UBER FARE PREDICTION PROJECT

By Tao Duc Minh - Group 5

The dataset contains the following fields:

- key: a unique identifier for each trip
- fare_amount: the cost of each trip in usd
- pickup_datetime: date and time when the meter was engaged
- passenger_count: the number of passengers in the vehicle (driver entered value)
- pickup_longitude: the longitude where the meter was engaged
- pickup_latitude: the latitude where the meter was engaged
- dropoff_longitude: the longitude where the meter was disengaged
- dropoff_latitude: he latitude where the meter was disengaged

1. Data Exploration

```
#Check for empty elements

nvc = pd.DataFrame(df.isnull().sum().sort_values(), columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null Values']/df.shape[0],3)*100
print(nvc)
df.dropna(inplace=True)
```

```
                   Total Null Values  Percentage
fare_amount                        0         0.0
pickup_datetime                    0         0.0
pickup_longitude                   0         0.0
pickup_latitude                    0         0.0
passenger_count                    0         0.0
dropoff_longitude                  1         0.0
dropoff_latitude                   1         0.0
```

```
# Reframing the columns

df = df[(df.pickup_latitude<90) & (df.dropoff_latitude<90) &
        (df.pickup_latitude>-90) & (df.dropoff_latitude>-90) &
        (df.pickup_longitude<180) & (df.dropoff_longitude<180) &
        (df.pickup_longitude>-180) & (df.dropoff_longitude>-180)]

df.pickup_datetime=pd.to_datetime(df.pickup_datetime)

df['year'] = df.pickup_datetime.dt.year
df['month'] = df.pickup_datetime.dt.month
df['weekday'] = df.pickup_datetime.dt.weekday
df['hour'] = df.pickup_datetime.dt.hour

df['Monthly_Quarter'] = df.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'})
df['Hourly_Segments'] = df.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'})

df['Distance']=[round(geopy.distance.distance((df.pickup_latitude[i], df.pickup_longitude[i]),(df.dr
opoff_latitude[i], df.dropoff_longitude[i])).m,2) for i in df.index]

df.drop(['pickup_datetime','month', 'hour',], axis=1, inplace=True)

original_df = df.copy(deep=True)

df.head()
```

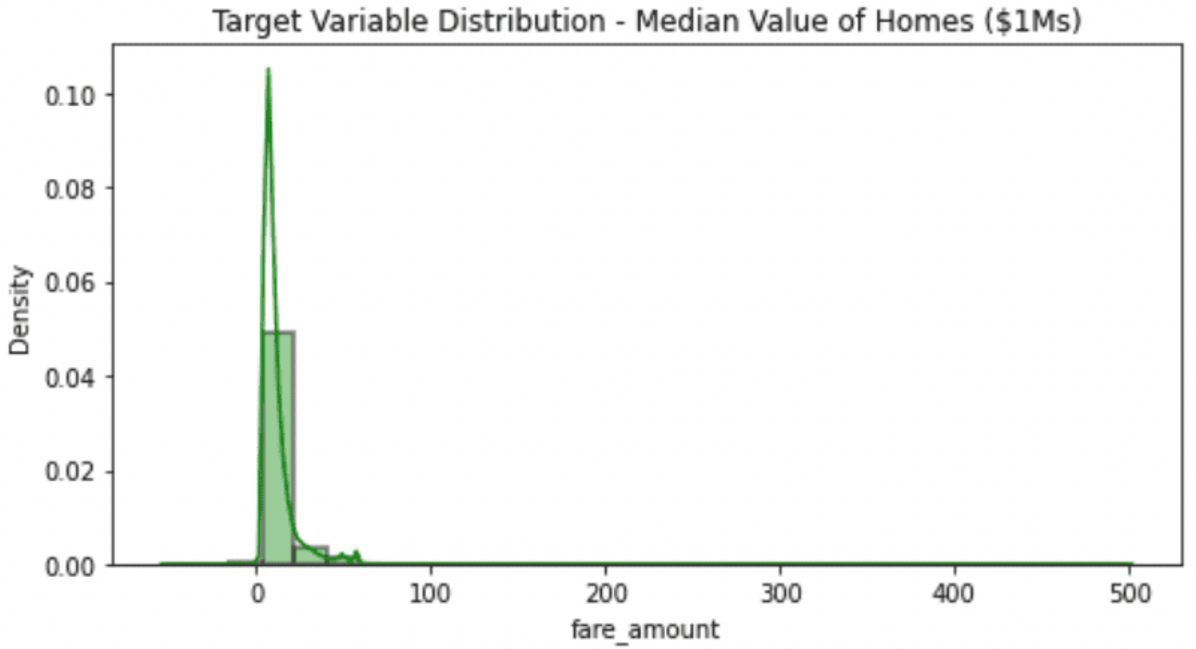
```
#Checking the stats of all the columns

display(df.describe())
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year
count	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000
mean	11.359849	-72.501786	39.917937	-72.511608	39.922031	1.684544	2011.742463
std	9.901868	10.449955	6.130412	10.412192	6.117669	1.385999	1.856438
min	-52.000000	-93.824668	-74.015515	-75.458979	-74.015750	0.000000	2009.000000
25%	6.000000	-73.992064	40.734793	-73.991407	40.733823	1.000000	2010.000000
50%	8.500000	-73.981822	40.752592	-73.980092	40.753042	1.000000	2012.000000
75%	12.500000	-73.967154	40.767157	-73.963658	40.768000	2.000000	2013.000000
max	499.000000	40.808425	48.018760	40.831932	45.031598	208.000000	2015.000000

Inference: The stats seem to be fine, let us do further analysis on the Dataset

2. Exploratory Data Analysis (EDA)

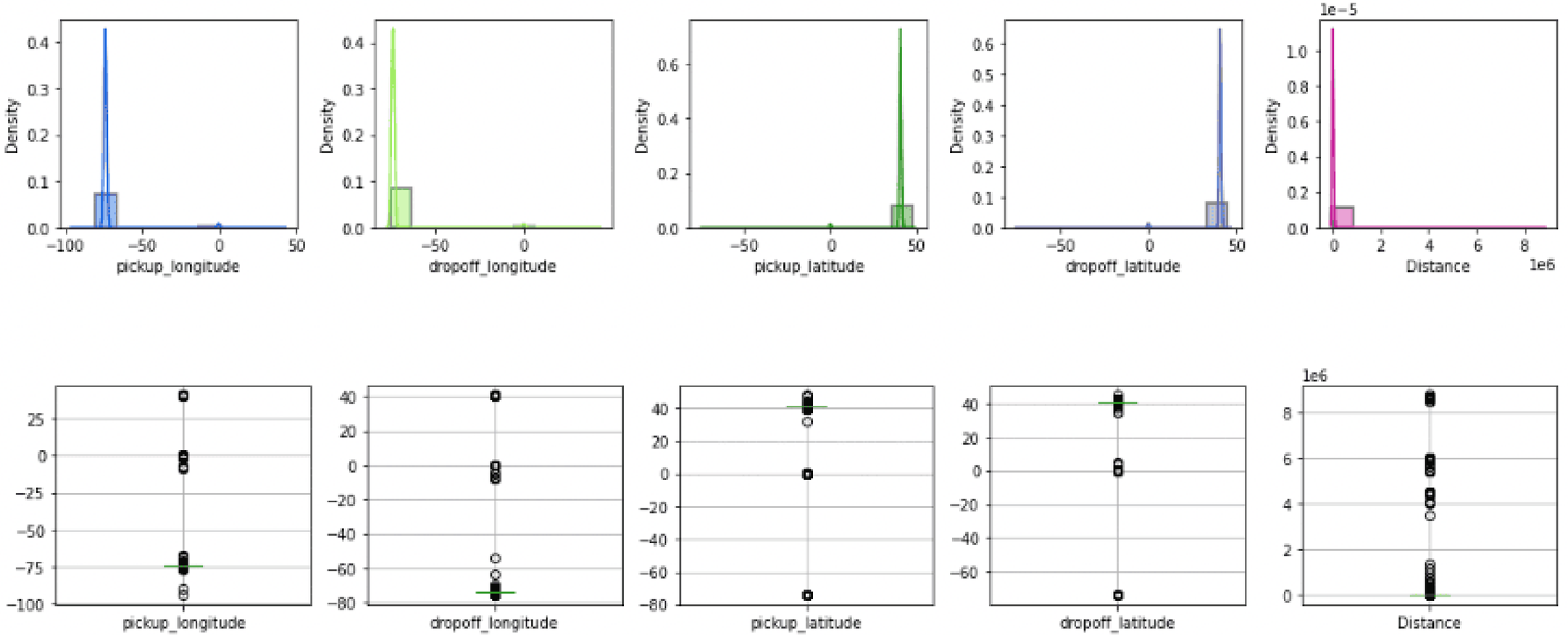
```
#Let us first analyze the distribution of the target variable

