In [199]:

```
import os
import math
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import display


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.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

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

In [200]:

```
df = pd.read_csv('../desktop/daily_sales.csv')


display(df.head())

original_df = df.copy(deep=True)

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

	DATE	STORE_ID	DAILY_SALES_AMOUNT	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIA
0	01-01-2020	1503	24291.92	1040	6.662225	
1	01-01-2020	1504	48414.88	2018	6.662225	
2	01-01-2020	1505	27791.83	1210	6.662225	
3	01-01-2020	1507	22157.77	899	6.662225	
4	01-01-2020	1515	54941.39	1574	6.662225	

Inference: The Datset consists of 8 features & 10346 samples.

In [201]:

```
df.Date=pd.to_datetime(df.DATE)

df['weekday'] = df.Date.dt.weekday
df['month'] = df.Date.dt.month
df['year'] = df.Date.dt.year

df.drop(['DATE'], axis=1, inplace=True)

df.head()
```

Out[201]:

	STORE_ID	DAILY_SALES_AMOUNT	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIAL_HOLI
0	1503	24291.92	1040	6.662225	
1	1504	48414.88	2018	6.662225	
2	1505	27791.83	1210	6.662225	
3	1507	22157.77	899	6.662225	
4	1515	54941.39	1574	6.662225	

In [202]:

```
target = 'DAILY_SALES_AMOUNT'
features = [i for i in df.columns if i not in [target]]
original_df = df.copy(deep=True)

df.head()
```

Out[202]:

	STORE_ID	DAILY_SALES_AMOUNT	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIAL_HOLI
0	1503	24291.92	1040	6.662225	
1	1504	48414.88	2018	6.662225	
2	1505	27791.83	1210	6.662225	
3	1507	22157.77	899	6.662225	
4	1515	54941.39	1574	6.662225	

In [203]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10346 entries, 0 to 10345
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   STORE_ID              10346 non-null  int64  
 1   DAILY_SALES_AMOUNT    10346 non-null  float64
 2   DAILY_SALES_QUANTITY  10346 non-null  int64  
 3   EURO_PARITY           9860 non-null   float64
 4   SPECIAL_HOLIDAY       10346 non-null  int64  
 5   WEEKEND               10346 non-null  int64  
 6   SEASONS               10346 non-null  object 
 7   weekday               10346 non-null  int64  
 8   month                 10346 non-null  int64  
 9   year                  10346 non-null  int64  
dtypes: float64(2), int64(7), object(1)
memory usage: 808.4+ KB
```

In [204]:

```
df.nunique().sort_values()
```

Out[204]:

```
SPECIAL_HOLIDAY            2
WEEKEND                    2
year                       3
SEASONS                    4
weekday                    7
month                     12
STORE_ID                  30
EURO_PARITY              490
DAILY_SALES_QUANTITY    1956
DAILY_SALES_AMOUNT     10083
dtype: int64
```

In [205]:

```
nu = df[features].nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0;

for i in range(df[features].shape[1]):
    if nu.values[i]<=30:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

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

nu.head()
```

Inference: The Datset has 2 numerical & 7 categorical features.

Out[205]:

```
SPECIAL_HOLIDAY    2
WEEKEND            2
year               3
SEASONS            4
weekday            7
dtype: int64
```

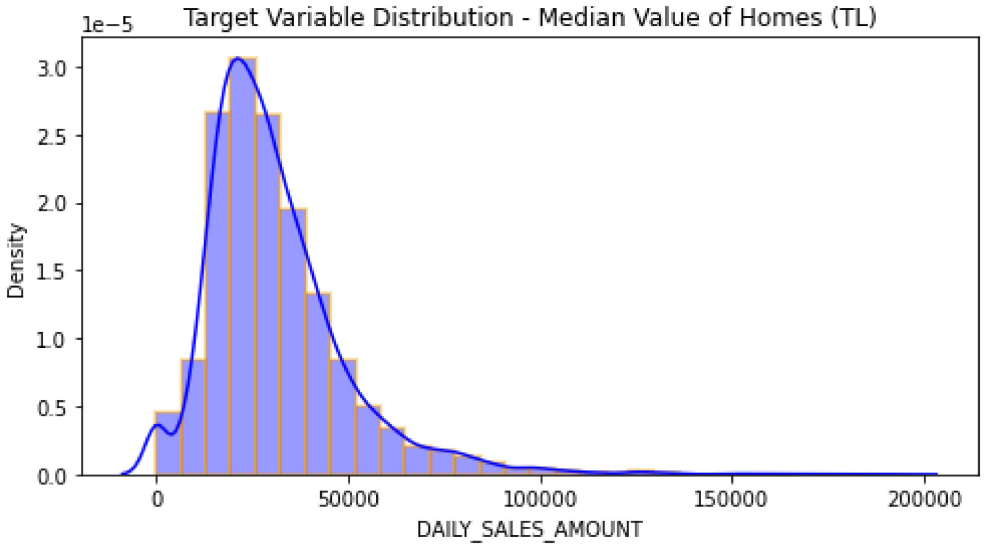
In [206]:

