In [1]:
```
#importing all the requried packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
from sklearn.linear_model import Lasso
lasso = Lasso()
from sklearn.linear_model import Ridge
ridge = Ridge()
from sklearn.linear_model import ElasticNet
EN = ElasticNet()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
from pymongo import MongoClient
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
svr = SVR()
```

In [2]:
```
#loading the dataset
df = pd.read_csv(r"C:\Users\hrush\Downloads\household_power_consumption\household_power_consumption.txt", sep= ";
```

```
C:\Users\hrush\anaconda5\lib\site-packages\IPython\core\interactiveshell.py:3444: DtypeWarning: Columns (2,3,4,5,
6,7) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

RANDOM SAMPLING

This is a very dataset. So have we are taking 30,000 samples from the total to perform various methods and functions such as EDA, preprocessing, model building etc

In [3]:
```
data = df.sample(n = 30000, ignore_index=True)
```

In [4]:
```
data.head()
```

Out[4]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	26/8/2008	09:46:00	0.080	0.000	240.180	0.200	0.000	0.000	1.0
1	26/3/2008	07:04:00	2.236	0.000	240.670	9.200	0.000	0.000	18.0
2	12/4/2008	21:55:00	3.578	0.650	239.660	15.200	0.000	2.000	18.0
3	27/6/2009	16:58:00	1.496	0.236	240.710	6.200	0.000	0.000	19.0
4	29/6/2009	00:29:00	0.350	0.236	245.400	1.600	0.000	0.000	1.0

Dropping the duplicate and unwanted data

In [5]:
```
data.drop_duplicates(subset={"Global_active_power","Global_reactive_power","Voltage","Global_intensity","Sub_mete
```

Out[5]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_meterir
0	26/8/2008	09:46:00	0.080	0.000	240.180	0.200	0.000	0.000	
1	26/3/2008	07:04:00	2.236	0.000	240.670	9.200	0.000	0.000	
2	12/4/2008	21:55:00	3.578	0.650	239.660	15.200	0.000	2.000	
3	27/6/2009	16:58:00	1.496	0.236	240.710	6.200	0.000	0.000	
4	29/6/2009	00:29:00	0.350	0.236	245.400	1.600	0.000	0.000	
...	...	...	...	...	...	...	...	...	
29995	6/12/2007	16:08:00	0.228	0.000	241.380	1.000	0.000	0.000	
29996	24/8/2010	00:25:00	0.172	0.000	240.780	0.800	0.000	0.000	
29997	27/6/2007	20:12:00	3.834	0.446	237.800	16.200	0.000	31.000	
29998	19/7/2008	23:49:00	2.090	0.214	240.330	8.600	0.000	0.000	
29999	14/4/2007	07:58:00	0.322	0.118	240.690	1.400	0.000	0.000	

29563 rows × 9 columns

In [6]:
```
print(data[data["Global_active_power"] == "?"].index)
data.drop(data.index[data["Global_active_power"] == "?"], inplace=True)
```

```
Int64Index([   69,   212,   283,   294,   439,   472,   492,   689,   760,
              777,
            ...
            29338, 29407, 29460, 29461, 29485, 29674, 29691, 29692, 29890,
            29895],
           dtype='int64', length=368)
```

In [7]:
```
data.drop(["Time","Date"], axis=1, inplace=True)
```

Changing the dtype of the features

In [8]:
```
data["Global_active_power"] = (data['Global_active_power'].astype("float"))
data["Global_reactive_power"] = (data['Global_reactive_power'].astype("float"))
data["Global_intensity"] =  (data['Global_intensity'].astype("float"))
data["Voltage"] = (data["Voltage"].astype('float'))
data["Sub_metering_1"] = (data['Sub_metering_1'].astype("float"))
data["Sub_metering_2"] = (data['Sub_metering_2'].astype("float"))
```

In [9]:
```
#merging the three columns
data["Total_metering"] = data["Sub_metering_1"] + data["Sub_metering_2"] + data["Sub_metering_3"]
```

DATA INFO AND DESCRIPTION

In [10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29632 entries, 0 to 29999
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Global_active_power    29632 non-null  float64
 1   Global_reactive_power  29632 non-null  float64
 2   Voltage                29632 non-null  float64
 3   Global_intensity       29632 non-null  float64
 4   Sub_metering_1         29632 non-null  float64
 5   Sub_metering_2         29632 non-null  float64
 6   Sub_metering_3         29632 non-null  float64
 7   Total_metering         29632 non-null  float64
dtypes: float64(8)
memory usage: 2.0 MB
```

In [11]:
```
data.describe().T
```

Out[11]:

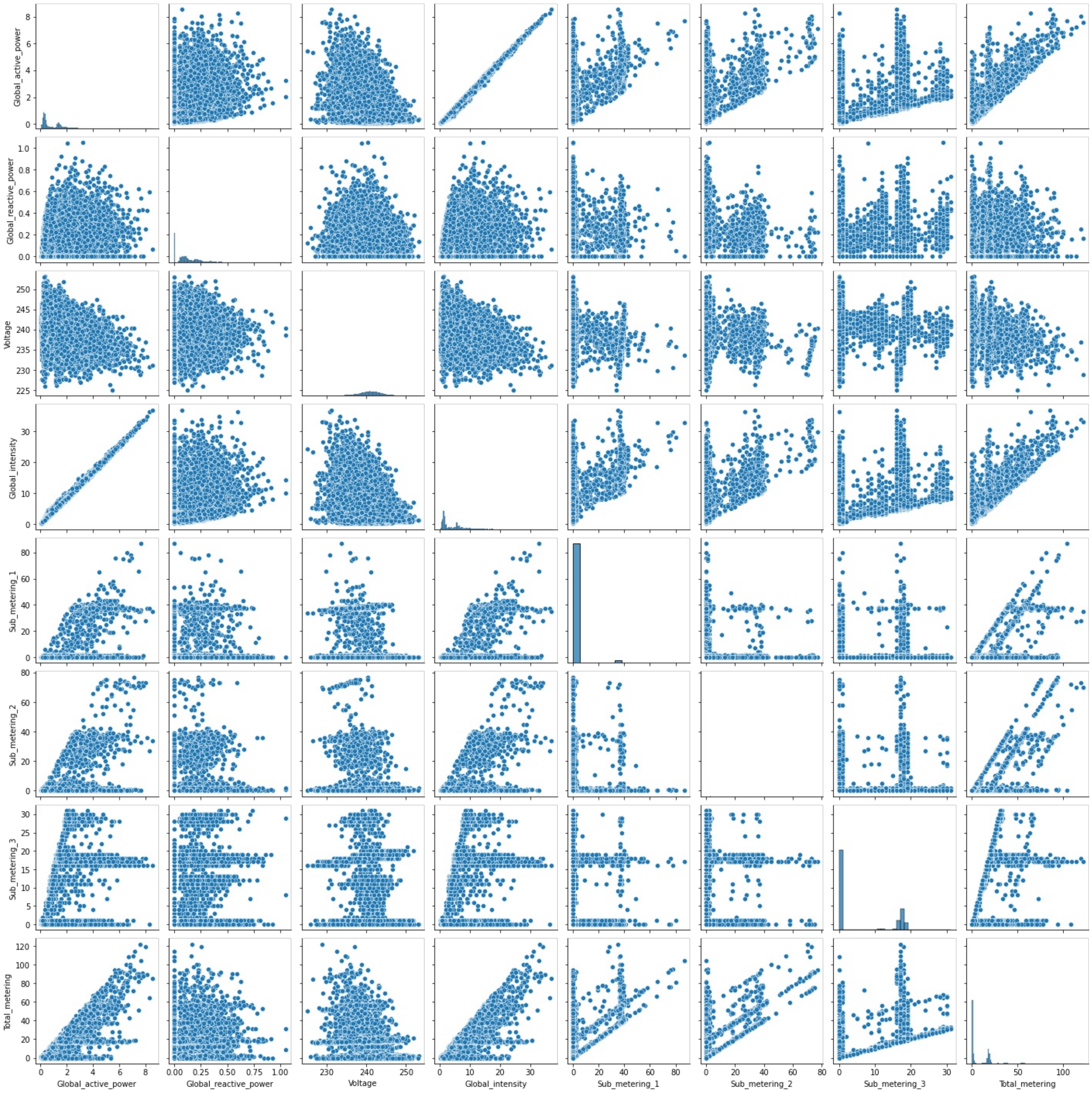
	count	mean	std	min	25%	50%	75%	max
Global_active_power	29632.0	1.095013	1.057088	0.078	0.310	0.604	1.5260	8.556
Global_reactive_power	29632.0	0.123816	0.112916	0.000	0.048	0.100	0.1940	1.048
Voltage	29632.0	240.839185	3.240869	225.020	238.990	240.980	242.8725	253.200
Global_intensity	29632.0	4.642123	4.442031	0.200	1.400	2.600	6.4000	37.000
Sub_metering_1	29632.0	1.200999	6.373842	0.000	0.000	0.000	0.0000	87.000
Sub_metering_2	29632.0	1.224285	5.542152	0.000	0.000	0.000	1.0000	77.000
Sub_metering_3	29632.0	6.446949	8.431714	0.000	0.000	1.000	17.0000	31.000
Total_metering	29632.0	8.872233	12.799234	0.000	0.000	1.000	18.0000	122.000

