Household_Power_Consumption_- Decision Tree Regression (GridSearchCV)

November 16, 2022

Import Necessary Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.impute import SimpleImputer
```

[2]:
```
df=pd.read_csv("household_power_consumption.txt",sep = ';',low_memory=False)
```

[3]:
```
df.head()
```

[3]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	16/12/2006	17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17.0
1	16/12/2006	17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16.0
2	16/12/2006	17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17.0
3	16/12/2006	17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17.0
4	16/12/2006	17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17.0

Statistical Info

[4]:
```
df.describe(include='all')
```

[4]:

	Date	Time	Global_active_power	Global_reactive_power
count	2075259	2075259	2075259	2075259
unique	1442	1440	4187	533
top	6/12/2008	17:24:00	?	0.000
freq	1440	1442	25979	481561
mean	NaN	NaN	NaN	NaN

```
std           NaN       NaN                NaN                NaN
min           NaN       NaN                NaN                NaN
25%           NaN       NaN                NaN                NaN
50%           NaN       NaN                NaN                NaN
75%           NaN       NaN                NaN                NaN
max           NaN       NaN                NaN                NaN

        Voltage Global_intensity Sub_metering_1 Sub_metering_2  Sub_metering_3
count   2075259          2075259        2075259        2075259    2.049280e+06
unique     2838              222             89             82             NaN
top           ?            1.000          0.000          0.000             NaN
freq      25979           172785        1880175        1436830             NaN
mean        NaN              NaN            NaN            NaN    6.458447e+00
std         NaN              NaN            NaN            NaN    8.437154e+00
min         NaN              NaN            NaN            NaN    0.000000e+00
25%         NaN              NaN            NaN            NaN    0.000000e+00
50%         NaN              NaN            NaN            NaN    1.000000e+00
75%         NaN              NaN            NaN            NaN    1.700000e+01
max         NaN              NaN            NaN            NaN    3.100000e+01
```

[5]:
```
df.describe().T
```

[5]:
```
                    count      mean       std  min  25%  50%   75%   max
Sub_metering_3  2049280.0  6.458447  8.437154  0.0  0.0  1.0  17.0  31.0
```

[6]:
```
df.isnull().any(axis = 1).sum()
```

[6]:
```
25979
```

[7]:
```
# fill missing values row wise and making the changes permanent in the original␣
 ↪dataframe

df.ffill(axis=0,inplace=True)
```

[8]:
```
# Feature Modification
df['Date'] = df['Date'].astype(str)
df['Time'] = df['Time'].astype(str)
df.replace(['?', 'nan', np.nan], -1, inplace=True)
num_vars= ['Global_active_power', 'Global_reactive_power', 'Voltage',
           'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',␣
 ↪'Sub_metering_3']
```

[9]:
```
for i in num_vars:
    df[i] = pd.to_numeric(df[i])
imp = SimpleImputer(missing_values=-1, strategy='mean')
df[num_vars] = imp.fit_transform(df[num_vars])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Date                   object
 1   Time                   object
 2   Global_active_power    float64
 3   Global_reactive_power  float64
 4   Voltage                float64
 5   Global_intensity       float64
 6   Sub_metering_1         float64
 7   Sub_metering_2         float64
 8   Sub_metering_3         float64
dtypes: float64(7), object(2)
memory usage: 142.5+ MB
```

Making a Target Variable using 3 main columns

[10]:
```
# Target Variable
eq1 = (df['Global_active_power']*1000/60)
eq2 = df['Sub_metering_1'] + df['Sub_metering_2'] + df['Sub_metering_3']
df['power_consumption'] = eq1 - eq2
df.head()
```

[10]:

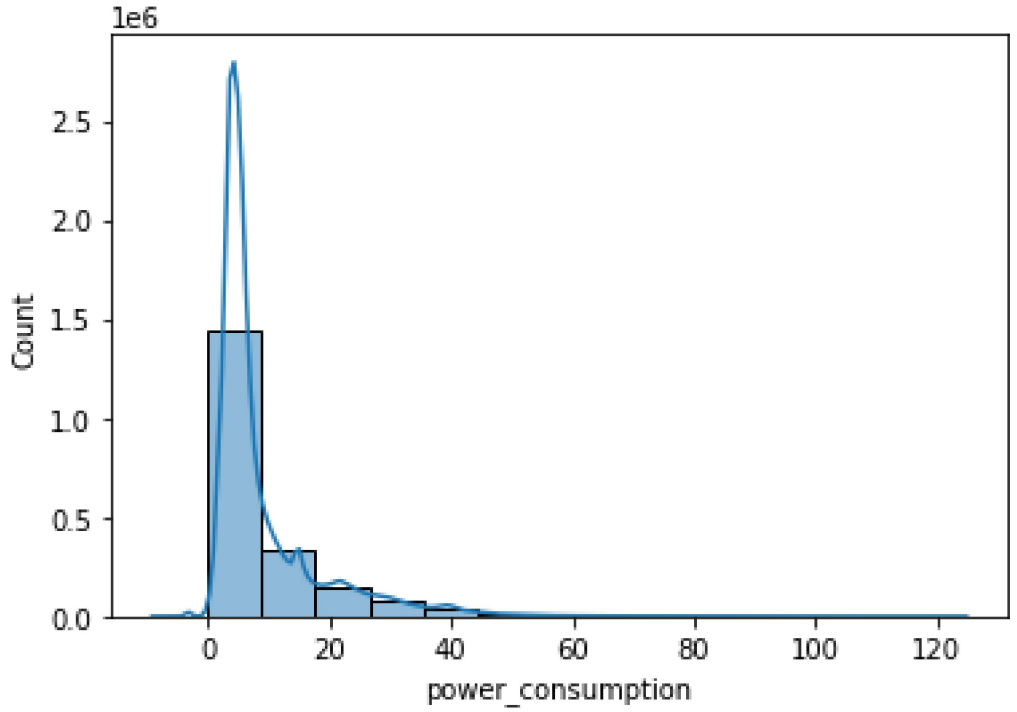
	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	power_consumption
0	16/12/2006	17:24:00	4.216	0.418	234.84	18.4	0.0	1.0	17.0	52.266667
1	16/12/2006	17:25:00	5.360	0.436	233.63	23.0	0.0	1.0	16.0	72.333333
2	16/12/2006	17:26:00	5.374	0.498	233.29	23.0	0.0	2.0	17.0	70.566667
3	16/12/2006	17:27:00	5.388	0.502	233.74	23.0	0.0	1.0	17.0	71.800000
4	16/12/2006	17:28:00	3.666	0.528	235.68	15.8	0.0	1.0	17.0	43.100000

[11]:
```
df.columns
```

[11]:
```
Index(['Date', 'Time', 'Global_active_power', 'Global_reactive_power',
       'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3', 'power_consumption'],
      dtype='object')
```

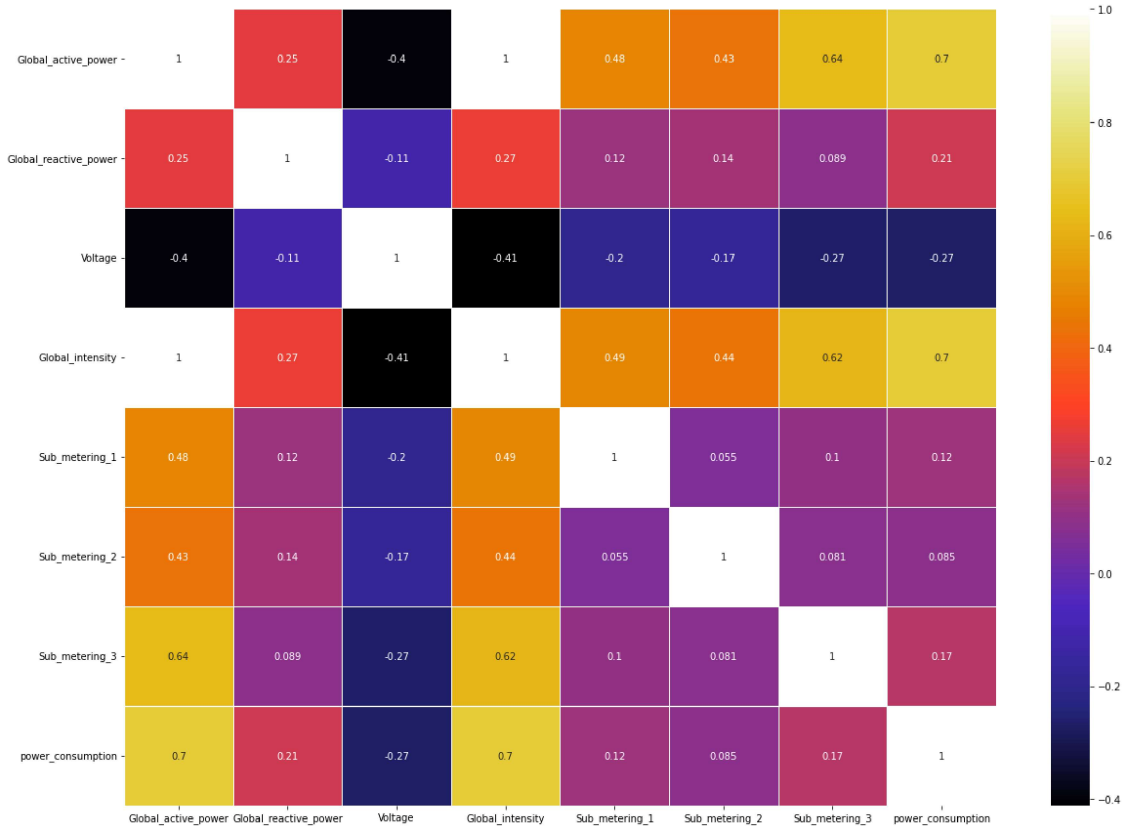
Exploratory Data Analysis

[12]:
```
#Distribution of the target variables
sns.histplot(data=df, x='power_consumption', bins=15, kde=True)
plt.show()
```



[13]:
```
# Heatmap
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot= True,linewidths=1, linecolor="white", cbar=True,
            cmap = "CMRmap",xticklabels="auto", yticklabels="auto")

