UBER Fare Prediction

The initiative is focused on Uber Inc., the largest taxi firm in the world. We want to forecast the cost for their upcoming transactional scenarios in this project. Every day, Uber serves thousands of consumers. It is now crucial for them to handle their data correctly in order to generate fresh business concepts that will yield the best outcomes. Over time, it becomes crucial to precisely anticipate the travel pricing.

In [ ]:

```python
import os
import math
import scipy
import requests
import numpy as np
import pandas as pd
import seaborn as sns
import datetime as dt
import geopy.distance
import urllib.request
from tqdm import tqdm
from IPython.display import display
from PIL import Image

from statsmodels.formula import api
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

import matplotlib
import matplotlib.image as img
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

import warnings
warnings.filterwarnings('ignore')
```

The following fields are included in the datset:

key: a special code that uniquely identifies each trip
fare_amount: the cost of each journey in US dollars
pickup_datetime: the time the metre was activated
passenger_count (driver entered value): the number of passengers in the car
pickup_longitude - the longitude where the metre was engaged
pickup_latitude - the latitude where the metre was engaged

dropoff_longitude - the longitude where the metre was disengaged
dropoff_latitude - the latitude where the metre was disengaged

Objective:

Recognise the Dataset and perform any necessary cleanup.
Create regression models to estimate the cost of an Uber ride.
Additionally, assess the models and contrast their individual scores, such as R2, RMSE, etc.

By developing a strategy, I want to address the problem statement. The following are some essential steps:

Data Exploration
Exploratory Data Analysis (EDA)
Data Pre-processing
Data Manipulation
Feature Selection/Extraction
Predictive Modelling
Project Outcomes & Conclusion

1. Data Exploration

In [ ]:
```
df = pd.read_csv('uber.csv')

df.drop(['Unnamed: 0','key'], axis=1, inplace=True)
display(df.head())

target = 'fare_amount'
features = [i for i in df.columns if i not in [target]]

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} sample
```

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

Inference: The Datset consists of 7 features & 200000 samples.

In [ ]:
```
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
```

In [ ]:
```
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',
                                      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
                                     9:'H3',10:'H3',11:'H3',12:'H4',13:'H4',14:'
                                     17:'H5',18:'H5',19:'H5',20:'H6',21:'H6',22:

df['Distance']=[round(geopy.distance.distance((df.pickup_latitude[i], df.pickup_

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

original_df = df.copy(deep=True)

df.head()
```

Out[ ]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199987 entries, 0 to 199999
Data columns (total 11 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        199987 non-null  float64
 1   pickup_longitude   199987 non-null  float64
 2   pickup_latitude    199987 non-null  float64
 3   dropoff_longitude  199987 non-null  float64
 4   dropoff_latitude   199987 non-null  float64
 5   passenger_count    199987 non-null  int64  
 6   year               199987 non-null  int64  
 7   weekday            199987 non-null  int64  
 8   Monthly_Quarter    199987 non-null  object 
 9   Hourly_Segments    199987 non-null  object 
 10  Distance           199987 non-null  float64
dtypes: float64(6), int64(3), object(2)
memory usage: 22.3+ MB
```

In [ ]:
```
df.nunique().sort_values()
```

Out[ ]:
```
Monthly_Quarter           4
Hourly_Segments           6
year                      7
weekday                   7
passenger_count           8
fare_amount            1244
pickup_longitude      71055
dropoff_longitude     76890
pickup_latitude       83831
dropoff_latitude      90582
Distance             164542
dtype: int64
```

In [ ]:
```
nu = df.drop([target], axis=1).nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df.drop([target], axis=1).shape[1]):
    if nu.values[i]<=24:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical f
```

Inference: The Datset has 5 numerical & 5 categorical features.

In [ ]:
```
display(df.describe())
```

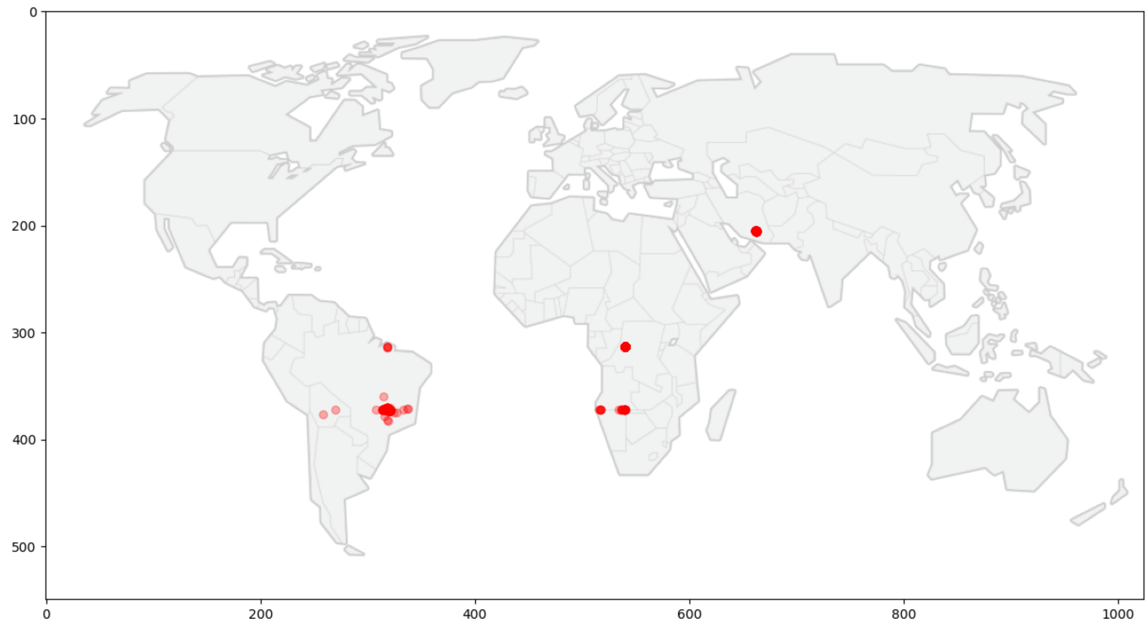
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	p
count	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	
mean	11.359849	-72.501786	39.917937	-72.511608	39.922031	
std	9.901868	10.449955	6.130412	10.412192	6.117669	
min	-52.000000	-93.824668	-74.015515	-75.458979	-74.015750	
25%	6.000000	-73.992064	40.734793	-73.991407	40.733823	
50%	8.500000	-73.981822	40.752592	-73.980092	40.753042	
75%	12.500000	-73.967154	40.767157	-73.963658	40.768000	
max	499.000000	40.808425	48.018760	40.831932	45.031598	

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

2. Exploratory Data Analysis (EDA)

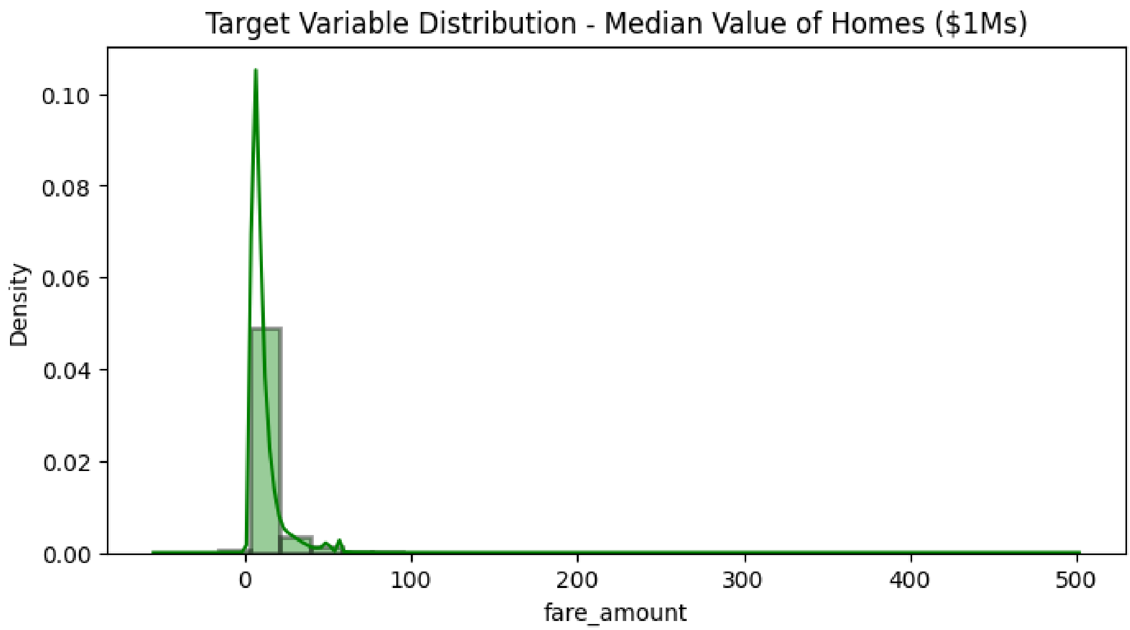
In [ ]:

```
plt.figure(figsize=[15,10])
url = "https://raw.githubusercontent.com/Masterx-AI/Project_Uber_Fare_Predictior
with urllib.request.urlopen(url) as url_obj:
    a = np.array(Image.open(url_obj))
plt.imshow(a, alpha=0.2)
plt.scatter( (df.pickup_longitude+180)*3,(df.pickup_latitude+215)*1.45555555,alp
#mdf.plot(kind='scatter',x='pickup_latitude',y='pickup_longitude',alpha=0.1)
plt.show()
```



In [ ]:

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



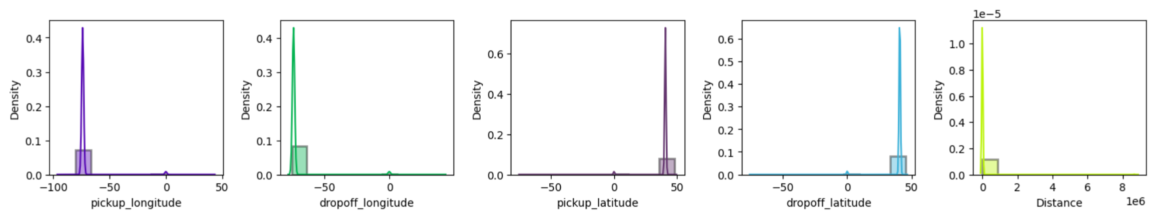
The Target Variable seems to be be highly skewed, with most datapoints lieing near 0.

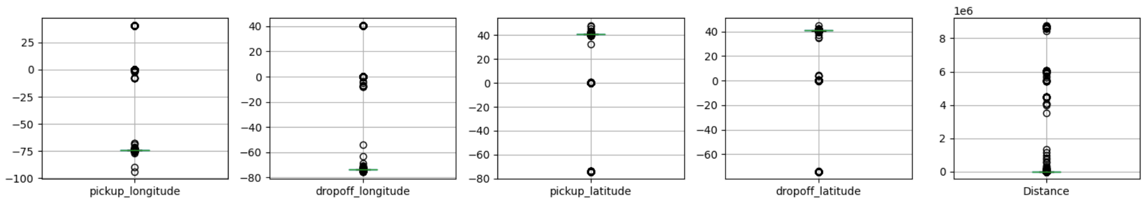
```
In [ ]: for col in cf:
            print(f"Column: {col}, Unique Values: {df[col].unique()}, Count of Unique Va
```

```
Column: Monthly_Quarter, Unique Values: ['Q2' 'Q3' 'Q1' 'Q4'], Count of Unique
Values: 4
Column: Hourly_Segments, Unique Values: ['H5' 'H6' 'H3' 'H1' 'H2' 'H4'], Count
of Unique Values: 6
Column: year, Unique Values: [2015 2009 2014 2011 2012 2010 2013], Count of Uni
que Values: 7
Column: weekday, Unique Values: [3 4 0 5 6 1 2], Count of Unique Values: 7
Column: passenger_count, Unique Values: [  1   3   5   2   4   6   0 208], Coun
t of Unique Values: 8
```

```
In [ ]: 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=1
        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





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

3. Data Preprocessing

In [ ]:
```
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 ---> {
```

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

In [ ]:
```
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[

df3.shape
```

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

Out[ ]:
```
(199878, 33)
```

In [ ]:
```
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 {
print('After removal of outliers, The dataset now has {} samples.'.format(df1.sh
```

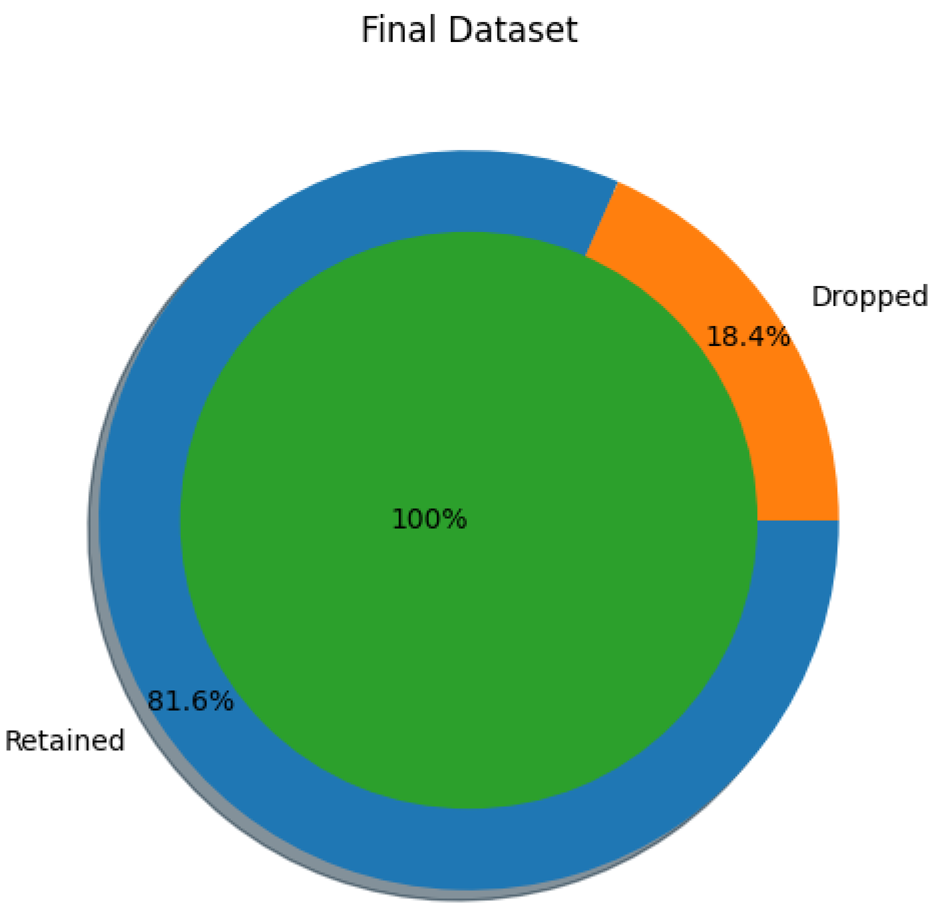
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	Distanc
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1681.1
1	7.7	-73.994355	40.728225	-73.994710	40.750325	2454.3
2	12.9	-74.005043	40.740770	-73.962565	40.772647	5039.6
3	5.3	-73.976124	40.790844	-73.965316	40.803349	1661.4
4	4.9	-73.969019	40.755910	-73.969019	40.755910	0.0

5 rows × 33 columns

```
Inference:
Before removal of outliers, The dataset had 199878 samples.
After removal of outliers, The dataset now has 163203 samples.
```

In [ ]:

```
df = df1.copy()
df.columns=[i.replace('-','_') for i in df.columns]

plt.title('Final Dataset')
plt.pie([df.shape[0], original_df.shape[0]-df.shape[0]], radius = 1, labels=['Re
        autopct='%1.1f%%', pctdistance=0.9, explode=[0,0], shadow=True)
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78)
plt.show()

print(f'\n\033[1mInference:\033[0m After the cleanup process, {original_df.shape
while retaining {round(100 - (df.shape[0]*100/(original_df.shape[0])),2)}% of th
```




```
Inference: After the cleanup process, 36784 samples were dropped, while retaini
ng 18.39% of the data.
```

4. Data Manipulation

In [ ]:
```
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_s
Train_X.reset_index(drop=True,inplace=True)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',Train_X.shap
```