EXPLORING DATA ANALYSIS (EDA)

In [12]:

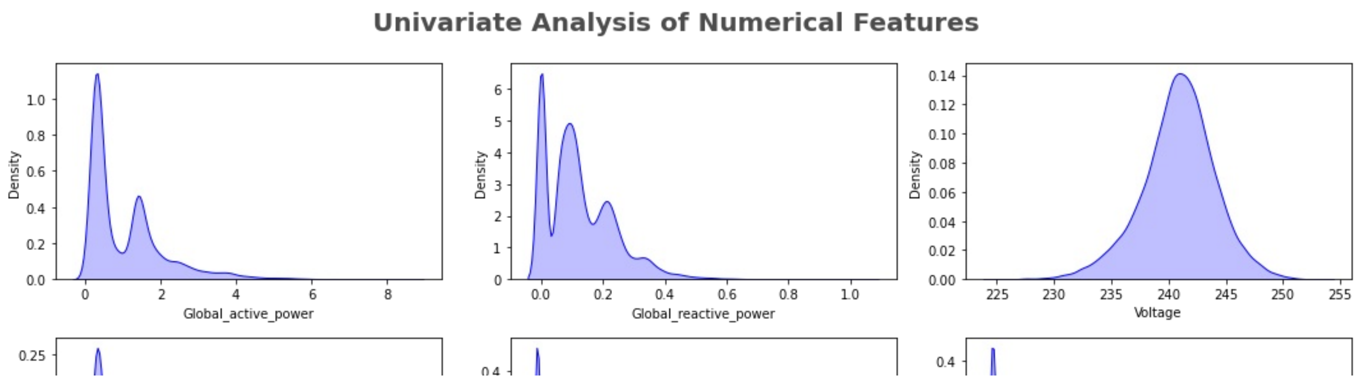
```python
sns.pairplot(data)
```

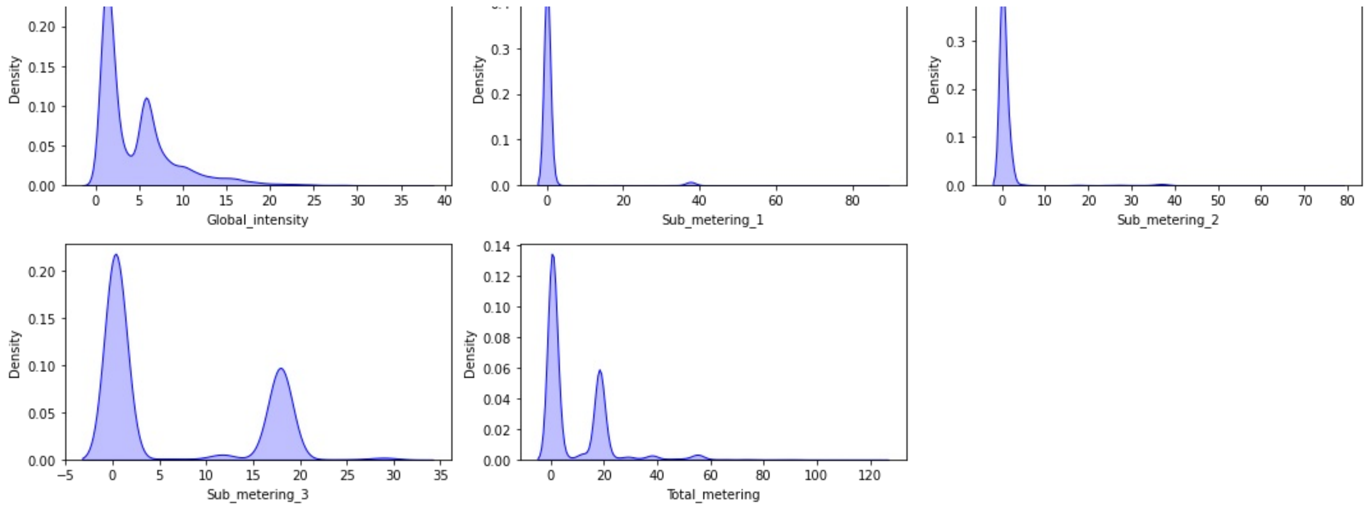
Out[12]: <seaborn.axisgrid.PairGrid at 0x250dc6e5b20>



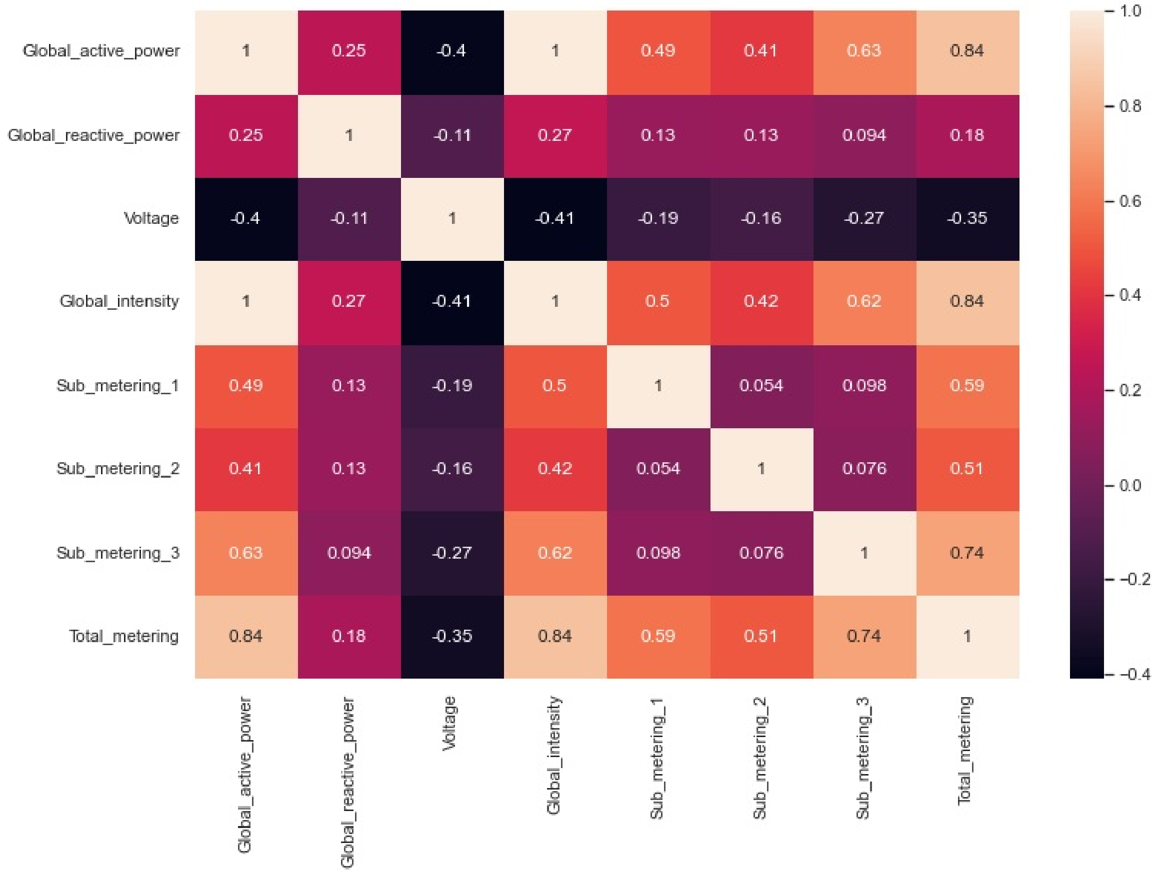
In [13]:

```python
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight='bold', alpha=0.7, y=1.)

for i in range(0, len(numeric_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=data[numeric_features[i]],shade=True, color='b')
    plt.xlabel(numeric_features[i])
    plt.tight_layout()
```