plt.savefig('multi2.png')
```



[14]:
```
corr = df.corr(method='pearson')
print("Correlation of the Dataset:",corr)
```

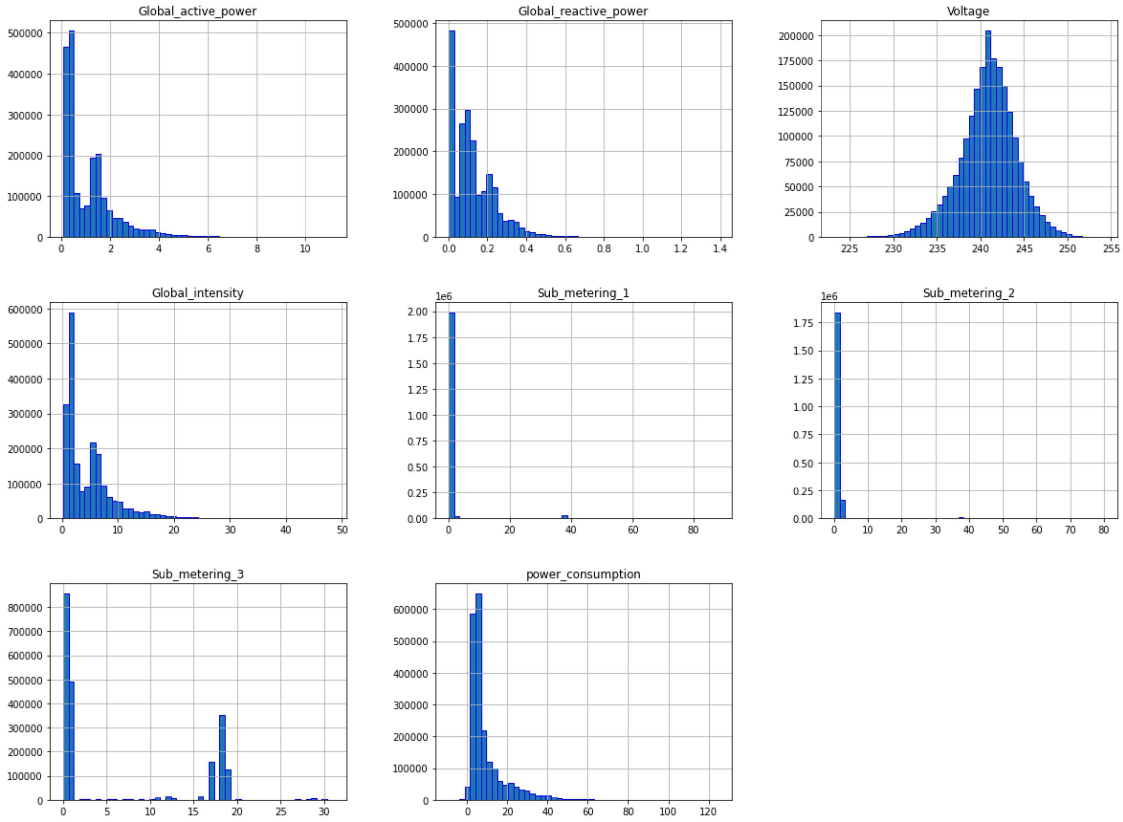
```
Correlation of the Dataset:                          Global_active_power
Global_reactive_power   Voltage  \
Global_active_power                 1.000000                0.247017 -0.399762
Global_reactive_power               0.247017                1.000000 -0.112246
Voltage                            -0.399762               -0.112246  1.000000
Global_intensity                    0.998889                0.266120 -0.411363
Sub_metering_1                      0.484401                0.123111 -0.195976
Sub_metering_2                      0.434569                0.139231 -0.167405
Sub_metering_3                      0.635876                0.089240 -0.267047
power_consumption                   0.699097                0.210935 -0.270488