plt.figure(figsize=[8,4])
sns.distplot(df[target], color='g',hist_kws=dict(edgecolor="black", linewidth=2), bins=30)
plt.title('Target Variable Distribution - Median Value of Homes ($1Ms)')
plt.show()
```



Inference: The Target Variable seems to be be highly skewed, with most datapoints lieing near 0.

```
#Visualising the numeric features

print('\033[1mNumeric Features Distribution'.center(100))

n=5

plt.figure(figsize=[15,5*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    sns.distplot(df[nf[i]],hist_kws=dict(edgecolor="black", linewidth=2), bins=10, color=list(np.ran
dom.randint([255,255,255])/255))
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,5*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    df.boxplot(nf[i])
plt.tight_layout()
plt.show()
```

Numeric Features Distribution



Inference: There seem to be some outliers. let us fix these in the upcoming section...

3. Data Preprocessing

```
#Removal of any Duplicate rows (if any)

counter = 0
rs,cs = original_df.shape

df.drop_duplicates(inplace=True)
df.drop(['pickup_latitude','pickup_longitude',
         'dropoff_latitude','dropoff_longitude'],axis=1)

if df.shape==(rs,cs):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped/fixed ---> {rs-df.shape[0]}')
```

Inference: Number of duplicates dropped/fixed ---> 109

```
Dummy Encoding on features:
Monthly_Quarter
Hourly_Segments
year
weekday
passenger_count


(199878, 33)
```

```
#Converting categorical Columns to Numeric

df1 = df.copy()
df3 = df1.copy()

ecc = nvc[nvc['Percentage']!=0].index.values
fcc = [i for i in cf if i not in ecc]
#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    #print(i)
    if df3[i].nunique()==2:
        if oh==True: print("\033[1mOne-Hot Encoding on features:\033[0m")
        print(i);oh=False
        df3[i]=pd.get_dummies(df3[i], drop_first=True, prefix=str(i))
    if (df3[i].nunique()>2 and df3[i].nunique()<17):
        if dm==True: print("\n\033[1mDummy Encoding on features:\033[0m")
        print(i);dm=False
        df3 = pd.concat([df3.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df3[i], drop_first=True,
prefix=str(i)))],axis=1)

df3.shape
```

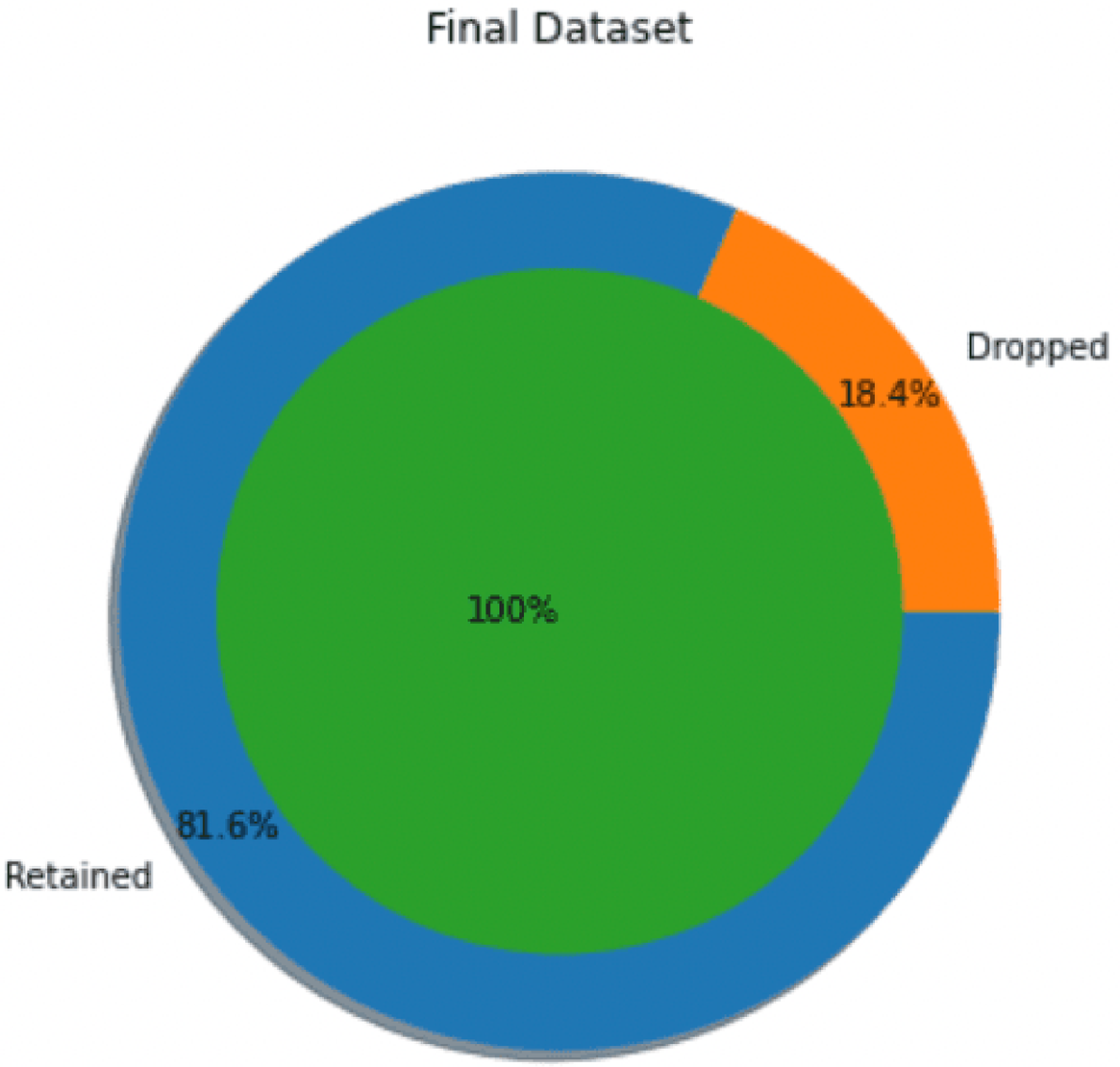
```
#Removal of outlier:

df1 = df3.copy()