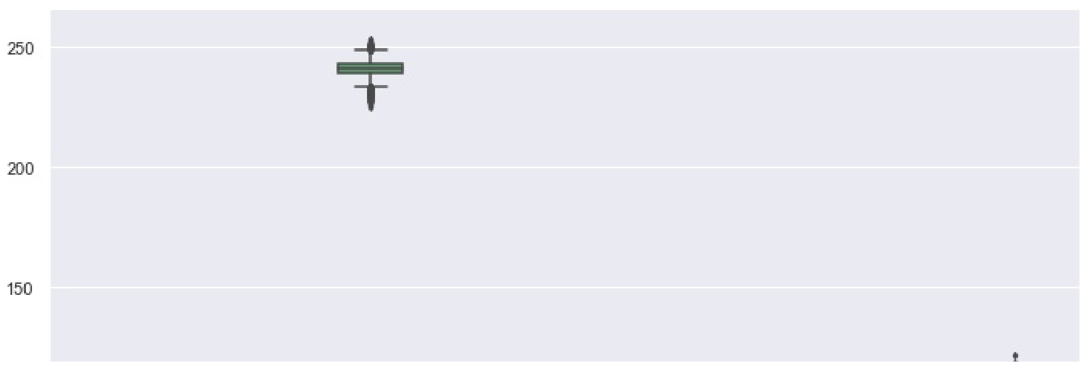


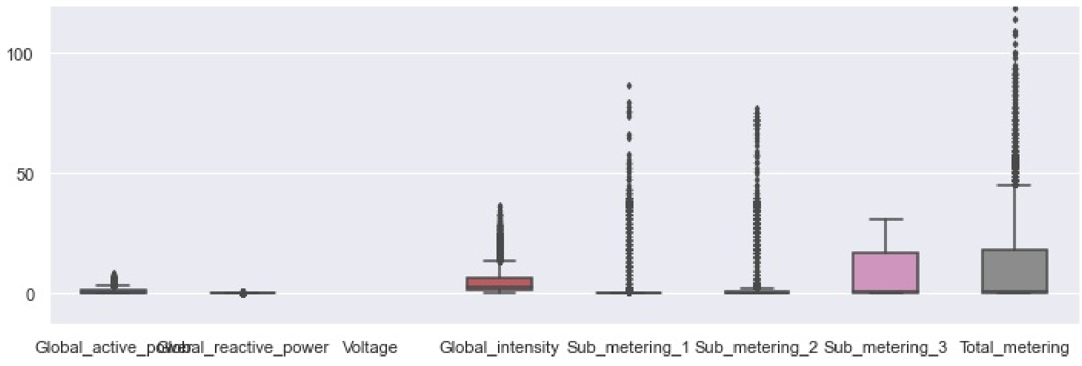


In [14]:
```
sns.set(rc= {"figure.figsize":(12,8)})
sns.heatmap(data.corr(),annot=True)
```
Out[14]: <AxesSubplot:>



In [15]:
```
fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(data=data, width= 0.5,ax=ax, fliersize=3)
```
Out[15]: <AxesSubplot:>

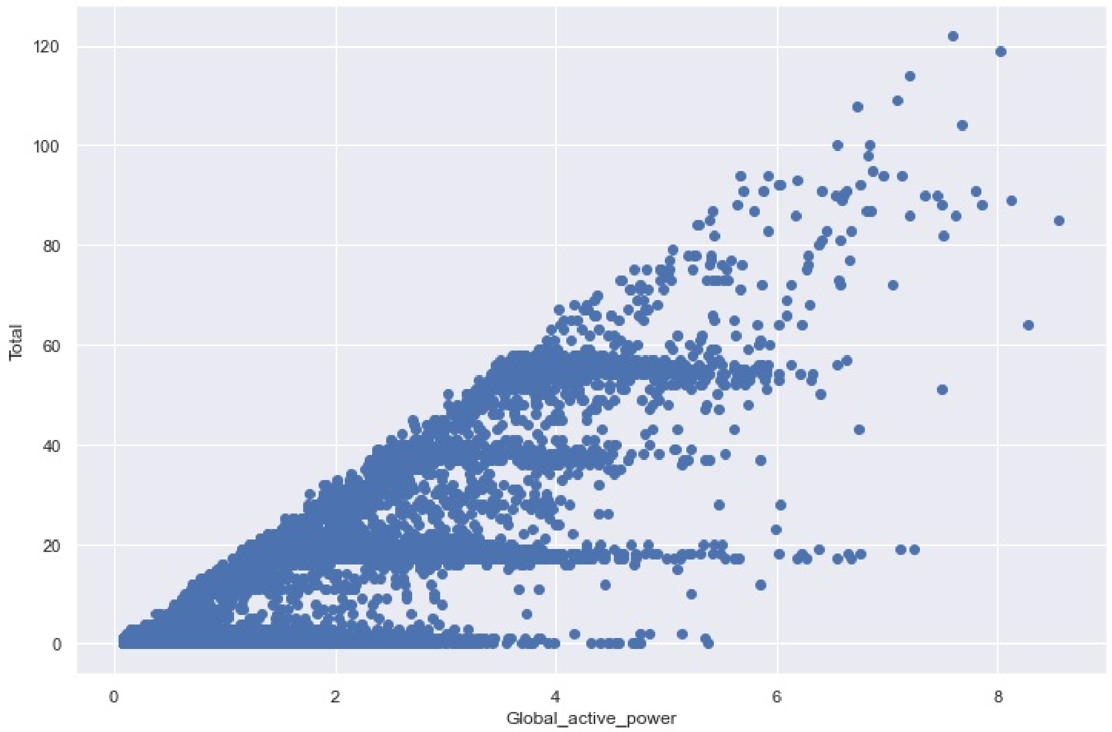




In [16]:

```
plt.scatter(data['Global_active_power'],data['Total_metering'])
plt.xlabel("Global_active_power")
plt.ylabel("Total")
```

Out[16]: Text(0, 0.5, 'Total')



In [17]:

```
plt.scatter(data['Global_reactive_power'],data['Total_metering'])
plt.xlabel("Global_reactive_power")
plt.ylabel("Total")
```

Out[17]: Text(0, 0.5, 'Total')