                     Global_intensity  Sub_metering_1  Sub_metering_2  \
Global_active_power          0.998889        0.484401        0.434569
Global_reactive_power        0.266120        0.123111        0.139231
Voltage                     -0.411363       -0.195976       -0.167405
Global_intensity             1.000000        0.489298        0.440347
Sub_metering_1               0.489298        1.000000        0.054721
Sub_metering_2               0.440347        0.054721        1.000000
```

```
Sub_metering_3              0.623914        0.102141        0.080533
power_consumption           0.700969        0.124660        0.084923
```

	Sub_metering_3	power_consumption
Global_active_power	0.635876	0.699097
Global_reactive_power	0.089240	0.210935
Voltage	-0.267047	-0.270488
Global_intensity	0.623914	0.700969
Sub_metering_1	0.102141	0.124660
Sub_metering_2	0.080533	0.084923
Sub_metering_3	1.000000	0.170017
power_consumption	0.170017	1.000000

[15]:
```
df.hist(bins=50, figsize=(20,15), ec = 'b')
plt.show()
```



Splitting the data into Features and Label & then performing train_test_split

[16]:
```
X = df.drop(['power_consumption','Date','Time'], axis=1).values
Y = df['power_consumption'].values
```

[17]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
```

[18]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.
↪33,random_state=10)
```

[19]:
```
'''from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler
sclr = StandardScaler()
scr.fit()
scr.transform()'''
```

[19]:
```
'from sklearn.preprocessing import StandardScaler\nfrom sklearn.preprocessing
import StandardScaler\nsclr = StandardScaler()\nscr.fit()\nscr.transform()'
```

Model Training and Model Building

[20]:
```
from sklearn.tree import DecisionTreeRegressor
model=DecisionTreeRegressor()
```

[21]:
```
model.fit(X_train,y_train)
```

[21]:
```
DecisionTreeRegressor()
```

[22]:
```
model.score(X_train,y_train)
```

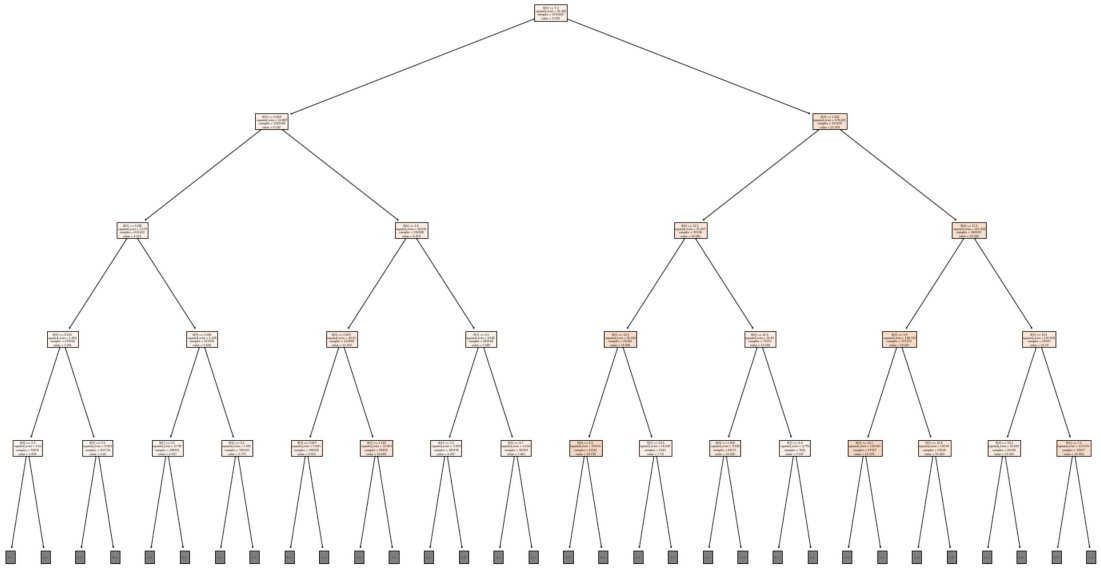
[22]:
```
1.0
```

[23]:
```
from sklearn import tree
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model, max_depth=4,filled=True)
```

[23]:
```
[Text(0.5, 0.9166666666666666, 'X[3] <= 7.3\nsquared_error = 91.382\nsamples =
1390423\nvalue = 9.359'),
 Text(0.25, 0.75, 'X[0] <= 0.459\nsquared_error = 22.887\nsamples =
1128188\nvalue = 6.342'),
 Text(0.125, 0.5833333333333334, 'X[0] <= 0.281\nsquared_error = 2.175\nsamples
= 612140\nvalue = 4.118'),
 Text(0.0625, 0.4166666666666667, 'X[0] <= 0.193\nsquared_error = 1.014\nsamples
= 278164\nvalue = 3.001'),
 Text(0.03125, 0.25, 'X[6] <= 0.5\nsquared_error = 0.52\nsamples = 76430\nvalue
= 1.895'),
 Text(0.015625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.046875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.09375, 0.25, 'X[5] <= 0.5\nsquared_error = 0.563\nsamples =
201734\nvalue = 3.42'),
 Text(0.078125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.109375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.1875, 0.4166666666666667, 'X[0] <= 0.361\nsquared_error = 1.236\nsamples
= 333976\nvalue = 5.049'),
```

