```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression , RANSACRegressor , TheilSenRegressor , Ridge , Lasso , ElasticNet , SGDRegressor
from sklearn.metrics import r2_score
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split , cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.ensemble import RandomForestRegressor , AdaBoostRegressor , GradientBoostingRegressor
from xgboost import  XGBRegressor
from scipy.stats.mstats import winsorize
```

```
data = pd.read_csv('data.csv')
```

```
data.head()
```

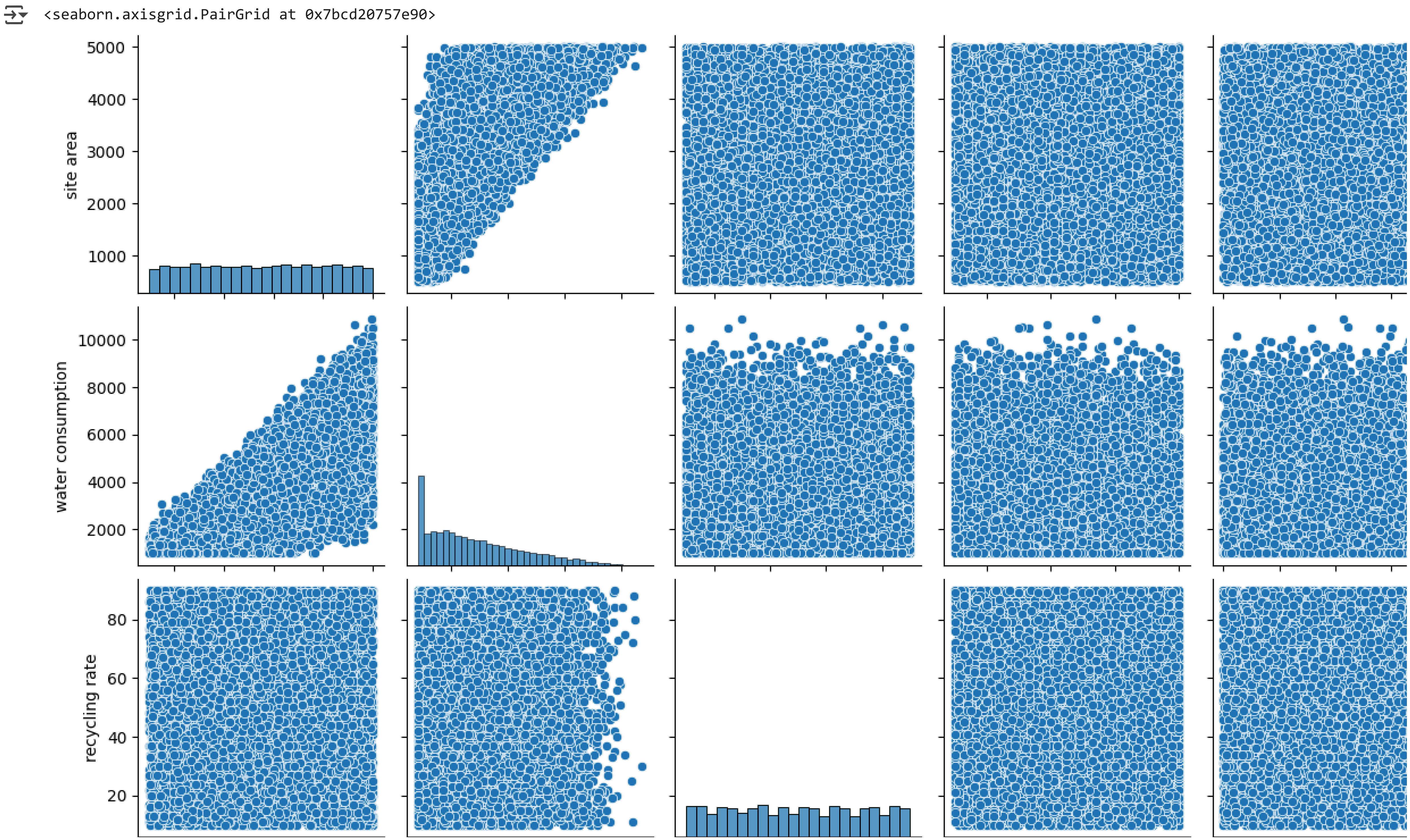
	site area	structure type	water consumption	recycling rate	utilisation rate	air qality index	issue reolution time	resident count	electricity cost
0	1360	Mixed-use	2519.0	69	52	188	1	72	1420.0
1	4272	Mixed-use	2324.0	50	76	165	65	261	3298.0
2	3592	Mixed-use	2701.0	20	94	198	39	117	3115.0
3	966	Residential	1000.0	13	60	74	3	35	1575.0
4	4926	Residential	5990.0	23	65	32	57	185	4301.0

Next steps: View recommended plots New interactive sheet

```
data.isnull().sum()
```

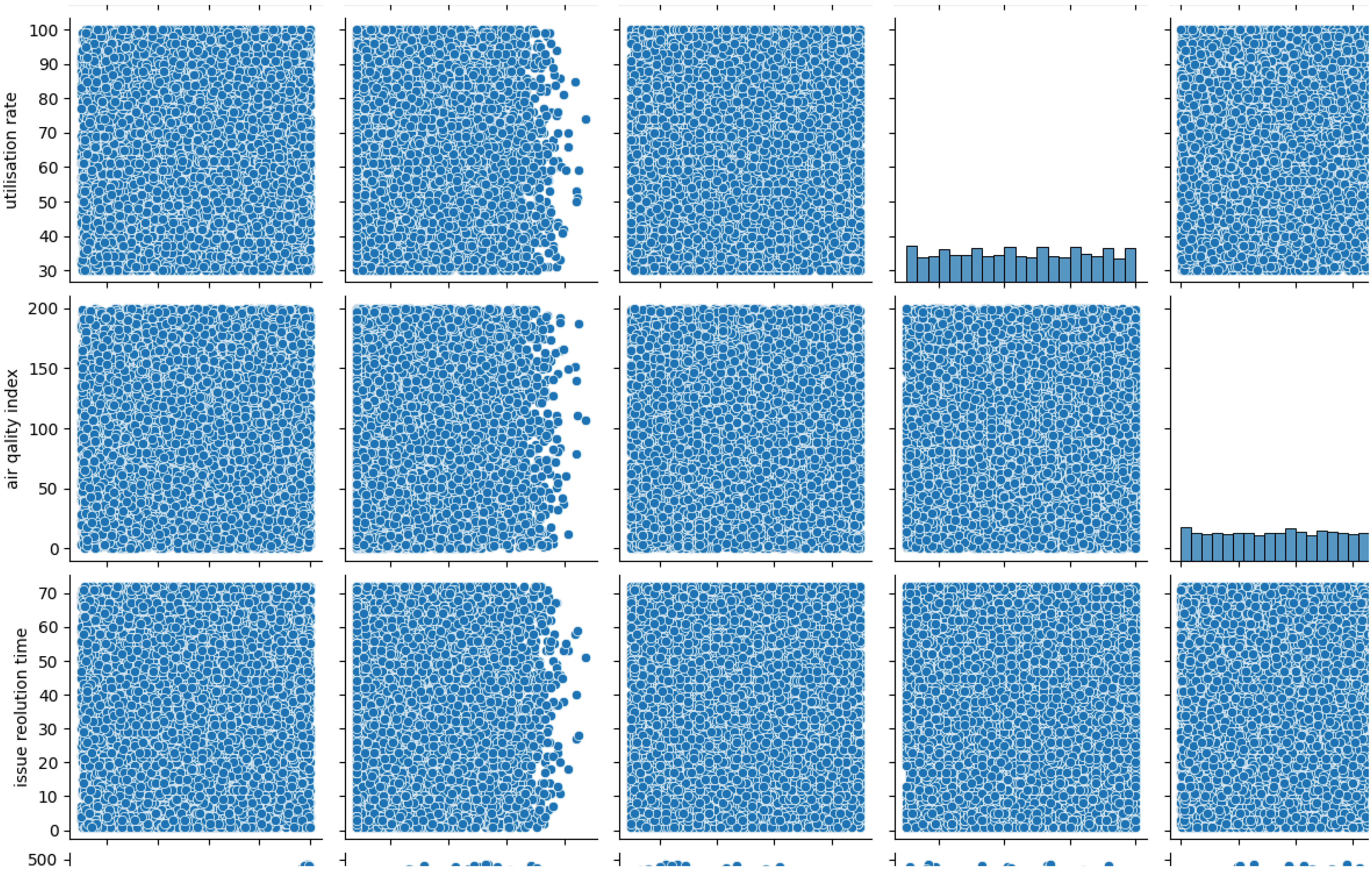
```
                           0
site area                  0
structure type             0
water consumption          0
recycling rate             0
utilisation rate           0
air qality index           0
issue reolution time       0
resident count             0
electricity cost           0

dtype: int64
```

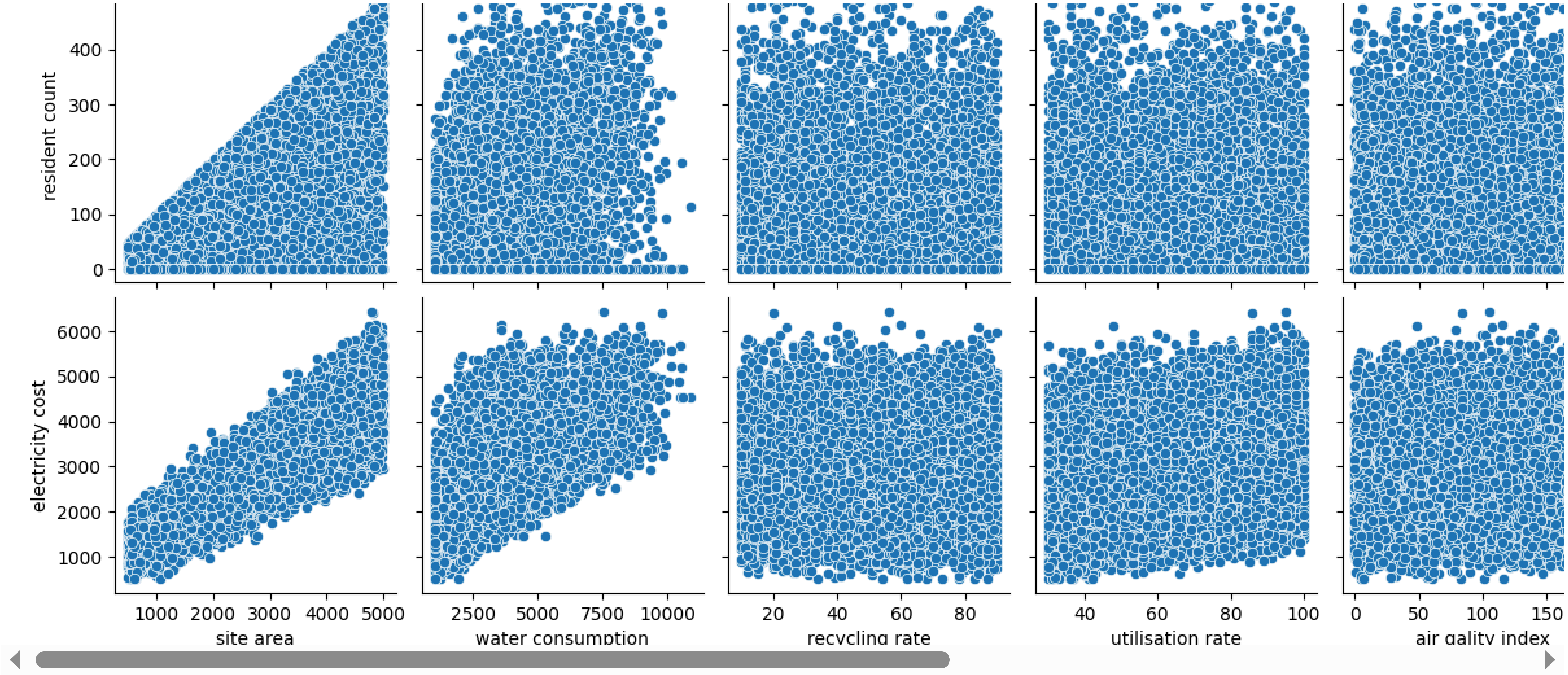
```
sns.pairplot(data)
```



```
<seaborn.axisgrid.PairGrid at 0x7bcd20757e90>
```

site area
5000
4000
3000
2000
1000
water consumption
10000
8000
6000
4000
2000
recycling rate
80
60
40
20


utilisation rate
100
90
80
70
60
50
40
30
air qality index
200
150
100
50
0
issue reolution time
70
60
50
40
30
20
10
0
500

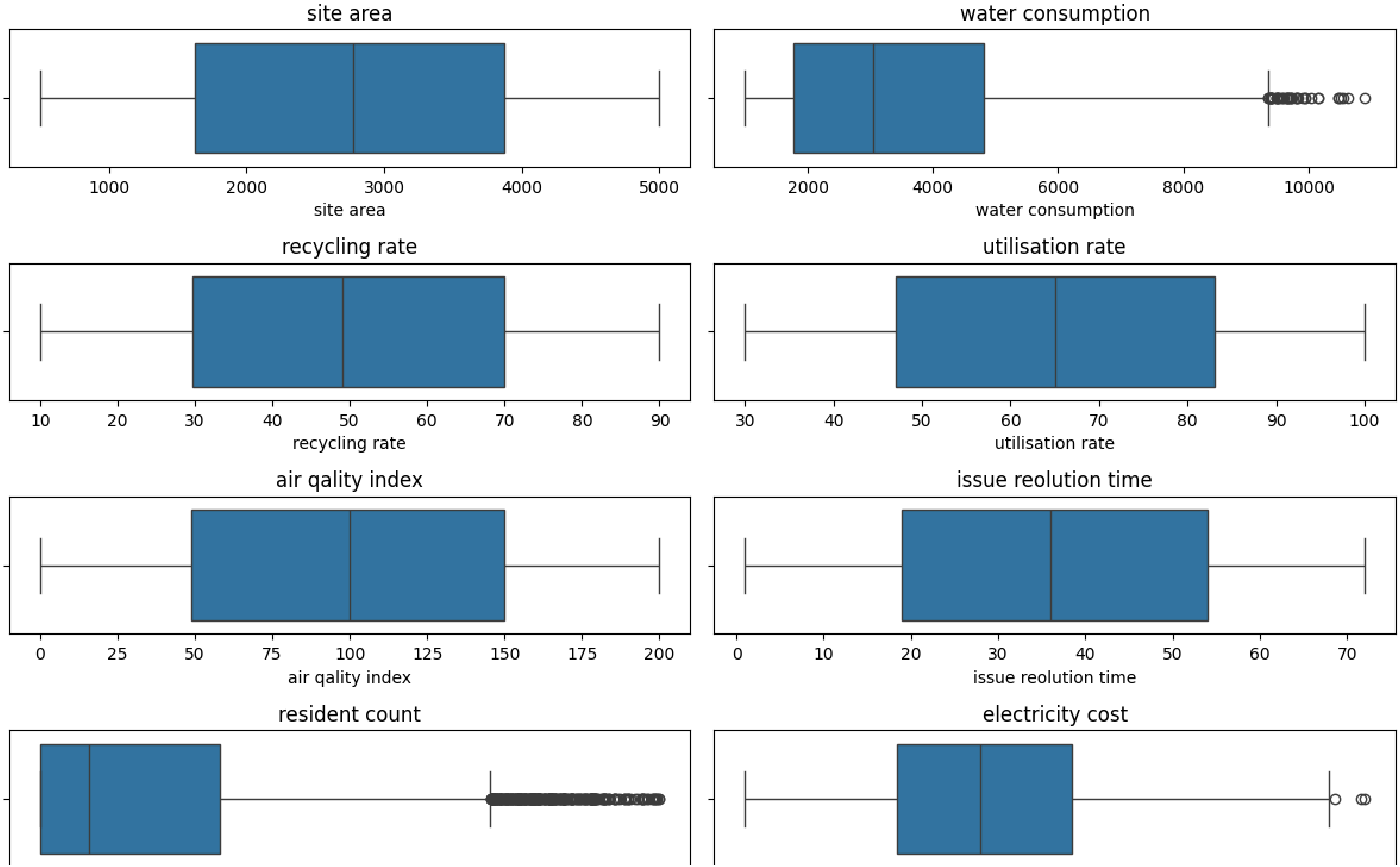

resident count
400
300
200
100
0
electricity cost
6000
5000
4000
3000
2000
1000
1000 2000 3000 4000 5000
site area
2500 5000 7500 10000
water consumption
20 40 60 80
recycling rate
40 60 80 100
utilisation rate
0 50 100 150
air qality index

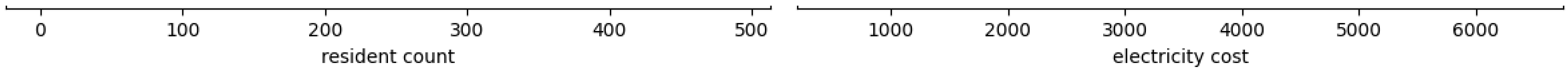
```
data.drop('structure type' , axis = 1 , inplace = True)
```

```
data.shape
```

```
(10000, 8)
```

```
fig, ax = plt.subplots(4,2, figsize=(12,8))
axes_ = [axes_row for axes in ax for axes_row in axes]

for i, j in enumerate(data.columns):
    g = sns.boxplot(x = data[j], ax = axes_[i])
    g.set_title(j)
    plt.tight_layout()
```


site area
1000
2000
3000
4000
5000
site area
water consumption
2000
4000
6000
8000
10000
water consumption
recycling rate
10
20
30
40
50
60
70
80
90
recycling rate
utilisation rate
30
40
50
60
70
80
90
100
utilisation rate
air qality index
0
25
50
75
100
125
150
175
200
air qality index
issue reolution time
0
10
20
30
40
50
60
70
issue reolution time
resident count
electricity cost



```
col = data.columns
```

```
def check_outliers(data, col):
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5*IQR
    upper_bound = Q3 + 1.5*IQR

    outliers = data[col][(data[col] < lower_bound) | (data[col] > upper_bound)]

    print("Outliers Report")

    print(f"The total number of outliers in Data: {len(outliers)}")

    plt.figure(figsize=(10,10))
    plt.subplot(211)
    plt.plot(data[col])
    plt.title(col + " with Outliers")
    plt.scatter(x=outliers.index, y=outliers.values, marker="X", color='r', s=100)

    plt.subplot(212)
    plt.title(col + " after removing Extreme Values")
    filter_data = data[col][-(data[col].isin(outliers))]
    sns.boxplot(filter_data)
```

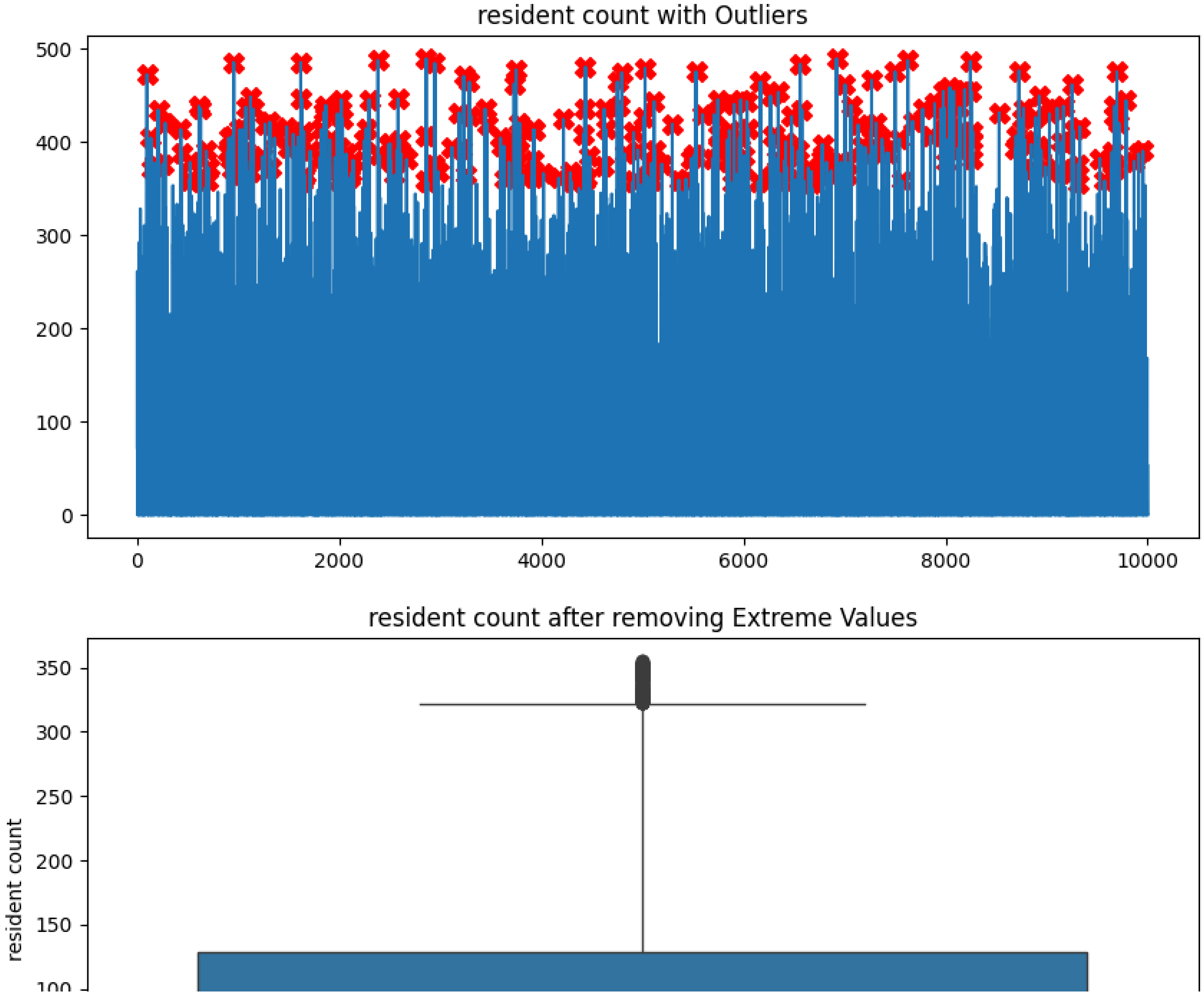
```
data.dtypes
```

	0
site area	int64
water consumption	float64
recycling rate	int64
utilisation rate	int64
air qality index	int64
issue reolution time	int64
resident count	int64
electricity cost	float64

dtype: object

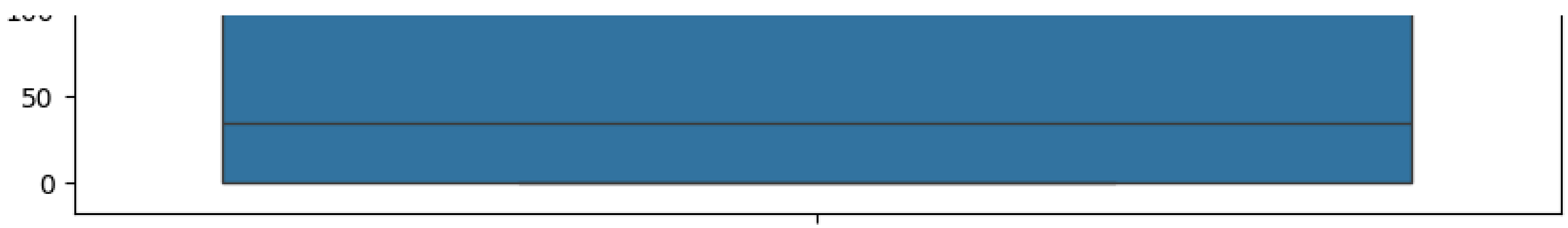
```
check_outliers(data, col[6])
```

```
Outliers Report
The total number of outliers in Data: 289
```



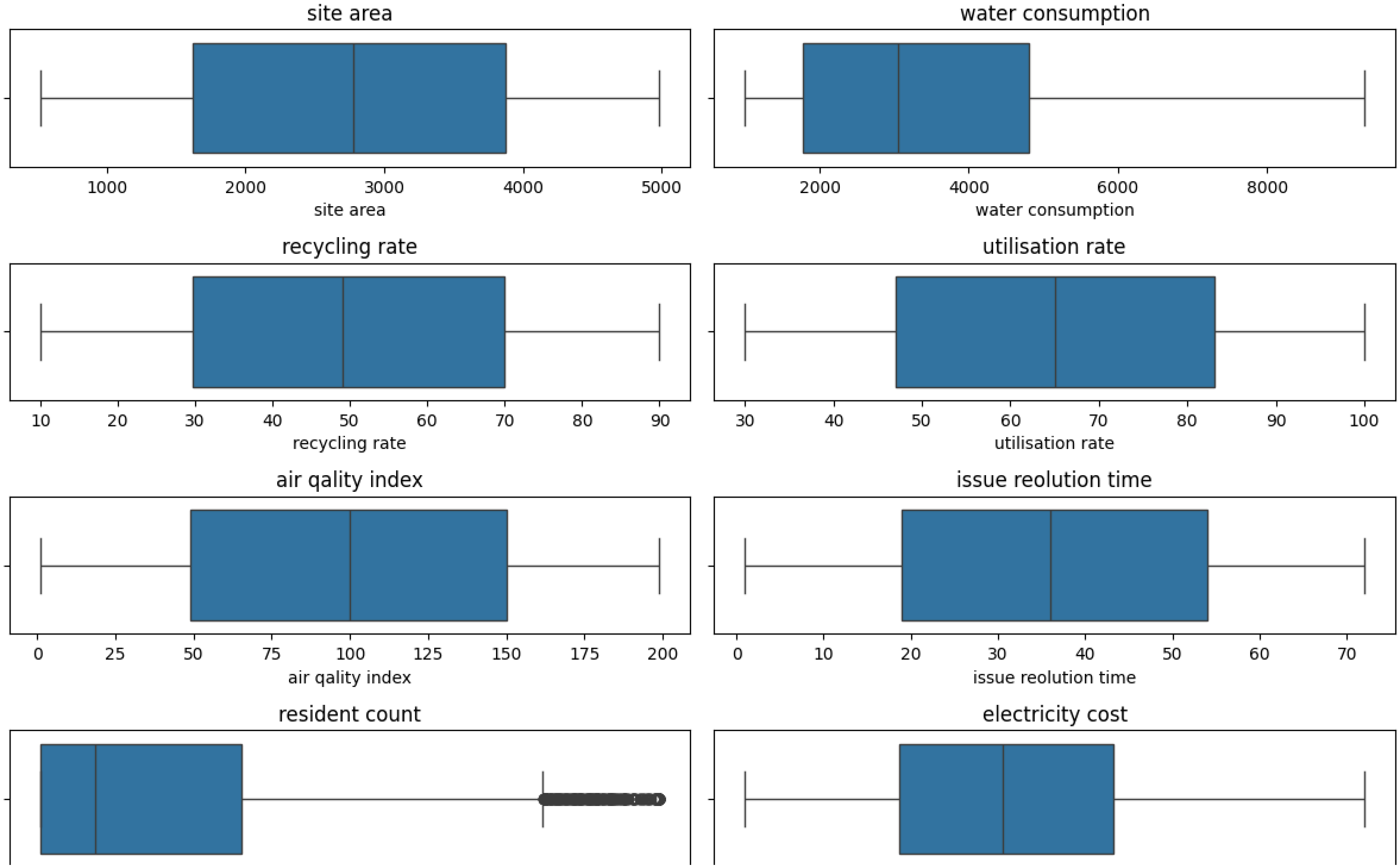
resident count with Outliers

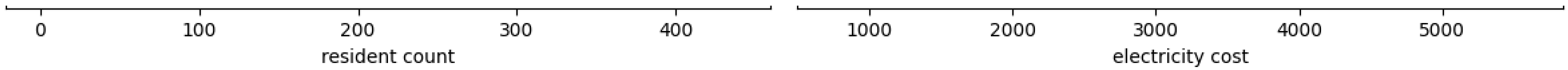
resident count after removing Extreme Values



```
data_copy = data.copy()
```

```
def apply_winsorize(data, col):
    winsorize(data[col], limits= [0.005, 0.005], inplace=True)
for i, j in data.items():
    apply_winsorize(data_copy, i)
fig, ax = plt.subplots(4,2, figsize=(12,8))
axes_ = [axes_row for axes in ax for axes_row in axes]

for i, j in enumerate(data.columns):
    g = sns.boxplot(x = data_copy[j], ax = axes_[i])
    g.set_title(j)
    plt.tight_layout()
```


site area
1000
2000
3000
4000
5000
site area
water consumption
2000
4000
6000
8000
water consumption
recycling rate
10
20
30
40
50
60
70
80
90
recycling rate
utilisation rate
30
40
50
60
70
80
90
100
utilisation rate
air qality index
0
25
50
75
100
125
150
175
200
air qality index
issue reolution time
0
10
20
30
40
50
60
70
issue reolution time
resident count
electricity cost



```
data_copy.dtypes
```

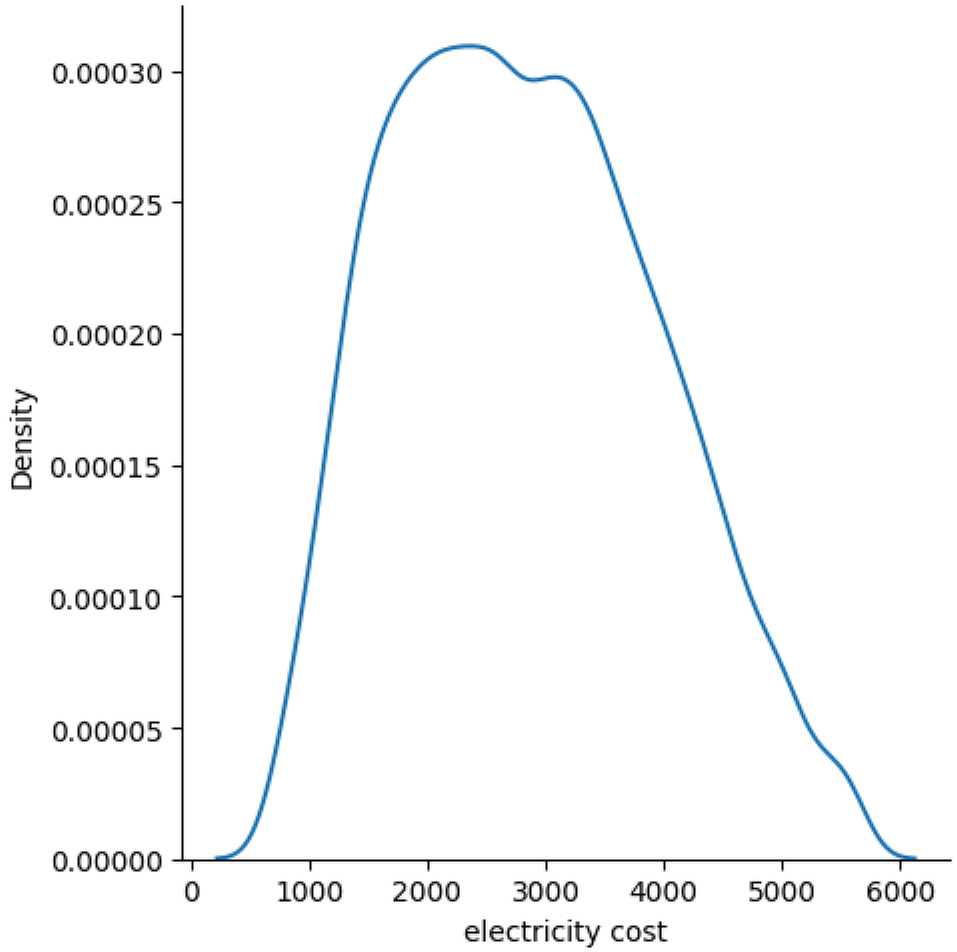
```
                              0
site area                 int64
water consumption       float64
recycling rate            int64
utilisation rate          int64
air qality index          int64
issue reolution time      int64
resident count            int64
electricity cost        float64

dtype: object
```

```
X = data_copy.iloc[: , data_copy.columns != 'electricity cost']
```

```
Y = data_copy['electricity cost']
```

```
sns.displot(Y , kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7bcd287bc810>
```



```
X_train , X_test ,y_train , y_test = train_test_split(X , Y , test_size=0.3 , random_state=0)
```

```
lr_pipeline = Pipeline([("scaler", StandardScaler()), ("linear_regression", LinearRegression())])
rb_pipeline = Pipeline([("scaler", StandardScaler()), ("robust_regression", RANSACRegressor(random_state=42))])
theil_pipeline = Pipeline([("scaler", StandardScaler()), ("theil_regressor", TheilSenRegressor(random_state=42))])
ridge_pipeline = Pipeline([("scaler", StandardScaler()), ("ridge_regressor", Ridge(random_state = 42))])
lasso_pipeline = Pipeline([("scaler", StandardScaler()), ("lasso_regressor", Lasso(random_state = 42))])
elastic_pipeline = Pipeline([("scaler", StandardScaler()), ("elastic_net", ElasticNet(random_state = 42))])
random_forest_pipeline = Pipeline([("scaler", StandardScaler()), ("randomforest_regression", RandomForestRegressor(random_state = 42))])
```

```
xgboost_pipeline = Pipeline([("scaler", StandardScaler()), ("xgboost_regression", XGBRegressor())])
adaboost_pipeline = Pipeline([("scaler", StandardScaler()), ("adaboost_regression", AdaBoostRegressor(random_state = 42))])
gradient_pipeline = Pipeline([("scaler", StandardScaler()), ("gradientboost_regression", GradientBoostingRegressor(random_state = 42))])
lightgbm_pipeline = Pipeline([("scaler", StandardScaler()), ("lightgbm_regression", LGBMRegressor(random_state = 42))])
decisiontree_pipeline = Pipeline([("scaler", StandardScaler()), ("decisiontree_regression", DecisionTreeRegressor(random_state = 42))])
knn_pipeline = Pipeline([("scaler", StandardScaler()), ("knn_regression", KNeighborsRegressor())])
sgc_pipeline = Pipeline([("scaler", StandardScaler()), ("sgd_regression", SGDRegressor(random_state = 42))])
```

```
pipelines = [lr_pipeline, rb_pipeline, theil_pipeline, ridge_pipeline, lasso_pipeline, elastic_pipeline,
             random_forest_pipeline, xgboost_pipeline, adaboost_pipeline, gradient_pipeline, lightgbm_pipeline,
              decisiontree_pipeline, knn_pipeline, sgc_pipeline]
```

```
pipe_dictionary  = pipe_dict = {0: "Linear Regression", 1: "Robust", 2: "Theil Sen", 3: "Ridge", 4: "Lasso", 5: "ElasticNet",
             6: "RandomForest", 7: "XGBoost", 8: "Adaboost", 9: "GradientBoost", 10: "LightGBM",
              11: "Decision Tree", 12: "KNN", 13: "SGD"}
```

```
for i , pipe in enumerate(pipelines):
  score = cross_val_score(pipe , X , Y , cv=10)
  print(pipe_dictionary[i] , score.mean())
```

Show hidden output

```
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from lightgbm import LGBMRegressor

class ModelTuner:
    def __init__(self, model_name, search_type='grid', random_state=42):
        self.model_name = model_name.lower()
        self.search_type = search_type
        self.random_state = random_state
        self.model = None
        self.param_grid = None
        self.best_estimator = None
        self.best_params = None
        self.best_score = None

        self._init_model_and_params()

    def _init_model_and_params(self):
        if self.model_name == 'random_forest':
            self.model = RandomForestRegressor(random_state=self.random_state)
            self.param_grid = {
                'n_estimators': [100, 200],
                'max_depth': [None, 10, 20],
                'min_samples_split': [2, 5],
                'min_samples_leaf': [1, 2],
                'max_features': ['sqrt', 'log2']
            }
        elif self.model_name == 'lightgbm':
            self.model = LGBMRegressor(random_state=self.random_state)
            self.param_grid = {
                'n_estimators': [100, 200],
                'learning_rate': [0.01, 0.05, 0.1],
                'max_depth': [-1, 5, 10],
                'num_leaves': [31, 50],
                'min_child_samples': [10, 20],
                'subsample': [0.7, 0.9],
                'colsample_bytree': [0.7, 0.9]
            }
        else:
            raise ValueError("Unsupported model. Use 'random_forest' or 'lightgbm'.")

    def fit(self, X, y, test_size=0.2, cv=5, n_jobs=-1):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=self.random_state)
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)

        if self.search_type == 'grid':
            search = GridSearchCV(self.model, self.param_grid, cv=cv, scoring='r2', n_jobs=n_jobs)
        else:
            from sklearn.model_selection import RandomizedSearchCV
            search = RandomizedSearchCV(self.model, self.param_grid, cv=cv, scoring='r2', n_jobs=n_jobs, n_iter=30, random_state=self.random_state)
```