April 19, 2024

```
Project Title : Energy Consumption Prediction
Group         : 28
Group Members : EG/2019/3629 - Karunasundara K.M.S.N.
                EG/2019/3633 - Kavinda B.G.K.
Github        : https://github.com/saranga97/Energy-consumption-prediction-using-LR-RF
```

1 1. Import libraries and input dataset

[80]:
```
# Importing common libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

[81]:
```
# Read data csv from github
url = 'https://raw.githubusercontent.com/saranga97/
 ↪Energy-consumption-prediction-using-LR-RF/master/Data/Energy_consumption.csv'
df = pd.read_csv(url)
```

[82]:
```
# Read first 5 datapoints of the dataset
df.head(5)
```

[82]:

	Timestamp	Temperature	Humidity	SquareFootage	Occupancy	HVACUsage	LightingUsage	RenewableEnergy	DayOfWeek	Holiday	EnergyConsumption
0	2022-01-01 00:00:00	25.139433	43.431581	1565.693999	5	On	Off	2.774699	Monday	No	
1	2022-01-01 01:00:00	27.731651	54.225919	1411.064918	1	On	On	21.831384	Saturday	No	
2	2022-01-01 02:00:00	28.704277	58.907658	1755.715009	2	Off	Off	6.764672	Sunday	No	
3	2022-01-01 03:00:00	20.080469	50.371637	1452.316318	1	Off	On	8.623447	Wednesday	No	
4	2022-01-01 04:00:00	23.097359	51.401421	1094.130359	9	On	Off	3.071969	Friday	No	

```
0          75.364373
1          83.401855
2          78.270888
3          56.519850
4          70.811732
```

2 2. Preprocessing data

[83]:
```
# Check whether if any column have null value and data type of each column
df.info()
```

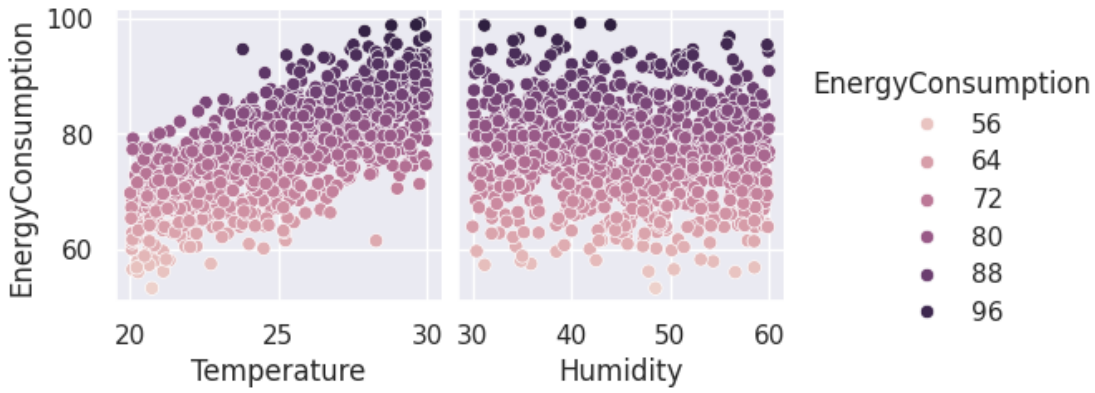
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Timestamp          1000 non-null   object
 1   Temperature        1000 non-null   float64
 2   Humidity           1000 non-null   float64
 3   SquareFootage      1000 non-null   float64
 4   Occupancy          1000 non-null   int64
 5   HVACUsage          1000 non-null   object
 6   LightingUsage      1000 non-null   object
 7   RenewableEnergy    1000 non-null   float64
 8   DayOfWeek          1000 non-null   object
 9   Holiday            1000 non-null   object
 10  EnergyConsumption  1000 non-null   float64
dtypes: float64(5), int64(1), object(5)
memory usage: 86.1+ KB
```

[84]:
```
# Drop the 'Timestamp' column as it's not needed
df.drop('Timestamp', inplace=True, axis=1)
```

[85]:
```
# Perform one-hot encoding for categorical variables
df = pd.get_dummies(data=df)
```

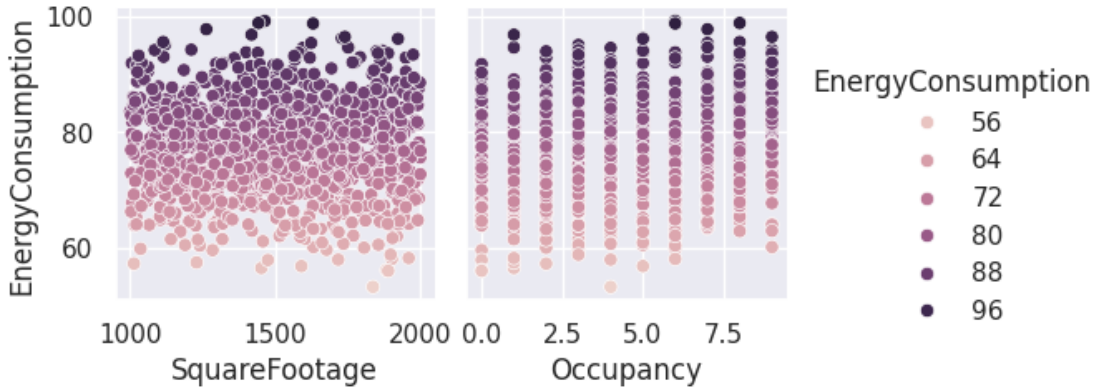
[86]:
```
# Exploratory Data Analysis (EDA) - Data pairplot
import seaborn as sns
sns.pairplot(df, x_vars=["Temperature", "Humidity"],␣
 ↪y_vars=["EnergyConsumption"], hue="EnergyConsumption")
```