```
display(df.describe())
```

	STORE_ID	DAILY_SALES_AMOUNT	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIA
count	10346.000000	10346.000000	10346.000000	9860.000000	1(
mean	1663.158999	31168.833548	904.284458	11.194242	
std	203.006581	18497.011628	480.525413	3.090578	
min	1050.000000	0.000000	0.000000	6.509569	
25%	1507.000000	19238.532500	578.000000	8.995233	
50%	1518.000000	27491.580000	829.500000	10.146104	
75%	1908.000000	38670.650000	1140.000000	15.112589	
max	1921.000000	194528.320000	5299.000000	19.677292	

In [207]:

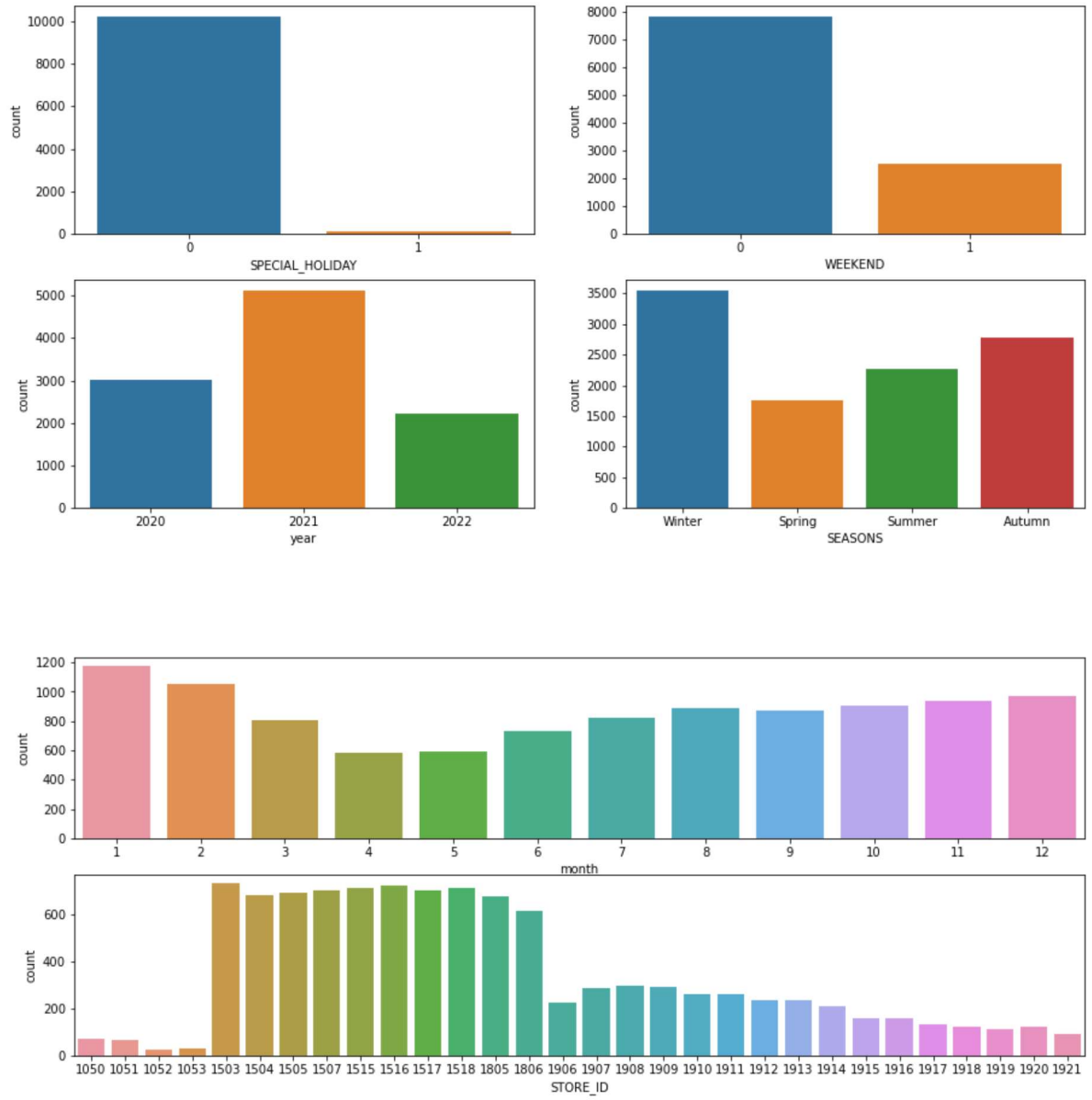
```
plt.figure(figsize=[8,4])
sns.distplot(df[target], color='b',hist_kws=dict(edgecolor="orange", linewidth=2), bins=30)
plt.title('Target Variable Distribution - Median Value of Homes (TL)')
plt.show()
```



In [208]:

```
print('\033[1mVisualising Categorical Features:'.center(100))

n=2
plt.figure(figsize=[15,4*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=7:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    else:
        plt.subplot(5,1,i-1)
        sns.countplot(df[cf[i]])

plt.tight_layout()
plt.show()
```

Visualising Categorical Features:

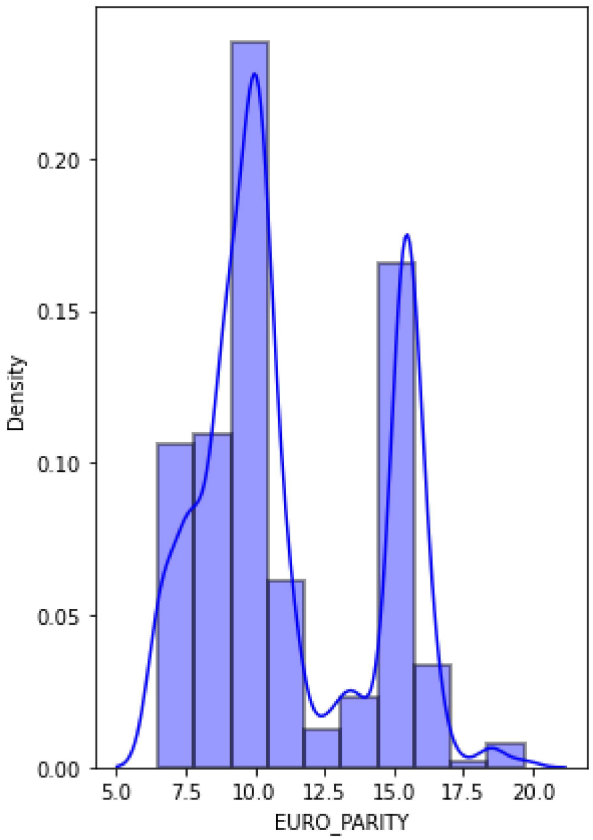


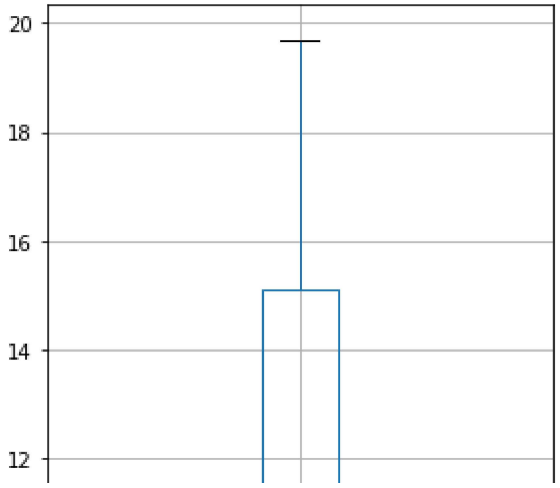
In [209]:

```
print('\033[1mNumeric Features Distribution'.center(130))


plt.figure(figsize=[15,6*math.ceil(len(nf)/4)])

plt.subplot(math.ceil(len(nf)/3),4,1)
sns.distplot(df[nf[0]],hist_kws=dict(edgecolor="black", linewidth=2), bins=10, color='b')
            #list(np.random.randint([255,255,255])/255))
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,6*math.ceil(len(nf)/4)])

plt.subplot(math.ceil(len(nf)/3),4,1)
df.boxplot(nf[0])
plt.tight_layout()
plt.show()
```

Numeric Features Distributio

n



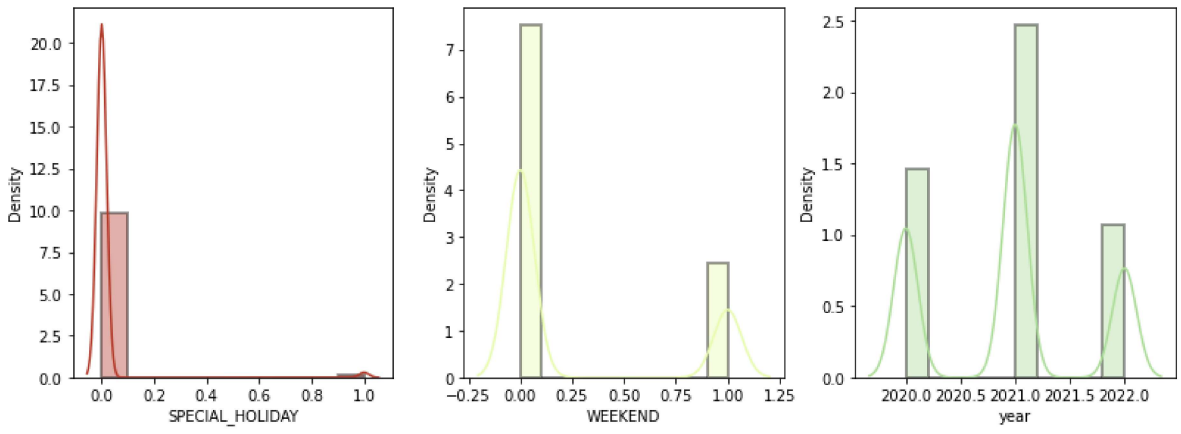

20
18
16
14
12

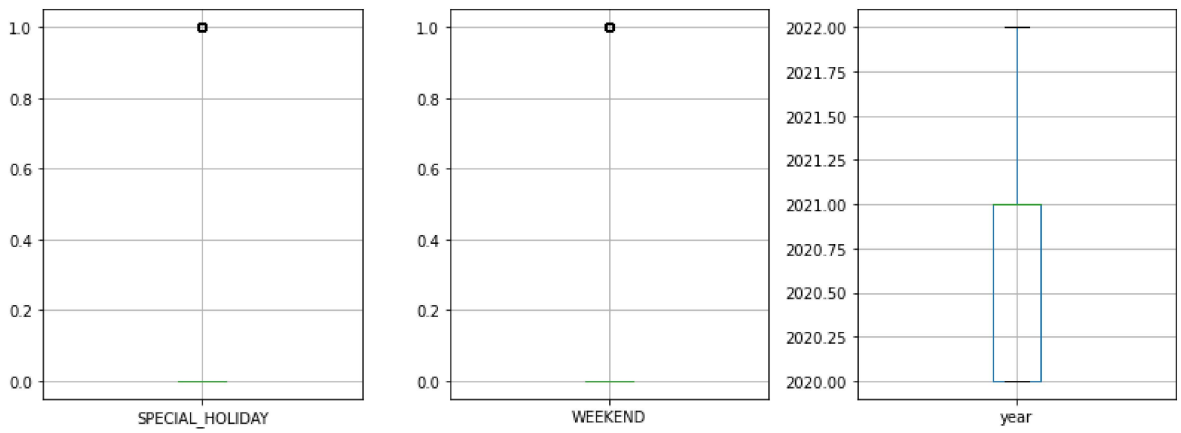
In [210]:

```
#Visualising the numeric features

print('\033[1mNumeric Features Distribution'.center(130))

n=4
a=len(cf)-4

clr=['r','g','b','g','b','r']

plt.figure(figsize=[15,6*math.ceil(len(cf)/n)])
for i in range(a):
    plt.subplot(math.ceil(len(cf)/3),n,i+1)
    sns.distplot(df[cf[i]],hist_kws=dict(edgecolor="black", linewidth=2), bins=10, color=li
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,6*math.ceil(len(cf)/n)])
for i in range(a):
    plt.subplot(math.ceil(len(cf)/3),n,i+1)
    df.boxplot(cf[i])
plt.tight_layout()
plt.show()
```

Numeric Features Distributio

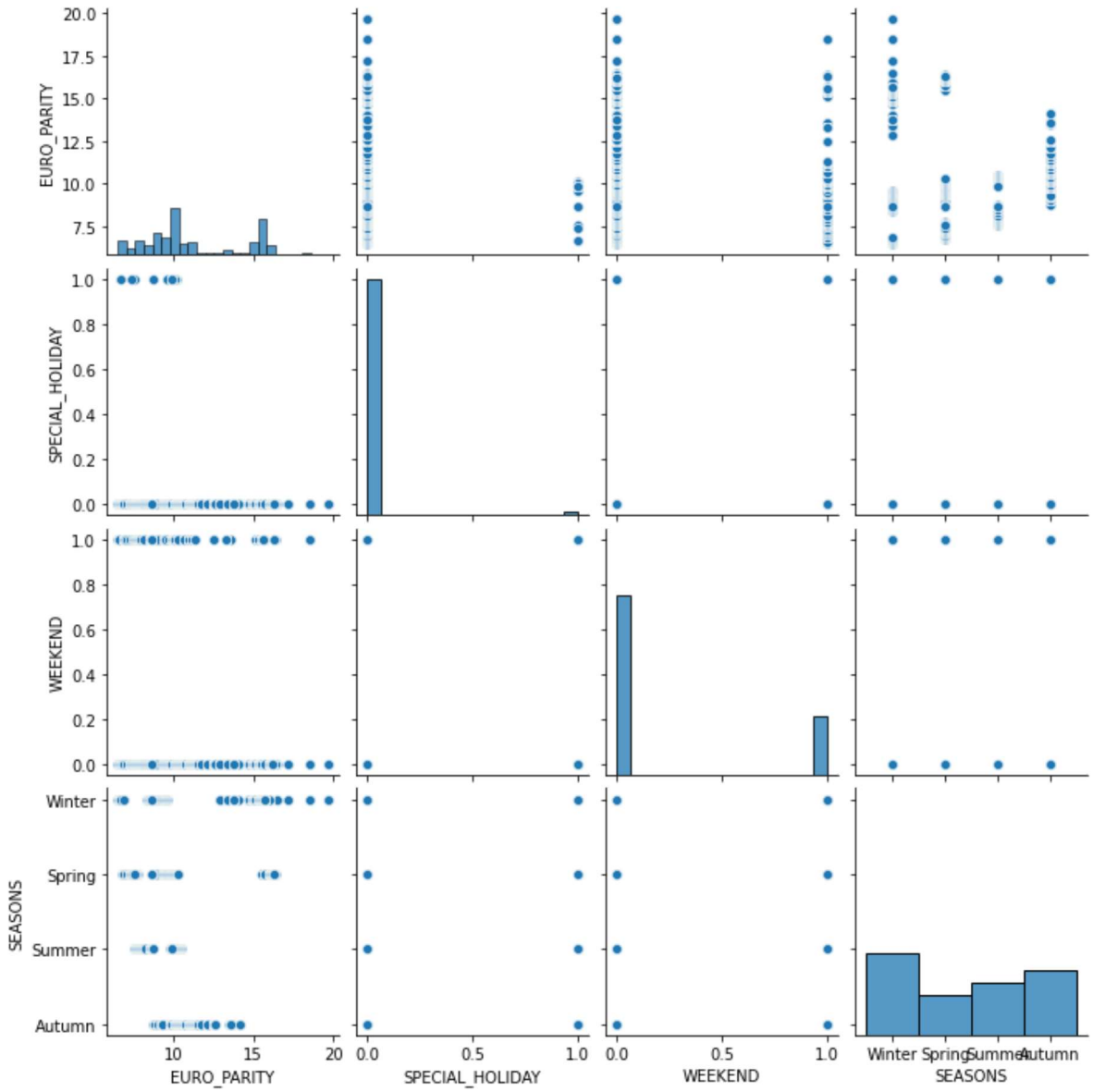
n





In [211]:

```
sns.pairplot(data = df, vars=['EURO_PARITY','SPECIAL_HOLIDAY','WEEKEND','SEASONS'])
plt.show()
```



In [212]:

```
counter = 0
rs,cs = original_df.shape

df.drop_duplicates(inplace=True)

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
```

Inference: The dataset doesn't have any duplicates

In [213]:

```
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)
```

```
                     Total Null Values  Percentage
STORE_ID                             0         0.0
DAILY_SALES_AMOUNT                   0         0.0
DAILY_SALES_QUANTITY                 0         0.0
SPECIAL_HOLIDAY                      0         0.0
WEEKEND                              0         0.0
SEASONS                              0         0.0
weekday                              0         0.0
month                                0         0.0
year                                 0         0.0
EURO_PARITY                        486         4.7
```

In [214]:

```
df=df.dropna()
```

In [215]:

```
#Kategorik veriler numeriğe çevrilmiştir.

df3 = df.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):
        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_fi

df3.shape
```

One-Hot Encoding on features:
SPECIAL_HOLIDAY
WEEKEND

Dummy Encoding on features:
year
SEASONS
weekday
month
STORE_ID

Out[215]:

(9860, 56)

In [216]:

```
#Outlier kontrolü

df1 = df3.copy()


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.'
print('After removal of outliers, The dataset now has {} samples.'.format(df1.shape[0]))
```

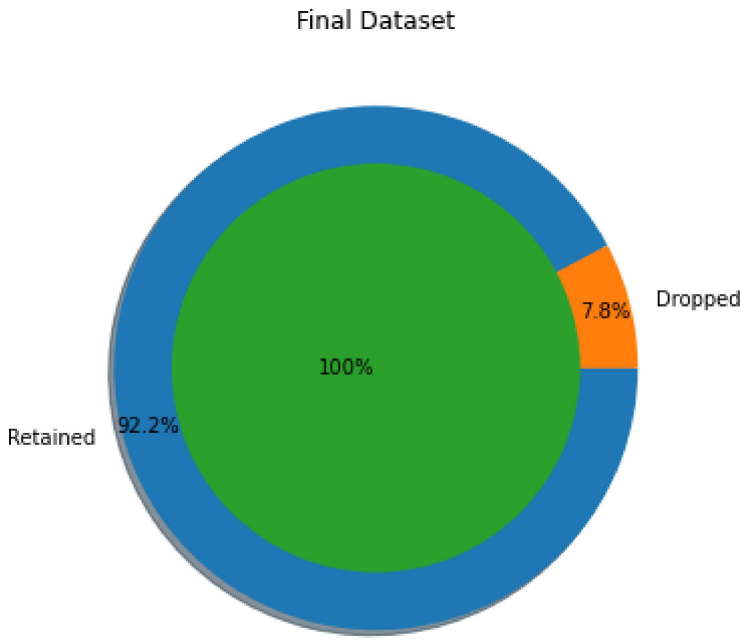
	DAILY_SALES_AMOUNT	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIAL_HOLIDAY	WEEK
0	24291.92	1040	6.662225	1	
1	27791.83	1210	6.662225	1	
2	22157.77	899	6.662225	1	
3	54941.39	1574	6.662225	1	
4	37232.62	1535	6.662225	1	

5 rows × 56 columns

Inference:
Before removal of outliers, The dataset had 9860 samples.
After removal of outliers, The dataset now has 9544 samples.

In [217]:

```
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=['Retained','Dr
        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[0]-df.shap
while retaining {round(100 - (df.shape[0]*100/(original_df.shape[0])),2)}% of the data.')
```



Inference: After the cleanup process, 802 samples were dropped, while retain
ing 7.75% of the data.

In [218]:

```
#Data Manupülasyonu
#Boşluklar silindi. Data %80 train ve %20 test için kullanılmıştır.

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, ra
Train_X.reset_index(drop=True,inplace=True)

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

```
Original set  --->  (9544, 55) (9544,)
Training set  --->  (7635, 55) (7635,)
Testing set   --->  (1909, 55)  (1909,)
```

In [219]:

```
#Özellikler standardize edildi

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(120))
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(120))
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

	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIAL_HOLIDAY	WEEKEND	year_
count	7.635000e+03	7.635000e+03	7.635000e+03	7.635000e+03	7.635000e
mean	-3.541880e-17	2.771413e-16	2.298387e-16	-3.797879e-16	8.759638
std	1.000065e+00	1.000065e+00	1.000065e+00	1.000065e+00	1.000065e
min	-2.163196e+00	-1.519945e+00	-1.247503e-01	-5.626778e-01	-9.8892
25%	-7.204049e-01	-7.132188e-01	-1.247503e-01	-5.626778e-01	-9.8892
50%	-1.106688e-01	-3.453717e-01	-1.247503e-01	-5.626778e-01	-9.8892
75%	6.132360e-01	1.269956e+00	-1.247503e-01	-5.626778e-01	1.011196e
max	2.830116e+00	2.753661e+00	8.016010e+00	1.777216e+00	1.011196e

8 rows × 55 columns

Standardardization on Testing set

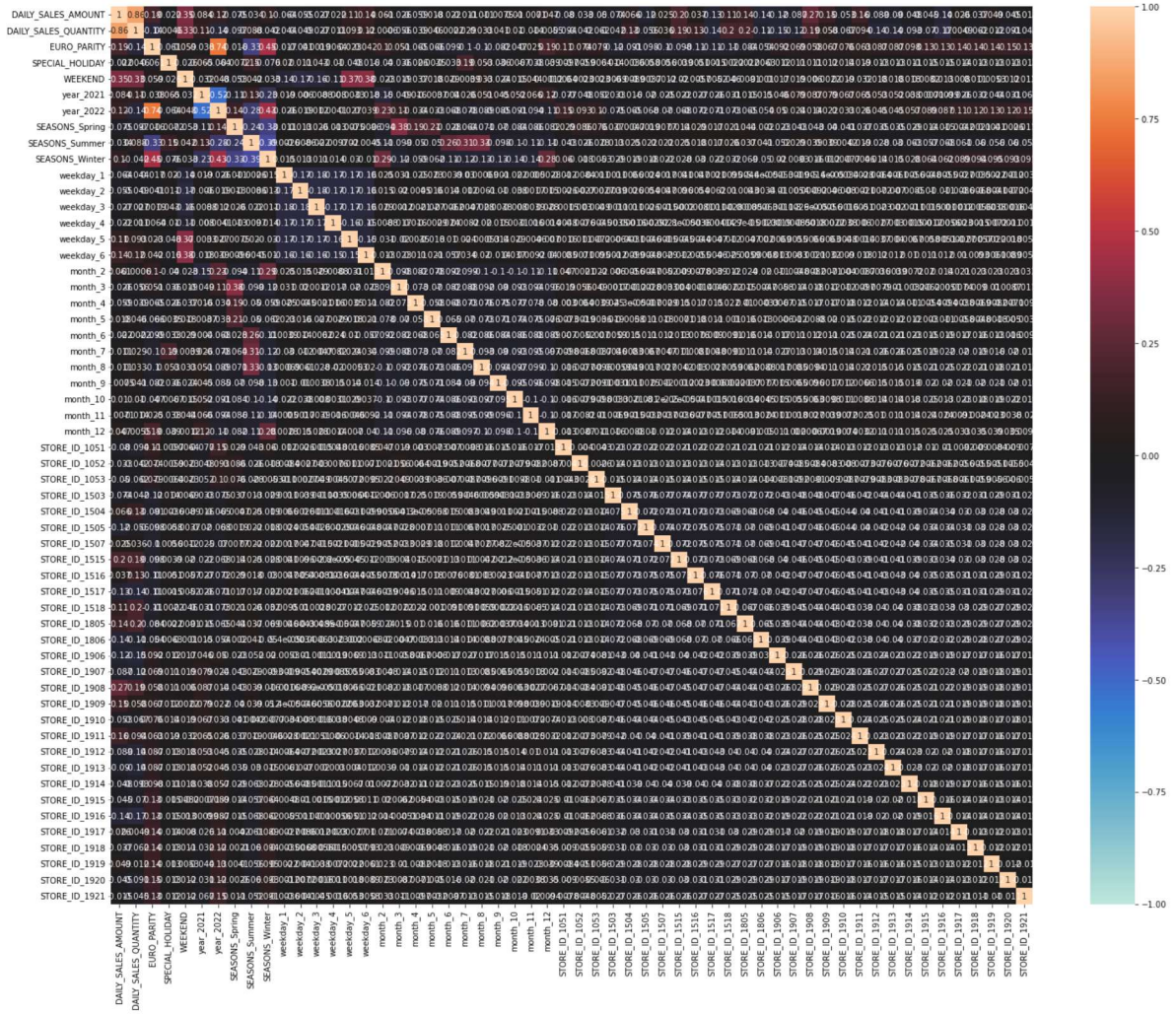
	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIAL_HOLIDAY	WEEKEND	year_202
count	1909.000000	1909.000000	1909.000000	1909.000000	1909.0000(
mean	-0.008585	0.015187	-0.043727	0.001152	-0.0166
std	1.000513	1.006880	0.808315	1.000961	0.9999
min	-2.163196	-1.517466	-0.124750	-0.562678	-0.9889
25%	-0.720405	-0.706111	-0.124750	-0.562678	-0.9889
50%	-0.120706	-0.339368	-0.124750	-0.562678	-0.9889
75%	0.596926	1.284812	-0.124750	-0.562678	1.0111

	DAILY_SALES_QUANTITY	EURO_PARITY	SPECIAL_HOLIDAY	WEEKEND	year_202
max	2.832625	2.753661	8.016010	1.777216	1.01119

In [220]:

```
#Hangi özellikler arasında korelasyon vardır bakıldı.

print('\033[1mCorrelation Matrix'.center(100))
plt.figure(figsize=[25,20])
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, center=0)
plt.show()
```

Correlation Matrix



In [221]:

```
#Resresyonda kullanılacak katsayıları buluyoruz

Train_xy = pd.concat([Train_X_std,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.columns)), data
#print(API.conf_int())
#print(API.pvalues)
API.summary()
```

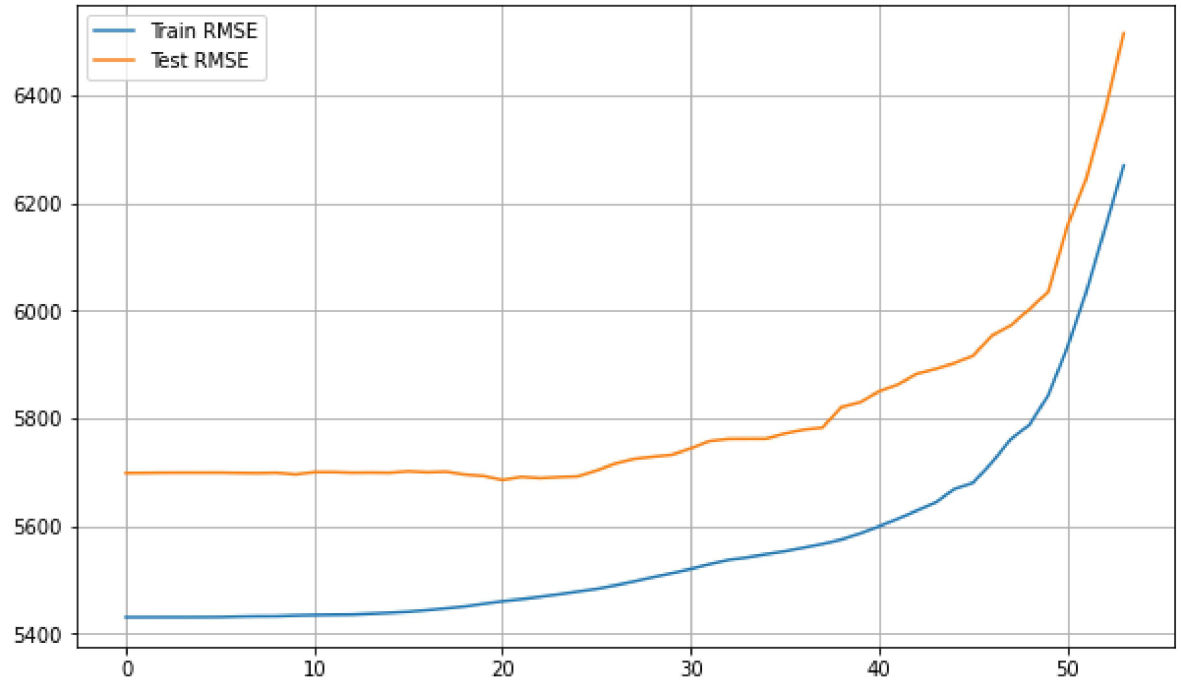
STORE_ID_1917	76.9253	105.434	0.730	0.466	-129.755	283.605
STORE_ID_1918	-53.2813	104.212	-0.511	0.609	-257.565	151.002
STORE_ID_1919	522.5303	98.827	5.287	0.000	328.803	716.258
STORE_ID_1920	219.2831	104.388	2.101	0.036	14.653	423.913
STORE_ID_1921	80.6420	94.038	0.858	0.391	-103.699	264.983

Omnibus:	3067.708	**Durbin-Watson:**	1.963
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	36092.373
Skew:	1.592	**Prob(JB):**	0.00
Kurtosis:	13.164	**Cond. No.**	20.9

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [222]:

```
#RFE
#Hedef değişkeninin tahmin edilmesi için en uygun özellikler seçilmiştir.

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 range(m):
    lm = LinearRegression()
    rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-i)
    rfe = rfe.fit(Train_X_std, Train_Y)

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

    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.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')

plt.legend()
plt.grid()
plt.show()
```



In [223]:

```
#İlk metod olarak Linear Regression kullanılmıştır.

lm = LinearRegression()
rfe = RFE(lm,n_features_to_select=Train_X_std.shape[1]-28)
rfe = rfe.fit(Train_X_std, Train_Y)

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


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_]
```

```
5504.273227008414
5728.282243843577
```

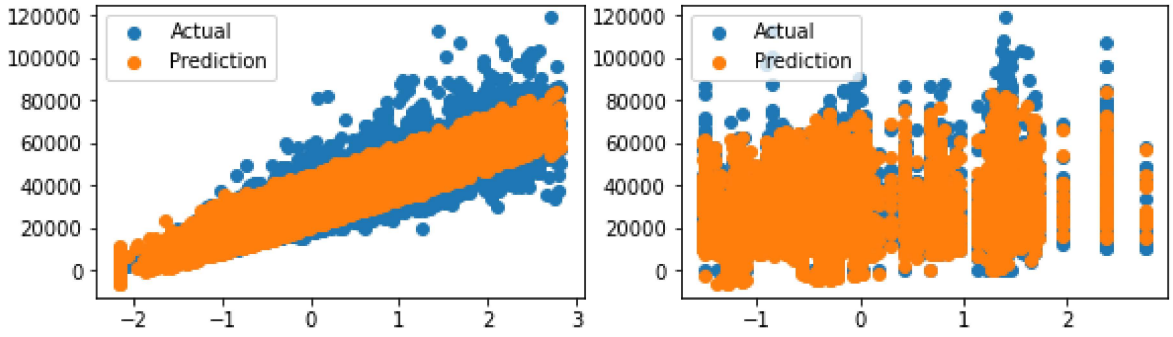
In [231]:

```
#Predictive Modeling

#Let us first define a function to evaluate our models

Model_Evaluation_Comparison_Matrix = pd.DataFrame(np.zeros([3,8]), columns=['Train-R2','Tes
                                                                            'Train-MSE','Te
rc=np.random.choice(Train_X_std.loc[:,Train_X_std.nunique()>=50].columns.values,2,replace=F
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.square(Trai
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Tr
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_

    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.square(Test
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Te
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_
    print('\n{}Residual Plots{}'.format('-'*20, '-'*20))

    Model_Evaluation_Comparison_Matrix.loc[n,'Train-R2']  = round(r2_score(Train_Y, pred1),
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-R2']   = round(r2_score(Test_Y, pred2),2
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RSS'] = round(np.sum(np.square(Train_Y-
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RSS']  = round(np.sum(np.square(Test_Y-p
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-MSE'] = round(mean_squared_error(Train_
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-MSE']  = round(mean_squared_error(Test_Y
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RMSE']= round(np.sqrt(mean_squared_erro
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RMSE'] = round(np.sqrt(mean_squared_erro

    # 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()
```

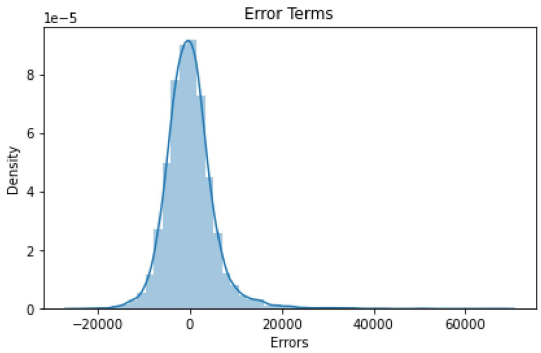
In [232]:

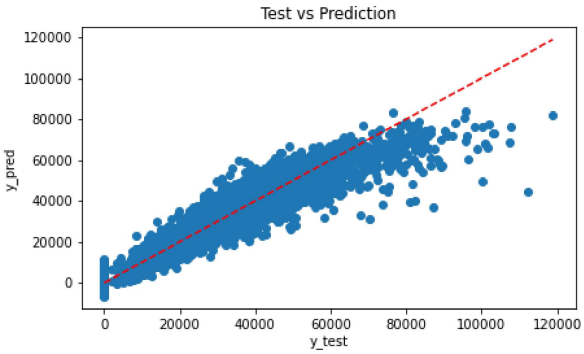
```
#Linear Regression
MLR = LinearRegression().fit(Train_X_std,Train_Y)
pred1 = MLR.predict(Train_X_std)
pred2 = MLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Multiple Linear Regression Model \033[0m{}{}\n'.format('<'*3,
#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 Intercept of the Regresion Model was found to be  29669.73140929928
```




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

R2-Score on Training set ---> 0.8720738278669876
Residual Sum of Squares (RSS) on Training set  ---> 231317776388.98303
Mean Squared Error (MSE) on Training set       ---> 30297023.757561628
Root Mean Squared Error (RMSE) on Training set ---> 5504.273227008414

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

R2-Score on Testing set ---> 0.8699851787595558
Residual Sum of Squares (RSS) on Training set  ---> 62640432140.94004
Mean Squared Error (MSE) on Training set       ---> 32813217.4651336
Root Mean Squared Error (RMSE) on Training set ---> 5728.282243843577

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


Error Terms
1e−5
Density
Errors
0
2
4
6
8
−20000
0
20000
40000
60000

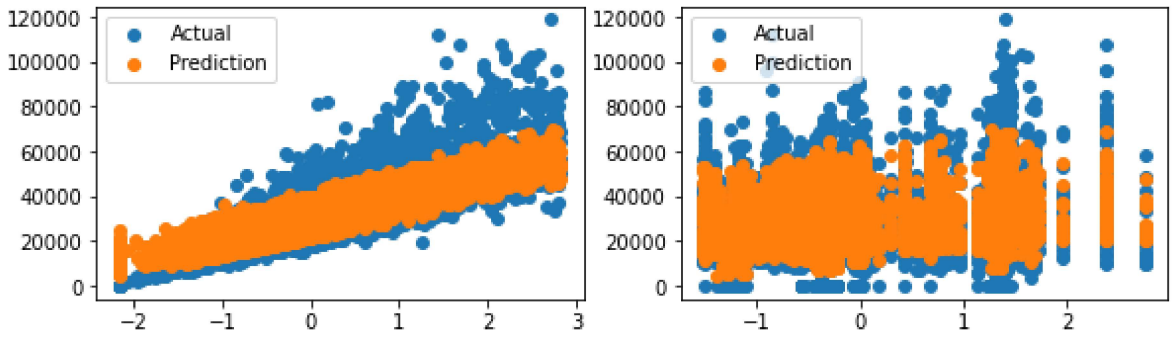

Test vs Prediction
y_pred
y_test
0
20000
40000
60000
80000
100000
120000

In [233]:

```
#Elastic Net Regression

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,'-'*
#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 Elastic-Net Regression Mod**
el ----------->>>

The Intercept of the Regresion Model was found to be 29669.73140929928



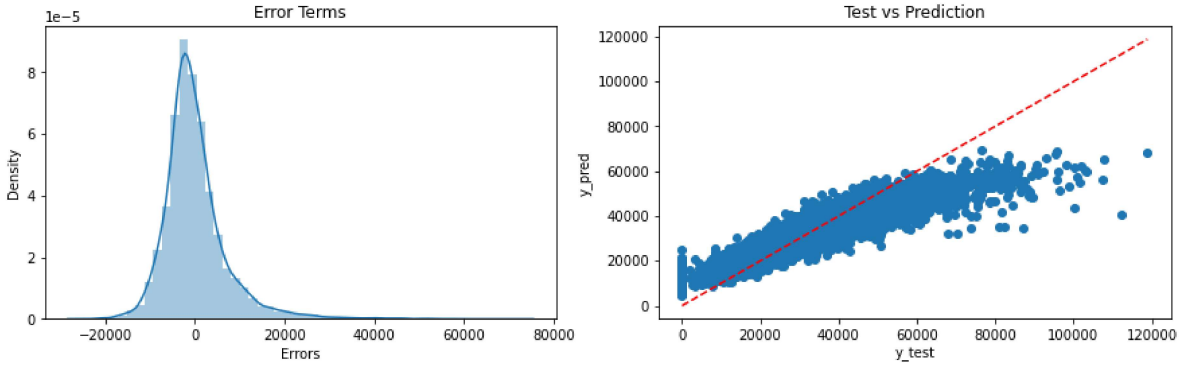
----------Training Set Metrics----------

R2-Score on Training set ---> 0.7855755451514148
Residual Sum of Squares (RSS) on Training set ---> 387725101689.29584
Mean Squared Error (MSE) on Training set   ---> 50782593.54149258
Root Mean Squared Error (RMSE) on Training set ---> 7126.190675353319

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

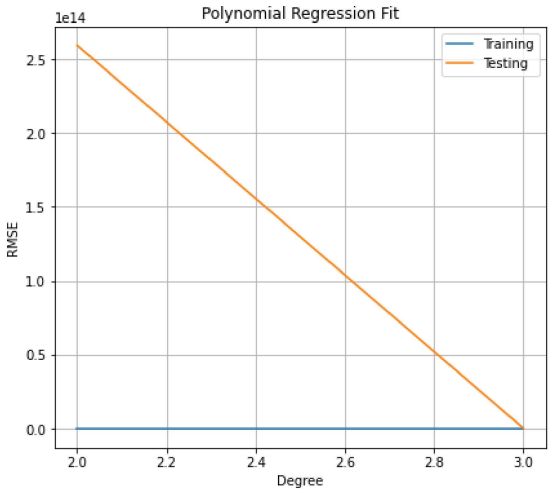
R2-Score on Testing set ---> 0.7787428933017406
Residual Sum of Squares (RSS) on Training set ---> 106600467897.43762
Mean Squared Error (MSE) on Training set   ---> 55840999.42243982
Root Mean Squared Error (RMSE) on Training set ---> 7472.683548929382

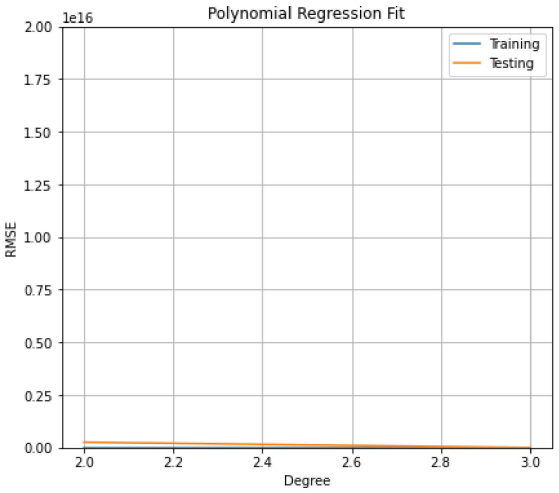
----------Residual Plots----------



In [234]:

```
#Polinomial Regression

Trr=[]; Tss=[]
n_degree=4

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([0,2e16])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()
plt.show()
```

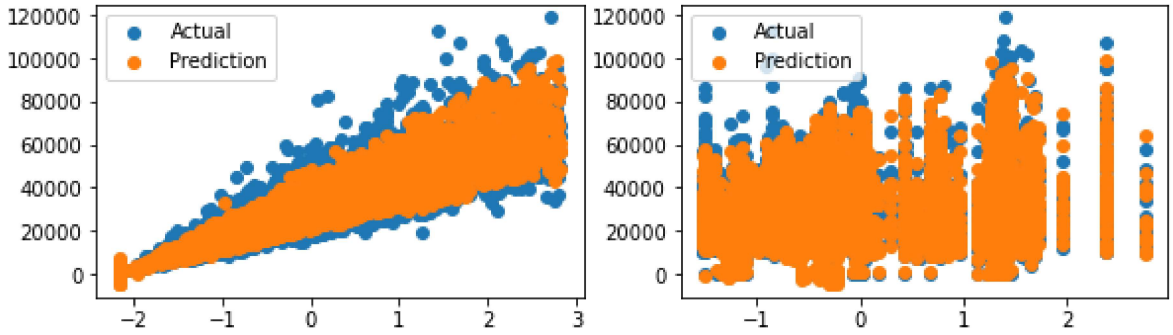

Polynomial Regression Fit
1e14
Training
Testing
RMSE
Degree
0.0
0.5
1.0
1.5
2.0
2.5
2.0
2.2
2.4
2.6
2.8
3.0


Polynomial Regression Fit
1e16
Training
Testing
RMSE
Degree
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.0
2.2
2.4
2.6
2.8
3.0

In [238]:

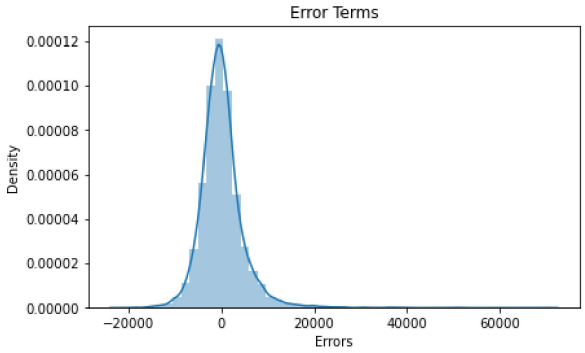
```
#Polinomial Regression Model

poly_reg = PolynomialFeatures(degree=2)
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,'-'*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)
```

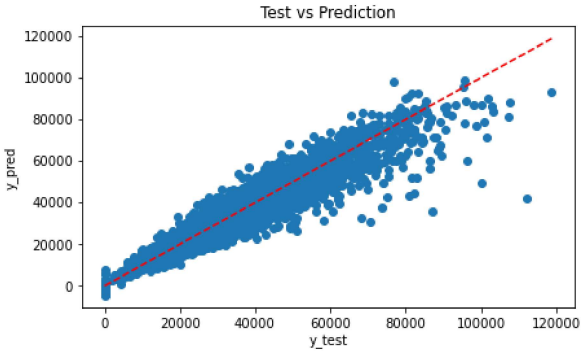
```
<<<---------------------------------- Evaluating Polynomial Regression Mode
l ----------------------------------->>>

The Coeffecient of the Regresion Model was found to be  [12465.64960421  378
3.32535791  1079.24617284   698.98187908
  1515.76646743  -454.17774591   417.68962413  -395.18670706
  1062.77083841  1007.69187743   485.88004283  1562.18921115
  2222.51601232   680.95191803  1362.37165765   847.51622346
  1177.7241863    626.86338205   526.95578976  2115.9565981
  1899.75799429   317.25804315  1579.33870176   542.76690109
   457.59707778   488.76019735   493.62910984]
The Intercept of the Regresion Model was found to be  29669.73140929928
```




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

R2-Score on Training set ---> 0.9048407188305309
Residual Sum of Squares (RSS) on Training set  ---> 172068255900.03894
Mean Squared Error (MSE) on Training set       ---> 22536772.220044393
Root Mean Squared Error (RMSE) on Training set ---> 4747.291040166422

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

R2-Score on Testing set ---> -2.662470757072175e+20
Residual Sum of Squares (RSS) on Training set  ---> 1.28276389718049e+32
Mean Squared Error (MSE) on Training set       ---> 6.719559440442589e+28
Root Mean Squared Error (RMSE) on Training set ---> 259221130320091.56
```

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





In [239]:

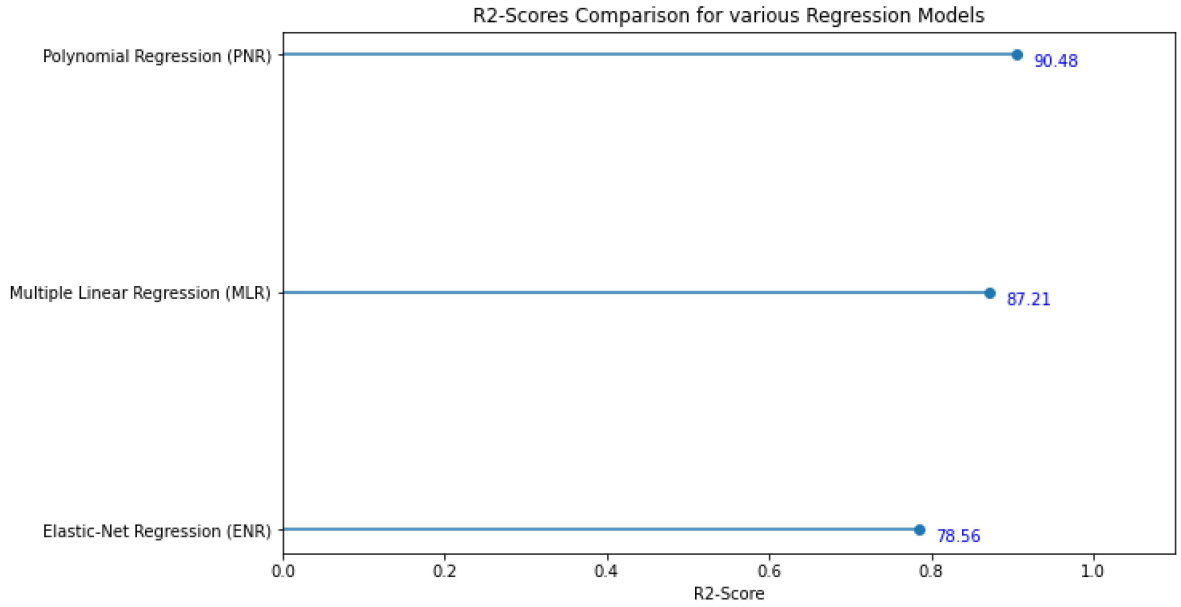
```
#Comparing
EMC = Model_Evaluation_Comparison_Matrix.copy()
EMC.index = ['Multiple Linear Regression (MLR)','Elastic-Net Regression (ENR)','Polynomial
EMC
```

Out[239]:

	Train-R2	Test-R2	Train-RSS	Test-RSS	Train-MSE	Test-MSE
Multiple Linear Regression (MLR)	0.872074	8.699852e-01	2.313178e+11	6.264043e+10	3.029702e+07	3.281322e+07
Elastic-Net Regression (ENR)	0.785576	7.787429e-01	3.877251e+11	1.066005e+11	5.078259e+07	5.584100e+07
Polynomial Regression (PNR)	0.904841	-2.662471e+20	1.720683e+11	1.282764e+32	2.253677e+07	6.719559e+28

In [240]:

```
#R2 Score

R2 = round(EMC['Train-R2'].sort_values(ascending=True),4)
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()
```



In [241]:

```
# Root Mean SquaredError Comparison for different Regression Models

cc = Model_Evaluation_Comparison_Matrix.columns.values
s=5

plt.bar(np.arange(3), Model_Evaluation_Comparison_Matrix[cc[6]].values, width=0.3, label='R
plt.bar(np.arange(3)+0.3, Model_Evaluation_Comparison_Matrix[cc[7]].values, width=0.3, labe
plt.xticks(np.arange(3),EMC.index, rotation =35)
plt.legend()
plt.ylim([0,500000])
plt.show()
```