#features1 = [i for i in features if i not in ['CHAS','RAD']]
features1 = nf

for i in features1:
    Q1 = df1[i].quantile(0.25)
    Q3 = df1[i].quantile(0.75)
    IQR = Q3 - Q1
    df1 = df1[df1[i] <= (Q3+(1.5*IQR))]
    df1 = df1[df1[i] >= (Q1-(1.5*IQR))]
    df1 = df1.reset_index(drop=True)
display(df1.head())
print('\n\033[1mInference:\033[0m\nBefore removal of outliers, The dataset had {} samples.'.format(df3.shape[0]))
print('After removal of outliers, The dataset now has {} samples.'.format(df1.shape[0]))
```

fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	Distance	Monthly_Quarter_Q2	Monthly_Quarter_Q
7.5	-73.999817	40.738354	-73.999512	40.723217	1681.11	1	0
7.7	-73.994355	40.728225	-73.994710	40.750325	2454.36	0	1
12.9	-74.005043	40.740770	-73.962565	40.772647	5039.60	0	1
5.3	-73.976124	40.790844	-73.965316	40.803349	1661.44	1	0
4.9	-73.969019	40.755910	-73.969019	40.755910	0.00	0	0

Inference:

Before removal of outliers, The dataset had 199878 samples.

After removal of outliers, The dataset now has 163203 samples.


Final Dataset
Dropped
18.4%
100%
81.6%
Retained

4. Data Manipulation

```
#Splitting the data intro training & testing sets

m=[]
for i in df.columns.values:
    m.append(i.replace(' ','_'))

df.columns = m
X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_stat
e=100)
Train_X.reset_index(drop=True,inplace=True)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',Train_X.shape,Train_Y.shape,'\nT
esting set   ---> ', Test_X.shape,'', Test_Y.shape)
```

```
Original set  --->  (163203, 32) (163203,)
Training set  --->  (130562, 32) (130562,)
Testing set   --->  (32641, 32)  (32641,)
```

```
#Feature Scaling (Standardization)

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(100))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(100))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

Standardardization on Training set

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	Distance	Monthly_Quarter_Q2	Monthly_Quarter
count	1.305620e+05	1.305620e+05	1.305620e+05	1.305620e+05	1.305620e+05	1.305620e+05	1.305620e+05
mean	-6.304892e-13	-2.335703e-15	-1.255051e-13	1.803562e-14	-1.397349e-16	-5.364534e-15	1.615100e-15
std	1.000004e+00	1.000004e+00	1.000004e+00	1.000004e+00	1.000004e+00	1.000004e+00	1.000004e+00
min	-2.961437e+00	-2.897556e+00	-2.919757e+00	-2.867446e+00	-1.622863e+00	-6.155085e-01	-5.316226e-01
25%	-6.783829e-01	-6.847366e-01	-6.710313e-01	-6.636110e-01	-7.629052e-01	-6.155085e-01	-5.316226e-01
50%	-6.341797e-02	3.271920e-02	-6.532058e-02	4.834418e-02	-2.347775e-01	-6.155085e-01	-5.316226e-01
75%	6.429738e-01	6.539807e-01	6.174684e-01	6.350862e-01	5.652279e-01	1.624673e+00	-5.316226e-01
max	3.255964e+00	2.807711e+00	3.137233e+00	2.805374e+00	2.933357e+00	1.624673e+00	1.881034e+00

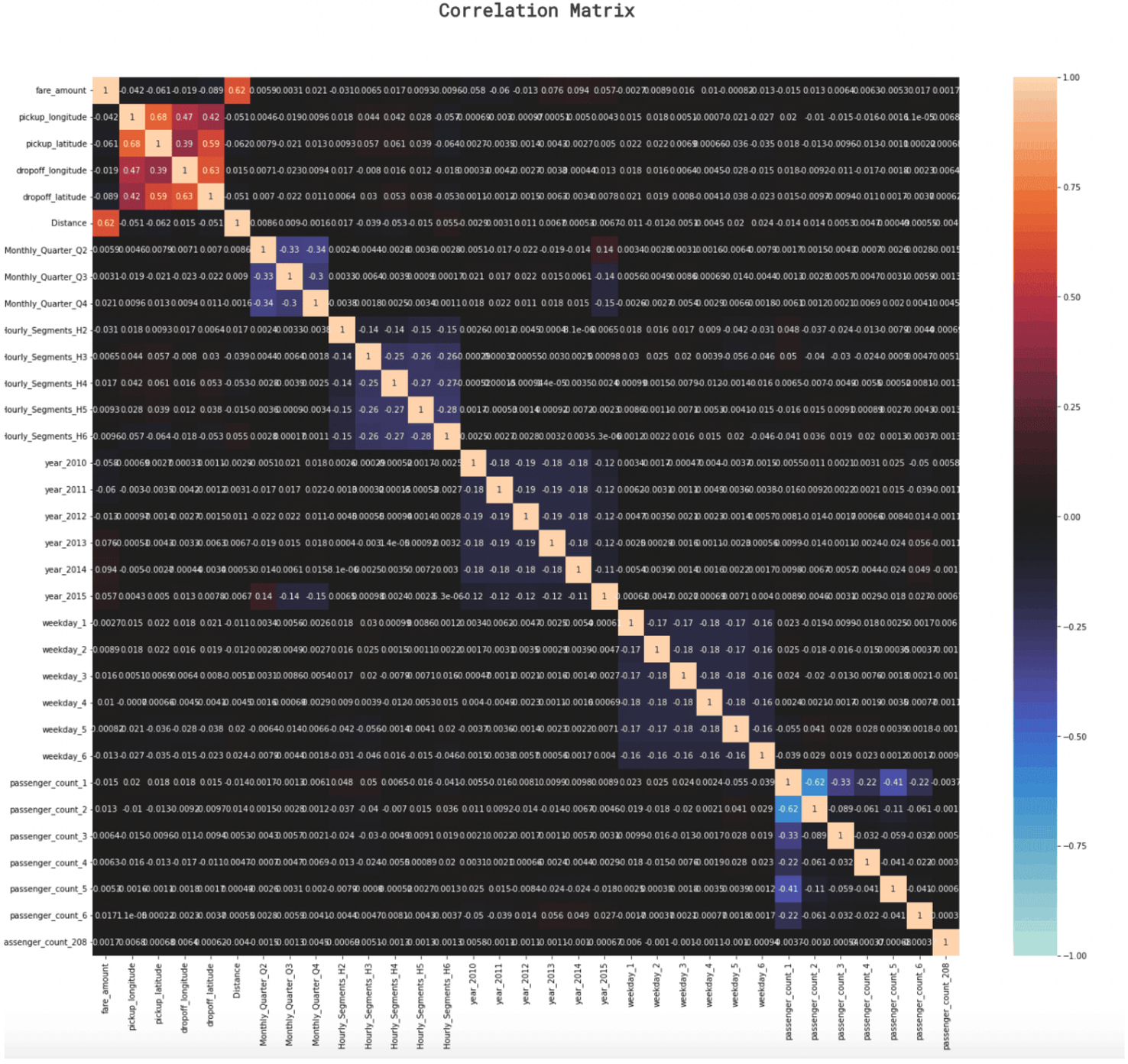
8 rows × 32 columns

Standardardization on Testing set

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	Distance	Monthly_Quarter_Q2	Monthly_Quarter_Q
count	32641.000000	32641.000000	32641.000000	32641.000000	32641.000000	32641.000000	32641.000000
mean	0.009496	0.003957	0.002645	0.010718	0.003829	-0.004419	0.008474
std	1.002215	1.000929	0.993715	1.001776	1.001768	0.997773	1.005681
min	-2.708732	-2.887521	-2.876670	-2.857408	-1.622863	-0.615509	-0.531623
25%	-0.674309	-0.688962	-0.664415	-0.652657	-0.762707	-0.615509	-0.531623
50%	-0.048188	0.048564	-0.060663	0.060627	-0.230528	-0.615509	-0.531623
75%	0.653268	0.661489	0.621690	0.657956	0.574644	1.624673	-0.531623
max	3.252830	2.800652	3.132185	2.804325	2.932838	1.624673	1.881034

8 rows × 32 columns

5. Feature Selection/Extraction



Inference: There seems to be strong multi-correlation between the features. Let us try to fix these...