[86]:
```
<seaborn.axisgrid.PairGrid at 0x7f2356a56890>
```



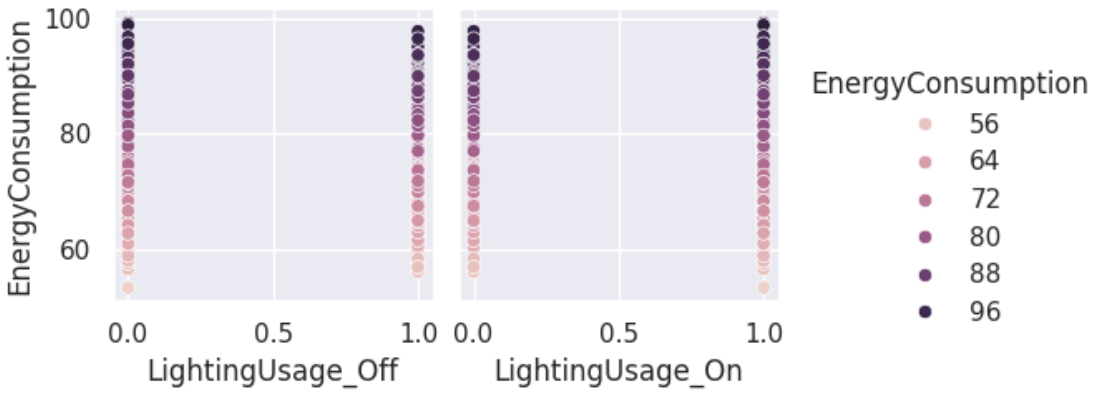
[87]:
```
sns.pairplot(df, x_vars=["SquareFootage", "Occupancy"],␣
↪y_vars=["EnergyConsumption"], hue="EnergyConsumption")
```

[87]: <seaborn.axisgrid.PairGrid at 0x7f2356683220>



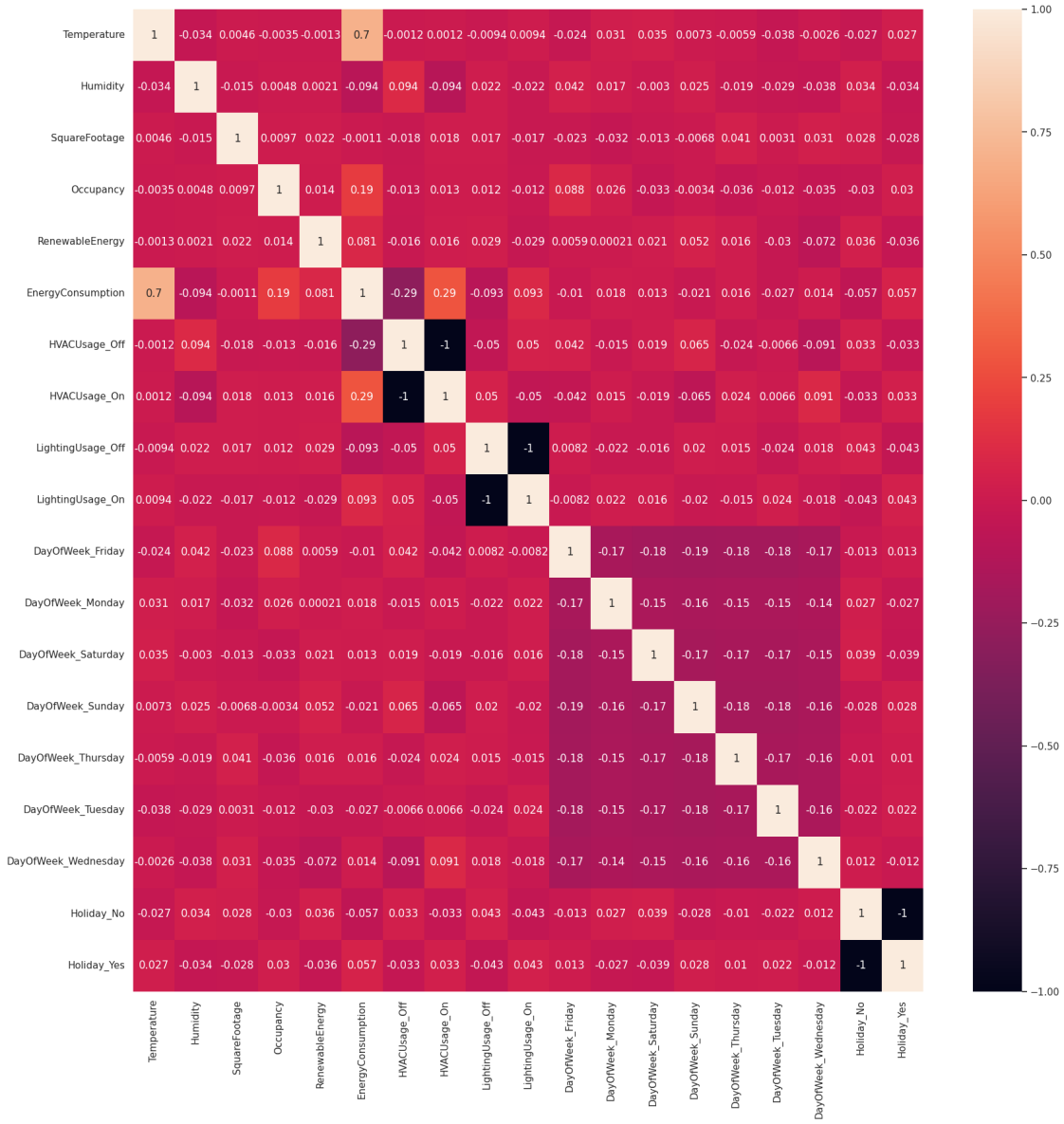
[88]:
```
sns.pairplot(df, x_vars=["LightingUsage_Off", "LightingUsage_On"],␣
↪y_vars=["EnergyConsumption"], hue="EnergyConsumption")
```

[88]: <seaborn.axisgrid.PairGrid at 0x7f23565e05e0>



[89]:
```
# Calculate the correlation matrix for the dataset
corr_matrix = df.corr()

# Set the figure size
sns.set(rc={'figure.figsize':(20 , 20)})

# Draw the correlation matrix as a heatmap
sns.heatmap(corr_matrix, annot=True)
```

[89]: <Axes: >

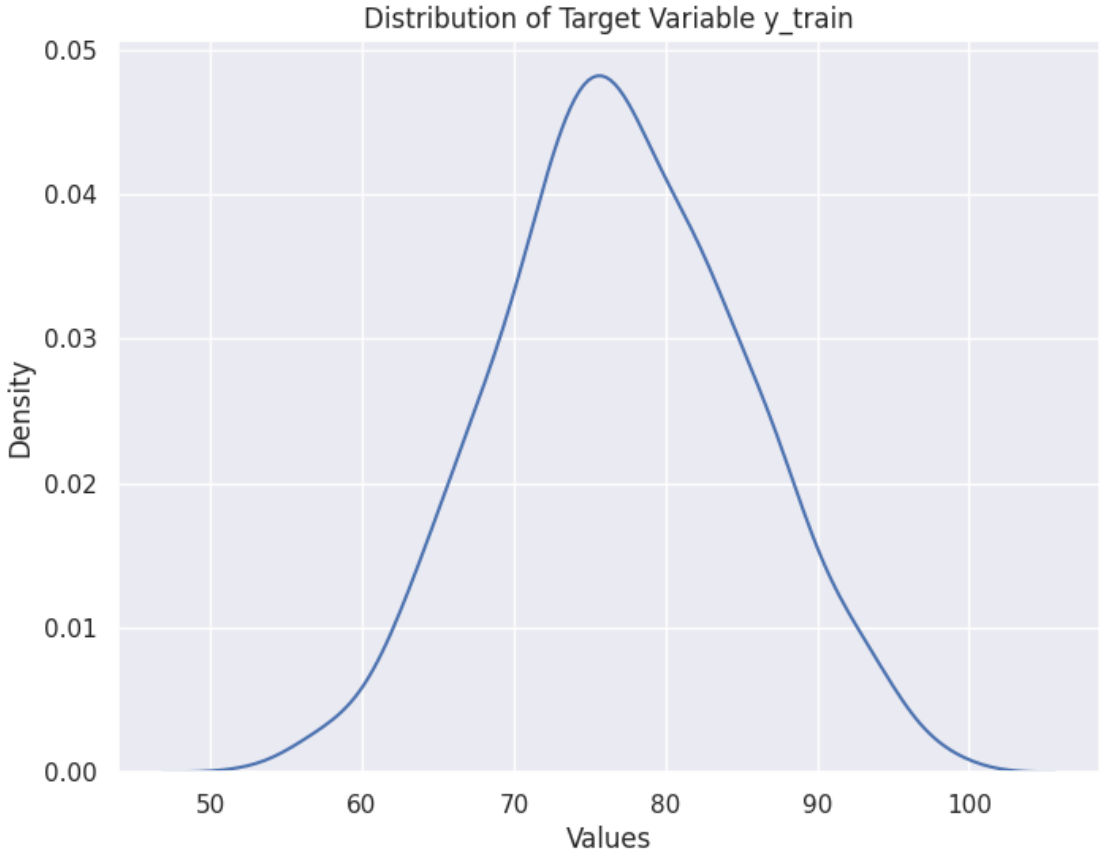


[90]:
```
# Split all the data points in to X and y
X = df.drop("EnergyConsumption", axis=1)
y = df["EnergyConsumption"]
```

[91]:
```
# Split data in to training and testing set
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=42)
```

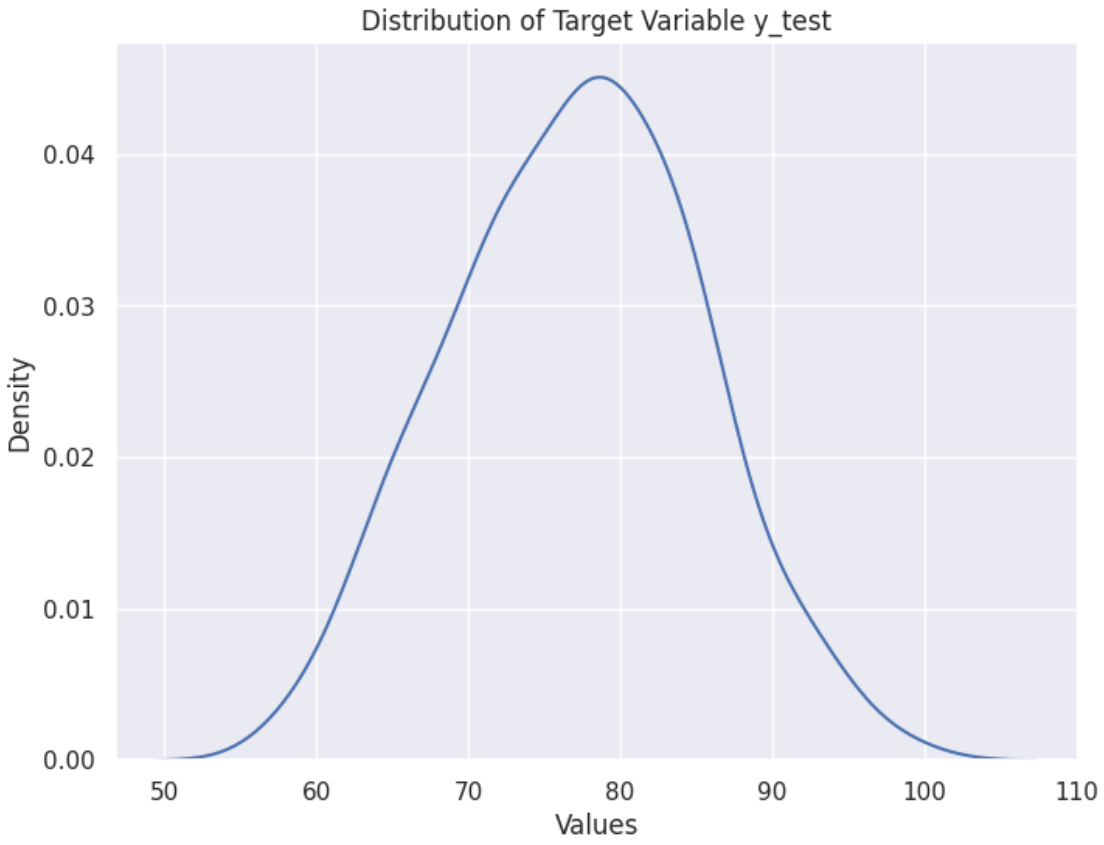
[92]:
```
# Check data distribution in each y_train and y_test
import matplotlib.pyplot as plt
```

```
# Plotting the hist plot
plt.figure(figsize=(8, 6))
sns.kdeplot(y_train)

plt.title('Distribution of Target Variable y_train')
plt.xlabel('Values')
plt.ylabel('Density')
plt.show()
```



[93]:
```
# Check data distribution in each y_train and y_test
import matplotlib.pyplot as plt

# Plotting the hist plot
plt.figure(figsize=(8, 6))
sns.kdeplot(y_test)

plt.title('Distribution of Target Variable y_test')
plt.xlabel('Values')
```

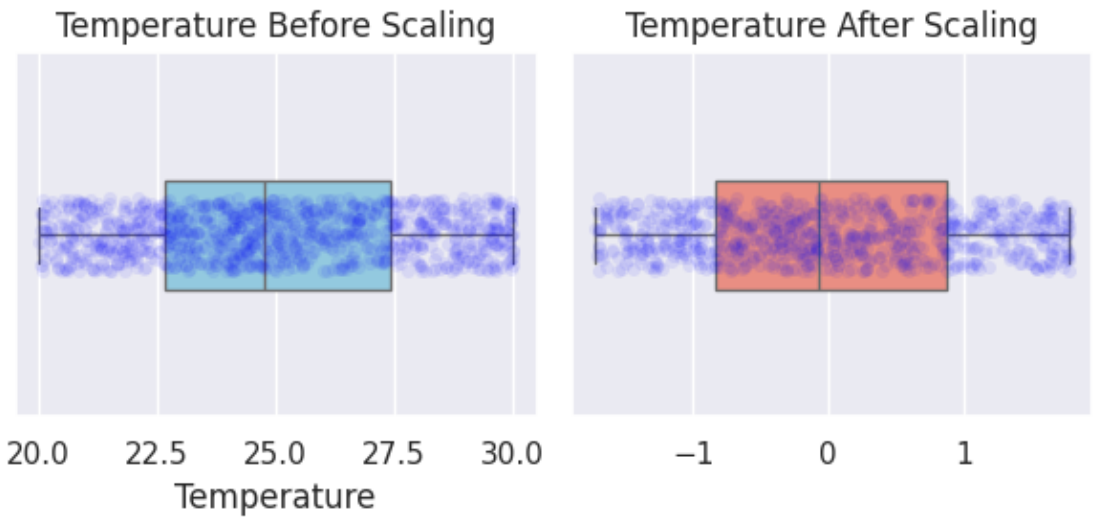
```
plt.ylabel('Density')
plt.show()
```



[94]:
```
# Feature scaling using standard scaler from sklearn to standardize the data in␣
 ↪a fixed range
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

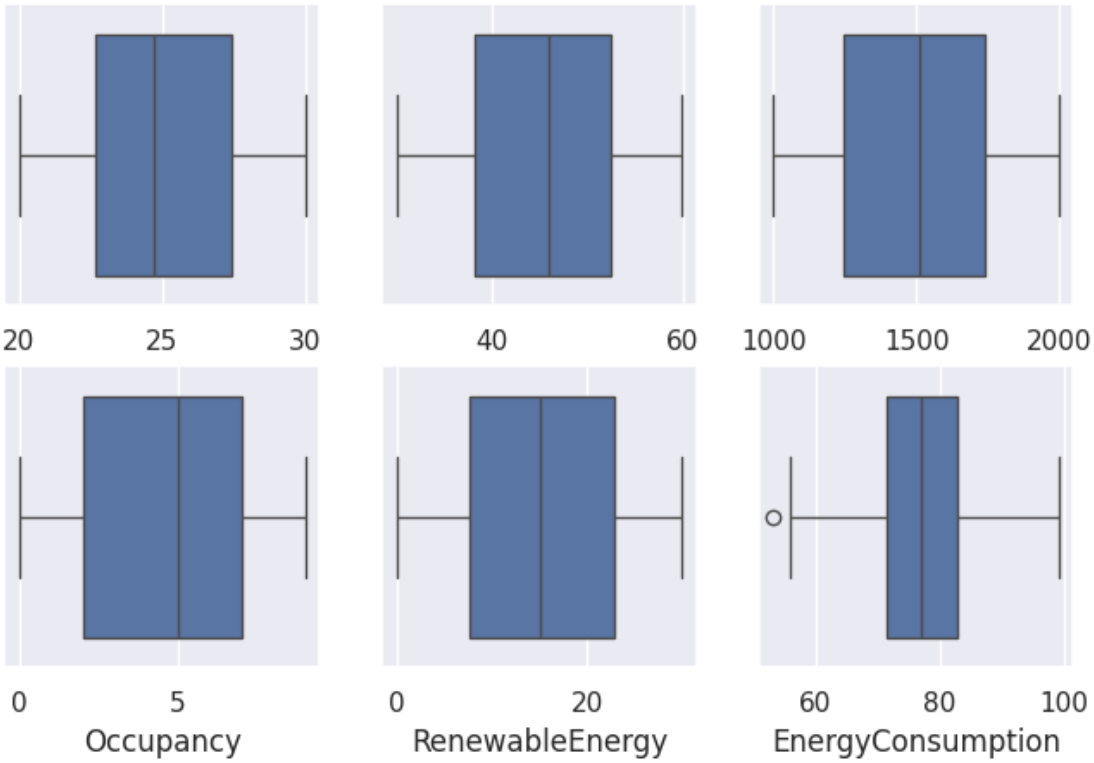
[95]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create boxplots of Temperature before and after scaling
plt.figure(figsize=(6, 3))

# Before scaling
plt.subplot(1, 2, 1)
sns.boxplot(x=df["Temperature"], color="skyblue", width=0.3)  # Customize␣
 ↪boxplot appearance
```

```
sns.stripplot(x=df["Temperature"], color="blue", alpha=0.08)  # Add data points␣
 ↪as a strip plot
plt.title("Temperature Before Scaling")

# After scaling
plt.subplot(1, 2, 2)
sns.boxplot(x=X_train_scaled[:, 0], color="salmon", width=0.3)  # Customize␣
 ↪boxplot appearance
sns.stripplot(x=X_train_scaled[:, 0], color="blue", alpha=0.08)  # Add data␣
 ↪points as a strip plot
plt.title("Temperature After Scaling")

plt.tight_layout()  # Ensures tight layout to prevent overlapping
plt.show()
```

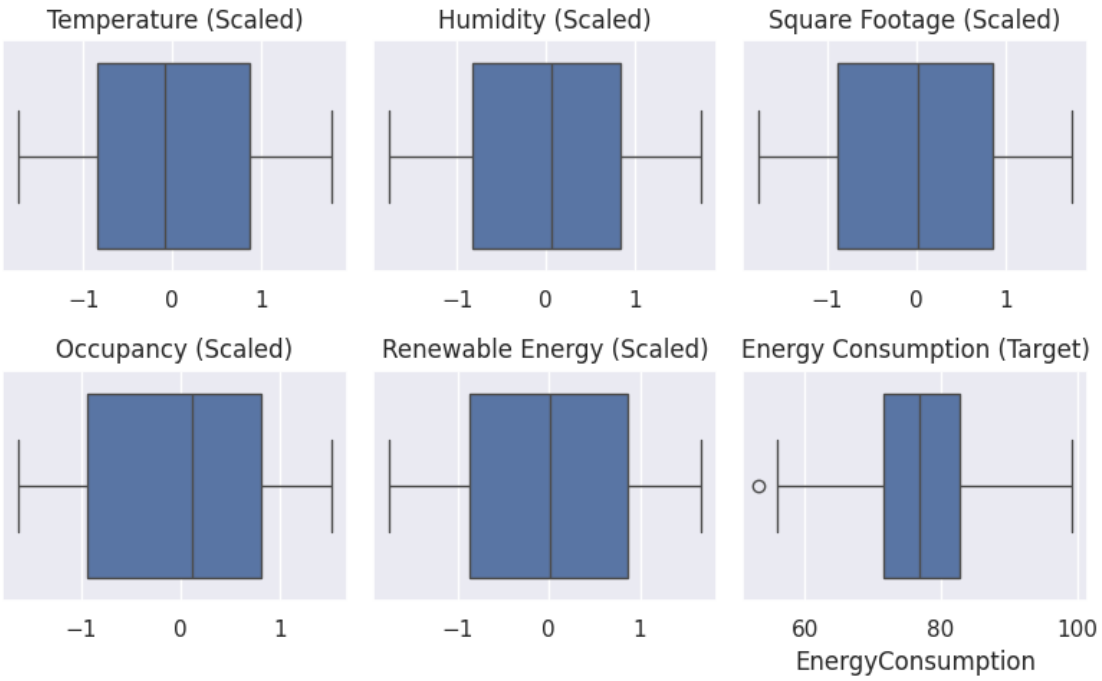


[96]:
```
# Before scaling
fig, axes = plt.subplots(2, 3, figsize=(8, 5))

# Plot boxplots for features and target variable before scaling
sns.boxplot(x=df['Temperature'], ax=axes[0][0])
sns.boxplot(x=df['Humidity'], ax=axes[0][1])
sns.boxplot(x=df['SquareFootage'], ax=axes[0][2])
sns.boxplot(x=df['Occupancy'], ax=axes[1][0])
sns.boxplot(x=df['RenewableEnergy'], ax=axes[1][1])
sns.boxplot(x=df['EnergyConsumption'],ax=axes[1][2])
```

[96]:
```
<Axes: xlabel='EnergyConsumption'>
```



[97]:
```
# After scaling
fig, axes = plt.subplots(2, 3, figsize=(8, 5))

# Plot boxplots for scaled features and target variable
sns.boxplot(x=X_train_scaled[:, 0], ax=axes[0][0])
sns.boxplot(x=X_train_scaled[:, 1], ax=axes[0][1])
sns.boxplot(x=X_train_scaled[:, 2], ax=axes[0][2])
sns.boxplot(x=X_train_scaled[:, 3], ax=axes[1][0])
sns.boxplot(x=X_train_scaled[:, 4], ax=axes[1][1])
sns.boxplot(x=y_train, ax=axes[1][2])

# Added captions for each feature scaled plot
axes[0][0].set_title('Temperature (Scaled)')
axes[0][1].set_title('Humidity (Scaled)')
axes[0][2].set_title('Square Footage (Scaled)')
axes[1][0].set_title('Occupancy (Scaled)')
axes[1][1].set_title('Renewable Energy (Scaled)')
axes[1][2].set_title('Energy Consumption (Target)')

plt.tight_layout()
plt.show()
```



3 3. Model Build

Model built using of two regression algorithms : Linear regression and Random forest regression

[100]:
```
# Linear regression
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train) # Train the Liner Regression model with␣
 ↪the best parameters
lr_predictions = lr_model.predict(X_test_scaled) # Make predictions using the␣
 ↪trained Liner Regression model
```

[101]:
```
# Random forest regression
rf_model = RandomForestRegressor(random_state=42) # Initialize Random Forest␣
 ↪Regressor
rf_model.fit(X_train_scaled,y_train)
rf_predictions = rf_model.predict(X_test_scaled)
```

4 4. Model evaluation

[106]:
```
# Evaluation metrics import
from sklearn.metrics import mean_squared_error, r2_score,mean_absolute_error

# for Linear regression
lr_mse = mean_squared_error(y_test, lr_predictions)
lr_mae = mean_absolute_error(y_test, lr_predictions)
lr_r2  = r2_score(y_test, lr_predictions)
lr_rmse = mean_squared_error(y_test, lr_predictions, squared=False)

# print evaluation metrics for linera regression

print(f'Linear regression MSE: {lr_mse}')
print(f'Linear regression MAE: {lr_mae}')
print(f'Linear regression R-squared error: {lr_r2}')
print(f'Linear regression RMSE: {lr_rmse}')
```

```
Linear regression MSE: 26.51953131219383
Linear regression MAE: 4.121254457226358
Linear regression R-squared error: 0.5951208900046383
Linear regression RMSE: 5.149711769817203
```

[108]:
```
# for Random forest regression
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_r2  = r2_score(y_test, rf_predictions)
rf_rmse = mean_squared_error(y_test, rf_predictions, squared=False)

# print evaluation metrics for linera regression
print(f'Random forest regression MSE: {rf_mse}')
print(f'Random forest regression MAE: {rf_mae}')
print(f'Random forest regression R-squared error: {rf_r2}')
print(f'Random forest regression RMSE: {rf_rmse}')
```

```
Random forest regression MSE: 29.947677697259383
Random forest regression MAE: 4.406812620621577
Random forest regression R-squared error: 0.5427826778024885
Random forest regression RMSE: 5.472447139740994
```

5 5. Hyper parameter testing

[113]:
```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid for Linear Regression
param_grid_lr = {
    'fit_intercept': [True, False],
```

```
    'positive': [True, False]  # Additional valid parameter for LinearRegression
}

# Create the GridSearchCV object for Linear Regression
grid_search_lr = GridSearchCV(LinearRegression(), param_grid_lr, cv=5)
grid_search_lr.fit(X_train_scaled, y_train)

# Get the best hyperparameters
best_params_lr = grid_search_lr.best_params_

# Print the best hyperparameters for Linear Regression
print("Best hyperparameters for Linear Regression:")
print(best_params_lr)
```

```
Best hyperparameters for Linear Regression:
{'fit_intercept': True, 'positive': True}
```

[114]:
```
# Define the hyperparameter grid for Random Forest Regression
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Create the GridSearchCV object for RF
grid_search_rf = GridSearchCV(RandomForestRegressor(), param_grid_rf, cv=5)
grid_search_rf.fit(X_train_scaled, y_train)

# best hyperparameters
best_params_rf = grid_search_rf.best_params_

# Print the best hyperparameters for RF
print("Best hyperparameters for Random Forest Regression:")
print(best_params_rf)
```

```
Best hyperparameters for Random Forest Regression:
{'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 300}
```

[117]:
```
# Best hyperparameters obtained from GridSearchCV
best_params_rf = {'max_depth': 5, 'min_samples_split': 10, 'n_estimators': 300}

# RandomForestRegressor with best hyperparameters
rf_model_best_hyperparams = RandomForestRegressor(**best_params_rf)
rf_model_best_hyperparams.fit(X_train_scaled, y_train)

# predictions using the trained model
rf_predictions = rf_model_best_hyperparams.predict(X_test_scaled)
```

[120]:
```
# for Random forest regression with best hyper parameters
rf_mse_best_hyperparams = mean_squared_error(y_test, rf_predictions)
rf_mae_best_hyperparams = mean_absolute_error(y_test, rf_predictions)
rf_r2_best_hyperparams  = r2_score(y_test, rf_predictions)
rf_rmse_best_hyperparams = mean_squared_error(y_test, rf_predictions,␣
 ↪squared=False)

# print evaluation metrics for linera regression
print(f'Random forest regression MSE: {rf_mse_best_hyperparams}')
print(f'Random forest regression MAE: {rf_mae_best_hyperparams}')
print(f'Random forest regression R-squared error: {rf_r2_best_hyperparams}')
print(f'Random forest regression RMSE: {rf_rmse_best_hyperparams}')
```

```
Random forest regression MSE: 29.253776982352605
Random forest regression MAE: 4.2686008245295355
Random forest regression R-squared error: 0.5533766019774389
Random forest regression RMSE: 5.40867608406647
```

6 6. Prediction

[124]:
```
# predicting values by two algorithms
linear_regression_predictions = lr_model.predict(X_test_scaled)
random_forest_predictions = rf_model_best_hyperparams.predict(X_test_scaled)

# Create a dataframe to compare actual and predicted values
prediction_df = pd.DataFrame({'Actual': y_test, 'Linear Regression':␣
 ↪linear_regression_predictions, 'Random Forest': random_forest_predictions})

# Print the dataframe
print(prediction_df.to_string())
```

```
        Actual  Linear Regression  Random Forest
521  86.920611          83.843847      82.783182
737  88.351606          81.599707      82.582565
740  79.431363          76.195410      73.680150
660  90.009188          88.365332      86.877820
411  83.891100          75.201269      74.504548
678  87.549041          80.789160      82.986137
626  79.697237          78.568457      77.882991
513  80.914057          72.935644      74.145816
859  85.133856          77.453222      79.493869
136  71.017140          68.021582      68.119407
811  79.558214          78.171972      78.840628
76   69.365745          73.318457      71.904269
636  80.821657          78.859472      78.794425
973  85.556236          78.724707      78.888472
938  71.081568          76.818457      79.145865
```

899	72.992079	71.271582	73.569141
280	71.568813	78.406347	75.614505
883	81.097206	78.318457	78.071535
761	73.976840	75.420019	77.783835
319	70.016667	66.670019	68.082234
549	76.374345	75.015722	71.938408
174	78.586138	73.974707	74.888015
371	78.979055	82.070410	78.833000
527	76.874451	78.115332	78.452900
210	86.550599	88.109472	85.217982
235	78.733298	73.568457	75.767915
101	85.181974	79.396582	78.067276
986	62.729873	70.461035	69.214816
902	77.545163	74.474707	76.942796
947	83.134823	80.490332	84.263959
346	68.243502	70.287207	71.461587
139	80.003847	81.695410	82.288378
621	67.391386	73.703222	75.967053
499	87.421468	88.662207	88.005576
370	84.010971	78.693457	77.287628
198	90.011593	86.703222	86.452920
687	86.268839	85.857519	84.031468
584	73.701477	78.060644	77.643170
901	80.132973	78.304785	78.133652
59	84.038684	79.959082	78.470598
328	71.682160	75.427832	76.669837
96	75.982381	81.984472	81.984905
312	69.615978	69.351660	68.721733
974	71.359781	76.890722	76.547158
299	71.792376	74.021582	72.261832
277	78.636885	79.693457	81.759684
924	78.657157	76.609472	76.949886
601	73.153608	70.568457	71.863703
439	72.791163	72.687597	74.695041
837	74.615378	77.070410	76.585975
570	82.223456	77.656347	76.942394
879	84.690911	84.287207	83.202148
261	72.993170	74.226660	72.747009
578	67.628111	70.906347	74.575802
23	78.394208	78.357519	77.985065
30	81.049581	89.445410	87.314175
617	74.441399	76.507910	76.776010
10	82.274434	78.109472	79.730919
221	86.715598	83.656347	82.507544
820	74.977281	74.693457	77.118020
296	64.159080	68.117285	66.475464
54	91.871965	88.789160	89.370241
542	67.262869	69.890722	72.047506

209	75.369438	68.453222	67.040232
604	64.054485	68.140722	68.059900
692	79.786300	85.859472	85.256702
662	79.257313	75.802832	76.923788
866	72.040034	69.109472	69.863642
70	64.963184	71.265722	71.168765
543	76.199312	74.185644	75.555653
107	63.861910	68.146582	70.566595
493	85.457155	81.271582	81.526993
590	60.396999	67.865332	68.318932
741	90.898507	82.242285	83.163169
292	85.925553	85.880957	87.047973
289	94.178518	86.390722	86.251776
652	72.805270	85.365332	85.067311
39	77.228457	74.203222	72.873960
589	83.501795	83.382910	79.002123
307	66.001780	76.570410	76.777118
679	84.959712	78.648535	78.395074
66	60.144238	76.234472	76.195033
275	74.126537	67.146582	70.571641
67	80.192646	77.693457	77.472044
318	81.912041	82.390722	82.391112
548	68.447654	67.779394	65.804461
998	76.950389	79.648535	79.018961
714	70.559427	74.591894	76.357189
753	58.242096	63.904394	65.816619
327	63.945190	70.052832	71.203931
382	79.855901	78.312597	78.249096
451	61.498862	66.505957	67.574721
522	63.978618	71.521582	71.313882
218	73.999839	82.031347	83.811038
787	70.069460	69.302832	70.675086
436	91.007122	88.515722	87.758999
764	79.766464	75.896582	72.760186
88	77.384478	73.336035	75.462653
63	78.187281	73.060644	74.807737
826	83.054685	89.031347	85.747736
716	81.756024	83.546972	81.763783
351	86.138042	81.171972	78.720725
936	63.298668	70.380957	70.955235
256	93.694807	86.312597	86.454086
635	78.675005	71.664160	71.976535
644	80.018592	68.265722	69.463342
554	66.265804	75.367285	76.887023
959	76.767503	81.875097	82.282646
168	75.614982	81.224707	81.514644
917	79.297405	76.123144	80.602792
528	84.759113	83.334082	82.973877

823	67.890218	74.703222	77.163032
985	85.271028	82.937597	82.086448
816	83.384026	81.779394	83.956762
86	65.972461	69.445410	67.660427
432	77.040569	74.701269	77.293789
184	78.286528	79.601660	78.691037
978	87.428844	78.521582	78.740288
534	70.170270	69.664160	70.293367
294	71.369356	64.609472	67.386784
892	89.392006	82.429785	81.750014
425	76.067385	87.615332	86.980913
713	75.805245	78.414160	79.367170
260	68.468534	73.779394	72.244261
237	72.431190	69.521582	70.853024
559	58.097010	66.045019	65.612357
583	83.446356	77.662207	79.099831
445	70.679166	75.023535	75.389387
867	77.120780	73.099707	75.127932
800	82.910470	80.255957	81.113569
599	77.667740	80.677832	82.960071
849	73.224626	70.828222	70.490398
265	91.908381	88.367285	85.818347
995	82.306692	82.162207	82.863746
529	64.290072	63.929785	63.930507
55	69.692263	69.951269	70.443756
120	76.936453	69.201269	71.422227
215	72.996281	76.888769	76.501276
25	72.929844	71.796972	74.245111
72	75.913906	74.802832	76.454060
44	77.851776	73.461035	71.761403
247	92.521211	88.945410	87.213640
721	76.737686	80.117285	82.908489
281	91.490982	88.359472	86.179626
893	82.976827	79.898535	82.203813
914	74.342438	81.263769	82.885411
810	71.542827	69.623144	71.013728
244	77.158927	70.904394	70.990689
822	86.122349	86.609472	87.316541
321	74.400792	79.828222	78.153145
643	76.431354	73.046972	74.157269
158	71.131062	72.357519	71.195938
977	67.449944	67.039160	68.129541
429	67.174573	70.873144	71.916674
941	64.627595	64.640722	65.940464
462	82.215644	76.218847	74.839649
309	83.219395	77.646582	77.757811
697	83.063549	83.211035	84.513240
60	77.673792	74.724707	74.459824

884	65.309203	65.453222	63.616683
595	72.366626	66.976660	65.693202
767	88.084489	81.591894	81.417659
649	73.479641	82.078222	82.353357
650	66.337141	77.623144	77.341617
865	65.356061	67.693457	68.140676
668	73.205783	81.209082	82.800634
298	83.957050	77.013769	77.082356
689	78.924743	83.021582	84.617761
314	84.492197	83.865332	84.740501
310	96.244796	87.742285	86.564220
361	83.730321	82.179785	81.671685
479	77.778027	82.507910	83.187308
110	81.893987	81.953222	82.248918
989	71.775781	66.125097	67.628453
486	84.219978	83.203222	82.922920
363	84.270071	77.084082	78.560566
254	79.496601	79.091894	78.170530
259	77.413859	80.334082	82.643613
802	71.583166	73.990332	71.871215
677	79.497795	78.390722	77.747371
494	69.009077	67.711035	67.295120
670	98.761754	89.459082	87.229431
377	93.617494	82.257910	79.019747
526	83.983753	88.375097	87.272218
845	78.484017	74.810644	71.829897
137	64.697910	72.851660	74.967877
355	81.719620	83.490332	82.854693
365	82.859636	77.287207	77.922060
942	69.598318	72.632910	71.035431
749	61.491394	78.765722	81.518337
948	69.887078	68.929785	70.566838
829	81.273754	79.388769	81.968683
656	81.385260	80.765722	82.550385
199	73.377047	69.412207	70.115159
213	73.586961	78.679785	77.303434
408	86.919371	86.125097	84.353505
332	66.833390	69.750097	70.143433
208	80.043926	80.857519	81.509640
613	76.399561	87.625097	86.871882
78	67.484788	67.711035	67.066884