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

In [ ]:
```
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	M
count	1.305620e+05	1.305620e+05	1.305620e+05	1.305620e+05	1.305620e+05	
mean	-6.304811e-13	-2.329963e-15	-1.254833e-13	1.805402e-14	-8.463960e-17	
std	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	
25%	-6.783829e-01	-6.847366e-01	-6.710313e-01	-6.636110e-01	-7.629052e-01	
50%	-6.341797e-02	3.271920e-02	-6.532058e-02	4.834418e-02	-2.347775e-01	
75%	6.429738e-01	6.539807e-01	6.174684e-01	6.350862e-01	5.652279e-01	
max	3.255964e+00	2.807711e+00	3.137233e+00	2.805374e+00	2.933357e+00	

8 rows × 32 columns

Standardardization on Testing set

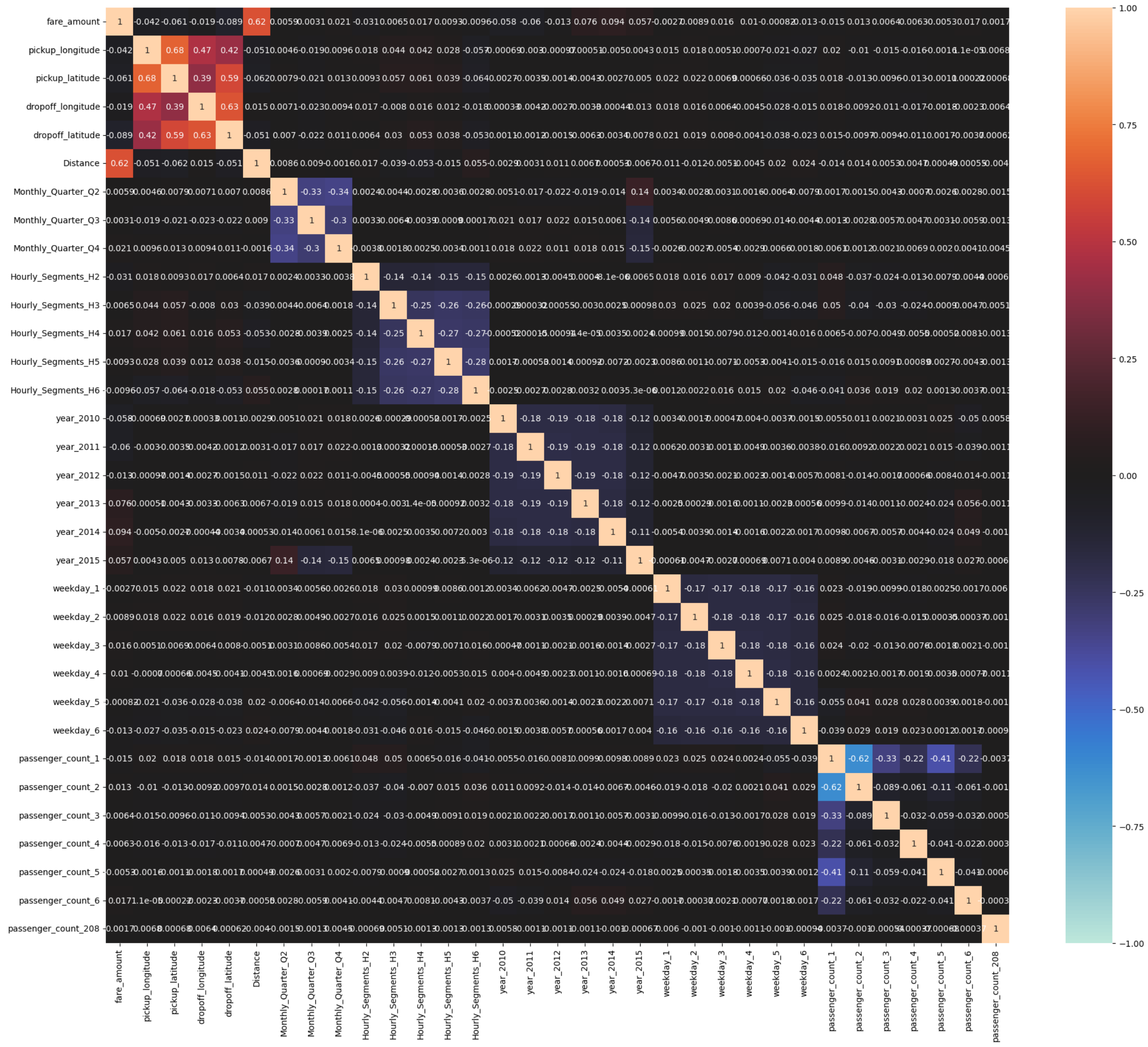
	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	Distance	M
count	32641.000000	32641.000000	32641.000000	32641.000000	32641.000000	
mean	0.009496	0.003957	0.002645	0.010718	0.003829	
std	1.002215	1.000929	0.993715	1.001776	1.001768	
min	-2.708732	-2.887521	-2.876670	-2.857408	-1.622863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

8 rows × 32 columns

5. Feature Selection/Extraction

In [ ]:
```
print('\033[1mCorrelation Matrix'.center(100))
plt.figure(figsize=[24,20])
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, center=0) #cmap='BuGn'
plt.show()
```

Correlation Matrix



In [ ]:
```
Train_xy = pd.concat([Train_X,Train_Y.reset_index(drop=True)],axis=1)
a = Train_xy.columns.values

API = api.ols(formula='{} ~ {}'.format(target,' + '.join(i for i in Train_X.colu
#print(API.conf_int())
#print(API.pvalues)
API.summary()
```

Out[ ]:

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:	Thu, 28 Sep 2023	**Prob (F-statistic):**	0.00
Time:	01:13:12	**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		

	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

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.

We can fix these multicollinearity with two techniques:

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

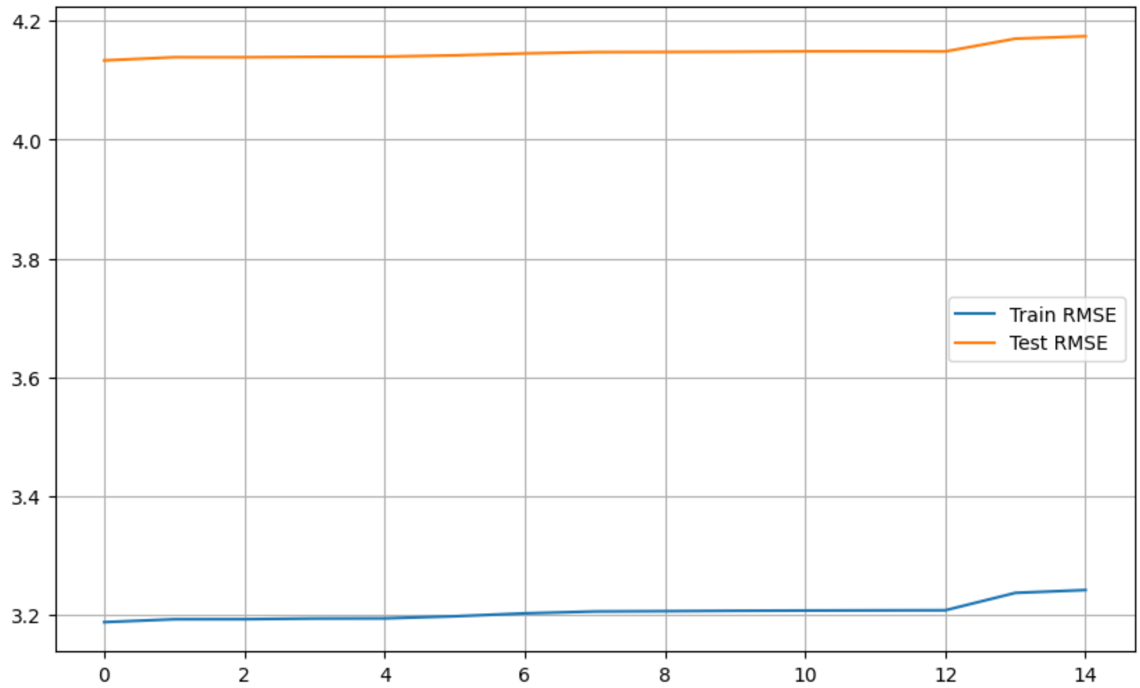
5a. Manual Method - VIF

In [ ]:

```
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
#Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
#Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

