```
#import libraries
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

```
#read the dataset
df = pd.read_csv("/content/WEC_Sydney_100.csv")
```

```
#data analysis
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2318 entries, 0 to 2317
Columns: 302 entries, X1 to Total_Power
dtypes: float64(302)
memory usage: 5.3 MB
```

```
#print the first five rows of df
df.head()
```

	X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5	X6	Y6	
0	1.0	1.0	1.00	51.00	1.00	101.00	1.00	151.0	398.0	0.0	397.46	75.07	397.
1	198.0	0.0	197.18	80.53	193.59	150.00	77.58	198.0	598.0	0.0	597.18	80.53	593.
2	198.0	0.0	197.07	76.64	192.74	155.74	84.67	198.0	798.0	0.0	797.07	76.64	792.
3	1.0	1.0	1.00	51.00	1.00	101.00	1.00	151.0	398.0	0.0	397.07	76.56	392.
4	198.0	0.0	197.46	75.07	197.18	149.14	149.00	198.0	598.0	0.0	597.46	75.07	597.

5 rows × 302 columns

```
df.describe()
```

	X1	Y1	X2	Y2	X3	Y3	
count	2318.000000	2318.000000	2318.000000	2318.000000	2318.000000	2318.000000	23
mean	177.162584	8.159819	204.669676	64.119892	228.071639	124.794698	2
std	174.211383	52.395345	172.438092	79.224562	181.670898	96.549059	2
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	48.000000	0.000000	100.000000	51.000000	192.570000	101.000000	
50%	198.000000	0.000000	197.070000	72.520000	193.700000	148.350000	1
75%	198.000000	1.000000	201.000000	77.580000	250.000000	150.000000	2
max	1398.000000	1381.090000	1414.000000	1316.750000	1400.000000	1413.430000	14

8 rows × 302 columns

```
#checking the correlation between the features
correlation = df.corr()
print(correlation)
```

```
                   X1        Y1        X2        Y2  ...   Power99  Power100        qW  Total_Power
X1           1.000000 -0.034881  0.783537  0.355123  ... -0.037404 -0.348752 -0.619046    -0.646574
Y1          -0.034881  1.000000  0.033723  0.114210  ... -0.210102  0.112654  0.081276     0.098505
X2           0.783537  0.033723  1.000000  0.060164  ... -0.263693 -0.032276 -0.473338    -0.497733
Y2           0.355123  0.114210  0.060164  1.000000  ...  0.057008 -0.235218 -0.258693    -0.269956
X3           0.664383  0.168309  0.865011 -0.021378  ... -0.413115  0.132155 -0.372082    -0.386883
...               ...       ...       ...       ...  ...       ...       ...       ...          ...
Power98     -0.039002 -0.183879 -0.249736  0.020680  ...  0.822807 -0.267783  0.063329     0.076519
Power99     -0.037404 -0.210102 -0.263693  0.057008  ...  1.000000 -0.203896  0.030976     0.059592
Power100    -0.348752  0.112654 -0.032276 -0.235218  ... -0.203896  1.000000  0.387624     0.416024
qW          -0.619046  0.081276 -0.473338 -0.258693  ...  0.030976  0.387624  1.000000     0.958786
Total_Power -0.646574  0.098505 -0.497733 -0.269956  ...  0.059592  0.416024  0.958786     1.000000

[302 rows x 302 columns]
```

```
#splitting the data into dependent and independent variables
I_v = df.drop(columns=['Total_Power'])
D_v = df['Total_Power']
```

```
#print the shape of the independent and dependent variables
I_v.shape
```

```
(2318, 301)
```

```
D_v.shape
```

```
(2318,)
```

```
#split the data into to 80% training - 20% testing
I_train, I_test, D_train, D_test = train_test_split(I_v, D_v, test_size=0.20)
```

```
#fitting the model
regressor = DecisionTreeRegressor(criterion = 'poisson', max_depth = 15, max_features= 'sqrt', random_state = 0)
regressor.fit(I_train,D_train)
```

```
DecisionTreeRegressor

DecisionTreeRegressor(criterion='poisson', max_depth=15, max_features='sqrt',
                      random_state=0)
```

```
y_pred_test = regressor.predict(I_test)
y_pred_test
```

```
array([7205436.37333333, 7304092.54      , 7258766.2       ,
       7185252.54428571, 7196990.22      , 7295406.54      ,
       7357367.85      , 7276371.11      , 7017266.39      ,
       7268309.15      , 7163270.145     , 7114007.29      ,
       7169825.7775    , 7038654.06      , 7237270.8040625 ,
       7230668.86      , 7226136.24      , 7165698.89      ,
       7242551.42      , 7258766.2       , 7076453.41      ,
       7168407.82      , 7283629.93      , 7146937.35      ,
       7309508.07      , 7147690.16      , 6814552.39      ,
       7126065.74      , 7227849.61      , 7258766.2       ,
       7276371.11      , 7181933.6725    , 7304092.54      ,
       7258766.2       , 7189834.05108108, 7208115.21      ,
       7206458.5       , 7349789.01      , 7291462.52      ,
       7242155.05      , 7172634.24666667, 6950219.93      ,
       6998846.87      , 7303589.25      , 7227849.61      ,
       6816198.02      , 7016760.05      , 7059628.12      ,
       7255804.23      , 7251063.        , 7220031.87      ,
       7227849.61      , 7142644.95      , 7222658.47      ,
       7188178.65      , 7066280.97      , 7112981.34      ,
       7168407.82      , 6860425.08      , 7121262.11      ,
       7186497.97      , 7259731.84      , 7162324.28      ,
       7102044.06      , 7189834.05108108, 7189834.05108108,
       7142644.95      , 7225451.87727273, 7137474.57      ,
       7240286.34      , 7136966.05      , 7357367.85      ,
       7155722.51909091, 7171109.75      , 7106146.32      ,
       7149866.79      , 7162324.28      , 7172366.8       ,
       7157264.42      , 7216360.14      , 7217706.26      ,
       7082012.41      , 7215804.98      , 7181933.6725    ,
       7221037.94      , 7276571.95      , 7163463.94      ,
       7228326.3       , 7142927.25      , 7186497.97      ,
       7076453.41      , 7168407.82      , 7199211.325     ,
       7283629.93      , 7162230.18      , 7175355.57      ,
       7157922.86      , 7152632.05      , 7169825.7775    ,
       7162324.28      , 7225451.87727273, 7230923.61      ,
       7215804.98      , 7284975.88      , 6885951.52      ,
       7164299.26      , 6993053.5       , 7145294.71      ,
       7153337.38      , 7256766.47      , 6915686.35      ,
       6936153.34      , 7281746.12      , 7248032.92      ,
       7222658.47      , 7142953.59      , 7227849.61      ,
       7224925.764     , 7115246.43      , 7227849.61      ,
       7185633.31      , 6950701.22      , 7300092.26      ,
       6939421.71      , 7148342.69      , 7162828.6725    ,
       7226226.76      , 7085192.54      , 7280289.68      ,
       7003293.26      , 6827667.72      , 7237270.8040625 ,
       7096067.01      , 7178962.84      , 6962332.43      ,
       7119962.04      , 7119312.6325    , 6865455.83      ,
       7004908.69375   , 6899891.58      , 7206977.62      ,
       7204305.58      , 7189834.05108108, 7183381.805     ,
       7047737.89      , 7140618.58      , 7106146.32      ,
       6927411.53      , 6956604.35      , 7153707.64      ,
       7196856.98      , 7255990.61      , 7175507.47      ,
       7182239.59      , 7227849.61      , 7259731.84      ,
       6937949.03      , 7085192.54      , 7227494.668     ,
       7025512.68      , 7029318.69      , 7104965.1       ,
       7244392.76      , 7171319.59      , 7266566.44      ,
       7013897.75      , 7226136.24      , 7169278.04      ,
       7232338.65      , 7065836.645     , 7015720.3       ,
       7202064.95      , 7257796.79      , 7276371.11      ,
```

```
# Calculate Mean Absolute Error (MAE)
mae_test = mean_absolute_error(D_test, y_pred_test)
print("The MAE is:", mae_test)
```

```
The MAE is: 18640.9590491914
```

```
mse = mean_squared_error(D_test, y_pred_test)
print("Mean Squared Error:", mse)
```

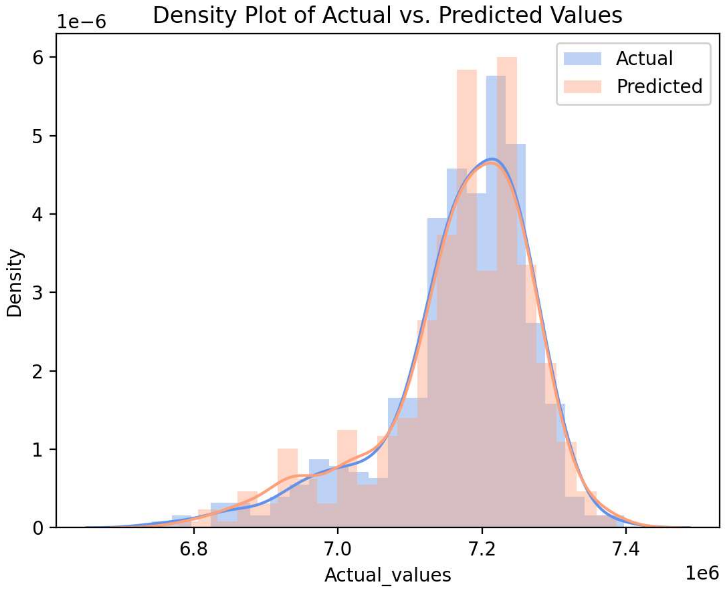
```
Mean Squared Error: 1837853157.9263809
```

```
#to check the accuracy
r2 = r2_score(D_test, y_pred_test)

print("The R-squared Score is:", r2)
```

```
The R-squared Score is: 0.8286421413212698
```

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.distplot(D_test, color='#6495ED', label='Actual')
sns.distplot(y_pred_test, color='#FFA07A', label='Predicted')
plt.xlabel('Actual_values')
plt.ylabel('Density')
plt.title('Density Plot of Actual vs. Predicted Values')
plt.legend()
plt.show()
```

```
/tmp/ipykernel_10602/734506330.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

/tmp/ipykernel_10602/734506330.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```



My observation to criterion was: when I am Using squared error, the error rate is increasing as well as the model accuracy performed only an accuracy of 77 with max depth 10.