Visualizing the target feature

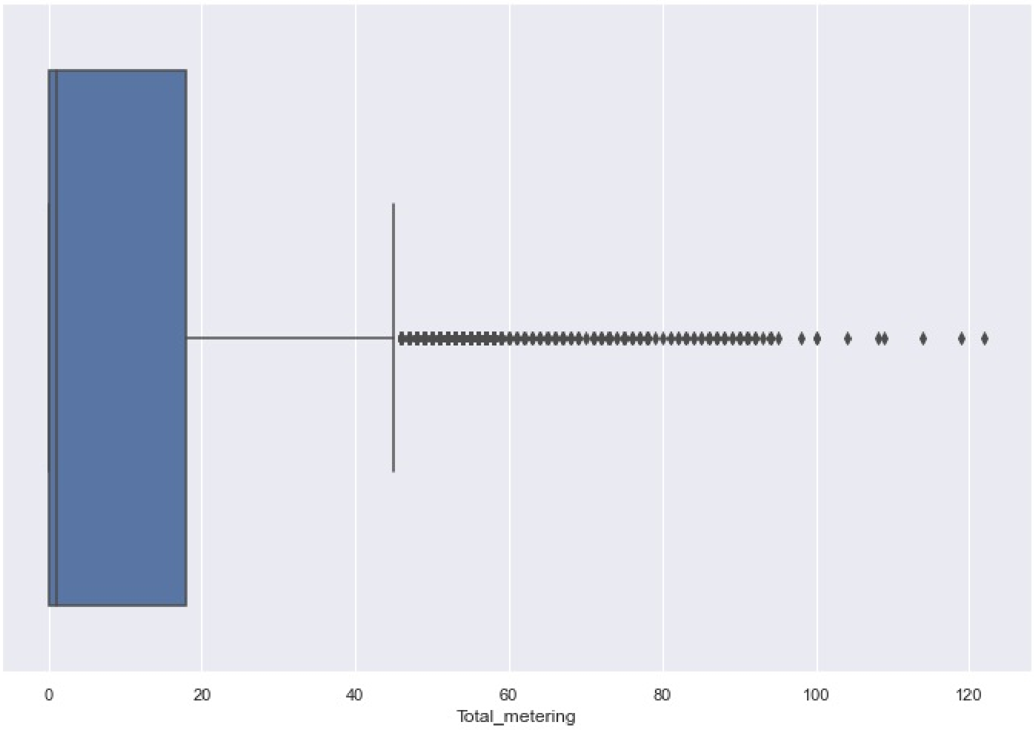
In [18]:

```
sns.boxplot(data['Total_metering'])
```

```
C:\Users\hrush\anaconda5\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

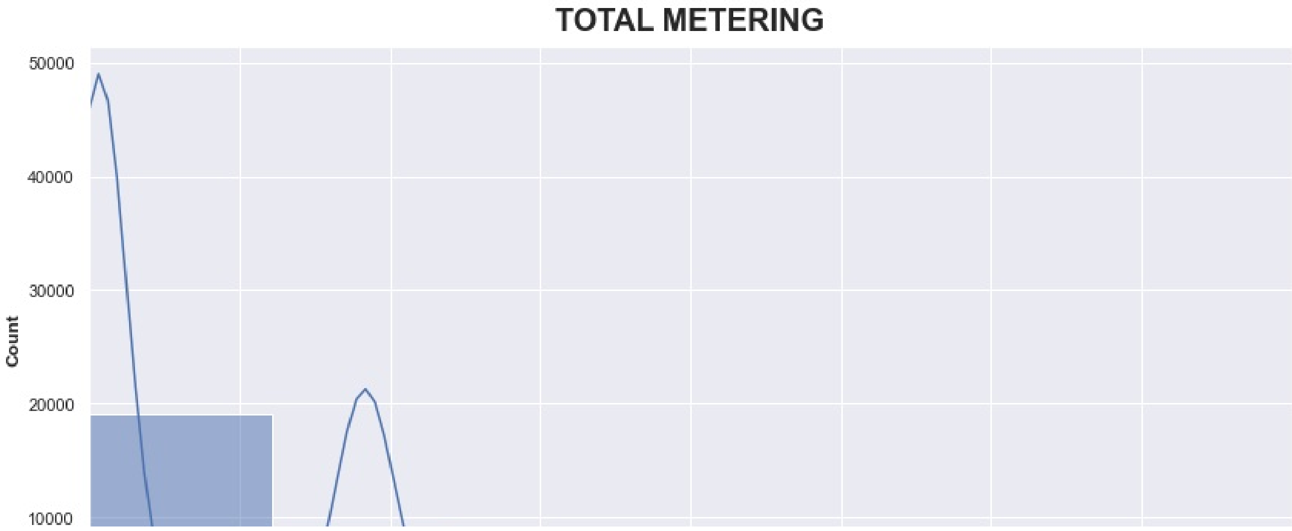
Out[18]:

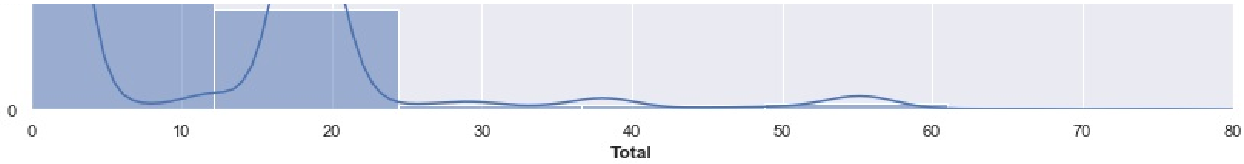
```
<AxesSubplot:xlabel='Total_metering'>
```



In [19]:

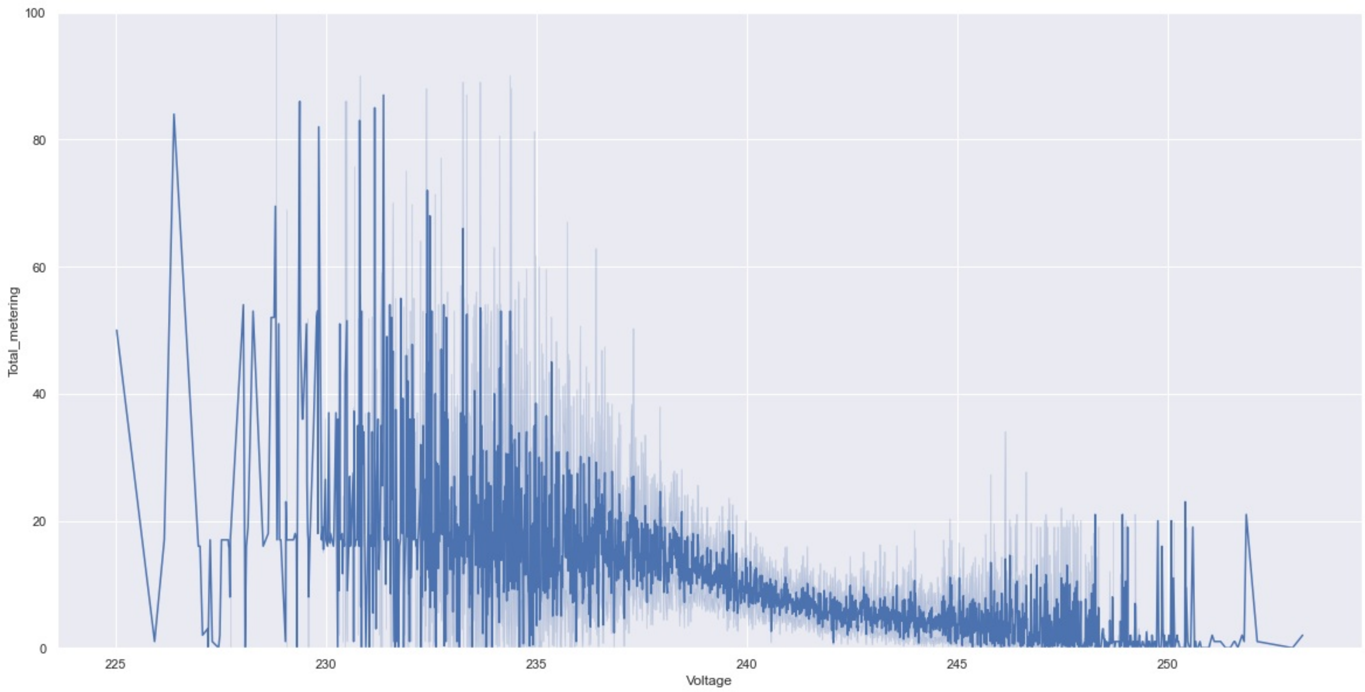
```
plt.subplots(figsize=(14,7))
sns.histplot(data.Total_metering, bins=10, kde=True, color = 'b')
plt.title("TOTAL METERING", weight="bold",fontsize=20, pad=10)
plt.ylabel("Count", weight="bold", fontsize=12)
plt.xlabel("Total", weight="bold", fontsize=12)
plt.xlim(0,80)
plt.show()
```





VOLTAGE VS TOTAL METERING

In [20]:
```
plt.subplots(figsize=(20,10))
sns.lineplot(x='Voltage',y='Total_metering',data=data,color='b')
plt.ylim(0,100)
plt.show()
```



SPLITTING THE DATASET INTO TRAINING AND TESTING DATA

In [21]:
```
x = data.drop(columns = ["Total_metering", "Sub_metering_1", "Sub_metering_2", "Sub_metering_3"])
y = data["Total_metering"]
```

In [22]:
```
x_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.33, random_state=10)
```

In [23]:
```
x_train.shape
```

Out[23]:
```
(19853, 4)
```

In [24]:
```
x_train=scaler.fit_transform(x_train)
```

In [25]:
```
x_test=scaler.transform(X_test)
```

In [26]:
```
x_train
```

Out[26]:
```
array([[ 5.35383263, -0.38583159, -1.37831419,  5.39082847],
       [-0.64410912,  0.87300796,  1.13297355, -0.6361476 ],
       [ 0.5713578 , -1.09503697, -1.55769189,  0.5782431 ],
       ...,
       [-0.6667928 , -0.27945078,  0.98143033, -0.68112504],
       [-0.78399178, -1.09503697,  0.34433023, -0.81605734],
       [ 0.6016027 , -1.09503697, -0.70410147,  0.5782431 ]])
```

In [27]:
```
y_train
```
Out[27]:
```
26918    18.0
17615     0.0
25012    17.0
8618     18.0
3912      1.0
         ...
28363     1.0
17955    17.0
29562     0.0
7396      1.0
17897    18.0
Name: Total_metering, Length: 19853, dtype: float64
```

In [28]:
```
x_test
```
Out[28]:
```
array([[-0.71594076, -0.26172065,  0.47731715, -0.72610247],
       [ 0.35019197,  1.08576957,  0.27938314,  0.3083785 ],
       [ 0.21219961,  0.00423137,  0.14021079,  0.1734462 ],
       ...,
       [ 1.2367456 ,  0.23472312,  0.2082506 ,  1.20792716],
       [-0.83313975, -0.5985932 , -1.44326129, -0.81605734],
       [-0.53825197,  0.96165863, -3.64836986, -0.5012153 ]])
```

LINEAR REGRESSION

In [29]:
```
regr.fit(x_train, y_train)
```
Out[29]:
```
LinearRegression()
```

In [30]:
```
print(regr.coef_)
```
```
[ 25.53452753  -0.05414028  -0.38024243 -14.86180486]
```

In [31]:
```
print(regr.intercept_)
```
```
8.852566362766334
```

PICKLING

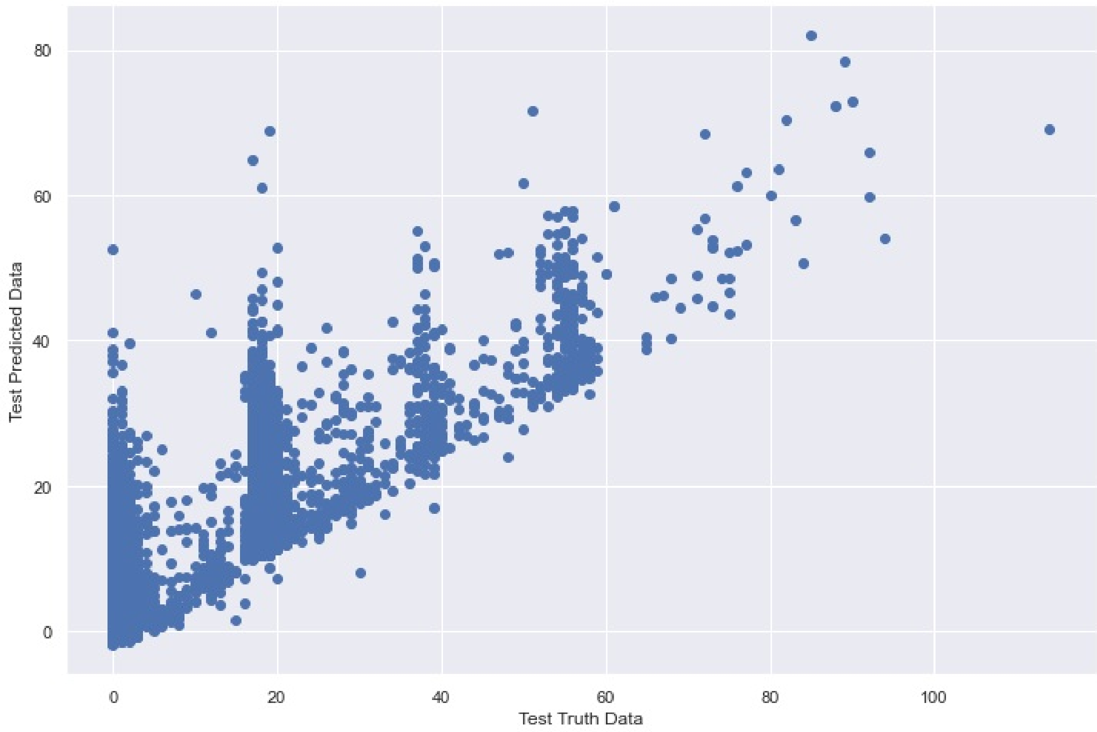
In [32]:
```
import pickle
# writing different model files to file
with open('modelForPrediction.sav', 'wb') as f:
    pickle.dump(regr,f)