DROP=[];b=[]

for i in tqdm(range(len(Train_X_std.columns)-1)):
    vif = pd.DataFrame()
    X = Train_X_std.drop(DROP,axis=1)
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    if vif.loc[0][1]>=1.1:
        DROP.append(vif.loc[0][0])
        LR = LinearRegression()
        LR.fit(Train_X_std.drop(DROP,axis=1), Train_Y)
```

```
        pred1 = LR.predict(Train_X_std.drop(DROP,axis=1))
        pred2 = LR.predict(Test_X_std.drop(DROP,axis=1))

        Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
        Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

        #Trd.loc[i,'ord-'+str(k)] = round(np.sqrt(mean_squared_error(Train_Y, pr
        #Tsd.loc[i,'ord-'+str(k)] = round(np.sqrt(mean_squared_error(Test_Y, pre

print('Dropped Features --> ',DROP)
#plt.plot(b)
#plt.show()
#print(API.summary())

# plt.figure(figsize=[20,4])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max()
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```

```
100%|██████████| 31/31 [04:35<00:00,  8.90s/it]
Dropped Features -->  ['passenger_count_1', 'Hourly_Segments_H5', 'pickup_latit
ude', 'weekday_4', 'dropoff_longitude', 'year_2012', 'Monthly_Quarter_Q4', 'Hou
rly_Segments_H6', 'weekday_5', 'year_2011', 'pickup_longitude', 'weekday_1', 'M
onthly_Quarter_Q2', 'year_2013', 'Hourly_Segments_H3']
```

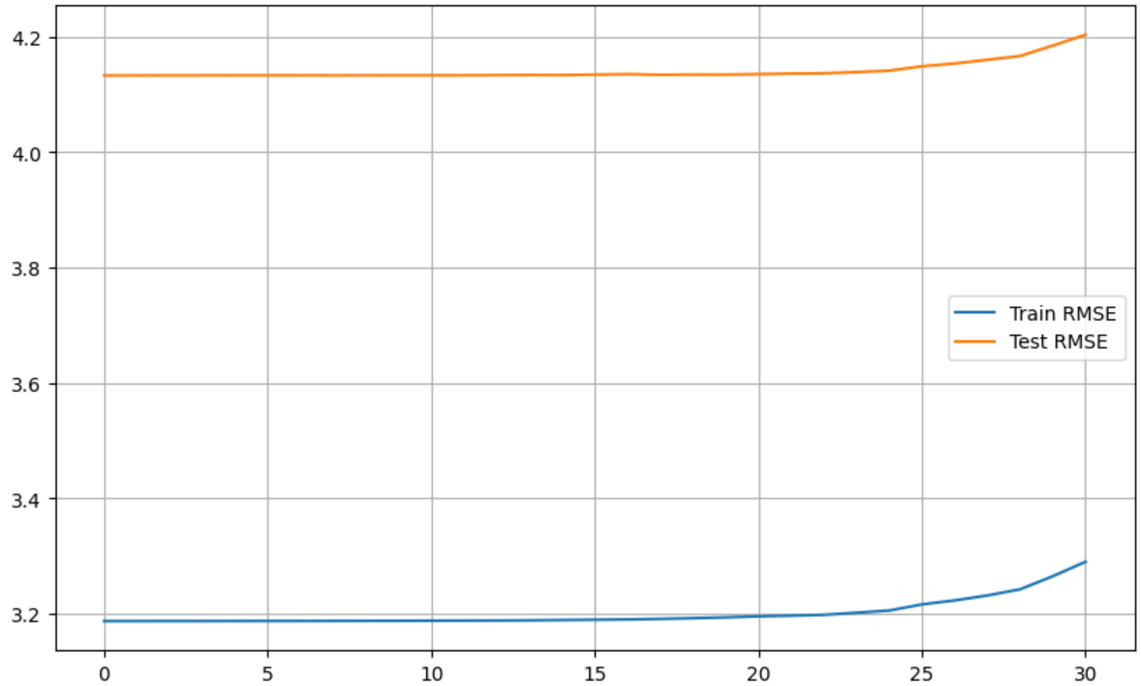


5b. Automatic Method - RFE

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

m=df.shape[1]-2
for i in tqdm(range(m)):
    lm = LinearRegression()
    #lm.fit(Train_X_std, Train_Y)

    rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-i)             # runr
    rfe = rfe.fit(Train_X_std, Train_Y)

    #print(Train_X_std.shape[1]-i)

    #Train_xy = pd.concat([Train_X_std[Train_X.columns[rfe.support_]],Train_Y.re
    #a = Train_xy.columns.values.tolist()
    #a.remove(target)

    #API = api.ols(formula='{} ~ {}'.format(target,' + '.join(i for i in a)), da
    #DROP.append(vif.loc[0][0])
    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_])

    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

# plt.figure(figsize=[20,4])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:,:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()
```

```
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max()
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```

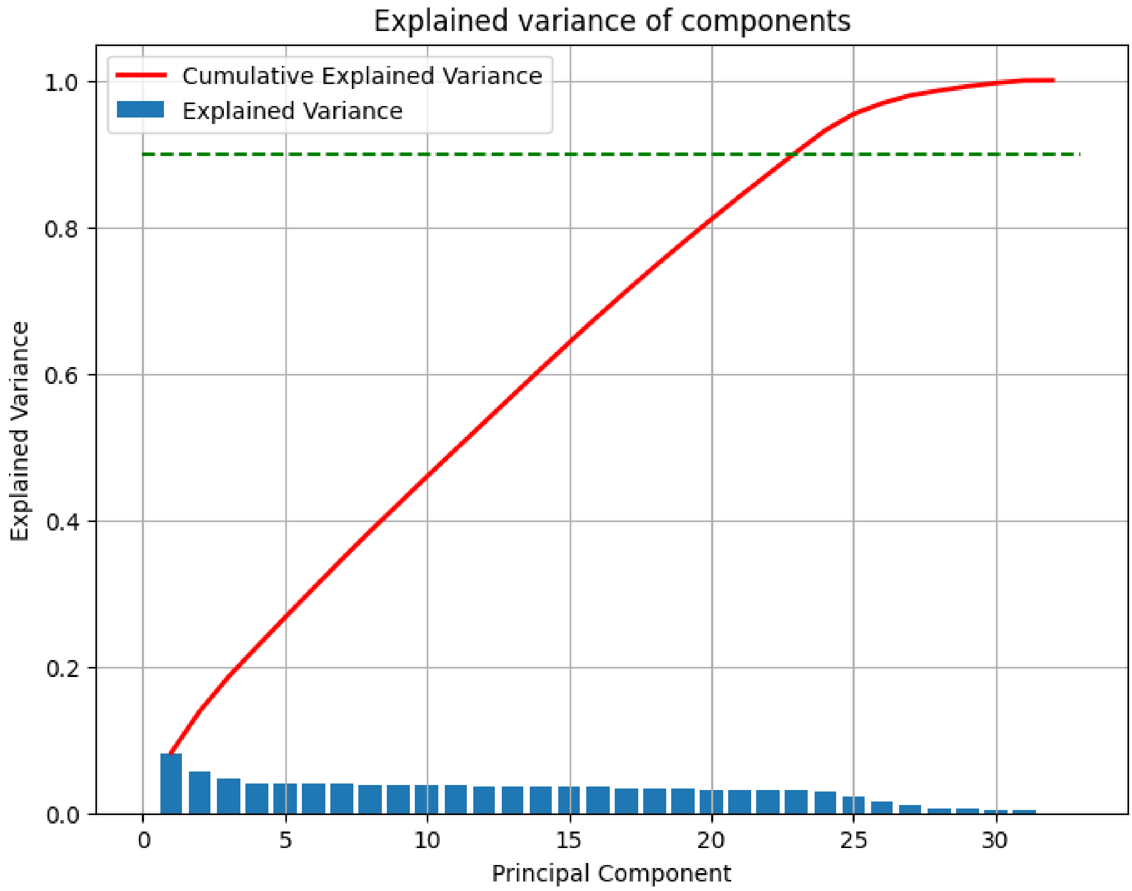
```
100%|██████████| 31/31 [01:43<00:00,  3.34s/it]
```



5c. Feature Elmination using PCA Decomposition

In [ ]:

```
from sklearn.decomposition import PCA

pca = PCA().fit(Train_X_std)

fig, ax = plt.subplots(figsize=(8,6))
x_values = range(1, pca.n_components_+1)
ax.bar(x_values, pca.explained_variance_ratio_, lw=2, label='Explained Variance'
ax.plot(x_values, np.cumsum(pca.explained_variance_ratio_), lw=2, label='Cumulat
plt.plot([0,pca.n_components_+1],[0.9,0.9],'g--')
ax.set_title('Explained variance of components')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Explained Variance')
plt.legend()
plt.grid()
plt.show()
```