```
 Text(0.15625, 0.25, 'X[5] <= 0.5\nsquared_error = 0.799\nsamples =
198651\nvalue = 4.557'),
 Text(0.140625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.171875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.21875, 0.25, 'X[5] <= 0.5\nsquared_error = 1.001\nsamples =
135325\nvalue = 5.771'),
 Text(0.203125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.234375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.375, 0.5833333333333334, 'X[6] <= 2.5\nsquared_error = 34.635\nsamples =
516048\nvalue = 8.979'),
 Text(0.3125, 0.4166666666666667, 'X[0] <= 0.919\nsquared_error = 36.01\nsamples
= 229899\nvalue = 13.323'),
 Text(0.28125, 0.25, 'X[0] <= 0.657\nsquared_error = 5.841\nsamples =
138030\nvalue = 9.353'),
 Text(0.265625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.296875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.34375, 0.25, 'X[0] <= 1.245\nsquared_error = 22.063\nsamples =
91869\nvalue = 19.289'),
 Text(0.328125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.359375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.4375, 0.4166666666666667, 'X[3] <= 6.1\nsquared_error = 6.185\nsamples =
286149\nvalue = 5.489'),
 Text(0.40625, 0.25, 'X[3] <= 5.5\nsquared_error = 3.059\nsamples =
189190\nvalue = 4.297'),
 Text(0.390625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.421875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.46875, 0.25, 'X[3] <= 6.7\nsquared_error = 4.104\nsamples = 96959\nvalue
= 7.815'),
 Text(0.453125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.484375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.75, 0.75, 'X[0] <= 2.261\nsquared_error = 178.423\nsamples =
262235\nvalue = 22.339'),
 Text(0.625, 0.5833333333333334, 'X[6] <= 11.5\nsquared_error = 71.267\nsamples
= 95596\nvalue = 16.685'),
 Text(0.5625, 0.4166666666666667, 'X[5] <= 10.5\nsquared_error = 91.024\nsamples
= 24384\nvalue = 26.956'),
 Text(0.53125, 0.25, 'X[4] <= 8.5\nsquared_error = 39.074\nsamples =
21241\nvalue = 29.799'),
 Text(0.515625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.546875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.59375, 0.25, 'X[5] <= 20.5\nsquared_error = 18.218\nsamples =
3143\nvalue = 7.74'),
 Text(0.578125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.609375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.6875, 0.4166666666666667, 'X[6] <= 21.5\nsquared_error = 16.01\nsamples
= 71212\nvalue = 13.168'),
 Text(0.65625, 0.25, 'X[0] <= 1.969\nsquared_error = 9.506\nsamples =
```

