4. Build a Linear model for the given dataset with regularisation. Attempt to interpret the variable coefficients of the Linear Model

Linear Regression

Importing Libraries

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Importing the data

In [2]:
```
# Importing Data
data = pd.read_csv('nyc_taxi_trip_duration.csv')
data.head()
```

Out[2]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	N	400
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	N	1100
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	N	1635
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	N	1141
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	N	848

creating a function to identify outliers:

In [3]:
```
def UVA_outlier(data, var):
#     import pdb
#     pdb.set_trace()
    # calculating descriptives of variable
    quant25 = data[var].quantile(0.25)
    quant75 = data[var].quantile(0.75)
    IQR = quant75 - quant25
    med = data[var].median()
    whis_low = quant25-(1.5*IQR)
    whis_high = quant75+(1.5*IQR)

    ls = data.index[(data[var] < whis_low) | (data[var] > whis_high)]

    return ls
```

Function to remove outliers

In [6]:
```
def remove(df,ls):
    ls = sorted(set(ls))
    df = df.drop(ls)
    return df
```

In [16]:
```
# import pdb
index_list1 = []

# for j in data.drop(['id','vendor_id','pickup_datetime','dropoff_datetime','store_and_fwd_flag'], axis=1).columns:
for j in ['trip_duration','pickup_longitude','dropoff_longitude','pickup_latitude','dropoff_latitude']:
# for j in data.columns:
#     pdb.set_trace()
    for i in [j]:
        index_list1.extend(UVA_outlier(data,i))
        data_cleaned = remove(data,index_list1)
        index_list1.clear()
```

In [17]:
```
data_cleaned.shape
```

Out[17]:
```
(693076, 11)
```

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

Out[18]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	-73.963875	40.771164	N	400
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	-73.994751	40.694931	N	1100
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	-73.948029	40.774918	N	1635
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	-73.956779	40.780628	N	1141
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	-73.988182	40.740631	N	848

In [19]:
```
data = data_cleaned
```

Extracting data from date time column

In [20]:
```
# creating an instance(date) of DatetimeIndex class using "pickup_datetime"
date_pick = pd.DatetimeIndex(data['pickup_datetime'])
# creating an instance(date) of DatetimeIndex class using "dropoff_datetime"
date_drop = pd.DatetimeIndex(data['dropoff_datetime'])

# extracting new columns from "pick datetime"

# last day of year when pickup was done
data['doy_pick'] = date_pick.dayofyear

# week of year when pickup was done
data['woy_pick'] = date_pick.weekofyear

# month of year when pickup was done
data['moy_pick'] = date_pick.month

# day of week when pickup was done
data['dow_pick'] = date_pick.dayofweek

# hour of day when pickup was done
data['hod_pick'] = date_pick.hour


# extracting new columns from "dropoff datetime"

# last day of year dropoff was done
data['doy_drop'] = date_drop.dayofyear

# week of year when dropoff was done
data['woy_drop'] = date_drop.weekofyear

# month of year when dropoff was done
data['moy_drop'] = date_drop.month

# day of week when dropoff was done
data['dow_drop'] = date_drop.dayofweek

# hour of day when dropoff was done
data['hod_drop'] = date_drop.hour
```

```
C:\Users\vempa\AppData\Local\Temp/ipykernel_6968/1098005334.py:12: FutureWarning: weekofyear and week have been deprecated, please use DatetimeIndex.isocalendar().week instead, which returns a Series.  To exactly reproduce the behavior of week and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar().week)
  data['woy_pick'] = date_pick.weekofyear
C:\Users\vempa\AppData\Local\Temp/ipykernel_6968/1098005334.py:30: FutureWarning: weekofyear and week have been deprecated, please use DatetimeIndex.isocalendar().week instead, which returns a Series.  To exactly reproduce the behavior of week and weekofyear and return an Index, you may call pd.Int64Index(idx.isocalendar().week)
  data['woy_drop'] = date_drop.weekofyear
```

In [21]:
```
data.dtypes
```

Out[21]:
```
id                     object
vendor_id               int64
pickup_datetime        object
dropoff_datetime       object
passenger_count         int64
pickup_longitude      float64
pickup_latitude       float64
dropoff_longitude     float64
dropoff_latitude      float64
store_and_fwd_flag     object
trip_duration           int64
doy_pick                int64
woy_pick                int64
moy_pick                int64
dow_pick                int64
hod_pick                int64
doy_drop                int64
woy_drop                int64
moy_drop                int64
dow_drop                int64
hod_drop                int64
dtype: object
```

In [22]:
```
data = pd.get_dummies(data.drop('id',axis=1), columns = ['store_and_fwd_flag'])
```

In [23]:
```
data.tail()
```

Out[23]:

	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration	doy_pick	...	moy_pick	dow_pick	hod_pick	doy_
729317	2	2016-05-21 13:29:38	2016-05-21 13:34:34	2	-73.965919	40.789780	-73.952637	40.789181	296	142	...	5	5	13	
729318	1	2016-02-22 00:43:11	2016-02-22 00:48:26	1	-73.996666	40.737434	-74.001320	40.731911	315	53	...	2	0	0	
729319	1	2016-04-15 18:56:48	2016-04-15 19:08:01	1	-73.997849	40.761696	-74.001488	40.741207	673	106	...	4	4	18	
729320	1	2016-06-19 09:50:47	2016-06-19 09:58:14	1	-74.006706	40.708244	-74.013550	40.713814	447	171	...	6	6	9	
729321	2	2016-01-01 17:24:16	2016-01-01 17:44:40	4	-74.003342	40.743839	-73.945847	40.712841	1224	1	...	1	4	17	

5 rows × 21 columns

In [25]:
```
data_cleaned = data.drop(['pickup_datetime','dropoff_datetime'], axis=1)
```