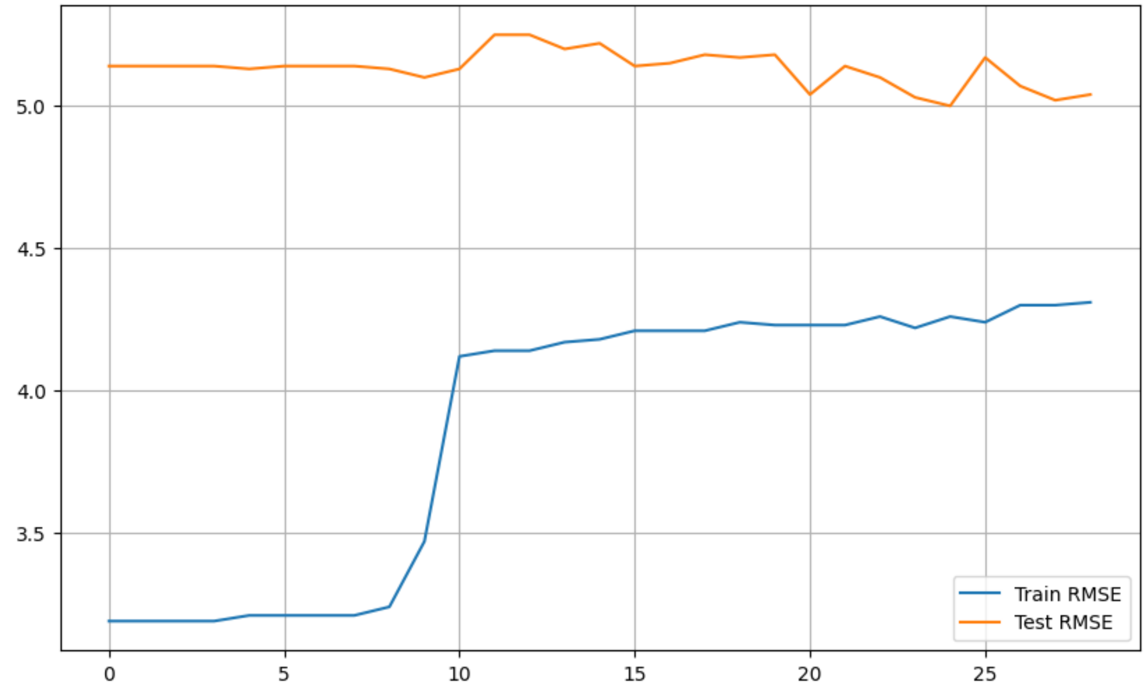


In [ ]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
m=df.shape[1]-4

for i in tqdm(range(m)):
    pca = PCA(n_components=Train_X_std.shape[1]-i)
    Train_X_std_pca = pca.fit_transform(Train_X_std)
    Test_X_std_pca = pca.fit_transform(Test_X_std)

    LR = LinearRegression()
    LR.fit(Train_X_std_pca, Train_Y)

    pred1 = LR.predict(Train_X_std_pca)
    pred2 = LR.predict(Test_X_std_pca)

    Trr.append(round(np.sqrt(mean_squared_error(Train_Y, pred1)),2))
    Tss.append(round(np.sqrt(mean_squared_error(Test_Y, pred2)),2))

# plt.figure(figsize=[20,4.5])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max()
# plt.title('Total RMSE')
```

```
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.5,20.75])
plt.legend()
plt.grid()
plt.show()
```

```
100%|██████████| 29/29 [00:33<00:00,  1.14s/it]
```



It is evident that the models' performance is quite equivalent when features are dropped using VIF, RFE, and PCA techniques. The ideal values for removing the majority of features using the manual RFE Technique were discovered by comparing the RMSE graphs.

In [ ]:

```
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_]
```

```
3.1983347651367957
4.137026089981123
```

6. Predictive Modelling

In [ ]:

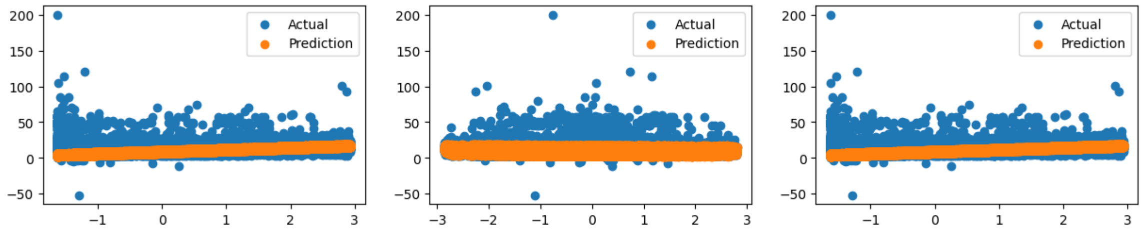
```
Model_Evaluation_Comparison_Matrix = pd.DataFrame(np.zeros([5,8]), columns=['Tra
                                                                              'Tra
rc=np.random.choice(Train_X_std.loc[:,Train_X_std.nunique()>50].columns,3)
def Evaluate(n, pred1,pred2):
    #Plotting predicted predicteds alongside the actual datapoints
    plt.figure(figsize=[15,6])
    for e,i in enumerate(rc):
        plt.subplot(2,3,e+1)
        plt.scatter(y=Train_Y, x=Train_X_std[i], label='Actual')
        plt.scatter(y=pred1, x=Train_X_std[i], label='Prediction')
        plt.legend()
    plt.show()

    #Evaluating the Multiple Linear Regression Model

    print('\n\n{}Training Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Training set --->',round(r2_score(Train_Y, pred1),20))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squar
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(me

    print('\n{}Testing Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Testing set --->',round(r2_score(Test_Y, pred2),20))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squar
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(me
    print('\n{}Residual Plots{}'.format('-'*20, '-'*20))

    Model_Evaluation_Comparison_Matrix.loc[n,'Train-R2']  = round(r2_score(Train
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-R2']   = round(r2_score(Test_
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RSS'] = round(np.sum(np.squa
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RSS']  = round(np.sum(np.squa
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-MSE'] = round(mean_squared_e
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-MSE']  = round(mean_squared_e
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RMSE']= round(np.sqrt(mean_s
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RMSE'] = round(np.sqrt(mean_s

    # Plotting y_test and y_pred to understand the spread.
    plt.figure(figsize=[15,4])

    plt.subplot(1,2,1)
    sns.distplot((Train_Y - pred1))
    plt.title('Error Terms')
    plt.xlabel('Errors')

    plt.subplot(1,2,2)
    plt.scatter(Train_Y,pred1)
    plt.plot([Train_Y.min(),Train_Y.max()],[Train_Y.min(),Train_Y.max()], 'r--')
    plt.title('Test vs Prediction')
    plt.xlabel('y_test')
    plt.ylabel('y_pred')
    plt.show()
```

Now let's attempt creating several regression models and comparing their assessment metrics to determine which model fits the training and testing sets the best.

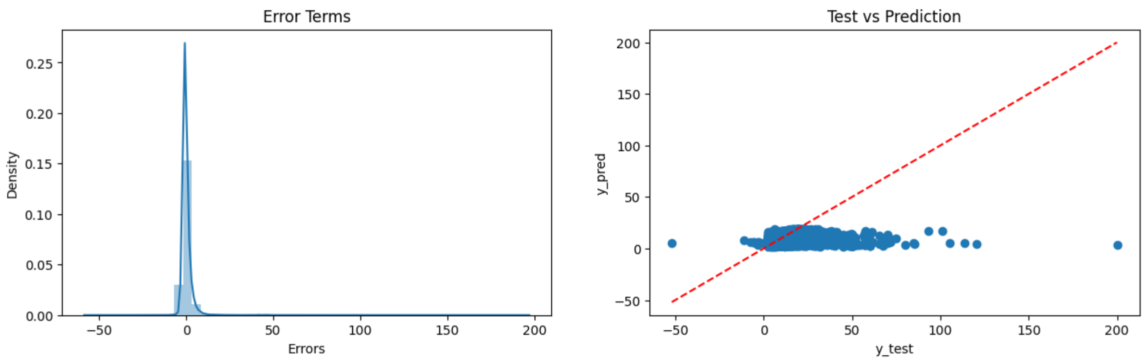
6a. Multiple Linear Regression(MLR)

In [ ]:
```
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'.fo
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)
```

```
<<<------------------------- Evaluating Multiple Linear Regression Model ------
------------------->>>

The Coeffecient of the Regresion Model was found to be  [-0.30499538  2.7970209
3  0.15508619  0.32086946  0.39552625  0.29085972
  0.16782406  0.50924681  0.59332707  0.44376597]
The Intercept of the Regresion Model was found to be  8.550616948269791
```




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

R2-Score on Training set ---> 0.4535189574339897
Residual Sum of Squares (RSS) on Training set  ---> 1335563.7771264175
Mean Squared Error (MSE) on Training set       ---> 10.229345269882643
Root Mean Squared Error (RMSE) on Training set ---> 3.1983347651367953

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

R2-Score on Testing set ---> 0.3282075840111336
Residual Sum of Squares (RSS) on Training set  ---> 558650.2211150512
Mean Squared Error (MSE) on Training set       ---> 17.114984869184497
Root Mean Squared Error (RMSE) on Training set ---> 4.137026089981123

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

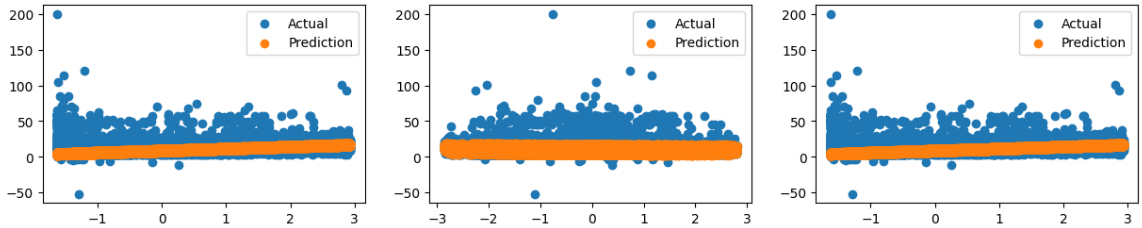


6b. Ridge Regression Model

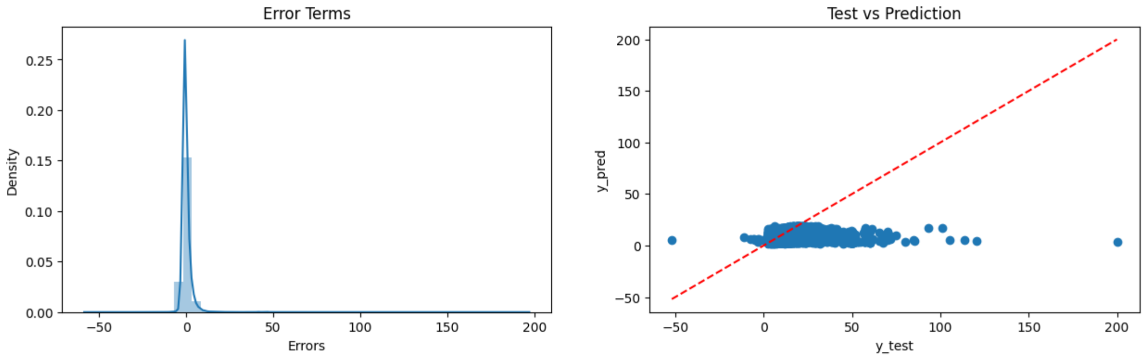
In [ ]:
```
RLR = Ridge().fit(Train_X_std,Train_Y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Ridge Regression Model \033[0m{}{}\n'.format('<'*3
```

```
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)
```

```
<<<------------------------- Evaluating Ridge Regression Model ----------------
--------->>>

The Coeffecient of the Regresion Model was found to be  [-0.30499538  2.7970209
3  0.15508619  0.32086946  0.39552625  0.29085972
  0.16782406  0.50924681  0.59332707  0.44376597]
The Intercept of the Regresion Model was found to be  8.550616948269791
```




```
--------------------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 ---> 0.3282076469313049
Residual Sum of Squares (RSS) on Training set  ---> 558650.1687917936
Mean Squared Error (MSE) on Training set       ---> 17.11498326619263
Root Mean Squared Error (RMSE) on Training set ---> 4.137025896243898

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

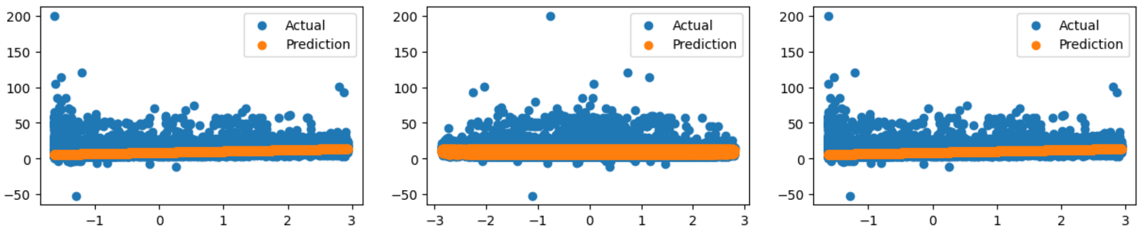


6c. Lasso Regression Model

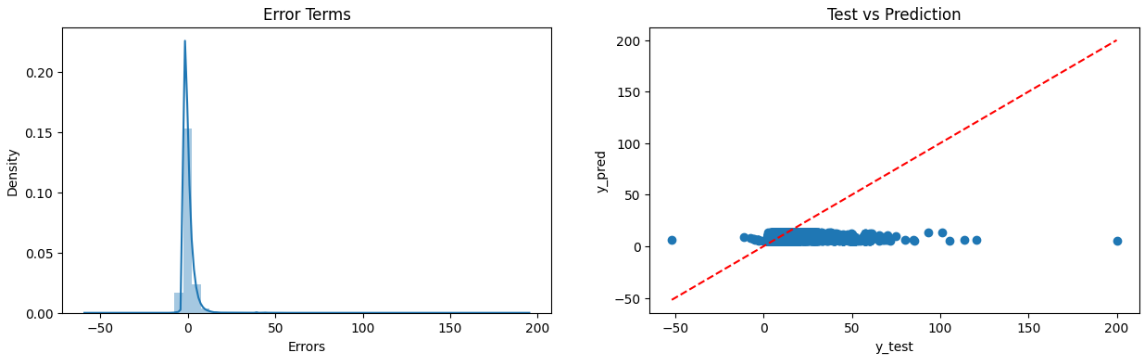
In [ ]:

```
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
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)
```

```
<<<------------------------- Evaluating Lasso Regression Model ----------------
--------->>>

The Coeffecient of the Regresion Model was found to be  [-0.30499538  2.7970209
3  0.15508619  0.32086946  0.39552625  0.29085972
  0.16782406  0.50924681  0.59332707  0.44376597]
The Intercept of the Regresion Model was found to be  8.550616948269791
```




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

R2-Score on Training set ---> 0.35855023186366475
Residual Sum of Squares (RSS) on Training set  ---> 1567661.105216741
Mean Squared Error (MSE) on Training set       ---> 12.007024288971838
Root Mean Squared Error (RMSE) on Training set ---> 3.46511533559445

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

R2-Score on Testing set ---> 0.26226052163439917
Residual Sum of Squares (RSS) on Training set  ---> 613490.5856410796
Mean Squared Error (MSE) on Training set       ---> 18.79509162222602
Root Mean Squared Error (RMSE) on Training set ---> 4.335330624326826

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

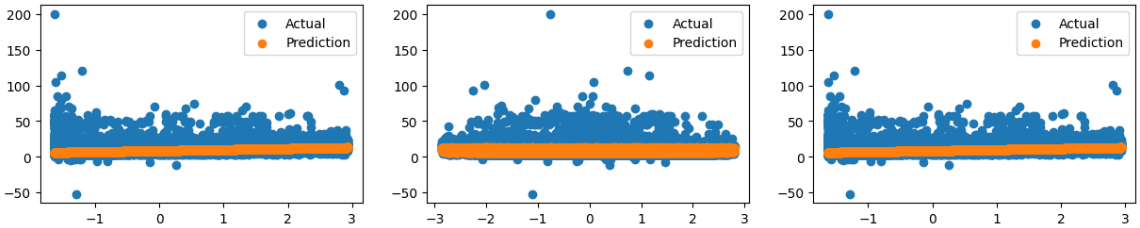


6d. Elastic-Net Regression

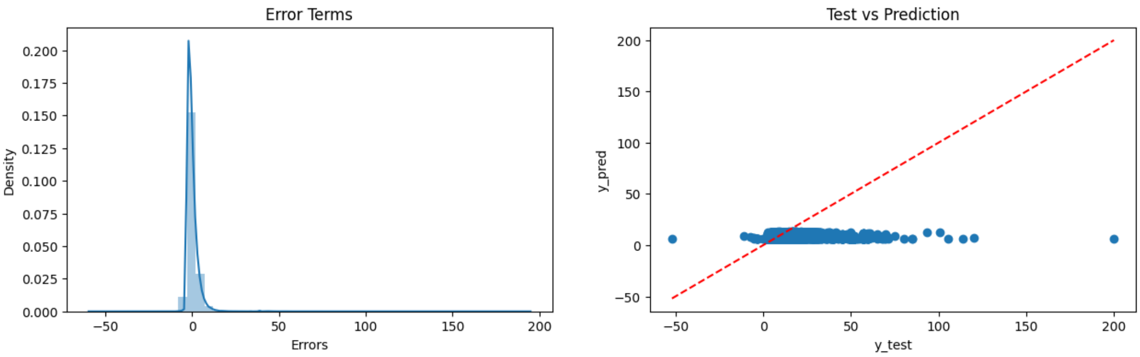
In [ ]:

```
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
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)
```

```
<<<------------------------ Evaluating Elastic-Net Regression Model ----------
---------------->>>

The Coeffecient of the Regresion Model was found to be  [-0.30499538  2.7970209
3  0.15508619  0.32086946  0.39552625  0.29085972
  0.16782406  0.50924681  0.59332707  0.44376597]
The Intercept of the Regresion Model was found to be  8.550616948269791
```




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

R2-Score on Training set ---> 0.3272949780463923
Residual Sum of Squares (RSS) on Training set  ---> 1644046.8928137538
Mean Squared Error (MSE) on Training set       ---> 12.592078038125594
Root Mean Squared Error (RMSE) on Training set ---> 3.5485318144446154

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

R2-Score on Testing set ---> 0.23952090035803794
Residual Sum of Squares (RSS) on Training set  ---> 632400.4365887304
Mean Squared Error (MSE) on Training set       ---> 19.37441979684233
Root Mean Squared Error (RMSE) on Training set ---> 4.4016383082714015

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

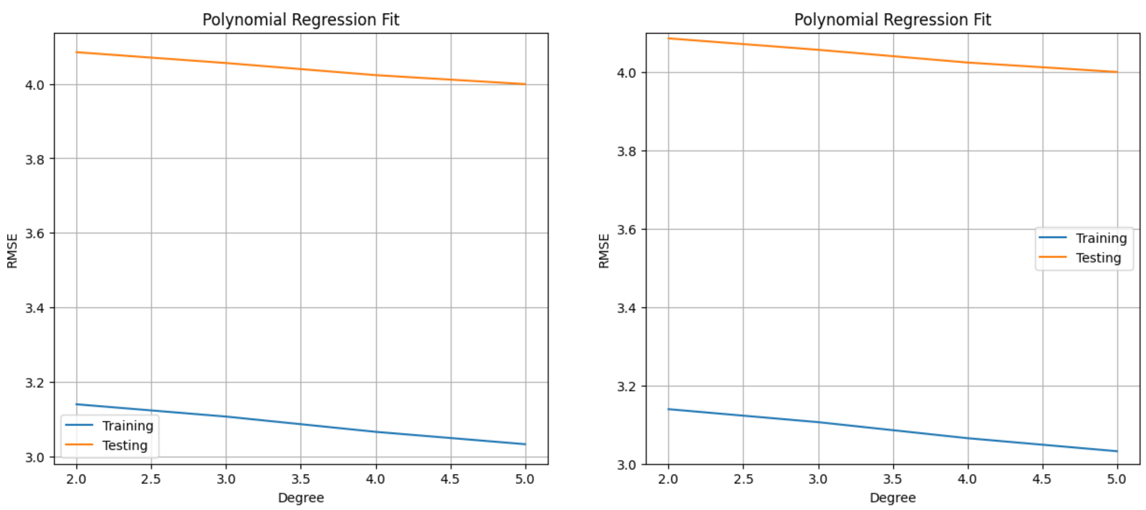


6e. Polynomial Regression Model

In [ ]:

```
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)
```

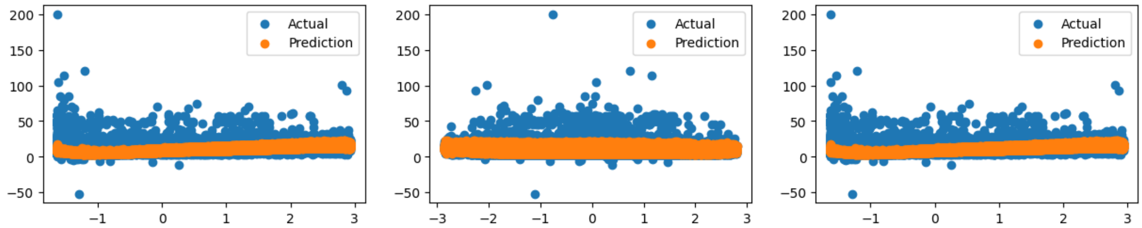
```
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

plt.figure(figsize=[15,6])
plt.subplot(1,2,1)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
#plt.plot([1,4],[1,4],'b--')
plt.title('Polynomial Regression Fit')
#plt.ylim([0,5])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()

plt.subplot(1,2,2)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
plt.title('Polynomial Regression Fit')
plt.ylim([3,4.1])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()
plt.show()
```



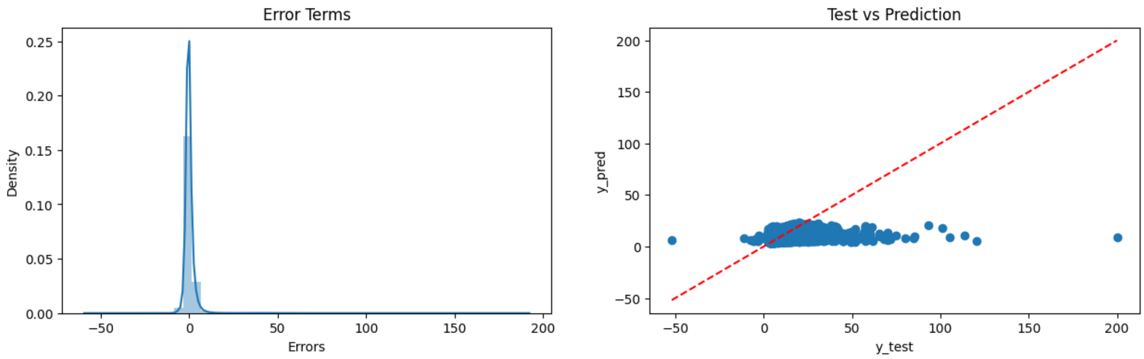
In [ ]:

```
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(
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)
```

```
<<<------------------------- Evaluating Polynomial Regression Model -----------
--------------->>>

The Coeffecient of the Regresion Model was found to be  [-0.30499538  2.7970209
3  0.15508619  0.32086946  0.39552625  0.29085972
  0.16782406  0.50924681  0.59332707  0.44376597]
The Intercept of the Regresion Model was found to be  8.550616948269791
```




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

R2-Score on Training set ---> 0.5088612277076756
Residual Sum of Squares (RSS) on Training set  ---> 1200310.903258342
Mean Squared Error (MSE) on Training set       ---> 9.193416945652963
Root Mean Squared Error (RMSE) on Training set ---> 3.0320647990524483

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

R2-Score on Testing set ---> 0.3720968157134218
Residual Sum of Squares (RSS) on Training set  ---> 522152.74300797563
Mean Squared Error (MSE) on Training set       ---> 15.996836586133258
Root Mean Squared Error (RMSE) on Training set ---> 3.999604553719437

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



6f. Comparing the Evaluation Metics of the Models

In [ ]:

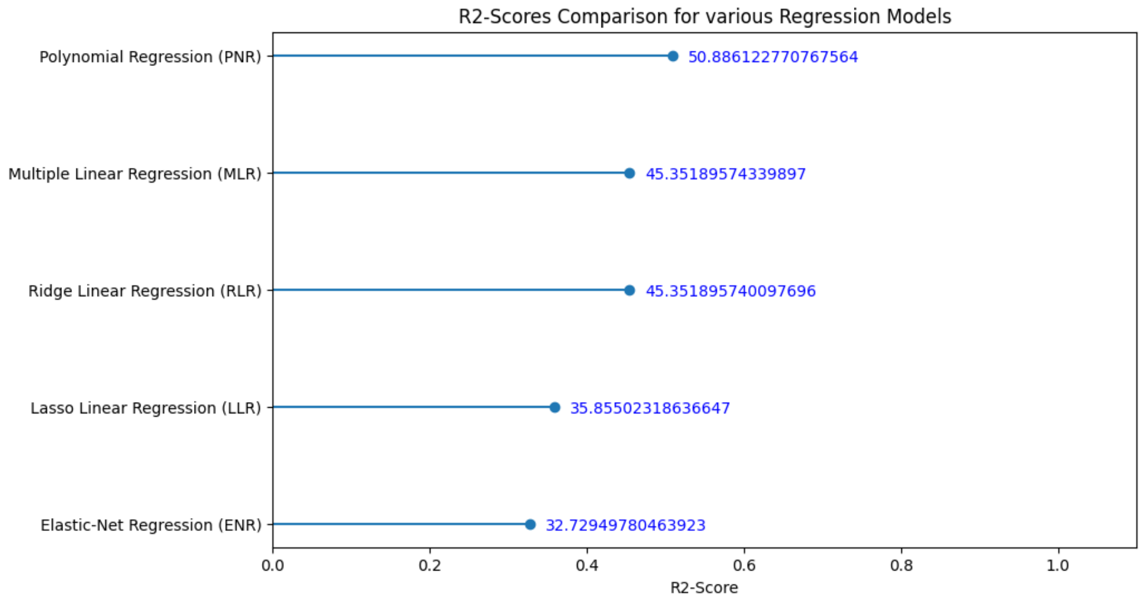
```
EMC = Model_Evaluation_Comparison_Matrix.copy()
EMC.index = ['Multiple Linear Regression (MLR)','Ridge Linear Regression (RLR)',
EMC
```

Out[ ]:

	Train-R2	Test-R2	Train-RSS	Test-RSS	Train-MSE	Test-MSE	Train-RMSE
Multiple Linear Regression (MLR)	0.453519	0.328208	1.335564e+06	558650.221115	10.229345	17.114985	3.198335
Ridge Linear Regression (RLR)	0.453519	0.328208	1.335564e+06	558650.168792	10.229345	17.114983	3.198335
Lasso Linear Regression (LLR)	0.358550	0.262261	1.567661e+06	613490.585641	12.007024	18.795092	3.465115
Elastic-Net Regression (ENR)	0.327295	0.239521	1.644047e+06	632400.436589	12.592078	19.374420	3.548532
Polynomial Regression (PNR)	0.508861	0.372097	1.200311e+06	522152.743008	9.193417	15.996837	3.032065

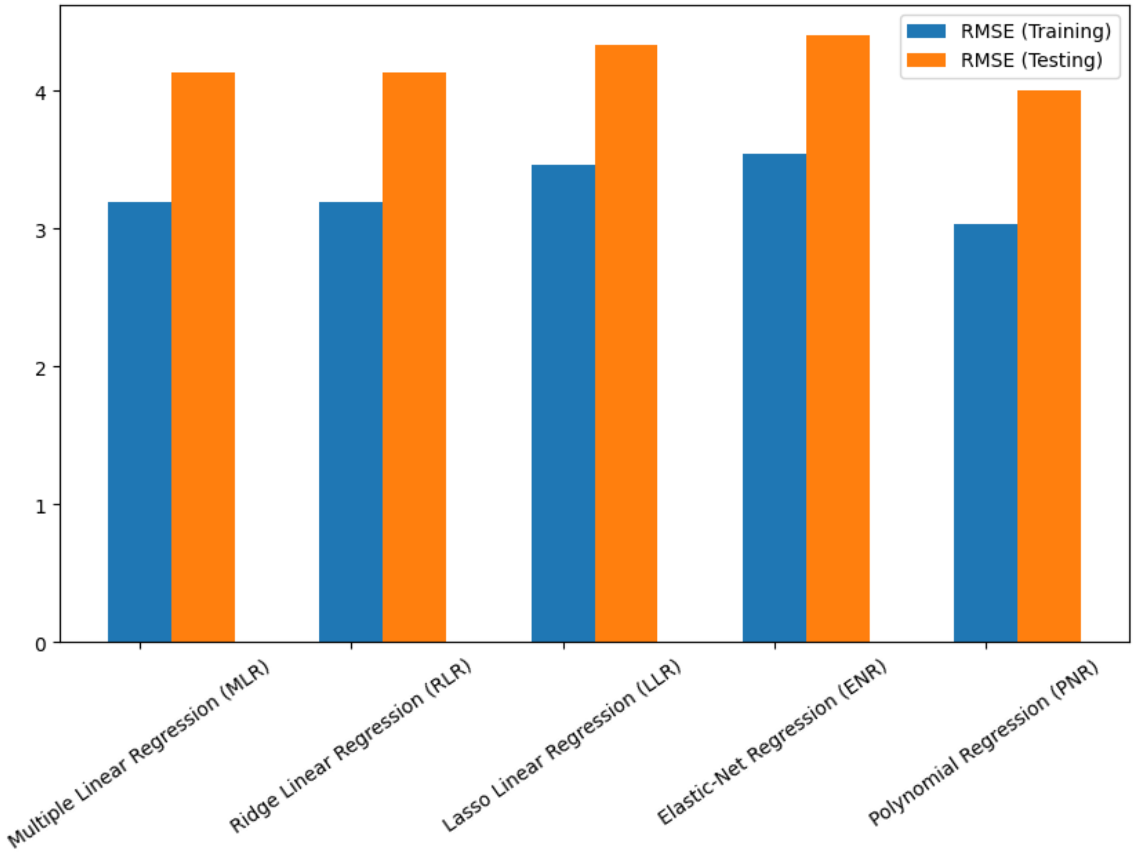
In [ ]:

```
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()
```



The polynomial regression models have the highest explainability capacity to comprehend the dataset, as is evident from the plot above.

```
In [ ]: cc = Model_Evaluation_Comparison_Matrix.columns.values
        # baxes = brokenaxes(ylims=((0,4),(524,532)))
        # baxes.bar(np.arange(s), Model_Evaluation_Comparison_Matrix[cc[-2]].values, wid
        # baxes.bar(np.arange(s)+0.3, Model_Evaluation_Comparison_Matrix[cc[-1]].values,
        # for index, value in enumerate(Model_Evaluation_Comparison_Matrix[cc[-2]].value
        #     plt.text(round(value,2), index, str(round(value,2)))
        # for index, value in enumerate(Model_Evaluation_Comparison_Matrix[cc[-1]].value
        #     plt.text(round(value,2), index, str(round(value,2)))
        plt.bar(np.arange(5), Model_Evaluation_Comparison_Matrix[cc[6]].values, width=0.
        plt.bar(np.arange(5)+0.3, Model_Evaluation_Comparison_Matrix[cc[7]].values, widt
        plt.xticks(np.arange(5),EMC.index, rotation=35)
        plt.legend()
        #plt.ylim([0,10])
        plt.show()
```



On the current dataset, sophisticated models like polynomials (degree-5) perform the best. It might be claimed that even basic regression can be a good solution for this issue.

7. Project Outcomes & Conclusions

The dataset has 2M samples in total, and after preprocessing, 18.4% of the data samples were eliminated.
We were able to gain some understanding of the feature-set by visualising the distribution of the data and their relationships.
The features had a high degree of multicollinearity, thus in the feature extraction step, we used the VIF Technique to narrow down the suitable features.
Testing numerous algorithms using the default hyperparameters helped us discover how different models performed on this particular dataset.
Although Polynomial Regression (Order-5) was the best option, using the multiple

regression approach is safe because their results were very comparable and more generalizable.

<<<--------------------------------------THE END---
----->>>

In [ ]:

```

```