with open('sandardScalar.sav', 'wb')as f:
    pickle.dump(scaler,f)
```

In [33]:
```
reg_pred=regr.predict(x_test)
```

In [34]:
```
reg_pred
```
Out[34]:
```
array([ 1.19522398, 13.04647462, 11.63971651, ..., 22.38840953,
        0.28801865,  3.89272119])
```

In [35]:
```
plt.scatter(y_test,reg_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```
```
Text(0, 0.5, 'Test Predicted Data')
```

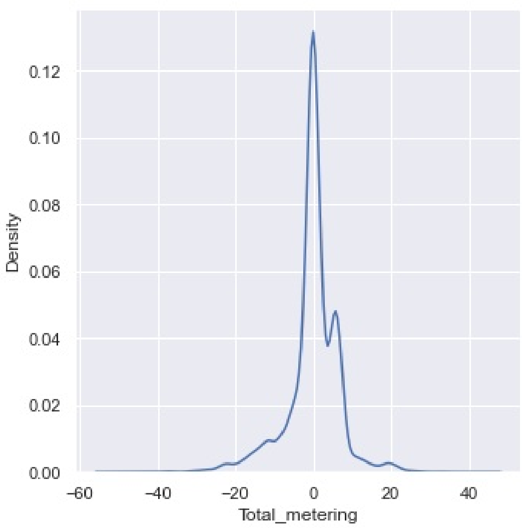
Out[35]: Text(0, 0.5, 'Test Predicted Data')



In [36]:
```
residuals=y_test-reg_pred
```

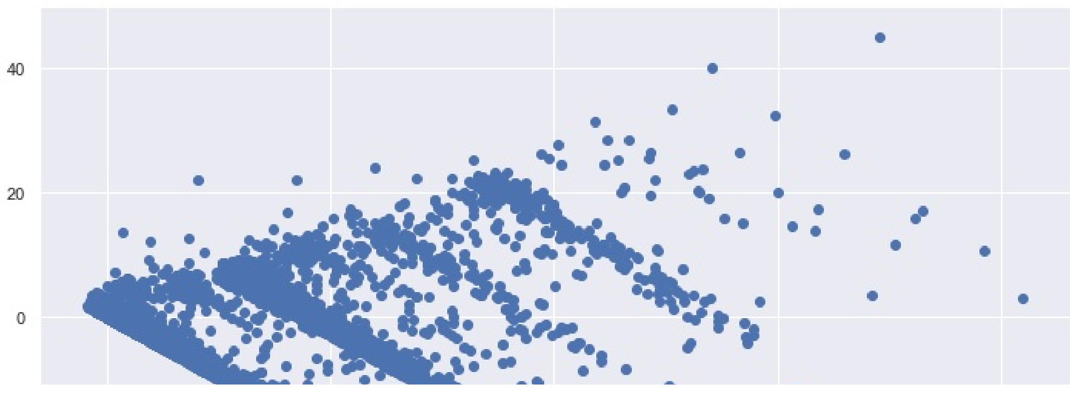
In [37]:
```
sns.displot(residuals,kind="kde")
```

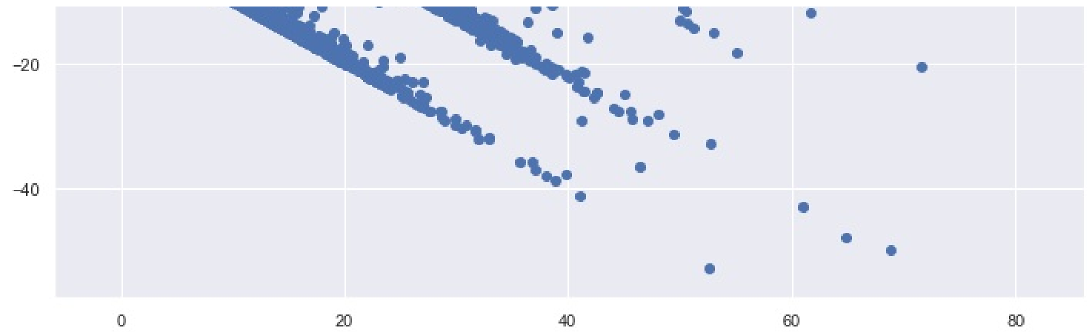
Out[37]: <seaborn.axisgrid.FacetGrid at 0x250ed8c1e80>



In [38]:
```
plt.scatter(reg_pred,residuals)
```

Out[38]: <matplotlib.collections.PathCollection at 0x250ede88190>





In [39]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,reg_pred))
print(mean_absolute_error(y_test,reg_pred))
print(np.sqrt(mean_squared_error(y_test,reg_pred)))
```

```
46.10869978933013
4.323198118017273
6.7903387094702525
```

In [40]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(score)
```

```
0.7155165877982992
```

In [41]:
```
# Adjusted R square
#display adjusted R squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

Out[41]:
```
0.715400163238364
```

RIDGE REGRESSION

In [42]:
```
ridge.fit(x_train,y_train)
```

Out[42]:
```
Ridge()
```

In [43]:
```
ridge.coef_
```

Out[43]:
```
array([ 24.42992919,  -0.07616832,  -0.36571069, -13.74591684])
```

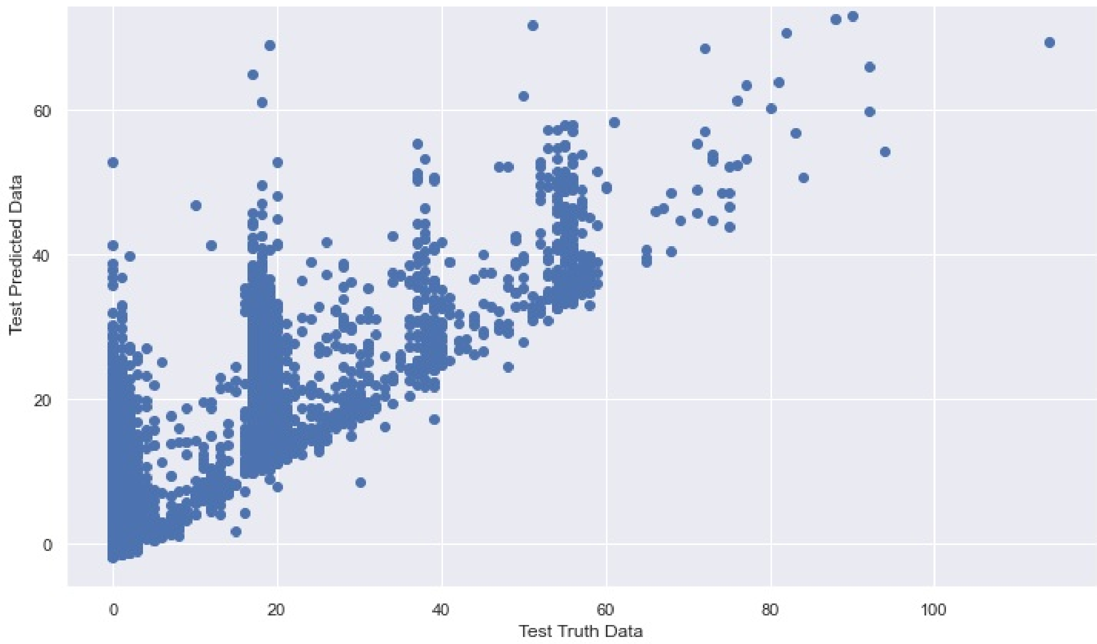
In [44]:
```
ridge.intercept_
```

Out[44]:
```
8.852566362766334
```

In [45]:
```
pred = ridge.predict(x_test)
```

In [46]:
```
plt.scatter(y_test,pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

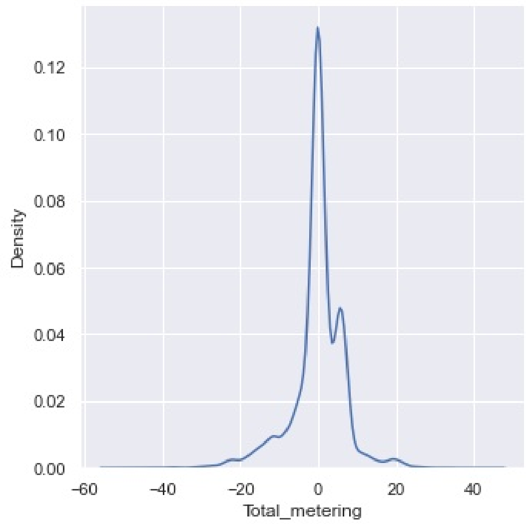
Out[46]:
```
Text(0, 0.5, 'Test Predicted Data')
```



In [47]:
```
rsd = y_test-pred
```

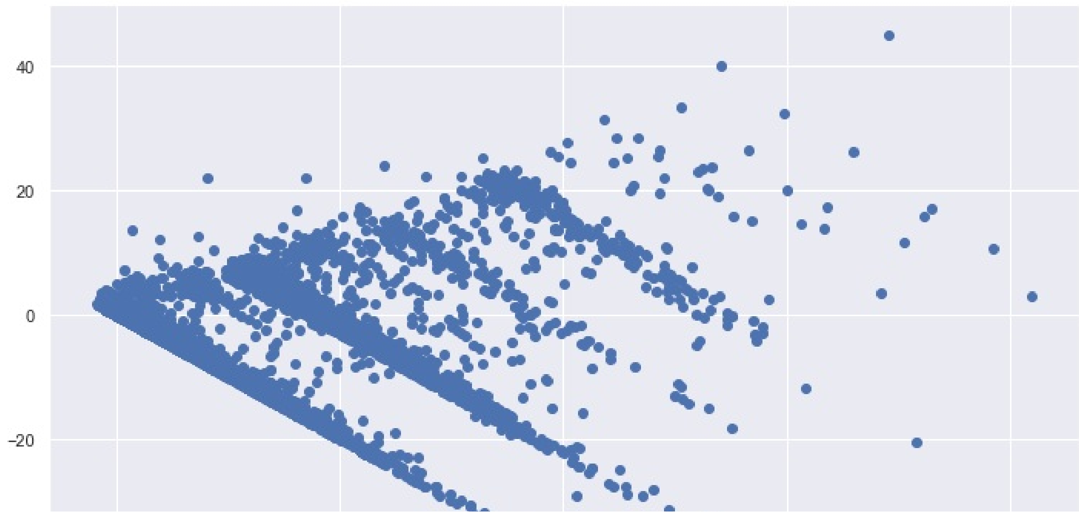
In [48]:
```
sns.displot(rsd,kind="kde")
```

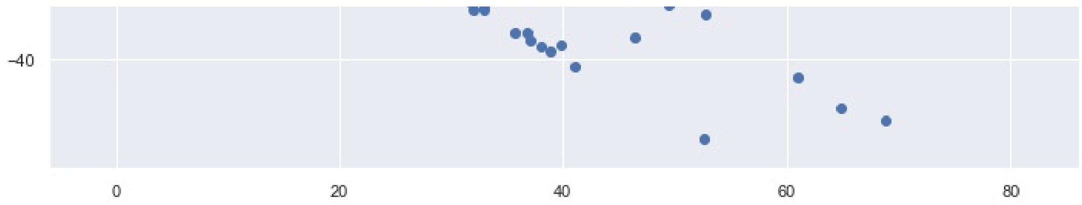
Out[48]: <seaborn.axisgrid.FacetGrid at 0x250ede9b460>



In [49]:
```
plt.scatter(reg_pred,residuals)
```

Out[49]: <matplotlib.collections.PathCollection at 0x250edf1a370>






```
In [50]: from sklearn.metrics import mean_squared_error
         from sklearn.metrics import mean_absolute_error
         print(mean_squared_error(y_test,pred))
         print(mean_absolute_error(y_test,pred))
         print(np.sqrt(mean_squared_error(y_test,pred)))
```

```
46.09706785152626
4.323342209713584
6.789482148995331
```

```
In [51]: from sklearn.metrics import r2_score
         score=r2_score(y_test,pred)
         print(score)
```

```
0.7155883550216666
```

```
In [52]: ## Adjusted R square
         #display adjusted R-squared
         1 - (1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

```
Out[52]: 0.7154719598323978
```

LASSO REGRESSION

```
In [53]: lasso.fit(x_train,y_train)
```

```
Out[53]: Lasso()
```

```
In [54]: lasso.coef_
```

```
Out[54]: array([ 9.82753708, -0.        , -0.        ,  0.        ])
```

```
In [55]: lasso.intercept_
```

```
Out[55]: 8.852566362766332
```

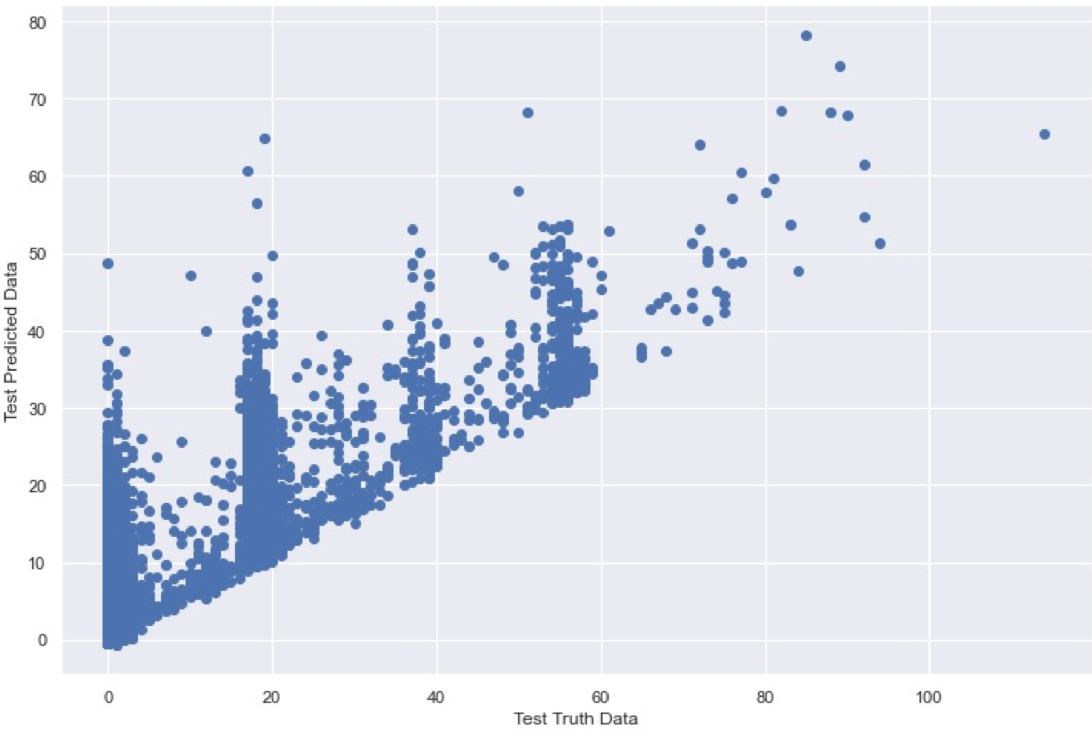
```
In [56]: predict = lasso.predict(x_test)
```

```
In [57]: predict
```

```
Out[57]: array([ 1.81663201, 12.29409092, 10.93796592, ..., 21.00672962,
                 0.66485461,  3.56287516])
```

```
In [58]: plt.scatter(y_test,predict)
         plt.xlabel("Test Truth Data")
         plt.ylabel("Test Predicted Data")
```

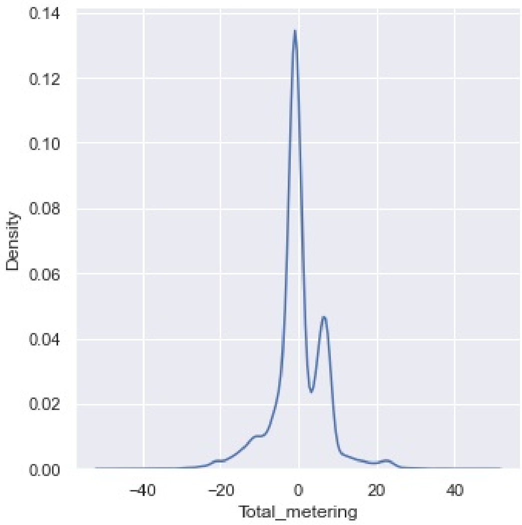
```
Out[58]: Text(0, 0.5, 'Test Predicted Data')
```



In [59]:
```
resd = y_test-predict
```

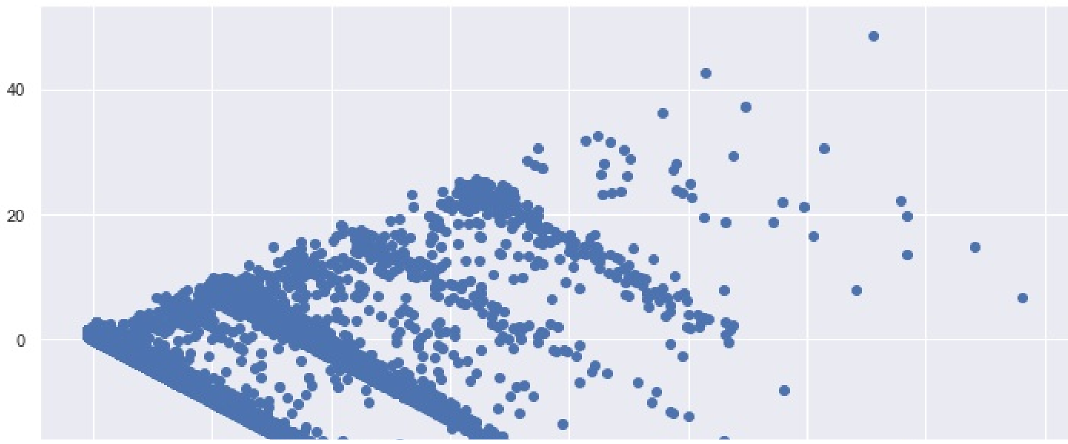
In [60]:
```
sns.displot(resd,kind="kde")
```

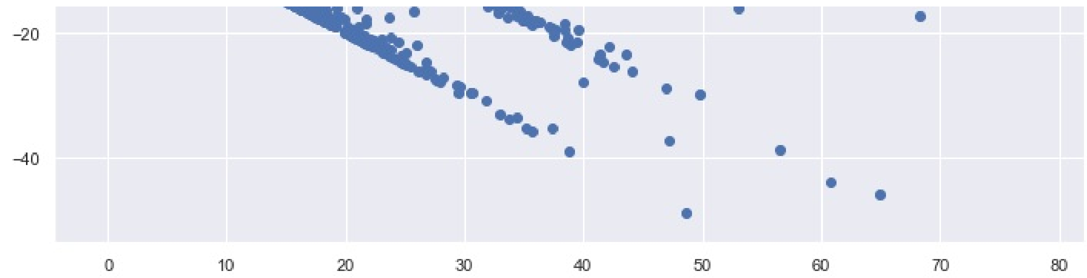
Out[60]: <seaborn.axisgrid.FacetGrid at 0x250edf667c0>



In [61]:
```
plt.scatter(predict,resd)
```

Out[61]: <matplotlib.collections.PathCollection at 0x250edda6190>





In [62]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,pred))
print(mean_absolute_error(y_test,pred))
print(np.sqrt(mean_squared_error(y_test,pred)))
```

```
46.09706785152626
4.323342209713584
6.789482148995331
```

In [63]:

```
from sklearn.metrics import r2_score
score=r2_score(y_test,predict)
print(score)
```

```
0.7075814285395345
```

In [64]:

```
## Adjusted R square
#display adjusted R-squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

Out[64]:

```
0.7074617565233854
```

ELASTIC-NET

In [65]:

```
EN.fit(x_train, y_train)
```

Out[65]:

```
ElasticNet()
```

In [66]:

```
EN.coef_
```

Out[66]:

```
array([ 4.09742862,  0.        , -0.46023824,  4.00264531])
```

In [67]:

```
EN.intercept_
```

Out[67]:

```
8.85256636276633
```

In [68]:

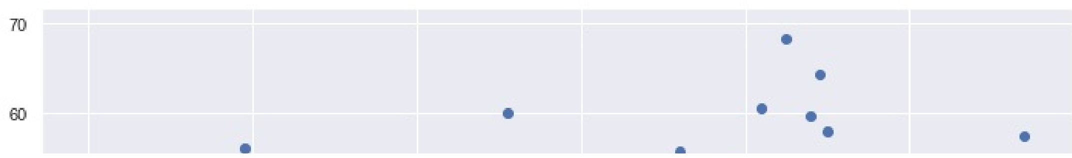
```
en_pred = EN.predict(x_test)
```

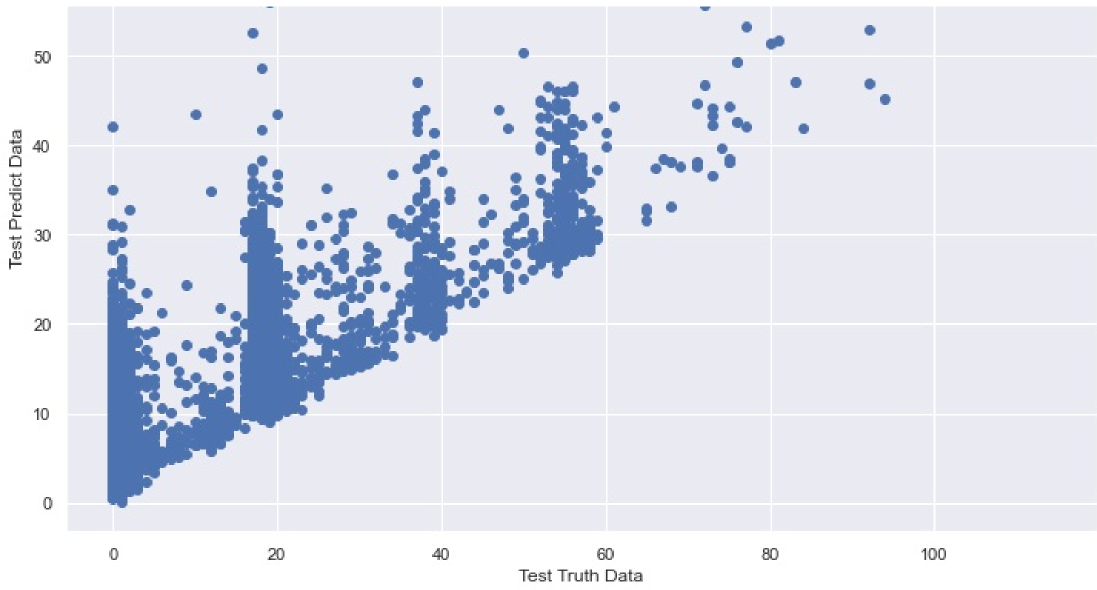
In [69]:

```
plt.scatter(y_test,en_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predict Data")
```

Out[69]:

```
Text(0, 0.5, 'Test Predict Data')
```

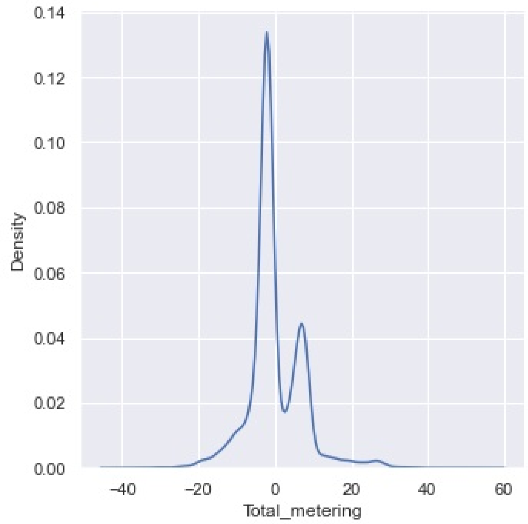




In [70]:
```
resi = y_test - en_pred
```

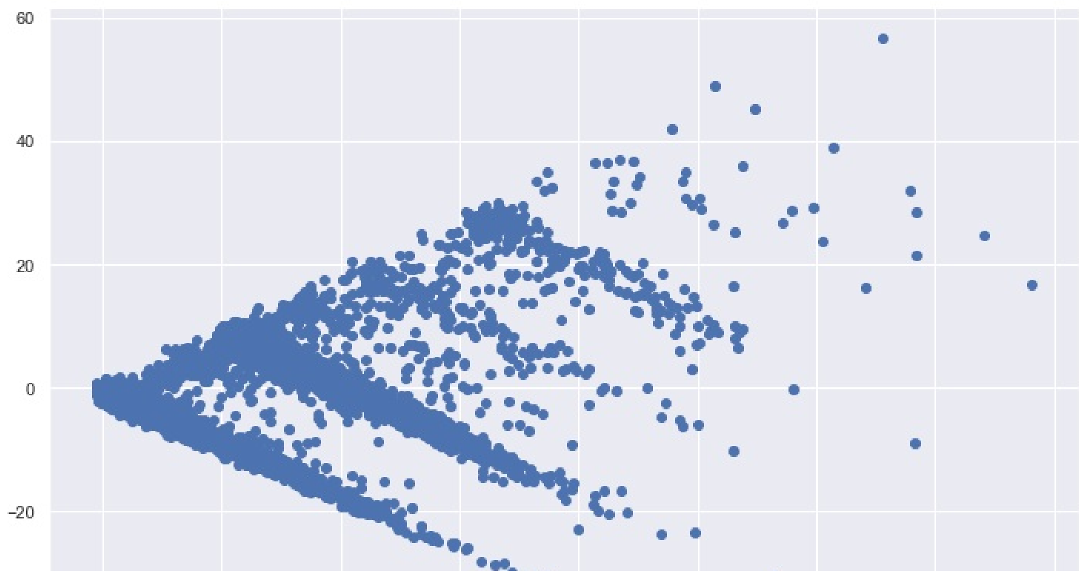
In [71]:
```
sns.displot(resi,kind="kde")
```

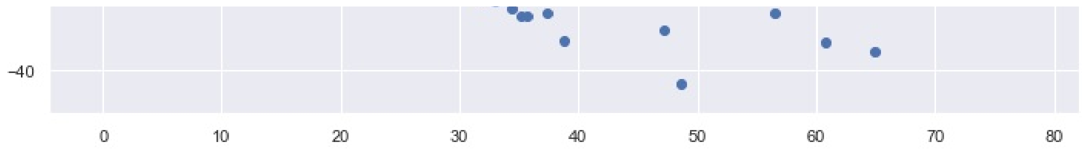
Out[71]: <seaborn.axisgrid.FacetGrid at 0x250ed39f5b0>



In [72]:
```
plt.scatter(predict,resi)
```

Out[72]: <matplotlib.collections.PathCollection at 0x250ec410df0>





In [73]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,predict))
print(mean_absolute_error(y_test,predict))
print(np.sqrt(mean_squared_error(y_test,predict)))
```

```
47.394820035186484
4.53180260066157
6.884389590601804
```

In [74]:

```
from sklearn.metrics import r2_score
score=r2_score(y_test,en_pred)
print(score)
```

```
0.6725304430977086
```

In [75]:

```
## Adjusted R square
#display adjusted R-squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

Out[75]:

```
0.6723964264998358
```

SUPPORT VECTOR REGRESSION

In [76]:

```
svr = SVR(kernel="rbf")
```

In [77]:

```
svr.fit(x_train, y_train)
```

Out[77]:

```
SVR()
```

In [78]:

```
prediction = svr.predict(x_test)
```

In [79]:

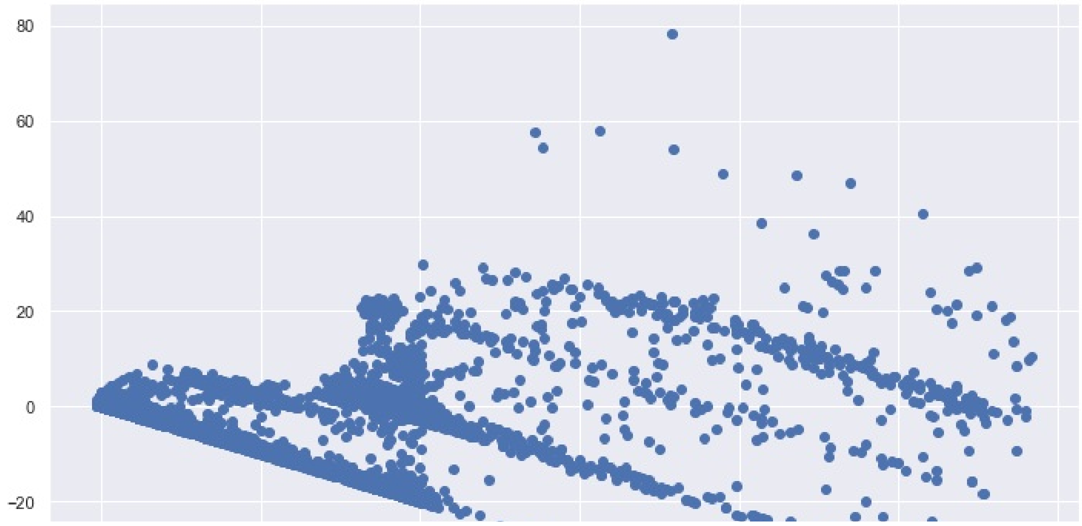
```
residual = y_test- prediction
```

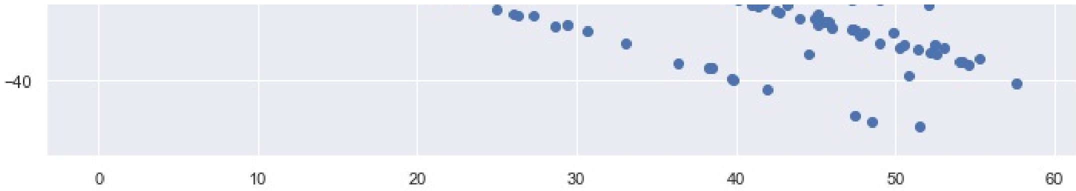
In [80]:

```
plt.scatter(prediction, residual)
```

Out[80]:

```
<matplotlib.collections.PathCollection at 0x250ec4dc610>
```





In [81]:
```
svr.score(x_test, y_test)
```
Out[81]:
```
0.7177480124788849
```

HYPERPARAMETER TUNING

In [82]:
```
from sklearn.model_selection import GridSearchCV
parameters = {'kernel':['rbf'], 'C':[1, 10]}
```
In [83]:
```
clf = GridSearchCV(svr, parameters)
```
In [84]:
```
clf.fit(x_train, y_train)
```
Out[84]:
```
GridSearchCV(estimator=SVR(), param_grid={'C': [1, 10], 'kernel': ['rbf']})
```
In [85]:
```
sorted(clf.cv_results_.keys())
```
Out[85]:
```
['mean_fit_time',
 'mean_score_time',
 'mean_test_score',
 'param_C',
 'param_kernel',
 'params',
 'rank_test_score',
 'split0_test_score',
 'split1_test_score',
 'split2_test_score',
 'split3_test_score',
 'split4_test_score',
 'std_fit_time',
 'std_score_time',
 'std_test_score']
```
In [86]:
```
y_pred = clf.predict(x_test)
```
In [87]:
```
accuracy = clf.best_score_ *100
```
In [88]:
```
accuracy
```
Out[88]:
```
72.72383387378987
```
In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js