```
64171\nvalue = 14.043'),
 Text(0.640625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.671875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.71875, 0.25, 'X[3] <= 8.9\nsquared_error = 4.791\nsamples = 7041\nvalue
= 5.197'),
 Text(0.703125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.734375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.875, 0.5833333333333334, 'X[4] <= 11.5\nsquared_error = 211.038\nsamples
= 166639\nvalue = 25.582'),
 Text(0.8125, 0.4166666666666667, 'X[5] <= 9.5\nsquared_error = 194.712\nsamples
= 127372\nvalue = 28.447'),
 Text(0.78125, 0.25, 'X[3] <= 14.1\nsquared_error = 143.922\nsamples =
97767\nvalue = 32.379'),
 Text(0.765625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.796875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.84375, 0.25, 'X[3] <= 16.9\nsquared_error = 142.81\nsamples =
29605\nvalue = 15.463'),
 Text(0.828125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.859375, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.9375, 0.4166666666666667, 'X[3] <= 18.1\nsquared_error =
151.016\nsamples = 39267\nvalue = 16.29'),
 Text(0.90625, 0.25, 'X[3] <= 16.3\nsquared_error = 55.632\nsamples =
26290\nvalue = 11.025'),
 Text(0.890625, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.921875, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.96875, 0.25, 'X[5] <= 7.5\nsquared_error = 174.372\nsamples =
12977\nvalue = 26.954'),
 Text(0.953125, 0.08333333333333333, '\n  (…)  \n'),
 Text(0.984375, 0.08333333333333333, '\n  (…)  \n')]
```



[24]:
```
fig.savefig("decistion_tree_classifier.png")
```

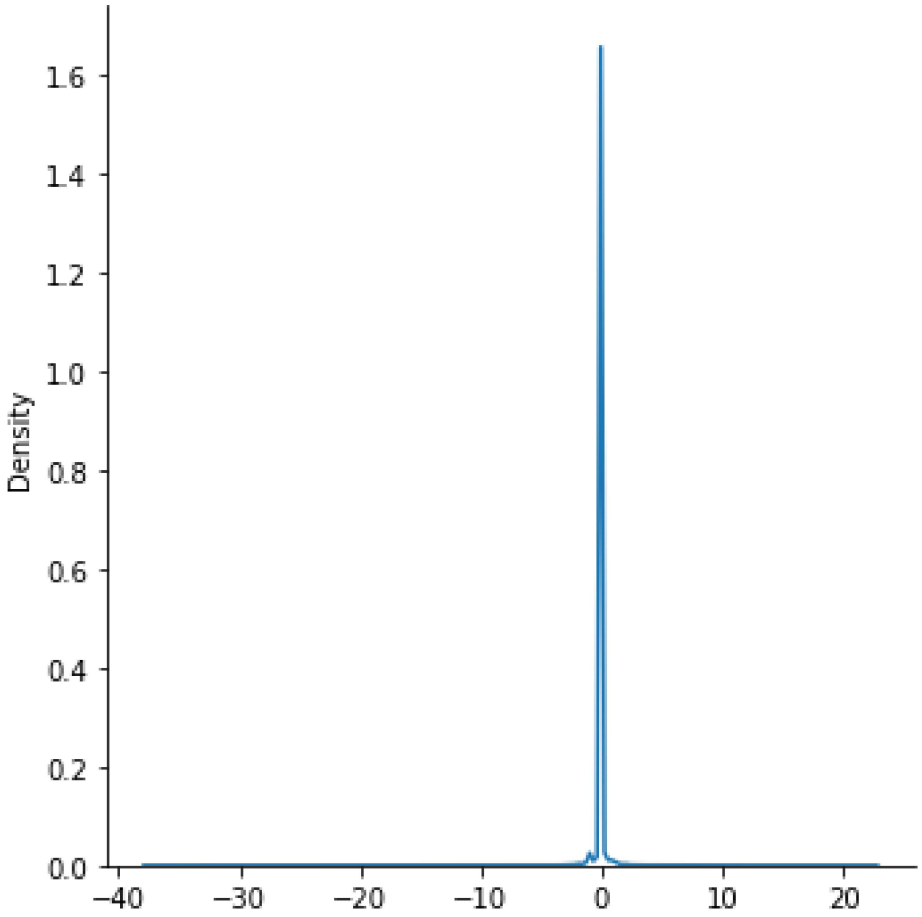
[25]:
```
y_predict=model.predict(X_test)
```

[26]:
```
## residuals
residuals=y_test-y_predict
residuals
```

[26]:
```
array([ 9.81437154e-14,  3.05533376e-13, -2.30926389e-14, …,
       -1.77635684e-15, -7.10542736e-15, -5.32907052e-15])
```

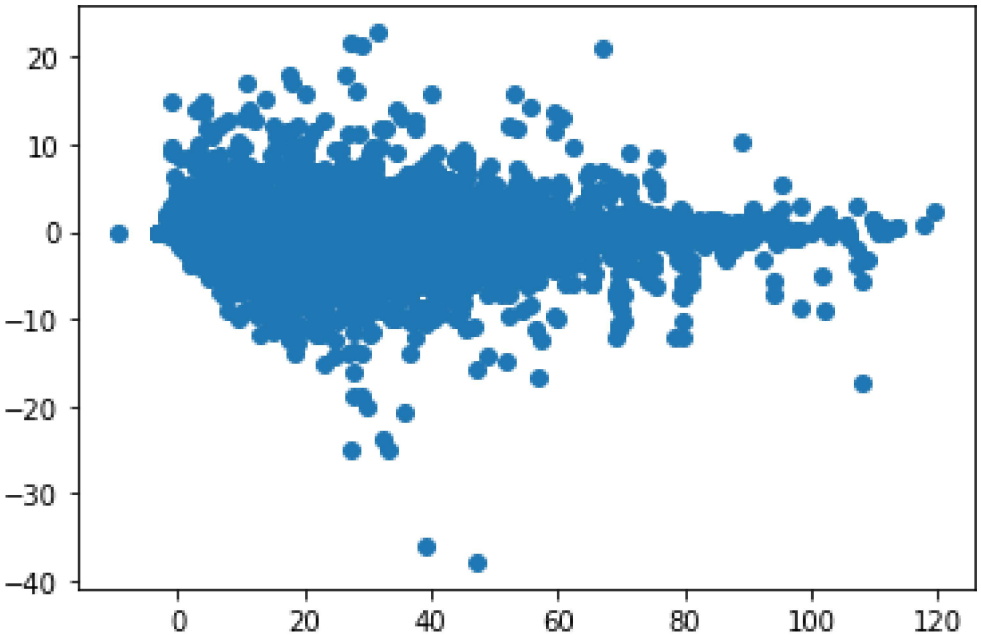
[27]:
```
sns.displot(residuals,kind="kde")
```

[27]:
```
<seaborn.axisgrid.FacetGrid at 0x1e7488b2c20>
```



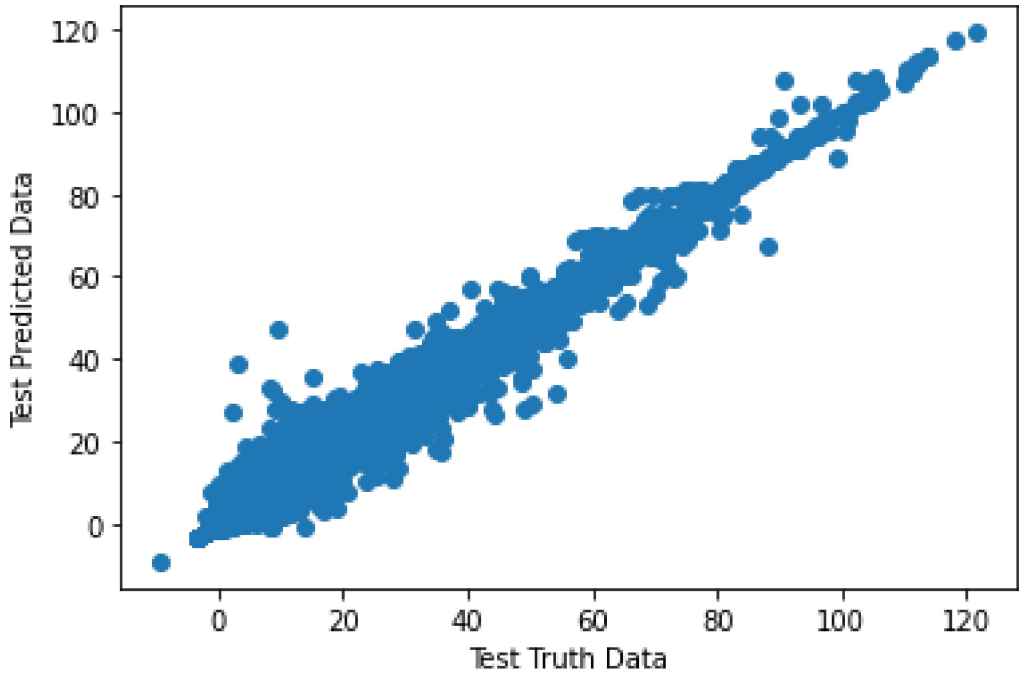
[28]:
```
plt.scatter(y_predict,residuals)
```

[28]:
```
<matplotlib.collections.PathCollection at 0x1e7487b5360>
```



[29]:
```
plt.scatter(y_test,y_predict)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