OLS Regression Results

Dep. Variable:	fare_amount	R-squared:	0.457
Model:	OLS	Adj. R-squared:	0.457
Method:	Least Squares	F-statistic:	3436.
Date:	Tue, 19 Apr 2022	Prob (F-statistic):	0.00
Time:	19:42:54	Log-Likelihood:	-3.3661e+05
No. Observations:	130562	AIC:	6.733e+05
Df Residuals:	130529	BIC:	6.736e+05
Df Model:	32		
Covariance Type:	nonrobust		

Omnibus:	204830.890	Durbin-Watson:	2.015
Prob(Omnibus):	0.000	Jarque-Bera (JB):	313047239.517
Skew:	9.705	Prob(JB):	0.00
Kurtosis:	242.098	Cond. No.	2.50e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.5e+07. This might indicate that there are strong multicollinearity or other numerical problems.

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1036.8864	85.753	12.092	0.000	868.812	1204.961
pickup_longitude	0.7547	0.802	0.941	0.347	-0.817	2.327
pickup_latitude	1.1043	0.663	1.665	0.096	-0.195	2.404
dropoff_longitude	4.4888	0.729	6.153	0.000	3.059	5.919
dropoff_latitude	-16.9776	0.604	-28.113	0.000	-18.161	-15.794
Distance	0.0021	6.6e-06	314.210	0.000	0.002	0.002
Monthly_Quarter_Q2	0.1479	0.024	6.174	0.000	0.101	0.195
Monthly_Quarter_Q3	0.3185	0.026	12.348	0.000	0.268	0.369
Monthly_Quarter_Q4	0.5197	0.025	20.525	0.000	0.470	0.569
Hourly_Segments_H2	-0.2480	0.045	-5.563	0.000	-0.335	-0.161
Hourly_Segments_H3	0.8033	0.036	22.239	0.000	0.733	0.874
Hourly_Segments_H4	0.9938	0.035	28.006	0.000	0.924	1.063
Hourly_Segments_H5	0.7011	0.035	20.017	0.000	0.632	0.770
Hourly_Segments_H6	0.0871	0.035	2.494	0.013	0.019	0.156
year_2010	0.1195	0.032	3.756	0.000	0.057	0.182
year_2011	0.0771	0.032	2.445	0.015	0.015	0.139
year_2012	0.5323	0.031	16.912	0.000	0.471	0.594
year_2013	1.4822	0.032	46.623	0.000	1.420	1.544
year_2014	1.7470	0.032	54.333	0.000	1.684	1.810
year_2015	1.9042	0.041	45.907	0.000	1.823	1.985
weekday_1	0.2489	0.034	7.300	0.000	0.182	0.316
weekday_2	0.3675	0.034	10.824	0.000	0.301	0.434
weekday_3	0.4200	0.034	12.465	0.000	0.354	0.486
weekday_4	0.3504	0.034	10.453	0.000	0.285	0.416
weekday_5	0.0497	0.034	1.464	0.143	-0.017	0.116
weekday_6	-0.1686	0.036	-4.732	0.000	-0.238	-0.099
passenger_count_1	0.2151	0.148	1.451	0.147	-0.075	0.506
passenger_count_2	0.3693	0.150	2.467	0.014	0.076	0.663
passenger_count_3	0.3946	0.154	2.569	0.010	0.094	0.696
passenger_count_4	0.4779	0.160	2.993	0.003	0.165	0.791
passenger_count_5	0.3229	0.152	2.130	0.033	0.026	0.620
passenger_count_6	0.2356	0.160	1.470	0.141	-0.078	0.550
passenger_count_208	7.7688	3.192	2.434	0.015	1.513	14.025

Inference: We can fix these multicollinearity with three techniques:

- Manual Method - Variance Inflation Factor (VIF)
- Automatic Method - Recursive Feature Elimination (RFE)
- Feature Elmination using PCA Decomposition

---> It can be seen that the performance of the models is quiet comparable unpon dropping features using VIF, RFE & PCA Techniques. Comparing the RMSE plots, the optimal values were found for dropping most features using manual RFE Technique.

```
#Shortlisting the selected Features (with RFE)

lm = LinearRegression()
rfe = RFE(lm,n_features_to_select=df.shape[1]-23)
rfe = rfe.fit(Train_X_std, Train_Y)

LR = LinearRegression()
LR.fit(Train_X_std.loc[:,rfe.support_], Train_Y)

#print(Train_X_std.loc[:,rfe.support_].columns)

pred1 = LR.predict(Train_X_std.loc[:,rfe.support_])
pred2 = LR.predict(Test_X_std.loc[:,rfe.support_])

print(np.sqrt(mean_squared_error(Train_Y, pred1)))
print(np.sqrt(mean_squared_error(Test_Y, pred2)))

Train_X_std = Train_X_std.loc[:,rfe.support_]
Test_X_std = Test_X_std.loc[:,rfe.support_]
```

```
3.198334765136796
4.1370260899811235
```

6. Predictive Modelling

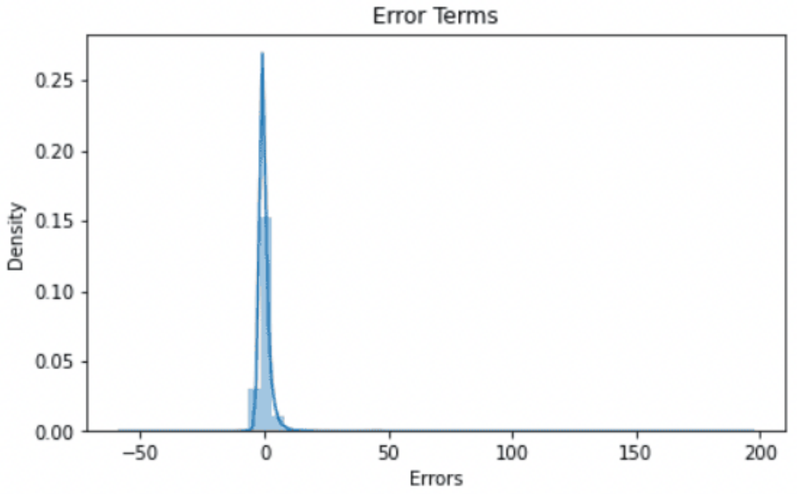
6a. Multiple Linear Regression(MLR)

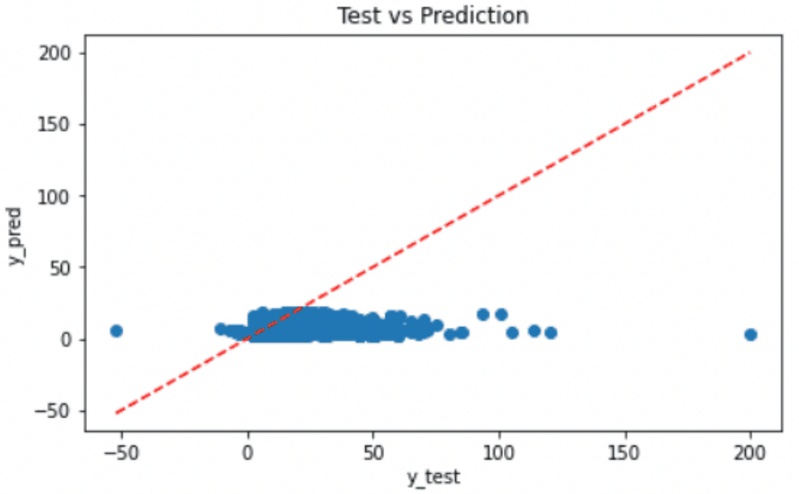
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ ... + \beta_i X_i$$

Y : Dependent variable
β_0 : Intercept
β_i : Slope for X_i
X = Independent variable

```
#Linear Regression

MLR = LinearRegression().fit(Train_X_std,Train_Y)
pred1 = MLR.predict(Train_X_std)
pred2 = MLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Multiple Linear Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25
,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(0, pred1, pred2)
```

```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.4535189574339896
Residual Sum of Squares (RSS) on Training set  ---> 1335563.7771264177
Mean Squared Error (MSE) on Training set       ---> 10.229345269882645
Root Mean Squared Error (RMSE) on Training set ---> 3.198334765136796

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.32820758401113337
Residual Sum of Squares (RSS) on Training set  ---> 558650.2211150513
Mean Squared Error (MSE) on Training set       ---> 17.1149848691845
Root Mean Squared Error (RMSE) on Training set ---> 4.1370260899811235

--------------------Residual Plots--------------------
```





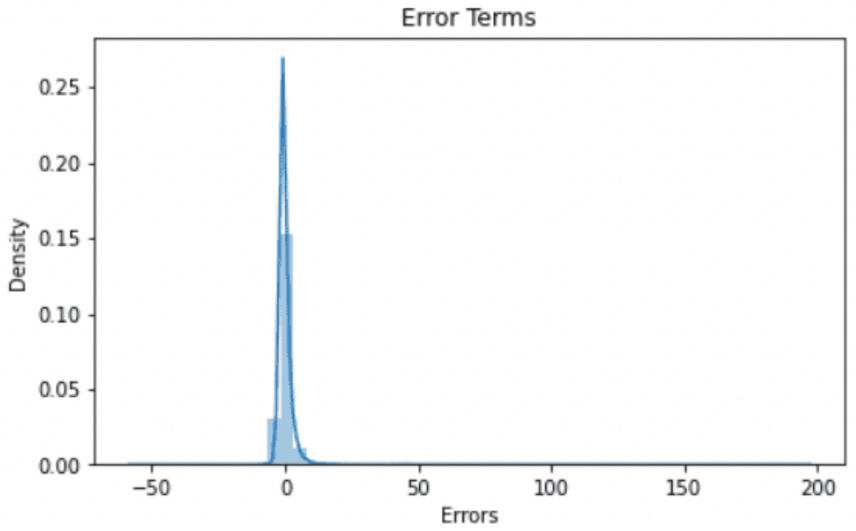
6b. Ridge Regression Model

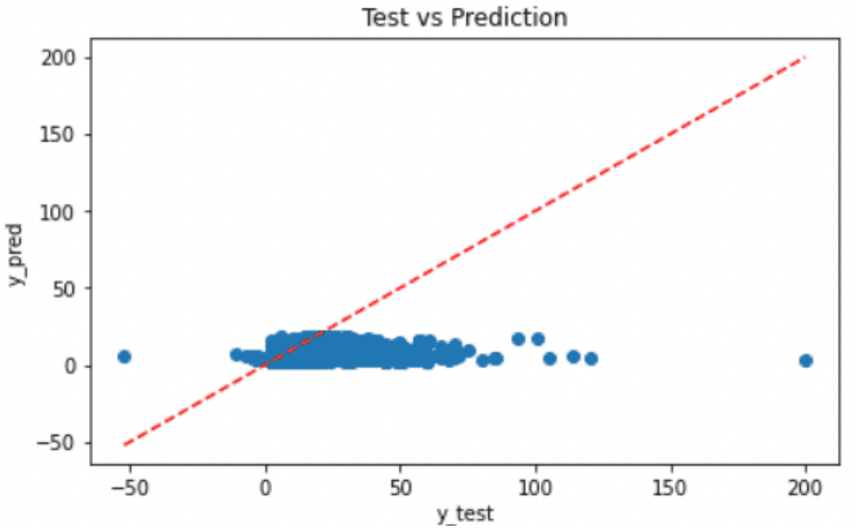
Ridge Formula: Sum of Error + Sum of the squares of coefficients

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum \beta^2$$

```
#Creating a Ridge Regression model

RLR = Ridge().fit(Train_X_std,Train_Y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Ridge Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*
3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(1, pred1, pred2)
```

```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.453518957400977
Residual Sum of Squares (RSS) on Training set  ---> 1335563.7772070984
Mean Squared Error (MSE) on Training set       ---> 10.229345270500593
Root Mean Squared Error (RMSE) on Training set ---> 3.1983347652334007

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.32820764693130466
Residual Sum of Squares (RSS) on Training set  ---> 558650.1687917938
Mean Squared Error (MSE) on Training set       ---> 17.114983266192635
Root Mean Squared Error (RMSE) on Training set ---> 4.137025896243899

--------------------Residual Plots--------------------
```





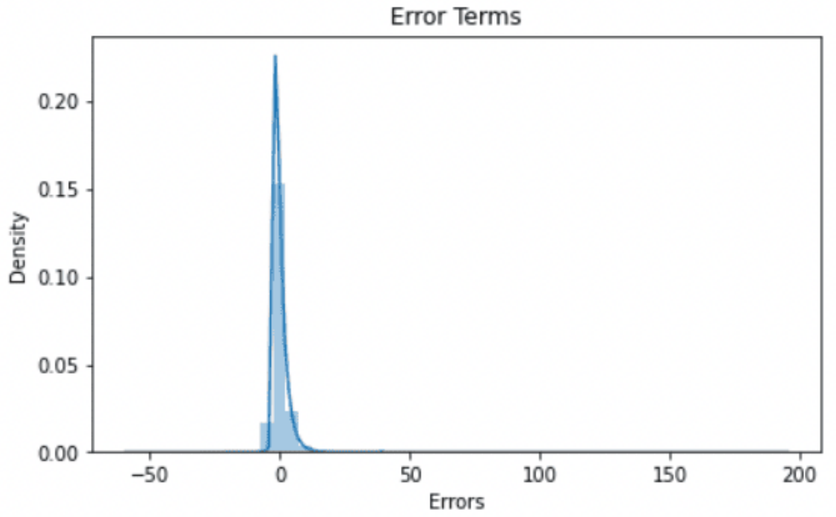
6c. Lasso Regression Model

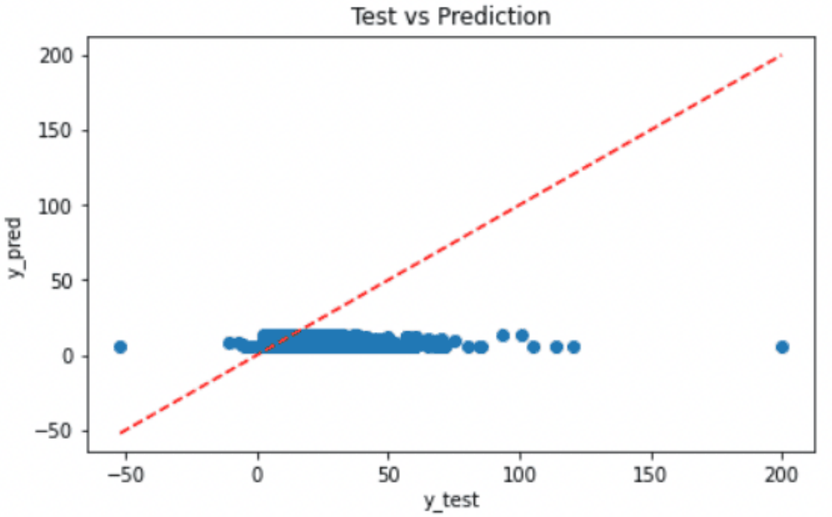
Lasso = Sum of Error + Sum of the absolute value of coefficients

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum |\beta|$$

```
#Creating a Ridge Regression model

LLR = Lasso().fit(Train_X_std,Train_Y)
pred1 = LLR.predict(Train_X_std)
pred2 = LLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Lasso Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*
3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(2, pred1, pred2)
```

```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.3585502318636643
Residual Sum of Squares (RSS) on Training set  ---> 1567661.1052167423
Mean Squared Error (MSE) on Training set       ---> 12.007024288971845
Root Mean Squared Error (RMSE) on Training set ---> 3.4651153355944513

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.2622605216343987
Residual Sum of Squares (RSS) on Training set  ---> 613490.5856410799
Mean Squared Error (MSE) on Training set       ---> 18.795091622226032
Root Mean Squared Error (RMSE) on Training set ---> 4.335330624326827

--------------------Residual Plots--------------------
```





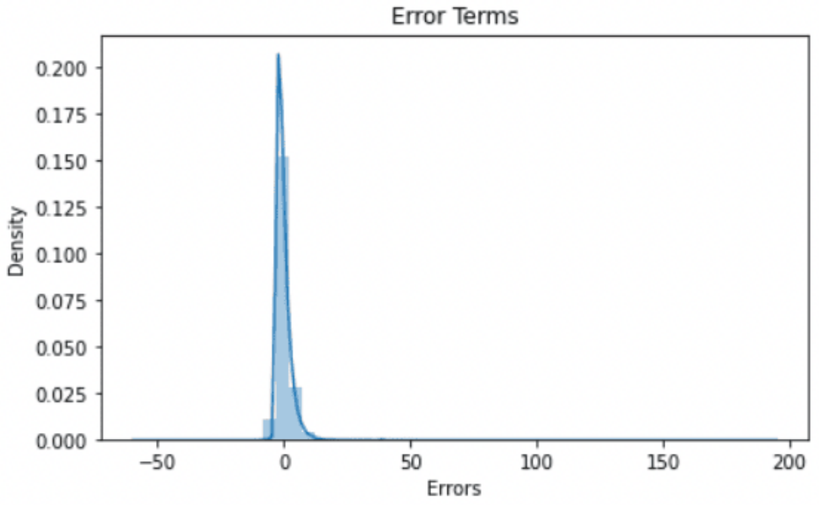
6d. Elastic-Net Regression

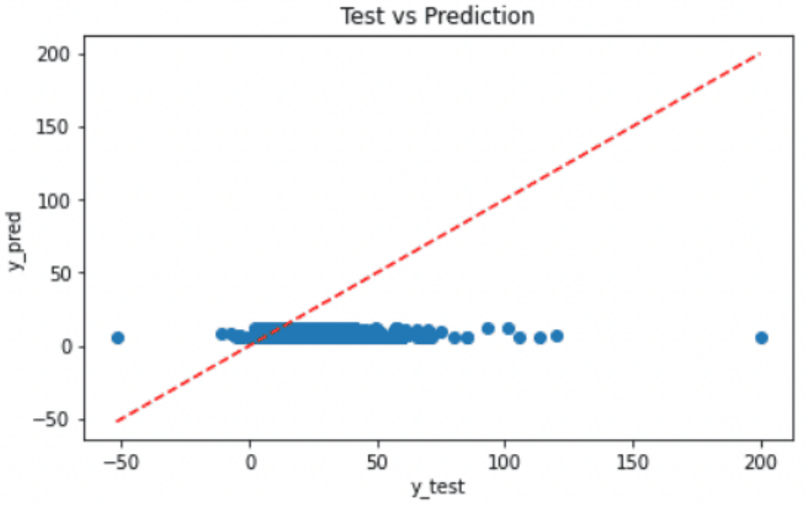
Elastic Net Formula: Ridge + Lasso

$$L = \sum(\hat{Y}i - Yi)^2 + \lambda \sum \beta^2 + \lambda \sum |\beta|$$

```
#Creating a ElasticNet Regression model

ENR = ElasticNet().fit(Train_X_std,Train_Y)
pred1 = ENR.predict(Train_X_std)
pred2 = ENR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Elastic-Net Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(3, pred1, pred2)
```

```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.32729497804639207
Residual Sum of Squares (RSS) on Training set  ---> 1644046.8928137545
Mean Squared Error (MSE) on Training set       ---> 12.5920780381256
Root Mean Squared Error (RMSE) on Training set ---> 3.5485318144446163

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.23952090035803764
Residual Sum of Squares (RSS) on Training set  ---> 632400.4365887307
Mean Squared Error (MSE) on Training set       ---> 19.374419796842336
Root Mean Squared Error (RMSE) on Training set ---> 4.401638308271402

--------------------Residual Plots--------------------
```



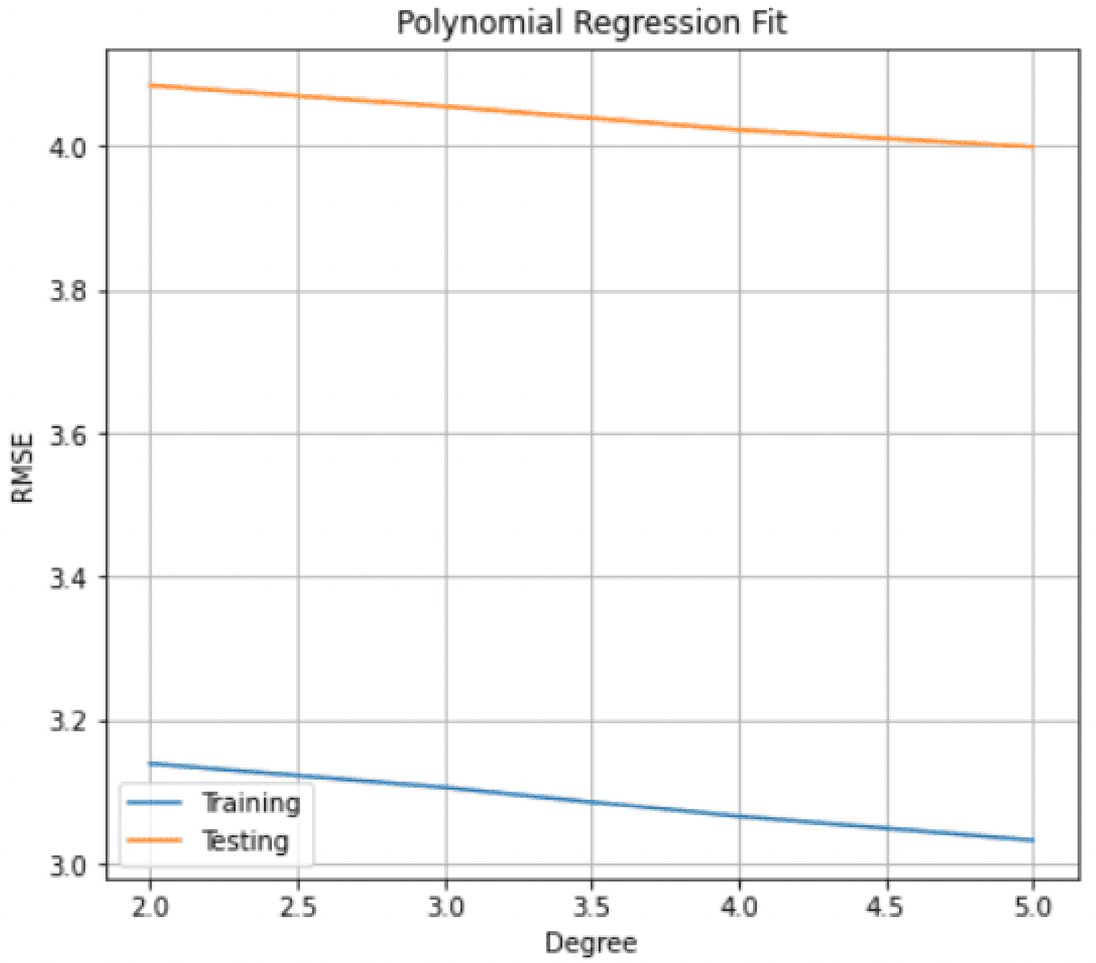


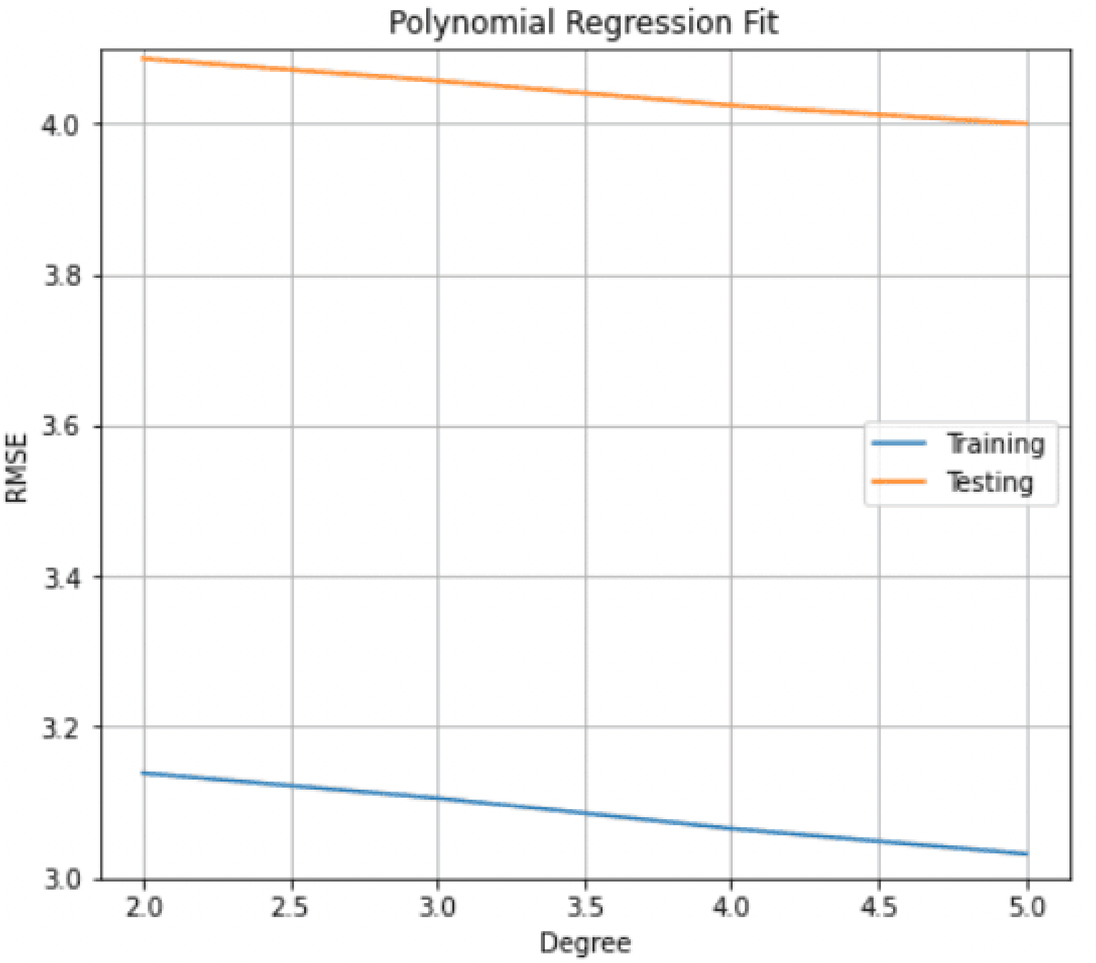
6e. Polynomial Regression Model

Polynomial Regression : Order-n

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

```
#Checking polynomial regression performance on various degrees

Trr=[]; Tss=[]
n_degree=6

for i in range(2,n_degree):
    #print(f'{i} Degree')
    poly_reg = PolynomialFeatures(degree=i)
    X_poly = poly_reg.fit_transform(Train_X_std)
    X_poly1 = poly_reg.fit_transform(Test_X_std)
    LR = LinearRegression()
    LR.fit(X_poly, Train_Y)

    pred1 = LR.predict(X_poly)
    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))

    pred2 = LR.predict(X_poly1)
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))
```

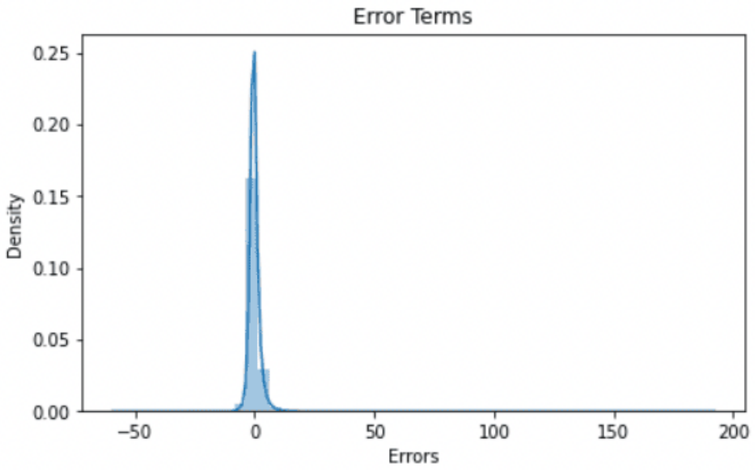


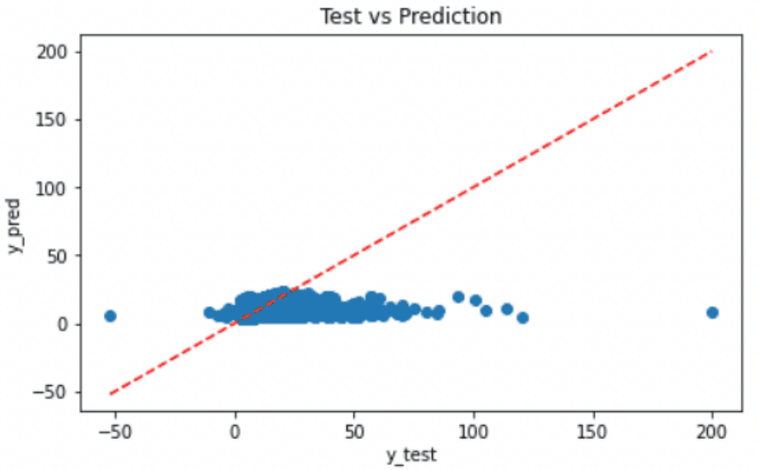


Inference: We can choose 5th order polynomial regression as it gives the optimal training & testing scores...

```
#Using the 5th Order Polynomial Regression model (degree=5)

poly_reg = PolynomialFeatures(degree=5)
X_poly = poly_reg.fit_transform(Train_X_std)
X_poly1 = poly_reg.fit_transform(Test_X_std)
PR = LinearRegression()
PR.fit(X_poly, Train_Y)

pred1 = PR.predict(X_poly)
pred2 = PR.predict(X_poly1)

print('{}{}\033[1m Evaluating Polynomial Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(4, pred1, pred2)
```

```
--------------------Training Set Metrics--------------------

R2-Score on Training set ---> 0.5088614966002526
Residual Sum of Squares (RSS) on Training set  ---> 1200310.2461025438
Mean Squared Error (MSE) on Training set       ---> 9.19341191236764
Root Mean Squared Error (RMSE) on Training set ---> 3.03206396904281

--------------------Testing Set Metrics--------------------

R2-Score on Testing set ---> 0.372092413105833
Residual Sum of Squares (RSS) on Training set  ---> 522156.4041354973
Mean Squared Error (MSE) on Training set       ---> 15.996948749593983
Root Mean Squared Error (RMSE) on Training set ---> 3.999618575513668

--------------------Residual Plots--------------------
```





6f. Comparing the Evaluation Metics of the Models

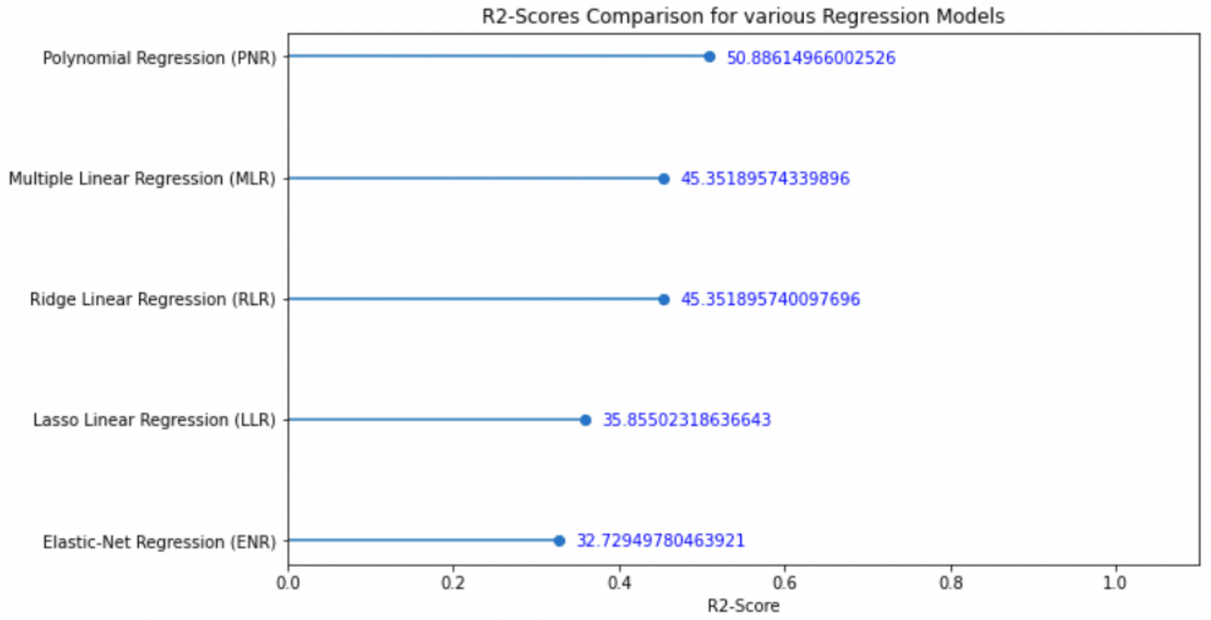
:

```
# Regression Models Results Evaluation

EMC = Model_Evaluation_Comparison_Matrix.copy()
EMC.index = ['Multiple Linear Regression (MLR)','Ridge Linear Regression (RLR)','Lasso Linear Regres
sion (LLR)','Elastic-Net Regression (ENR)','Polynomial Regression (PNR)']
EMC
```

:

	Train-R2	Test-R2	Train-RSS	Test-RSS	Train-MSE	Test-MSE	Train-RMSE	Test-RMSE
Multiple Linear Regression (MLR)	0.453519	0.328208	1.335564e+06	558650.221115	10.229345	17.114985	3.198335	4.137026
Ridge Linear Regression (RLR)	0.453519	0.328208	1.335564e+06	558650.168792	10.229345	17.114983	3.198335	4.137026
Lasso Linear Regression (LLR)	0.358550	0.262261	1.567661e+06	613490.585641	12.007024	18.795092	3.465115	4.335331
Elastic-Net Regression (ENR)	0.327295	0.239521	1.644047e+06	632400.436589	12.592078	19.374420	3.548532	4.401638
Polynomial Regression (PNR)	0.508861	0.372092	1.200310e+06	522156.404135	9.193412	15.996949	3.032064	3.999619

```
# R2-Scores Comparison for different Regression Models

R2 = EMC['Train-R2'].sort_values(ascending=True)
plt.hlines(y=R2.index, xmin=0, xmax=R2.values)
plt.plot(R2.values, R2.index,'o')
plt.title('R2-Scores Comparison for various Regression Models')
plt.xlabel('R2-Score')
#plt.ylabel('Regression Models')
for i, v in enumerate(R2):
    plt.text(v+0.02, i-0.05, str(v*100), color='blue')
plt.xlim([0,1.1])
plt.show()
```



Inference: From the above plot, it is clear that the polynomial regresion models have the highest explainability power to understand the dataset.

7. Project Outcomes & Conclusions

Here are some of the key outcomes of the project:

- The Dataset was large enough totalling 200,000 samples & after preprocessing 18.4% of the datasamples were dropped.
- Visualising the distribution of data & their relationships, helped us to get some insights on the feature-set.
- The features had high multicollinearity, hence in Feature Extraction step, we shortlisted the appropriate features with REF Technique.
- Testing multiple algorithms with default hyperparamters gave us some understanding for various models performance on this specific dataset.
- While, Polynomial Regression (Order-5) was the best choise, yet it is safe to use multiple regression algorithm, as their scores were quiet comparable & also they're more generalisable.

Thank for listening!