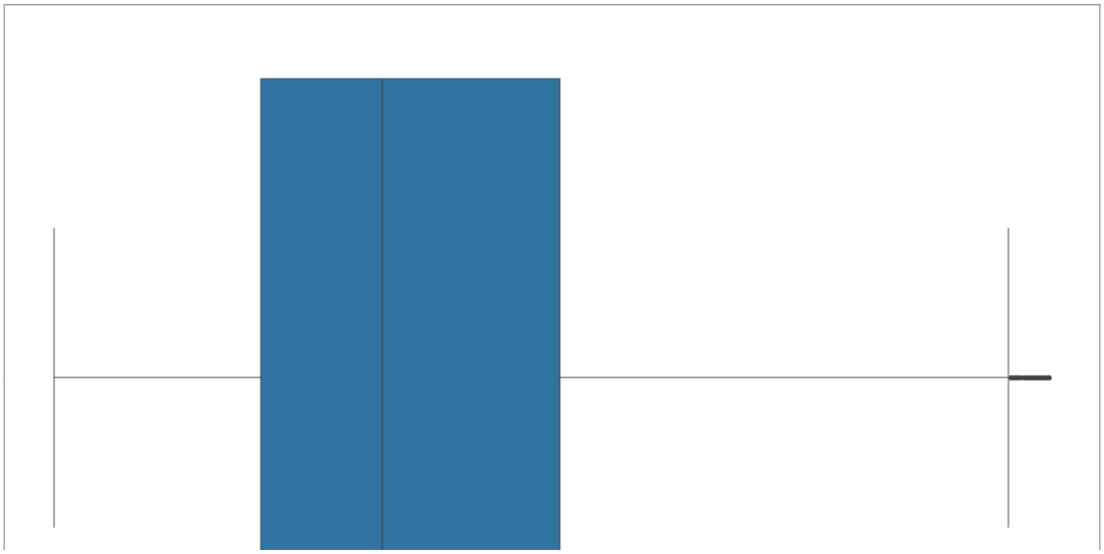
In [139]:
```
ax1, ax = plt.subplots()
dis = uber_clean['Distance']
ax.boxplot(dis)
plt.show()
```



In [87]:
```
sns.boxplot(data=uber_clean, x='fare_amount')
plt.show()
```



```
In [134]: sns.boxplot(data=uber_clean, x='Distance')
plt.show()
```

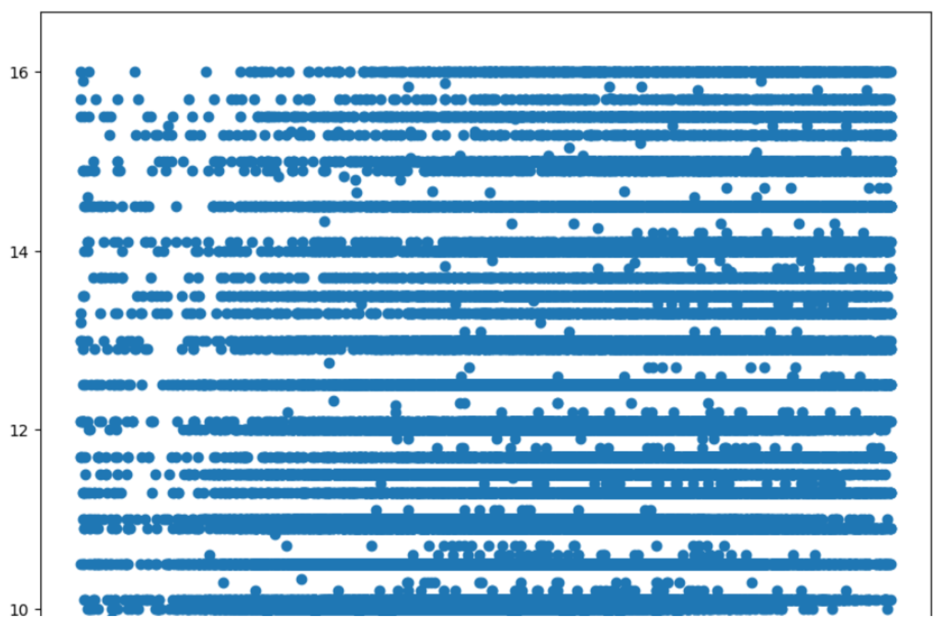


Box plots were displayed to help visualize the distribution which helped us to identify the outliers

Scatter Plots:

```
import matplotlib.pyplot as plt
plt.scatter(uber_clean['Distance'], uber_clean['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

```
Out[128]: Text(0, 0.5, 'fare_amount')
```

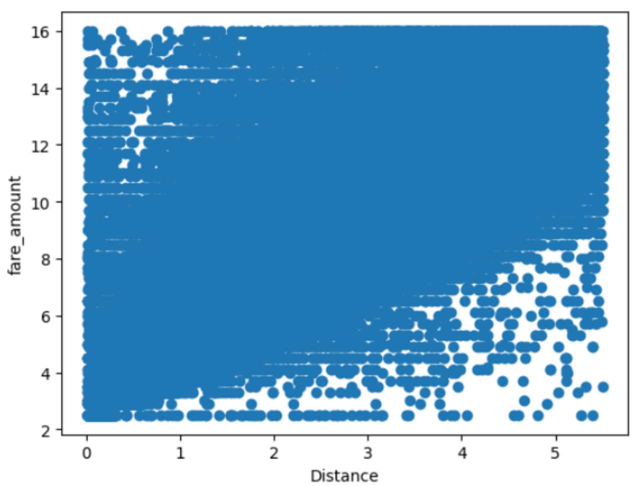


Scatter Plots of fare vs distance were displayed to show the distribution and identify the relation between these two variables to check whether they fit for our purpose

```
In [65]: #we must avoid the outliers(includes those with very large distances and with 0 distances or negative distances)
uber_clean.drop(uber_clean[uber_clean['Distance'] > 5.5].index, inplace = True)
uber_clean.drop(uber_clean[uber_clean['Distance'] == 0].index, inplace = True)
uber_clean.drop(uber_clean[uber_clean['Distance'] < 0].index, inplace = True)

uber_clean.drop(uber_clean[uber_clean['fare_amount'] == 0].index, inplace = True)
uber_clean.drop(uber_clean[uber_clean['fare_amount'] < 0].index, inplace = True)
uber_clean.drop(uber_clean[uber_clean['Distance'] > 100].index, inplace = True)
uber_clean.drop(uber_clean[uber_clean['fare_amount'] > 16].index, inplace = True)
uber_clean.drop(uber_clean[(uber_clean['fare_amount']>100) & (uber_clean['Distance']<1)].index, inplace = True )
uber_clean.drop(uber_clean[(uber_clean['fare_amount']<100) & (uber_clean['Distance']>100)].index, inplace = True )
```

Particular set of data was dropped carefully which were acting as outliers, affecting the distributions

```
In [68]: #if we re plot the scatter plot the change can be observed
plt.scatter(uber_clean['Distance'], uber_clean['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")

Out[68]: Text(0, 0.5, 'fare_amount')
```



Scatterplot was redrawn. This shows better stronger relation and better distribution as outliers were removed

In [70]:
```python
#Date and time are simply seperated to extract more information from the data
uber_clean['pickup_datetime'] = pd.to_datetime(uber_clean['pickup_datetime'])

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

day_map = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
uber_clean['Day of Week'] = uber_clean['Day of Week'].map(day_map)

uber_clean['counter'] = 1
```

In [71]:
```python
#seperate columns for pickup and dropoff for clear understanding of the data
uber_clean['pickup'] = uber_clean['pickup_latitude'].astype(str) + "," + uber_clean['pickup_longitude'].astype(str)
uber_clean['drop off'] = uber_clean['dropoff_latitude'].astype(str) + "," + uber_clean['dropoff_longitude'].astype(str)
```

In [72]:
```python
uber_clean.head(10)
```

Out[72]:

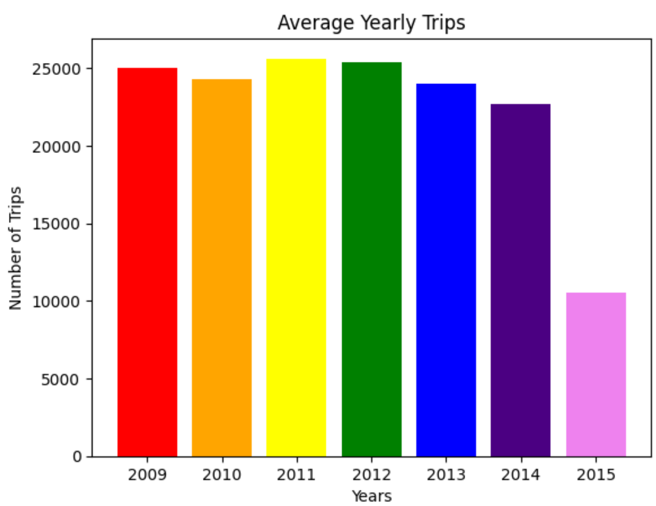
	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	Distance	Year	Month	Day	Day of Week
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723217	1	1.68	2015	5	7	Thu
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750325	1	2.46	2009	7	17	Fri
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772647	1	5.04	2009	8	24	Mon
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803349	3	1.66	2009	6	26	Fri
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761247	5	4.48	2014	8	28	Thu
8	9.7	2012-02-17 09:32:00+00:00	-73.975187	40.745767	-74.002720	40.743537	1	2.33	2012	2	17	Fri

Date and time were separated into days, months and years to make data useful and identify trends clearly.

Bar charts:

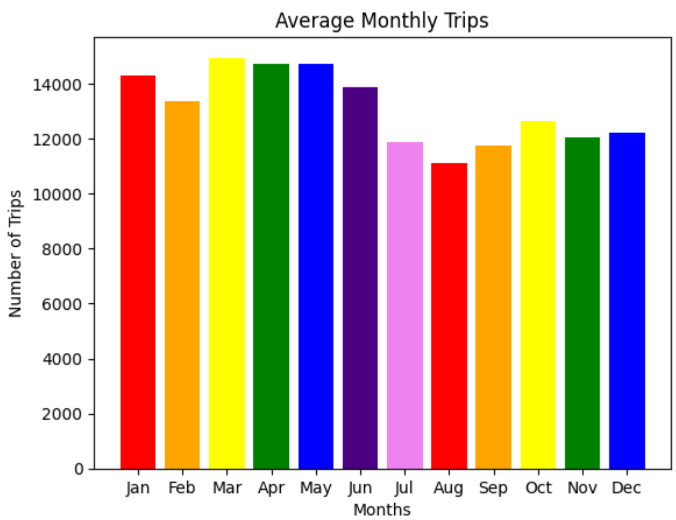
In [73]:
```
#needed to detect trends in the data to extract useful information for future predictions
trips = []
year = [2009, 2010, 2011, 2012, 2013, 2014, 2015]

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']

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

plt.title("Average Yearly Trips")
plt.xlabel("Years")
plt.ylabel("Number of Trips")

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

Out[73]:
```
<BarContainer object of 7 artists>
```



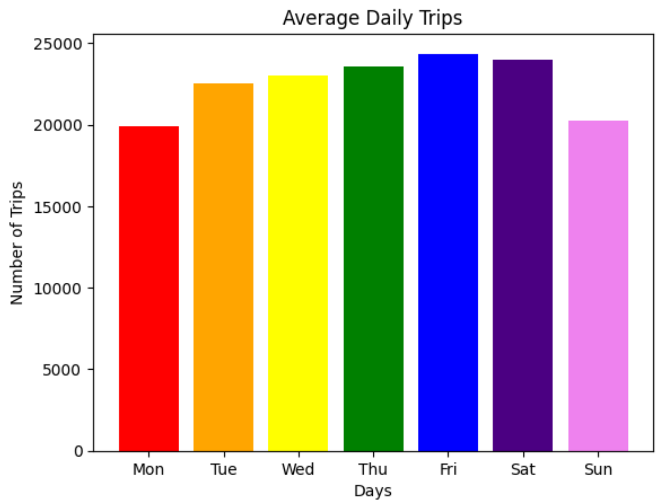
```
trips = []
month = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']


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


plt.title("Average Monthly Trips")
plt.xlabel("Months")
plt.ylabel("Number of Trips")

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

Out[74]: <BarContainer object of 12 artists>



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

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']


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


plt.title("Average Daily Trips")
plt.xlabel("Days")
plt.ylabel("Number of Trips")

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

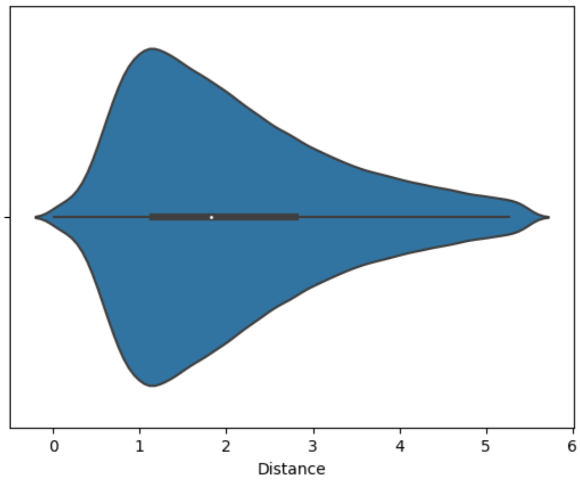
```
Out[75]: <BarContainer object of 7 artists>
```



Bar charts were plotted to visualize the trips completed against days, months and years to help identify the trends

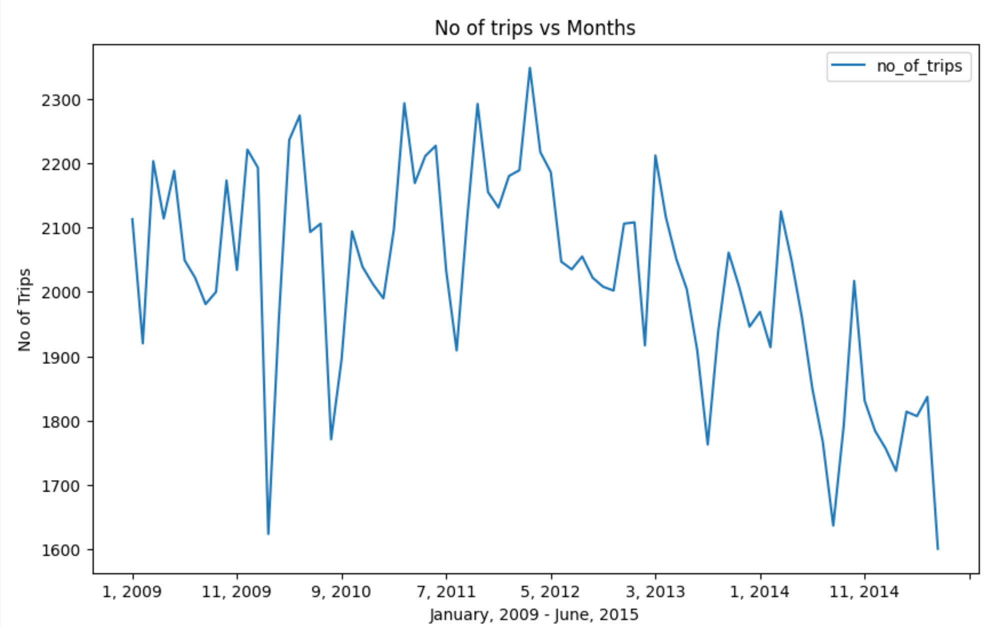
Violin Plots:

```
In [76]: sns.violinplot(data=uber_clean, x='Distance')
         plt.show()
```



A violin plot is more informative than a plain box plot. While a box plot only shows summary statistics such as mean/median and interquartile ranges, the violin plot shows the full distribution of the data.

Line Plots:



Line Plot of No. of trips vs months were plotted to check the trends.

Model Training

```
In [93]: #Seperate Test and Train set
         X = uber_clean['Distance'].values
         Y = uber_clean['fare_amount'].values
```

```
In [94]: X = X.reshape(-1,1)
         Y = Y.reshape(-1,1)
```

```
In [95]: X.shape

Out[95]: (157593, 1)
```

Linear Regression Model

```
In [101]: 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)

          [[-0.15349139]
           [-0.08631261]
           [ 1.66033545]
           ...
           [-0.15349139]
           [ 2.19776563]
           [ 2.06340808]]
          [[-0.33658053]
           [ 0.31539556]
           [ 2.47193184]
           ...
           [-0.16940717]
           [ 1.21813168]
```

Data was reshaped and scaled using StandardScaler to make it fit to perform linear regression on it.

Train Test Split

Train Test Split

```
In [ ]: X = uber_clean['Distance'].values
        Y = uber_clean['fare_amount'].values
```

```
In [ ]: x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size=0.3, random_state=42)
```

```
In [ ]: x_train = x_train.reshape(-1,1)
        y_train = y_train.reshape(-1,1)
        x_test = x_test.reshape(-1,1)
        y_test = y_test.reshape(-1,1)
```

```
In [ ]: lr_model = LinearRegression()
```

```
In [ ]: lr_model.fit(x_train, y_train)
```

```
In [ ]: y_pred = lr_model.predict(x_test)
```

```
In [143]: r2_score(y_test, y_pred)

Out[143]: 0.6083634494745023
```

We used the train test split holdout approach as well. The accuracy output was 60.8%

Ada Boosting with K-Fold Cross Validation Approach:

We also performed Ada boosting with K-Fold approach to see if it increases accuracy and but there wasn't much of a difference.

Boosted Linear Regression Model

```
In [155]: #Seperate features and target label
X = uber_clean['Distance'].values
Y = uber_clean['fare_amount'].values
```

```
In [156]: X = X.reshape(-1,1)
Y = Y.reshape(-1,1)
```

```
In [157]: base_model = LinearRegression()
ada_boost = AdaBoostRegressor(estimator = base_model, n_estimators=400, learning_rate=1, loss='square')
accuracy_list = []
mse_list = []
```

```
In [ ]: kf = KFold(n_splits = 10, shuffle=True, random_state=42)
for train_index, test_index in kf.split(X):
    x_train, x_test = X[train_index], X[test_index]
    y_train, y_test = Y[train_index], Y[test_index]

    #x_train, x_test = x_train[0], x_test[0]
    #y_train, y_test = y_train[0], y_test[0]

    ada_boost.fit(x_train, y_train)
    y_pred = ada_boost.predict(x_test)

    accuracy_list.append(r2_score(y_test, y_pred))
    mse_list.append(mean_squared_error(y_test, y_pred))
```

```
In [164]: avg_score = np.array(accuracy_list).mean()
avg_error = np.array(mse_list).mean()
print("Average Coeficient of Determination: {}".format(round(avg_score,3)))
print("Average Mean Square Error: {}".format(round(avg_error,3)))

Average Coeficient of Determination: 0.575
Average Mean Square Error: 3.768
```

References:

- https://www.kaggle.com/code/yasserh/uber-fare-prediction-comparing-best-ml-models
- https://medium.com/@rishabh21071/uber-fare-and-demand-prediction-data-analysis-fc26201b03f
- https://www.youtube.com/watch?v=bLEp-8V-F0I&ab_channel=Simply AI