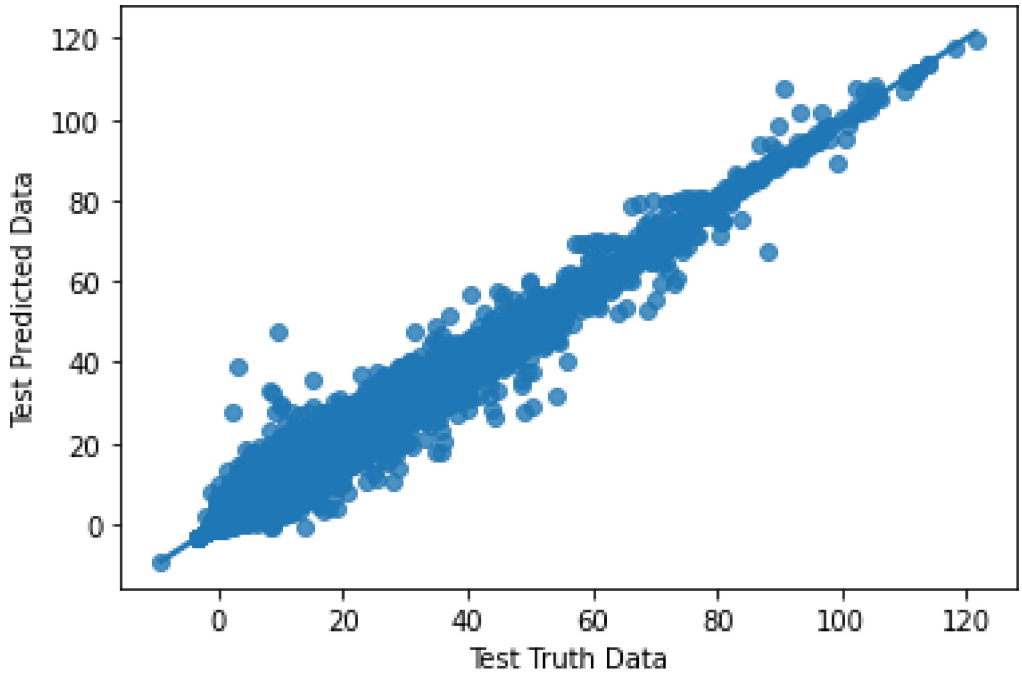
[29]: Text(0, 0.5, 'Test Predicted Data')



[30]:
```
sns.regplot(y_test,y_predict)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. 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(
```

[30]:
```
Text(0, 0.5, 'Test Predicted Data')
```



[31]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import explained_variance_score
from sklearn.metrics import r2_score

## Performance Metrics
from sklearn.metrics import mean_squared_error   ## MSE
from sklearn.metrics import mean_absolute_error  ## MAE
print(mean_squared_error(y_test,y_predict))
```

```
print(mean_absolute_error(y_test,y_predict))
print(np.sqrt(mean_squared_error(y_test,y_predict)))
```

```
0.13670053203329782
0.04970382203430955
0.3697303504356896
```

[32]:
```
# grid_param = {
#     'criterion': ['gini', 'entropy'],
#     'max_depth' : range(2,10,1),
#     'min_samples_leaf' : range(1,8,1),
#     'min_samples_split': range(2,8,1),
#     'splitter' : ['best', 'random']

# }
```

Improving score through Hyperparameter Tuning

[33]:
```
# Hyper parameters range intialization for tuning

parameters={"splitter":["best","random"],
            "max_depth" : [1,3,5,7,9,11,12],
           "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
           "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
           "max_features":["auto","log2","sqrt",None],
           "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90] }
```

[35]:
```
from sklearn.model_selection import GridSearchCV
grid_searh=GridSearchCV(estimator=model,param_grid=parameters,cv=3,verbose=1)
```

[36]:
```
grid_searh.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 50400 candidates, totalling 151200 fits
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
```

```
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
```

```
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py:306: FutureWarning: `max_features='auto'` has
been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0'`.
  warnings.warn(
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:378: FitFailedWarning:
67200 fits failed out of a total of 151200.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
16800 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 1342, in fit
    super().fit(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 286, in fit
    check_scalar(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py", line 1489, in check_scalar
    raise ValueError(
ValueError: min_weight_fraction_leaf == 0.6, must be <= 0.5.

--------------------------------------------------------------------------------
16800 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 1342, in fit
    super().fit(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 286, in fit
    check_scalar(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py", line 1489, in check_scalar
```

```
    raise ValueError(
ValueError: min_weight_fraction_leaf == 0.7, must be <= 0.5.

--------------------------------------------------------------------------------
16800 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 1342, in fit
    super().fit(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 286, in fit
    check_scalar(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py", line 1489, in check_scalar
    raise ValueError(
ValueError: min_weight_fraction_leaf == 0.8, must be <= 0.5.

--------------------------------------------------------------------------------
16800 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 1342, in fit
    super().fit(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\tree\_classes.py", line 286, in fit
    check_scalar(
  File "C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py", line 1489, in check_scalar
    raise ValueError(
ValueError: min_weight_fraction_leaf == 0.9, must be <= 0.5.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:953: UserWarning: One or more of the
test scores are non-finite: [0.42836754 0.01063067 0.42278095 …        nan
nan        nan]
  warnings.warn(
```

[36]:
```
GridSearchCV(cv=3, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 12],
                         'max_features': ['auto', 'log2', 'sqrt', None],
```