Segregating variables: Independent and Dependent Variables

In [26]:
```
#seperating independent and dependent variables
x = data.drop(['trip_duration','pickup_datetime','dropoff_datetime'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[26]:
```
((693076, 18), (693076,))
```

Splitting the data into train set and the test set

In [27]:
```
# Importing the train test split function
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 42)
```

Implementing Linear Regression

In [28]:
```
#importing Linear Regression and metric mean square error
from sklearn.linear_model import LinearRegression as LR
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import r2_score
```

In [29]:
```
# Creating instance of Linear Regresssion
lr = LR()

# Fitting the model
lr.fit(train_x, train_y)
```

Out[29]:
```
▾ LinearRegression
LinearRegression()
```

In [30]:
```
# Predicting over the Train Set and calculating error
train_predict = lr.predict(train_x)
k = mae(train_predict, train_y)
print('Training Mean Absolute Error', k )
R_squared = r2_score(train_predict,train_y)
print('R2 score on train set', R_squared )
```

```
Training Mean Absolute Error 935.9382019789173
R2 score on train set 0.887845941706592
```

In [31]:
```
# Predicting over the Test Set and calculating error
test_predict = lr.predict(test_x)
k = mae(test_predict, test_y)
print('Test Mean Absolute Error    ', k )
R_squared = r2_score(test_predict,test_y)
print('R2 score on test set', R_squared )
```

```
Test Mean Absolute Error     933.0378564243354
R2 score on test set 0.8268991472379279
```

Parameters of Linear Regression

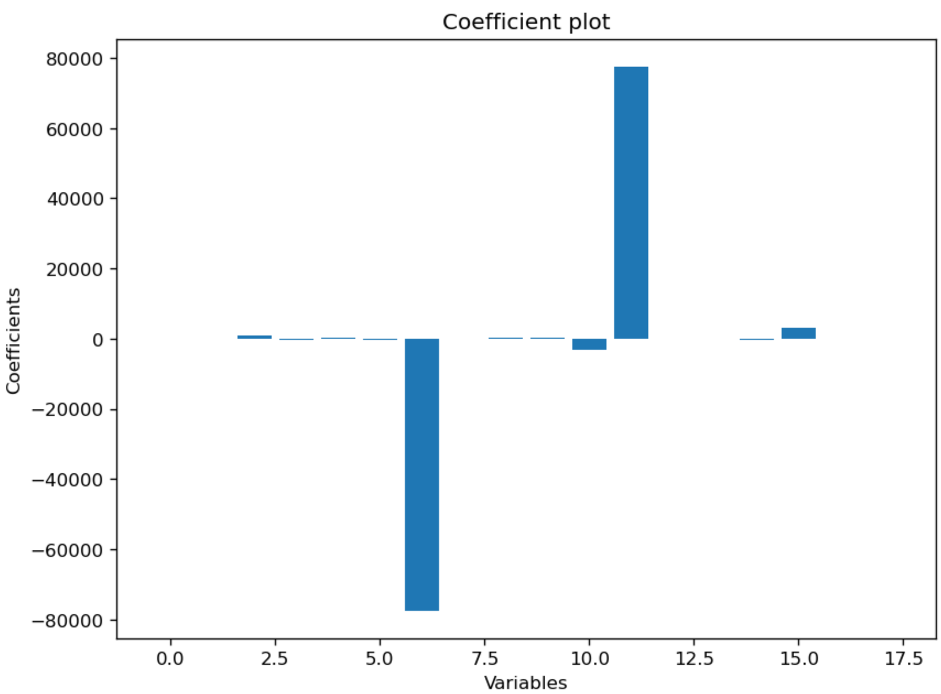
In [32]:
```
lr.coef_
```

Out[32]:
```
array([ 1.97460507e+01,  1.00938024e+00,  8.25841764e+02, -3.60814511e+02,
        1.99492352e+02, -4.55845213e+02, -7.75964570e+04, -2.45443053e+01,
        1.87602423e+02,  2.12251139e+02, -3.22946315e+03,  7.75963920e+04,
        2.48700620e+01, -1.85726162e+02, -2.14279628e+02,  3.23019609e+03,
        3.81055761e+00, -3.81055761e+00])
```

Plotting the coefficients

In [33]:
```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Coefficient plot')
```

Out[33]:
```
Text(0.5, 1.0, 'Coefficient plot')
```



Here we can see that the model depends upon some Independent variables too much, But these coefficients are not suitable for interpretation because these are not scaled, therefore we will normalize and interpret later.

Checking assumptions of Linear Model

In [34]:
```python
# Arranging and calculating the Residuals
residuals = pd.DataFrame({
    'fitted values' : test_y,
    'predicted values' : test_predict,
})

residuals['residuals'] = residuals['fitted values'] - residuals['predicted values']
residuals.head()
```

Out[34]:

	fitted values	predicted values	residuals
22461	1550	95.523576	1454.476424
606882	2004	3411.637743	-1407.637743
509380	1611	94.518419	1516.481581
105494	91	96.257752	-5.257752
54890	730	3195.555017	-2465.555017

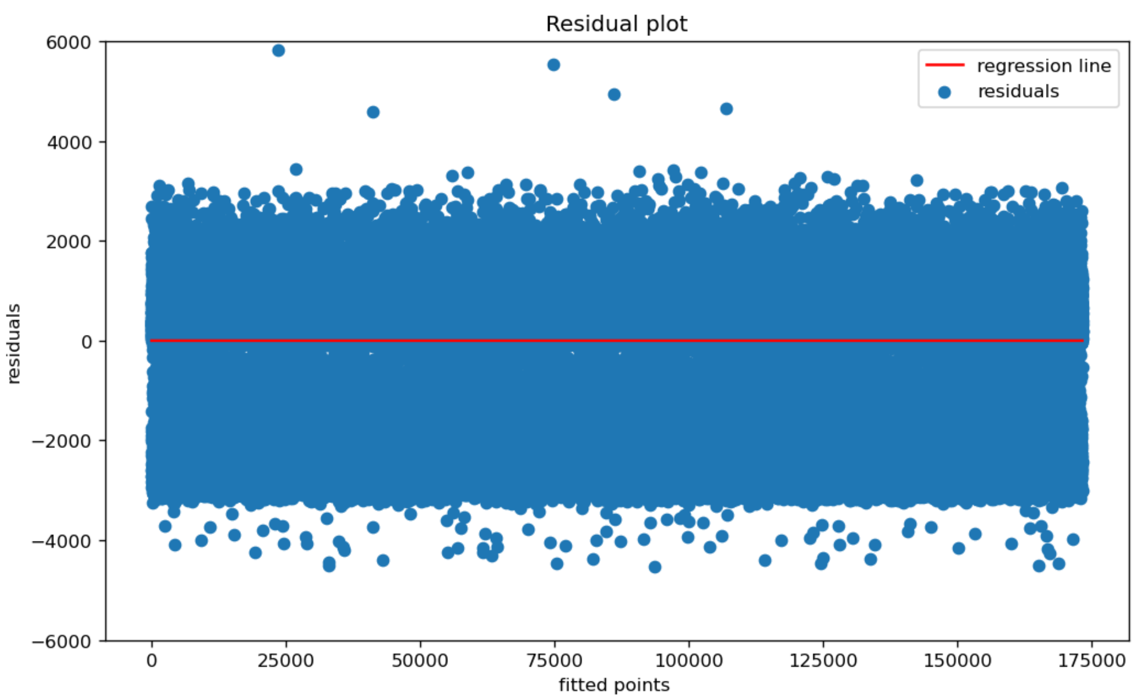
Plotting residual curve (Is there constant Variance OR Homoscedastic?)

In [35]:
```python
residuals.residuals[:]
```

Out[35]:
```
22461     1454.476424
606882   -1407.637743
509380    1516.481581
105494      -5.257752
54890    -2465.555017
             ...
279147     188.122587
71621     -533.849650
242060   -2420.875546
152036      35.267649
665089    1229.155245
Name: residuals, Length: 173269, dtype: float64
```

In [36]:
```python
plt.figure(figsize=(10, 6), dpi=120, facecolor='w', edgecolor='b')
f = range(0,173269)
k = [0 for i in range(0,173269)]
plt.scatter( f, residuals.residuals[:], label = 'residuals')
plt.plot( f, k , color = 'red', label = 'regression line' )
plt.xlabel('fitted points ')
plt.ylabel('residuals')
plt.title('Residual plot')
plt.ylim(-6000, 6000)
plt.legend()
```

Out[36]: <matplotlib.legend.Legend at 0x15c897e1e80>

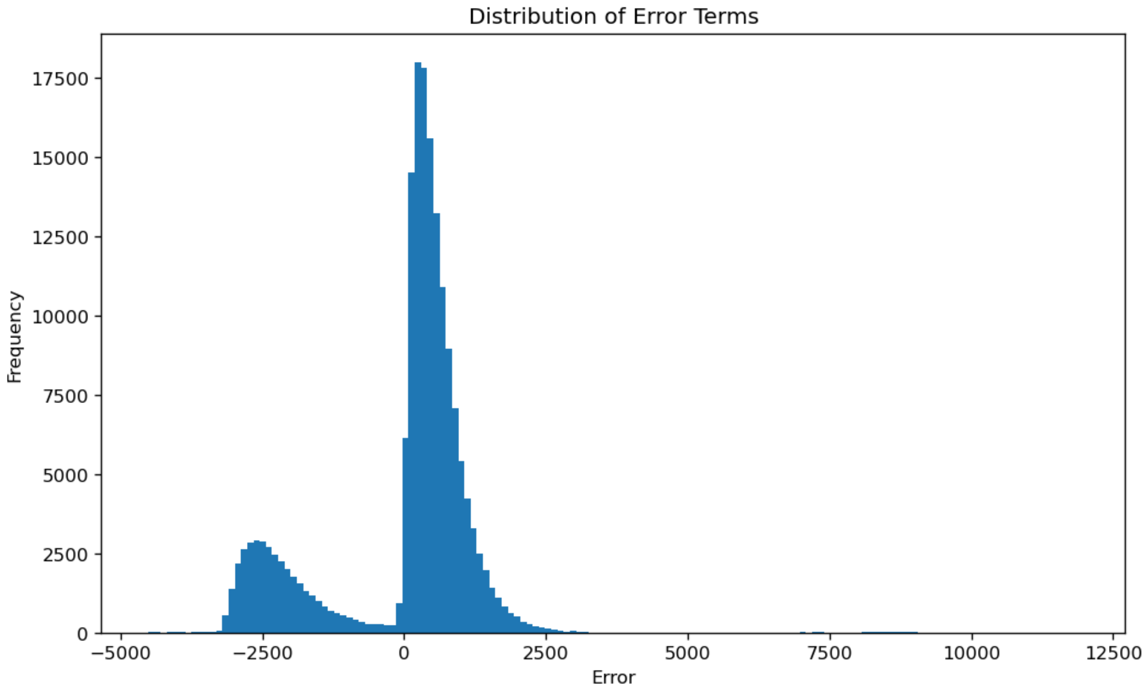


The Residual plot clearly Looks Homoscedastic, i.e. the the variance of the error across the dataset is nearly constant.

Checking Distribution of Residuals

In [37]:

```
# Histogram for distribution
plt.figure(figsize=(10, 6), dpi=120, facecolor='w', edgecolor='b')
plt.hist(residuals.residuals, bins = 150)
plt.xlabel('Error')
plt.ylabel('Frequency')
plt.title('Distribution of Error Terms')
# plt.xlim(-10000, 20000)
plt.show()
```

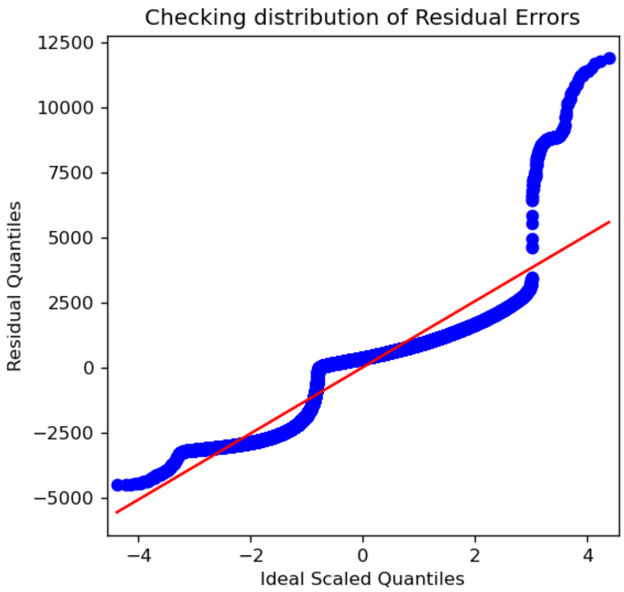


According to the Histogram, the distribution of error is not normal

QQ-Plot (Is the data Normally Distributed?)

In [38]:
```python
# importing the QQ-plot from the from the statsmodels
from statsmodels.graphics.gofplots import qqplot

## Plotting the QQ plot
fig, ax = plt.subplots(figsize=(5,5) , dpi = 120)
qqplot(residuals.residuals, line = 's' , ax = ax)
plt.ylabel('Residual Quantiles')
plt.xlabel('Ideal Scaled Quantiles')
plt.title('Checking distribution of Residual Errors')
plt.show()
```

```
C:\Users\vempa\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:993: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fm
t string "bo" (-> marker='o'). The keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```




```
The QQ-plot clearly verifies our findings from the the histogram of the residuals, the data is not normal, Even after removing outliers with interquartile range method,
there are still outliers all over the plot.
```

Variance Inflation Factor (VIF) (Checking for multi collinearity)

In [39]:
```python
# Importing Variance_inflation_Factor funtion from the Statsmodels
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Calculating VIF for every column (only works for the not Catagorical)
VIF = pd.Series([variance_inflation_factor(data_cleaned.values, i) for i in range(data_cleaned.shape[1])], index =data_cleaned.columns)
VIF
```

Out[39]:
```
vendor_id              1.097122e+00
passenger_count        1.090781e+00
pickup_longitude       1.100462e+00
pickup_latitude        1.203827e+00
dropoff_longitude      1.250740e+00
dropoff_latitude       1.380683e+00
trip_duration          8.978217e+00
doy_pick               1.047890e+07
woy_pick               6.959414e+02
moy_pick               9.132378e+03
dow_pick               9.525813e+01
hod_pick               2.811622e+02
doy_drop               1.047914e+07
woy_drop               6.972797e+02
moy_drop               9.132563e+03
dow_drop               9.535981e+01
hod_drop               2.880422e+02
store_and_fwd_flag_N   1.671513e+07
store_and_fwd_flag_Y   8.833919e+04
dtype: float64
```

In [40]:
```python
round(VIF[0:14],2)
```

Out[40]:
```
vendor_id                   1.10
passenger_count             1.09
pickup_longitude            1.10
pickup_latitude             1.20
dropoff_longitude           1.25
dropoff_latitude            1.38
trip_duration               8.98
doy_pick             10478897.96
woy_pick                  695.94
moy_pick                 9132.38
dow_pick                   95.26
hod_pick                  281.16
doy_drop             10479141.97
woy_drop                  697.28
dtype: float64
```

From this list, we clearly see that there happens to be many Independent Variable over the value of 5, which means that there are Many features that exhibit the Multicollinearity in the dataset. Note that VIF only works for the Continuous Variables.

In [41]:
```
data_cleaned.dtypes
```

Out[41]:
```
vendor_id                 int64
passenger_count           int64
pickup_longitude        float64
pickup_latitude         float64
dropoff_longitude       float64
dropoff_latitude        float64
trip_duration             int64
doy_pick                  int64
woy_pick                  int64
moy_pick                  int64
dow_pick                  int64
hod_pick                  int64
doy_drop                  int64
woy_drop                  int64
moy_drop                  int64
dow_drop                  int64
hod_drop                  int64
store_and_fwd_flag_N      uint8
store_and_fwd_flag_Y      uint8
dtype: object
```

Model Interpretability

So far we have simply been predicting the values using the linear regression, But in order to Interpret the model, the normalising of the data is essential.

In [42]:
```
train_x['vendor_id']
```

Out[42]:
```
5342      2
326132    2
288391    1
205218    2
433541    1
         ..
272816    2
385069    2
138799    2
706259    1
128270    1
Name: vendor_id, Length: 519807, dtype: int64
```

In [43]:
```
# Creating instance of Linear Regresssion
lr = LR(normalize = True)

# Fitting the model
lr.fit(train_x, train_y)
```

```
C:\Users\vempa\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

Out[43]:
```
        LinearRegression
LinearRegression(normalize=True)
```

In [44]:
```
# Predicting over the Train Set and calculating error
train_predict = lr.predict(train_x)
k = mae(train_predict, train_y)
print('Training Mean Absolute Error', k )
R_squared = r2_score(train_predict,train_y)
print('R2 score on test set', R_squared )
```

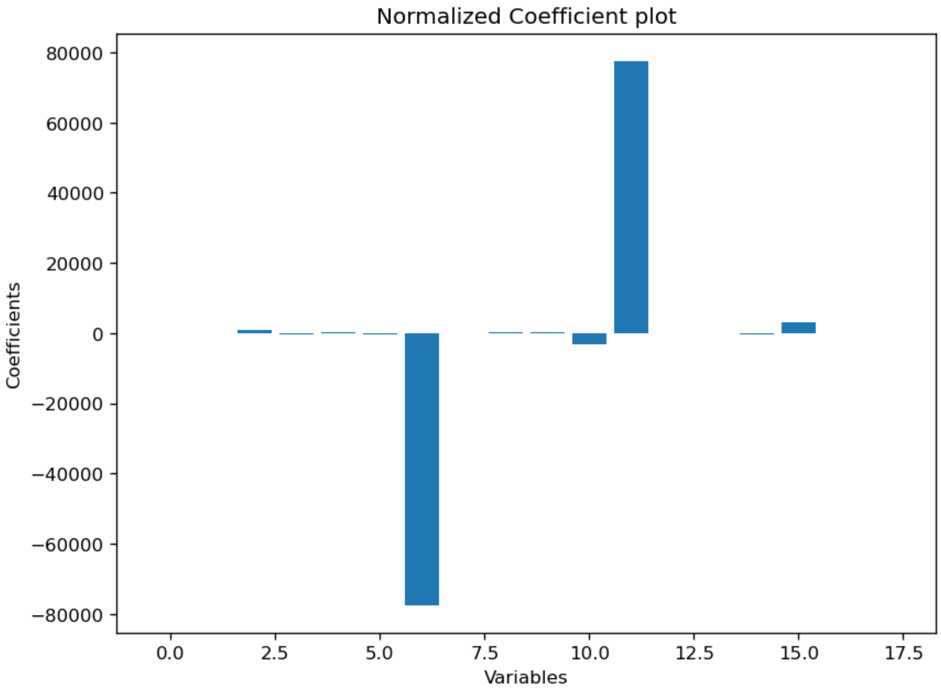
```
Training Mean Absolute Error 935.9382019791508
R2 score on test set 0.8878459417066005
```

In [45]:
```
# Predicting over the Test Set and calculating error
test_predict = lr.predict(test_x)
k = mae(test_predict, test_y)
print('Test Mean Absolute Error    ', k )
R_squared = r2_score(test_predict,test_y)
print('R2 score on test set', R_squared )
```

```
Test Mean Absolute Error     933.0378564245725
R2 score on test set 0.8268991472379501
```

In [46]:
```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Normalized Coefficient plot')
```

Out[46]: `Text(0.5, 1.0, 'Normalized Coefficient plot')`



Now the coefficients we see are normalised and we can easily make final inferences out of it.

Here we can see that there are a lot of Coefficients which are near to zero and not Significant. So let us try removing them and build the model again.

Creating new subsets of data

In [47]:
```
#seperating independent and dependent variables
x = data.drop(['trip_duration','pickup_datetime','dropoff_datetime'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[47]: `((693076, 18), (693076,))`

Arranging coefficients with features

In [48]:
```
Coefficients = pd.DataFrame({
    'Variable'    : x.columns,
    'coefficient' : lr.coef_
})
Coefficients.head()
```

Out[48]:

	Variable	coefficient
0	vendor_id	19.746051
1	passenger_count	1.009380
2	pickup_longitude	825.841764
3	pickup_latitude	-360.814511
4	dropoff_longitude	199.492352

Choosing variables with sigificance greater than 0.5 (Filtering Significant Features)

In [49]:
```
sig_var = Coefficients[Coefficients.coefficient > 0.5]
```

Extracting the significant subset of independent Variables

In [50]:
```
subset = data[sig_var['Variable'].values]
subset.head()
```

Out[50]:

	vendor_id	passenger_count	pickup_longitude	dropoff_longitude	moy_pick	dow_pick	doy_drop	woy_drop	hod_drop	store_and_fwd_flag_N
0	2	1	-73.953918	-73.963875	2	0	60	9	16	1
1	1	2	-73.988312	-73.994751	3	4	71	10	23	1
2	2	2	-73.997314	-73.948029	2	6	52	7	18	1
3	2	6	-73.961670	-73.956779	1	1	5	1	10	1
4	1	1	-74.017120	-73.988182	2	2	48	7	6	1

Splitting the data into train set and the test set

In [51]:
```
# Importing the train test split function
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(subset, y , random_state = 42)
```

Implementing Linear Regression

In [52]:
```
#importing Linear Regression and metric mean square error
from sklearn.linear_model import LinearRegression as LR
from sklearn.metrics import mean_absolute_error as mae
```

Training Model

In [53]:
```
# Creating instance of Linear Regresssion with Normalised Data
lr = LR(normalize = True)

# Fitting the model
lr.fit(train_x, train_y)
```

```
C:\Users\vempa\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normalize' was deprecated in version 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LinearRegression())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)


  warnings.warn(
```

Out[53]:
```
▾ LinearRegression
LinearRegression(normalize=True)
```

Predicting over the train set

In [54]:
```
# Predicting over the Train Set and calculating error
train_predict = lr.predict(train_x)
k = mae(train_predict, train_y)
print('Training Mean Absolute Error', k )
```

```
Training Mean Absolute Error 560.9403158158182
```

Predicting over the test set

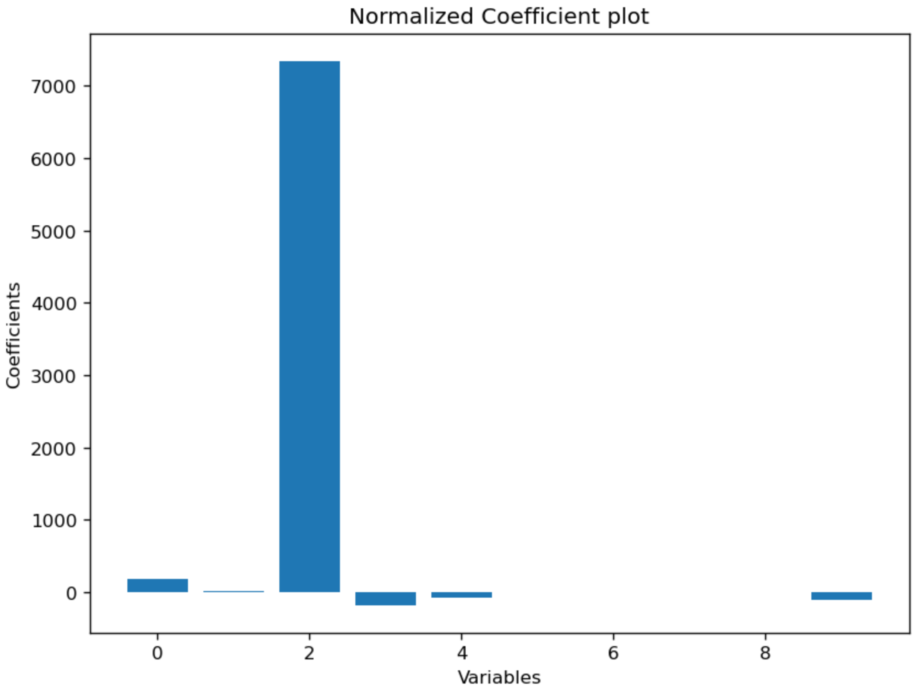
In [55]:
```
# Predicting over the Test Set and calculating error
test_predict = lr.predict(test_x)
k = mae(test_predict, test_y)
print('Test Mean Absolute Error    ', k )
```

```
Test Mean Absolute Error     556.8090417448929
```

Plotting the coefficients

In [56]:
```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Normalized Coefficient plot')
```

Out[56]: `Text(0.5, 1.0, 'Normalized Coefficient plot')`



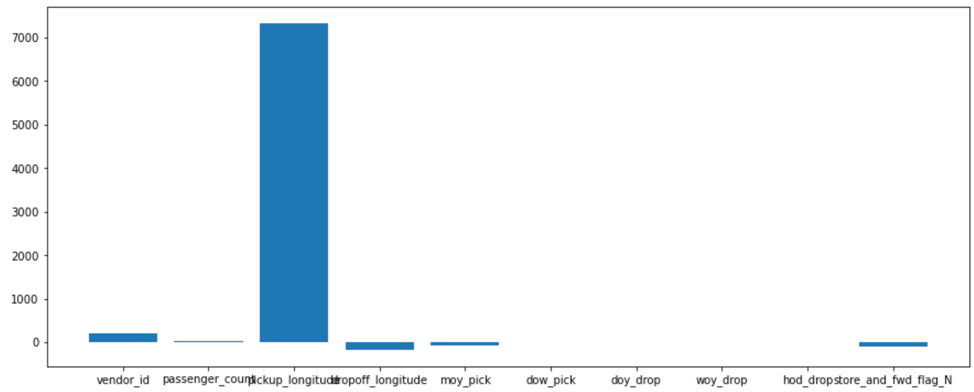
Ridge Regression

In [66]:

```python
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

ridgeRegressor = Ridge(alpha = 0.5 )
ridgeRegressor.fit(train_x,train_y)
y_test_predict_ridge  = ridgeRegressor.predict(test_x)
y_train_predict_ridge  = ridgeRegressor.predict(train_x)

# R_squared = r2_score(y_predict_ridge,test_y)
# print('R2 Score on test set', R_squared )

# Predicting over the Train Set and calculating error
# train_predict = lr.predict(train_x)
k1 = mae(y_train_predict_ridge, train_y)
print('Training Mean Absolute Error', k1 )

#### Predicting over the test set

# Predicting over the Test Set and calculating error
# test_predict = lr.predict(test_x)
k2 = mae(y_test_predict_ridge, test_y)
print('Test Mean Absolute Error    ', k2 )

coefficient_df = pd.DataFrame()
coefficient_df['Column_Name'] = train_x.columns
coefficient_df['Coefficient_Value'] = pd.Series(ridgeRegressor.coef_)
print(coefficient_df.head(15))


plt.rcParams["figure.figsize"] = (15,6)
plt.bar(coefficient_df['Column_Name'],coefficient_df['Coefficient_Value'])
```

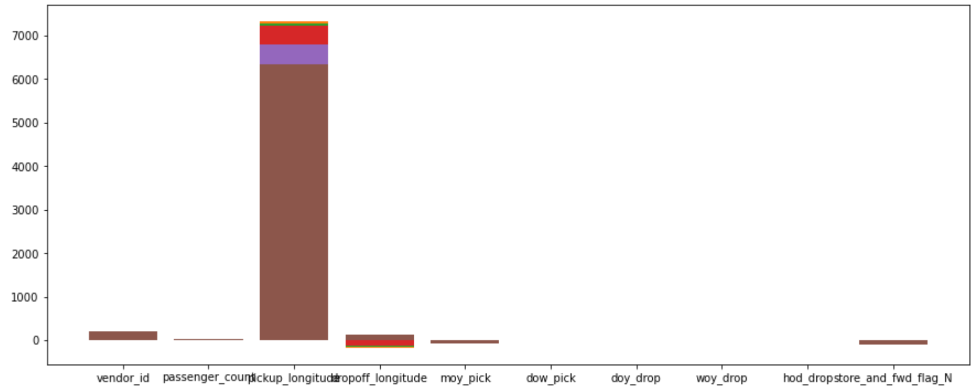
```
Training Mean Absolute Error 560.9328633555573
Test Mean Absolute Error     556.8012764347889
            Column_Name  Coefficient_Value
0             vendor_id         189.728970
1       passenger_count          16.085504
2      pickup_longitude        7335.928229
3     dropoff_longitude        -179.946308
4              moy_pick         -71.467552
5              dow_pick           0.076024
6              doy_drop           2.945474
7              woy_drop           0.056995
8              hod_drop           1.224168
9    store_and_fwd_flag_N         -99.704619
```

Out[66]: <BarContainer object of 10 artists>



In [67]:
```
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

list2 = [0.5,1,5,10,50,100]

for i in list2:
    ridgeRegressor = Ridge(alpha = i )
    ridgeRegressor.fit(train_x,train_y)
#     y_predict_ridge  = ridgeRegressor.predict(test_x)
    y_test_predict_ridge  = ridgeRegressor.predict(test_x)
    y_train_predict_ridge  = ridgeRegressor.predict(train_x)

# R_squared = r2_score(y_predict_ridge,test_y)
# print('R2 Score on test set', R_squared )

# Predicting over the Train Set and calculating error
# train_predict = lr.predict(train_x)
    k1 = mae(y_train_predict_ridge, train_y)
    print('Training Mean Absolute Error', k1 )

#### Predicting over the test set

# Predicting over the Test Set and calculating error
# test_predict = lr.predict(test_x)
    k2 = mae(y_test_predict_ridge, test_y)
    print('Test Mean Absolute Error    ', k2 )


    coefficient_df = pd.DataFrame()
    coefficient_df['Column_Name'] = train_x.columns
    coefficient_df['Coefficient_Value'] = pd.Series(ridgeRegressor.coef_)
    print(coefficient_df.head(15))


    plt.rcParams["figure.figsize"] = (15,6)
    plt.bar(coefficient_df['Column_Name'],coefficient_df['Coefficient_Value'])
```

```
Training Mean Absolute Error 560.9328633555573
Test Mean Absolute Error     556.8012764347889
            Column_Name  Coefficient_Value
0             vendor_id         189.728970
1       passenger_count          16.085504
2      pickup_longitude        7335.928229
3     dropoff_longitude        -179.946308
4              moy_pick         -71.467552
5              dow_pick           0.076024
6              doy_drop           2.945474
7              woy_drop           0.056995
8              hod_drop           1.224168
9   store_and_fwd_flag_N         -99.704619
Training Mean Absolute Error 560.9254744431873
Test Mean Absolute Error     556.7935956156483
            Column_Name  Coefficient_Value
0             vendor_id         189.735586
1       passenger_count          16.085377
2      pickup_longitude        7329.996764
3     dropoff_longitude        -177.754599
4              moy_pick         -71.466526
5              dow_pick           0.073072
6              doy_drop           2.945426
7              woy_drop           0.057266
8              hod_drop           1.225140
9   store_and_fwd_flag_N         -99.748400
Training Mean Absolute Error 560.8684072619743
Test Mean Absolute Error     556.734181647583
            Column_Name  Coefficient_Value
0             vendor_id         189.788113
1       passenger_count          16.084358
2      pickup_longitude        7282.927011
3     dropoff_longitude        -160.521745
4              moy_pick         -71.458304
5              dow_pick           0.049619
6              doy_drop           2.945038
7              woy_drop           0.059420
8              hod_drop           1.232827
9   store_and_fwd_flag_N        -100.094594
Training Mean Absolute Error 560.8016783831772
Test Mean Absolute Error     556.6649950914696
            Column_Name  Coefficient_Value
0             vendor_id         189.852770
1       passenger_count          16.083074
2      pickup_longitude        7225.026697
3     dropoff_longitude        -139.711222
4              moy_pick         -71.447980
5              dow_pick           0.020696
6              doy_drop           2.944554
7              woy_drop           0.062077
8              hod_drop           1.242203
9   store_and_fwd_flag_N        -100.517385
Training Mean Absolute Error 560.4343604536364
Test Mean Absolute Error     556.2840884199773
            Column_Name  Coefficient_Value
0             vendor_id         190.332841
1       passenger_count          16.072598
2      pickup_longitude        6795.948003
3     dropoff_longitude           1.434118
4              moy_pick         -71.363948
5              dow_pick          -0.196121
6              doy_drop           2.940712
7              woy_drop           0.082056
8              hod_drop           1.308996
9   store_and_fwd_flag_N        -103.534618
Training Mean Absolute Error 560.2968631920423
Test Mean Absolute Error     556.1405541289549
            Column_Name  Coefficient_Value
0             vendor_id         190.851843
1       passenger_count          16.059593
2      pickup_longitude        6331.565163
3     dropoff_longitude         129.148546
4              moy_pick         -71.256822
5              dow_pick          -0.435524
6              doy_drop           2.936015
7              woy_drop           0.104204
8              hod_drop           1.376135
9   store_and_fwd_flag_N        -106.528160
```



MAE reduced a bit for Ridge regressor

As we increase the alpha, the coefficients are changing but not becoming zero completely as expected in lasso model

Bar plot shows that pick longitude, drop off longitude has significant impact on the target variable - trip duration

surprisingly pick up latitude,dropoff latitude has no significant impact.

storeand flag co-efficient is showing impact on duration which is not practical

pickup longitude is more significant than the date and time - maybe the location(Traffic,roads) could be the major factor

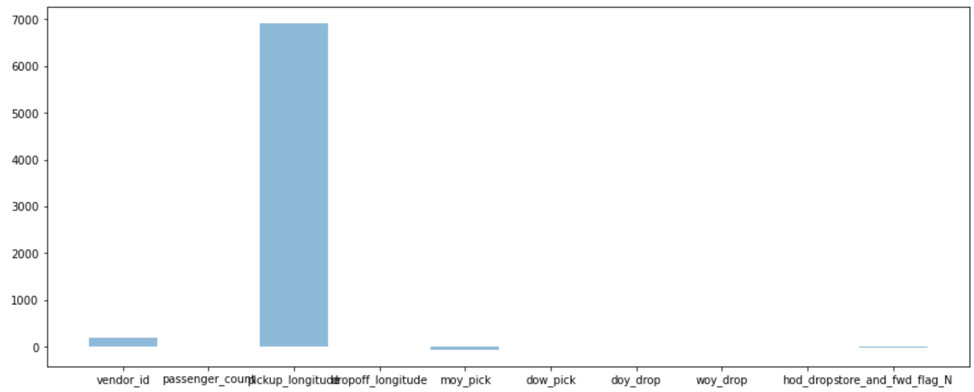
Lasso Regression

In [64]:

```
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score


LassoRegressor = Lasso(alpha = 0.5)
LassoRegressor.fit(train_x,train_y)
y_test_predict_lasso  = LassoRegressor.predict(test_x)
y_train_predict_lasso  = LassoRegressor.predict(train_x)

k3 = mae(y_train_predict_lasso, train_y)
print('Training Mean Absolute Error', k1 )

k4 = mae(y_test_predict_lasso, test_y)
print('Test Mean Absolute Error    ', k2 )


coefficient_df = pd.DataFrame()
coefficient_df['Column_Name'] = train_x.columns
coefficient_df['Coefficient_Value'] = pd.Series(LassoRegressor.coef_)
print(coefficient_df.head(15))


plt.rcParams["figure.figsize"] = (15,6)
plt.bar(coefficient_df['Column_Name'],coefficient_df['Coefficient_Value'],alpha = 0.5)
```

```
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
            Column_Name  Coefficient_Value
0             vendor_id         187.203868
1       passenger_count          16.000716
2      pickup_longitude        6920.123193
3     dropoff_longitude          -0.000000
4              moy_pick         -64.780914
5              dow_pick          -0.000000
6              doy_drop           2.737195
7              woy_drop           0.010440
8              hod_drop           1.291366
9  store_and_fwd_flag_N          -8.556934
```

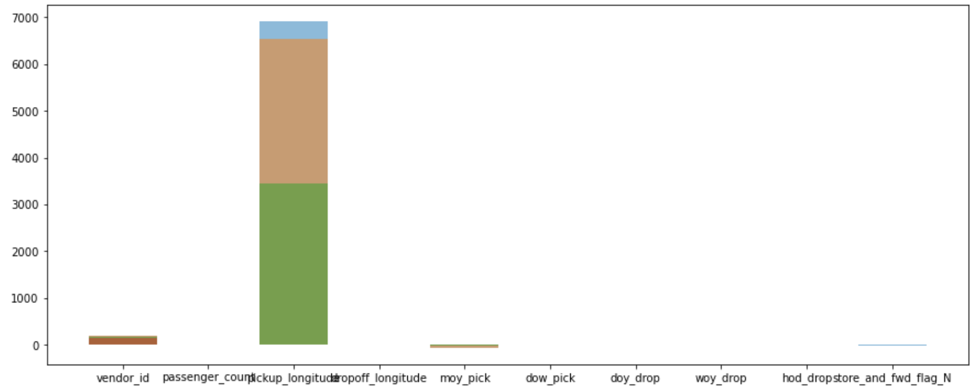
Out[64]: <BarContainer object of 10 artists>



In [65]:

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score

list1 = [0.5,1,5,10,50,100]

for i in list1:

    LassoRegressor = Lasso(alpha = i )
    LassoRegressor.fit(train_x,train_y)
    y_test_predict_lasso  = LassoRegressor.predict(test_x)
    y_train_predict_lasso  = LassoRegressor.predict(train_x)

    k3 = mae(y_train_predict_lasso, train_y)
    print('Training Mean Absolute Error', k1 )

    k4 = mae(y_test_predict_lasso, test_y)
    print('Test Mean Absolute Error    ', k2 )

    coefficient_df = pd.DataFrame()
    coefficient_df['Column_Name'] = train_x.columns
    coefficient_df['Coefficient_Value'] = pd.Series(LassoRegressor.coef_)
    print(coefficient_df.head(15))


    plt.rcParams["figure.figsize"] = (15,6)
    plt.bar(coefficient_df['Column_Name'],coefficient_df['Coefficient_Value'],alpha = 0.5)
```

```
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
          Column_Name  Coefficient_Value
0           vendor_id         187.203868
1     passenger_count          16.000716
2    pickup_longitude        6920.123193
3   dropoff_longitude          -0.000000
4            moy_pick         -64.780914
5            dow_pick          -0.000000
6            doy_drop           2.737195
7            woy_drop           0.010440
8            hod_drop           1.291366
9  store_and_fwd_flag_N         -8.556934
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
          Column_Name  Coefficient_Value
0           vendor_id         185.690914
1     passenger_count          15.889112
2    pickup_longitude        6535.353194
3   dropoff_longitude           0.000000
4            moy_pick         -58.124709
5            dow_pick          -0.050373
6            doy_drop           2.525743
7            woy_drop          -0.000000
8            hod_drop           1.317949
9  store_and_fwd_flag_N         -0.000000
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
          Column_Name  Coefficient_Value
0           vendor_id         174.379631
1     passenger_count          14.999367
2    pickup_longitude        3453.626906
3   dropoff_longitude           0.000000
4            moy_pick          -4.903665
5            dow_pick          -0.639785
6            doy_drop           0.836382
7            woy_drop          -0.095635
8            hod_drop           1.524114
9  store_and_fwd_flag_N         -0.000000
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
          Column_Name  Coefficient_Value
0           vendor_id         159.634495
1     passenger_count          13.909178
2    pickup_longitude           0.000000
3   dropoff_longitude           0.000000
4            moy_pick          -0.000000
5            dow_pick          -1.268891
6            doy_drop           0.677285
7            woy_drop          -0.000000
8            hod_drop           1.753835
9  store_and_fwd_flag_N         -0.000000
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
          Column_Name  Coefficient_Value
0           vendor_id           3.675598
1     passenger_count           7.626249
2    pickup_longitude           0.000000
3   dropoff_longitude           0.000000
4            moy_pick          -0.000000
5            dow_pick          -0.000000
6            doy_drop           0.655778
7            woy_drop           0.000000
8            hod_drop           0.943642
9  store_and_fwd_flag_N         -0.000000
Training Mean Absolute Error 580.6874771869304
Test Mean Absolute Error     576.2106723643577
          Column_Name  Coefficient_Value
0           vendor_id           0.000000
1     passenger_count           0.000000
2    pickup_longitude           0.000000
3   dropoff_longitude           0.000000
4            moy_pick          -0.000000
5            dow_pick          -0.000000
6            doy_drop           0.636098
7            woy_drop           0.000000
8            hod_drop           0.000000
9  store_and_fwd_flag_N         -0.000000
```



As we increase the alpha, the coefficients are becoming zero as expected in lasso model

Bar plot shows that pick longitude has significant impact on the target variable - trip duration

surprisingly pick up latitude has no significant impact

pickup longitude is more significant than the date and time - maybe the location(Traffic,roads) could be the major factor

In [ ]:

In [ ]: