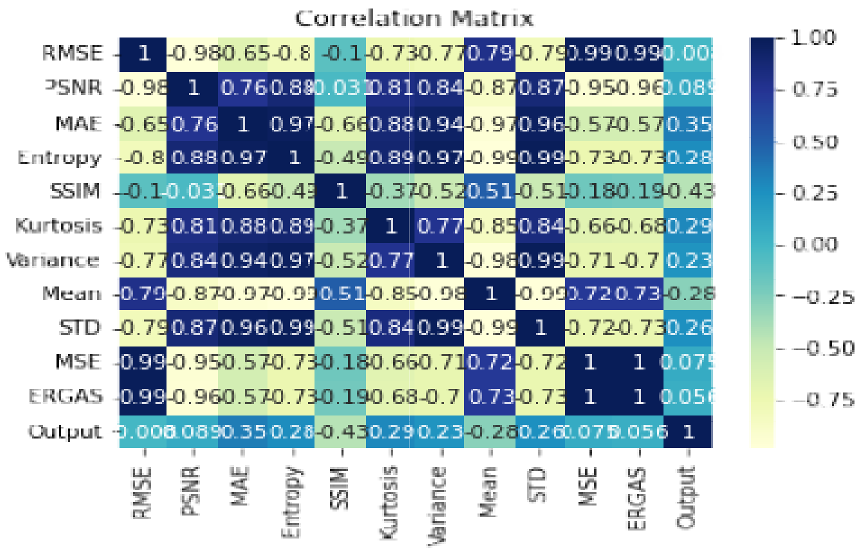
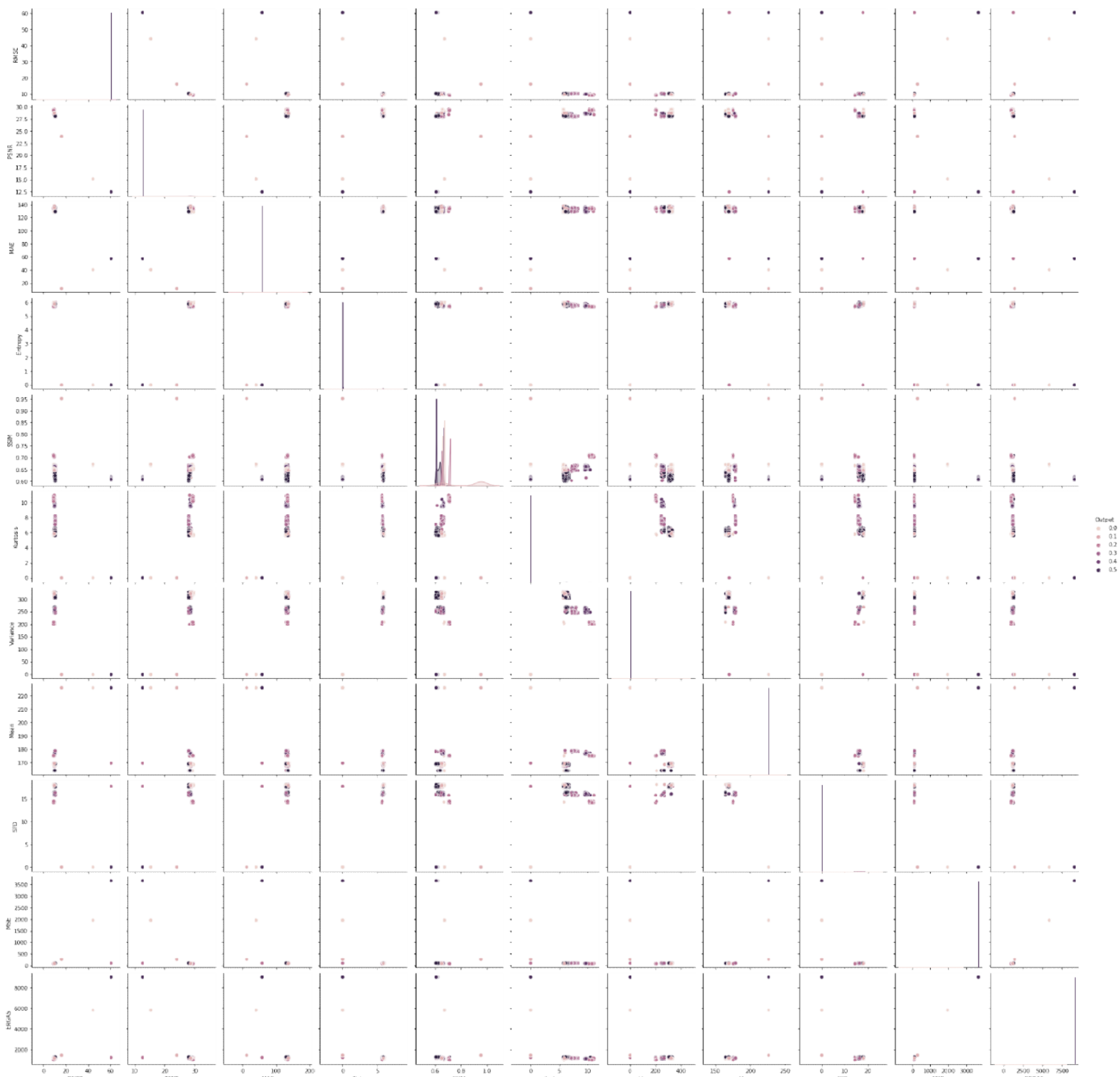
EDA for Data-2 Dataset

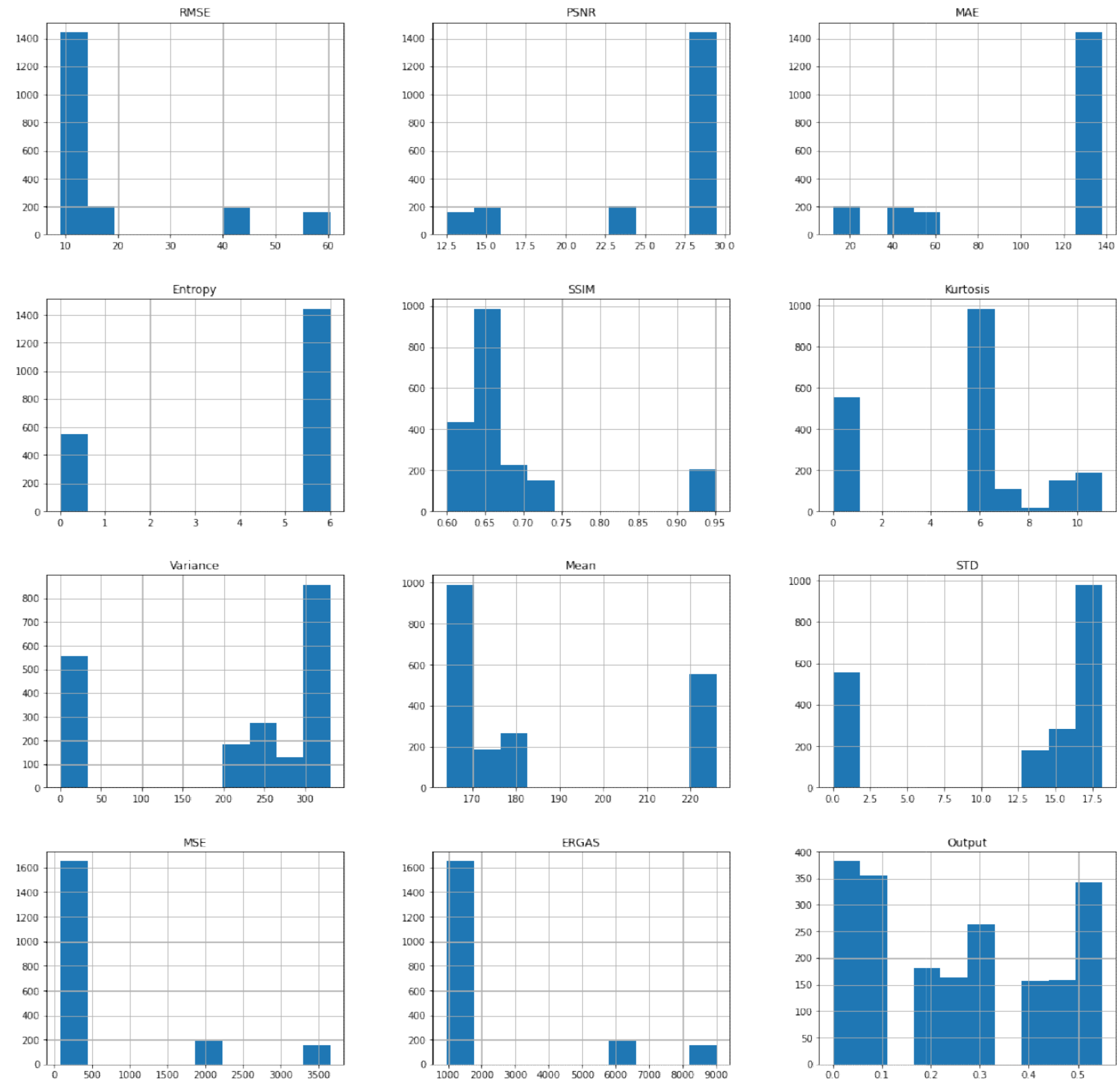
```
# Correlation Matrix
corr = data.corr()
sns.heatmap(corr, annot=True, cmap="YlGnBu");
plt.title("Correlation Matrix");
plt.show()
```




```
sns.pairplot(data,hue='Output')
<seaborn.axisgrid.PairGrid at 0x7f97db212510>
```




```
# histogram of data
data.hist(figsize=(20,20));
```



Machine Learning Algorithms

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
org_data = pd.read_excel('Data II.xlsx')
org_data.head(10)
```

	RMSE	PSNR	MAE	Entropy	SSIM	Kurtosis	Variance	Mean	STD	MSE	ERGAS	Output
0	9.803700	28.597781	132.383910	5.711144	0.660056	9.640824	246.495956	177.237922	15.790069	92.074388	1089.209115	0.44
1	10.236862	28.071782	129.207262	5.902248	0.624875	6.280826	328.842777	169.088223	18.174401	101.460239	1244.503026	0.55
2	9.893585	28.552832	135.041501	5.778902	0.641492	6.056850	304.278410	168.824486	17.498489	93.228629	1170.546730	0.08
3	16.231725	23.923827	12.111197	0.000163	0.952117	-0.000699	-0.000217	225.999929	0.000048	263.440207	1405.991063	0.08
4	16.231877	23.923816	12.110698	0.000017	0.951983	-0.000536	-0.000236	225.999978	0.000058	263.440175	1405.990935	0.08
5	9.757236	28.644019	129.694023	5.769120	0.663479	9.494948	262.140552	177.014573	16.268146	90.824981	1078.515542	0.44
6	9.867062	28.497257	134.061911	5.865035	0.646494	5.615578	310.551636	169.817527	17.659147	92.791974	1171.122032	0.08
7	9.737337	28.666077	133.959820	5.876081	0.647189	5.655271	307.013525	168.001495	17.563662	90.737701	1160.488323	0.31
8	10.166565	28.134280	134.292574	5.951885	0.649870	5.714806	328.121264	169.999224	18.140052	100.837558	1210.937585	0.23
9	10.024548	28.424311	133.675589	5.895320	0.635106	6.257486	268.280183	164.282044	16.461225	95.418100	1239.200299	0.55

Checking for NULL values

```python
org_data.isna().sum()
```

```
RMSE        0
PSNR        0
MAE         0
Entropy     0
SSIM        0
Kurtosis    0
Variance    0
Mean        0
STD         0
MSE         0
ERGAS       0
Output      0
dtype: int64
```

```python
data = org_data
```

```python
# taking all numerical values
ndata = data.select_dtypes(include = np.number)
ndata.head()
```

	RMSE	PSNR	MAE	Entropy	SSIM	Kurtosis	Variance	Mean	STD	MSE	ERGAS	Output
0	9.803700	28.597781	132.383910	5.711144	0.660056	9.640824	246.495956	177.237922	15.790069	92.074388	1089.209115	0.44
1	10.236862	28.071782	129.207262	5.902248	0.624875	6.280826	328.842777	169.088223	18.174401	101.460239	1244.503026	0.55
2	9.893585	28.552832	135.041501	5.778902	0.641492	6.056850	304.278410	168.824486	17.498489	93.228629	1170.546730	0.08
3	16.231725	23.923827	12.111197	0.000163	0.952117	-0.000699	-0.000217	225.999929	0.000048	263.440207	1405.991063	0.08
4	16.231877	23.923816	12.110698	0.000017	0.951983	-0.000536	-0.000236	225.999978	0.000058	263.440175	1405.990935	0.08

```
# getting range
ndata.max() - ndata.min()
```

```
RMSE           51.476807
PSNR           17.003716
MAE           126.094614
Entropy         6.019127
SSIM            0.352919
Kurtosis       11.020072
Variance      330.952975
Mean           61.941852
STD            18.186520
MSE          3583.117105
ERGAS        8063.487317
Output          0.550000
dtype: float64
```

```
# first Quartile
Q1 = ndata.quantile(0.25)
Q1
```

```
RMSE            9.721114
PSNR           23.923850
MAE            57.420428
Entropy         0.000089
SSIM            0.638516
Kurtosis        0.000429
Variance       -0.000208
Mean          168.850698
STD             0.000066
MSE            90.428433
ERGAS        1131.493426
Output          0.080000
Name: 0.25, dtype: float64
```

```
# second Quartile
Q2 = ndata.quantile(0.50)
Q2
```

```
RMSE           10.070360
PSNR           28.287276
MAE           131.749546
Entropy         5.802359
SSIM            0.652574
Kurtosis        5.778257
Variance      262.179782
Mean          174.982803
STD            16.245528
MSE            99.724794
ERGAS        1203.716318
Output          0.230000
Name: 0.5, dtype: float64
```

```python
# third Quartile
Q3 = ndata.quantile(0.75)
Q3
```

```
RMSE           16.231762
PSNR           28.670266
MAE           133.787037
Entropy         5.890909
SSIM            0.671308
Kurtosis        6.376148
Variance      313.458119
Mean          225.999965
STD            17.744357
MSE           263.440171
ERGAS        1405.990966
Output          0.452500
Name: 0.75, dtype: float64
```

```python
# Inter Quartile Range
IQR = Q3 - Q1
IQR
```

```
RMSE            6.510648
PSNR            4.746416
MAE            76.366609
Entropy         5.890819
SSIM            0.032792
Kurtosis        6.375719
Variance      313.458327
Mean           57.149267
STD            17.744290
MSE           173.011739
ERGAS         274.497540
Output          0.372500
dtype: float64
```

```
# removing outliers
final_data = ndata[~((ndata < (Q1 - 1.5 * IQR)) |(ndata > (Q3 + 1.5 *
IQR))).any(axis=1)]
final_data.shape
```

```
(1444, 12)
```

Machine Learning Algorithms

```
# site and pop are correlated features
X = final_data.drop(['Output'], axis=1)
y = final_data['Output']
X
```

	RMSE	PSNR	MAE	Entropy	SSIM	Kurtosis	Variance	Mean	STD	MSE	ERGAS
0	9.803700	28.597781	132.383910	5.711144	0.660056	9.640824	246.495956	177.237922	15.790069	92.074388	1089.209115
1	10.236862	28.071782	129.207262	5.902248	0.624875	6.280826	328.842777	169.088223	18.174401	101.460239	1244.503026
2	9.893585	28.552832	135.041501	5.778902	0.641492	6.056850	304.278410	168.824486	17.498489	93.228629	1170.546730
5	9.757236	28.644019	129.694023	5.769120	0.663479	9.494948	262.140552	177.014573	16.268146	90.824981	1078.515542
6	9.867062	28.497257	134.061911	5.865035	0.646494	5.615578	310.551636	169.817527	17.659147	92.791974	1171.122032
7	9.737337	28.666077	133.959820	5.876081	0.647189	5.655271	307.013525	168.001495	17.563662	90.737701	1160.488323
8	10.166565	28.134280	134.292574	5.951885	0.649870	5.714806	328.121264	169.999224	18.140052	100.837558	1210.937585
9	10.024548	28.424311	133.675589	5.895320	0.635106	6.257486	268.280183	164.282044	16.461225	95.418100	1239.200299
11	9.505491	28.824681	129.607648	5.969466	0.653286	5.620150	326.550954	168.602413	18.107595	85.693073	1118.055040
12	9.098326	29.328128	133.089994	5.714889	0.711866	10.375758	208.392831	175.238884	14.474208	78.490162	953.119352

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

from sklearn.linear_model import LinearRegression
from sklearn.kernel_ridge import KernelRidge
from xgboost.sklearn import XGBRegressor
from sklearn.linear_model import BayesianRidge

model1 = LinearRegression()
model2 = KernelRidge()
model3 = XGBRegressor()
model4 = BayesianRidge()

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

1. **LinearRegression**

```
model1.fit(X_train,y_train)
# precting on training
```

```
pred = model1.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train,pred)))
r2 = r2_score(y_train, pred)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# predicting on test
pred2 = model1.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, pred2)))
r2 = r2_score(y_test, pred2)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.07929732879366719
R2 score is 0.8200355424205997


The model performance for testing set
--------------------------------------
RMSE is 0.08552623414846985
R2 score is 0.7932692980307987
```

2. **KernelRidge**

```
model2.fit(X_train,y_train)

# precting on training
pred3 = model2.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train,pred3)))
r2 = r2_score(y_train, pred)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# predicting on test
```

```
pred4 = model2.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, pred4)))
r2 = r2_score(y_test, pred4)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.08415008768605448
R2 score is 0.8200355424205997


The model performance for testing set
--------------------------------------
RMSE is 0.09494753852985079
R2 score is 0.7452150913571423
```

3. **XGBRegressor**

```
model3.fit(X_train,y_train)

# precting on training
pred5 = model3.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train,pred5)))
r2 = r2_score(y_train, pred5)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# predicting on test
pred6 = model3.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, pred6)))
r2 = r2_score(y_test, pred6)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.03246799751143454
R2 score is 0.9698296301639215


The model performance for testing set
--------------------------------------
RMSE is 0.05050505800132889
R2 score is 0.9279098335753537
```

4. **BayesianRidge**

```
model4.fit(X_train,y_train)

# precting on training
pred7 = model4.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train,pred7)))
r2 = r2_score(y_train, pred7)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# predicting on test
pred8 = model4.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, pred8)))
r2 = r2_score(y_test, pred8)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 0.07934151105326072
R2 score is 0.8198349442010398


The model performance for testing set
--------------------------------------
RMSE is 0.08625571105027513
R2 score is 0.7897277320207432
```

Comparison:

We have done EDA of the given dataset and applied the preprocessing, training-testing and following 4 algorithms are used:

1. LinearRegression
2. KernelRidge
3. XGBRegressor
4. BayesianRidge

We can conclude that XGBRegressor has the highest R2 score means there is a high correlation between the attributes and XGBRegressor has the lowest RMSE value means it is the most accurate model. So overall the XGBRegressor model is best among the selected four models.