```
                            'max_leaf_nodes': [None, 10, 20, 30, 40, 50, 60, 70,
                                               80, 90],
                            'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                            'min_weight_fraction_leaf': [0.1, 0.2, 0.3, 0.4, 0.5,
                                                         0.6, 0.7, 0.8, 0.9],
                            'splitter': ['best', 'random']},
                   verbose=1)
```

[37]:
```
grid_searh.best_params_
```

[37]:
```
{'max_depth': 5,
 'max_features': 'sqrt',
 'max_leaf_nodes': 50,
 'min_samples_leaf': 6,
 'min_weight_fraction_leaf': 0.1,
 'splitter': 'best'}
```

[38]:
```
model_with_best_params=DecisionTreeRegressor(max_depth= 5,min_samples_leaf=␣
↪3,min_samples_split= 5,splitter='random')
```

[39]:
```
model_with_best_params.fit(X_train,y_train)
```

[39]:
```
DecisionTreeRegressor(max_depth=5, min_samples_leaf=3, min_samples_split=5,
                      splitter='random')
```

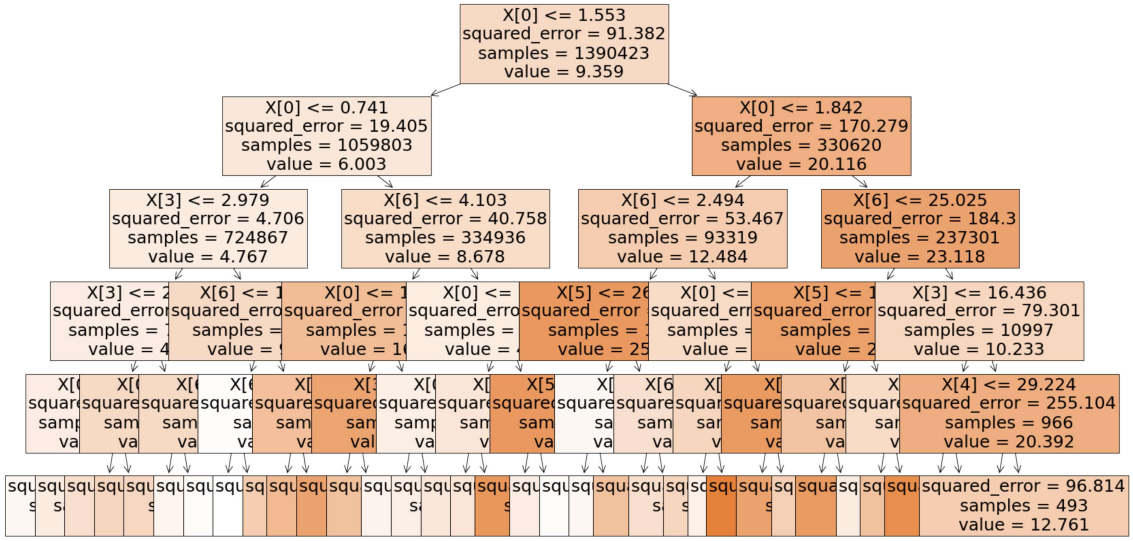
[40]:
```
from sklearn import tree
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model_with_best_params,filled=True,fontsize=25)
```

[40]:
```
[Text(0.5, 0.9166666666666666, 'X[0] <= 1.553\nsquared_error = 91.382\nsamples =
1390423\nvalue = 9.359'),
 Text(0.25, 0.75, 'X[0] <= 0.741\nsquared_error = 19.405\nsamples =
1059803\nvalue = 6.003'),
 Text(0.125, 0.5833333333333334, 'X[3] <= 2.979\nsquared_error = 4.706\nsamples
= 724867\nvalue = 4.767'),
 Text(0.0625, 0.4166666666666667, 'X[3] <= 2.727\nsquared_error = 4.065\nsamples
= 706416\nvalue = 4.646'),
 Text(0.03125, 0.25, 'X[0] <= 0.191\nsquared_error = 3.54\nsamples =
690881\nvalue = 4.539'),
 Text(0.015625, 0.08333333333333333, 'squared_error = 0.512\nsamples =
74808\nvalue = 1.879'),
 Text(0.046875, 0.08333333333333333, 'squared_error = 2.945\nsamples =
616073\nvalue = 4.862'),
 Text(0.09375, 0.25, 'X[0] <= 0.629\nsquared_error = 3.933\nsamples =
15535\nvalue = 9.436'),
 Text(0.078125, 0.08333333333333333, 'squared_error = 4.444\nsamples =
```

```
5582\nvalue = 7.911'),
 Text(0.109375, 0.08333333333333333, 'squared_error = 1.611\nsamples =
9953\nvalue = 10.291'),
 Text(0.1875, 0.4166666666666667, 'X[6] <= 10.226\nsquared_error =
7.357\nsamples = 18451\nvalue = 9.387'),
 Text(0.15625, 0.25, 'X[6] <= 6.492\nsquared_error = 6.888\nsamples =
18326\nvalue = 9.446'),
 Text(0.140625, 0.08333333333333333, 'squared_error = 4.149\nsamples =
17153\nvalue = 9.893'),
 Text(0.171875, 0.08333333333333333, 'squared_error = 1.303\nsamples =
1173\nvalue = 2.91'),
 Text(0.21875, 0.25, 'X[6] <= 11.703\nsquared_error = 0.223\nsamples =
125\nvalue = 0.697'),
 Text(0.203125, 0.08333333333333333, 'squared_error = 0.112\nsamples = 92\nvalue
= 0.912'),
 Text(0.234375, 0.08333333333333333, 'squared_error = 0.048\nsamples = 33\nvalue
= 0.1'),
 Text(0.375, 0.5833333333333334, 'X[6] <= 4.103\nsquared_error = 40.758\nsamples
= 334936\nvalue = 8.678'),
 Text(0.3125, 0.4166666666666667, 'X[0] <= 1.444\nsquared_error =
18.537\nsamples = 110347\nvalue = 16.783'),
 Text(0.28125, 0.25, 'X[0] <= 0.896\nsquared_error = 14.429\nsamples =
100749\nvalue = 16.19'),
 Text(0.265625, 0.08333333333333333, 'squared_error = 2.328\nsamples =
26445\nvalue = 12.488'),
 Text(0.296875, 0.08333333333333333, 'squared_error = 12.123\nsamples =
74304\nvalue = 17.507'),
 Text(0.34375, 0.25, 'X[3] <= 7.414\nsquared_error = 19.116\nsamples =
9598\nvalue = 23.015'),
 Text(0.328125, 0.08333333333333333, 'squared_error = 15.966\nsamples =
9348\nvalue = 23.213'),
 Text(0.359375, 0.08333333333333333, 'squared_error = 80.681\nsamples =
250\nvalue = 15.616'),
 Text(0.4375, 0.4166666666666667, 'X[0] <= 1.487\nsquared_error = 3.535\nsamples
= 224589\nvalue = 4.695'),
 Text(0.40625, 0.25, 'X[0] <= 0.925\nsquared_error = 3.103\nsamples =
192254\nvalue = 4.378'),
 Text(0.390625, 0.08333333333333333, 'squared_error = 2.523\nsamples =
8562\nvalue = 3.158'),
 Text(0.421875, 0.08333333333333333, 'squared_error = 3.058\nsamples =
183692\nvalue = 4.435'),
 Text(0.46875, 0.25, 'X[0] <= 1.499\nsquared_error = 1.949\nsamples =
32335\nvalue = 6.58'),
 Text(0.453125, 0.08333333333333333, 'squared_error = 1.659\nsamples =
6555\nvalue = 6.16'),
 Text(0.484375, 0.08333333333333333, 'squared_error = 1.966\nsamples =
25780\nvalue = 6.687'),
```

```
 Text(0.75, 0.75, 'X[0] <= 1.842\nsquared_error = 170.279\nsamples =
330620\nvalue = 20.116'),
 Text(0.625, 0.5833333333333334, 'X[6] <= 2.494\nsquared_error = 53.467\nsamples
= 93319\nvalue = 12.484'),
 Text(0.5625, 0.4166666666666667, 'X[5] <= 26.151\nsquared_error =
37.093\nsamples = 18560\nvalue = 25.489'),
 Text(0.53125, 0.25, 'X[5] <= 21.072\nsquared_error = 35.516\nsamples =
18504\nvalue = 25.56'),
 Text(0.515625, 0.08333333333333333, 'squared_error = 21.346\nsamples =
17901\nvalue = 26.266'),
 Text(0.546875, 0.08333333333333333, 'squared_error = 2.659\nsamples =
603\nvalue = 4.614'),
 Text(0.59375, 0.25, 'X[0] <= 1.73\nsquared_error = 1.047\nsamples = 56\nvalue =
1.924'),
 Text(0.578125, 0.08333333333333333, 'squared_error = 0.202\nsamples = 8\nvalue
= 0.9'),
 Text(0.609375, 0.08333333333333333, 'squared_error = 0.984\nsamples = 48\nvalue
= 2.095'),
 Text(0.6875, 0.4166666666666667, 'X[0] <= 1.669\nsquared_error = 5.12\nsamples
= 74759\nvalue = 9.256'),
 Text(0.65625, 0.25, 'X[6] <= 14.433\nsquared_error = 2.953\nsamples =
36858\nvalue = 8.068'),
 Text(0.640625, 0.08333333333333333, 'squared_error = 10.644\nsamples =
565\nvalue = 15.494'),
 Text(0.671875, 0.08333333333333333, 'squared_error = 1.962\nsamples =
36293\nvalue = 7.953'),
 Text(0.71875, 0.25, 'X[6] <= 22.633\nsquared_error = 4.525\nsamples =
37901\nvalue = 10.41'),
 Text(0.703125, 0.08333333333333333, 'squared_error = 3.85\nsamples =
37294\nvalue = 10.517'),
 Text(0.734375, 0.08333333333333333, 'squared_error = 2.373\nsamples =
607\nvalue = 3.861'),
 Text(0.875, 0.5833333333333334, 'X[6] <= 25.025\nsquared_error = 184.3\nsamples
= 237301\nvalue = 23.118'),
 Text(0.8125, 0.4166666666666667, 'X[5] <= 16.368\nsquared_error =
180.942\nsamples = 226304\nvalue = 23.744'),
 Text(0.78125, 0.25, 'X[6] <= 12.076\nsquared_error = 172.561\nsamples =
192707\nvalue = 25.268'),
 Text(0.765625, 0.08333333333333333, 'squared_error = 221.456\nsamples =
45642\nvalue = 31.773'),
 Text(0.796875, 0.08333333333333333, 'squared_error = 140.179\nsamples =
147065\nvalue = 23.249'),
 Text(0.84375, 0.25, 'X[0] <= 4.773\nsquared_error = 139.259\nsamples =
33597\nvalue = 15.001'),
 Text(0.828125, 0.08333333333333333, 'squared_error = 76.292\nsamples =
26409\nvalue = 12.243'),
 Text(0.859375, 0.08333333333333333, 'squared_error = 239.918\nsamples =
```

```
7188\nvalue = 25.136'),
 Text(0.9375, 0.4166666666666667, 'X[3] <= 16.436\nsquared_error =
79.301\nsamples = 10997\nvalue = 10.233'),
 Text(0.90625, 0.25, 'X[3] <= 9.403\nsquared_error = 51.475\nsamples =
10031\nvalue = 9.254'),
 Text(0.890625, 0.08333333333333333, 'squared_error = 3.279\nsamples =
5979\nvalue = 4.947'),
 Text(0.921875, 0.08333333333333333, 'squared_error = 54.826\nsamples =
4052\nvalue = 15.61'),
 Text(0.96875, 0.25, 'X[4] <= 29.224\nsquared_error = 255.104\nsamples =
966\nvalue = 20.392'),
 Text(0.953125, 0.08333333333333333, 'squared_error = 296.139\nsamples =
473\nvalue = 28.345'),
 Text(0.984375, 0.08333333333333333, 'squared_error = 96.814\nsamples =
493\nvalue = 12.761')]
```

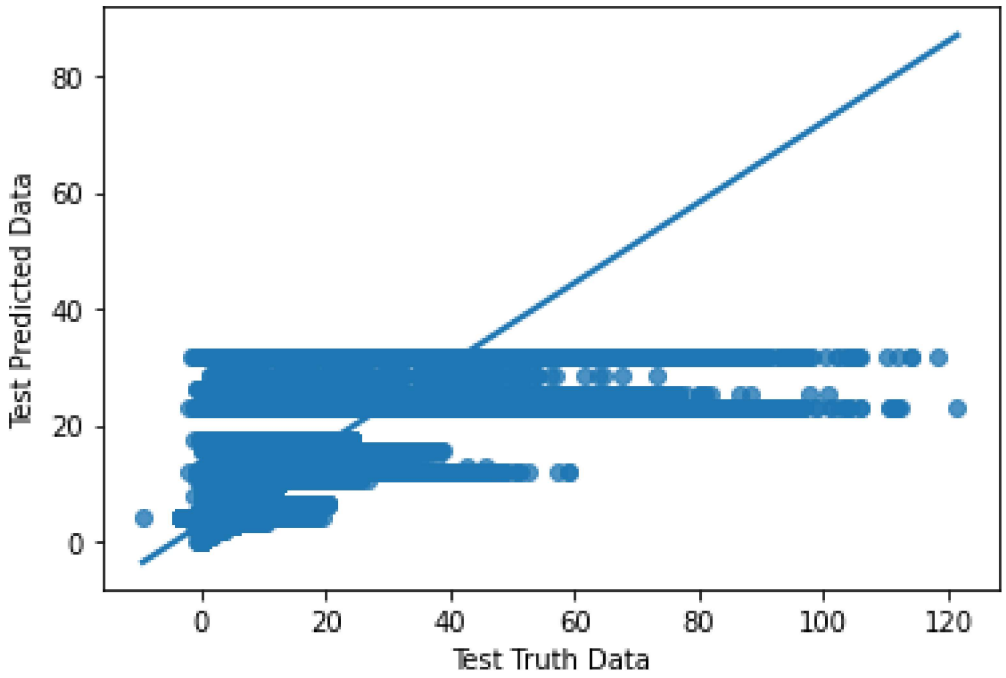


[45]:
```
y_prediction2=model_with_best_params.predict(X_test)
```

[47]:
```
sns.regplot(y_test,y_prediction2)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

```
C:\Users\Shobhandeb\AppData\Local\Programs\Python\Python310\lib\site-
packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. 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(
```

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



[46]:
```
## Performance Metrics
from sklearn.metrics import mean_squared_error   ## MSE
from sklearn.metrics import mean_absolute_error  ## MAE
print(mean_squared_error(y_test,y_prediction2))
print(mean_absolute_error(y_test,y_prediction2))
print(np.sqrt(mean_squared_error(y_test,y_prediction2)))
```

```
28.10619532659214
2.6829304714347084
5.301527640840151
```

[ ]: