# Plano de Implementação de Modelo de Machine Learning

## Este protótipo contém:

#### Geração de Dados Sintéticos baseados nas KPIs geradas pelo SmartGrid

#### Implementação de Modelo de Isolation Forest para detecção de anomalias

#### Identificação de padrões de consumo

### Import de libraries e tools que serão utilizados

In [1]:
```
import pandas as pd
import random
from datetime import datetime, timedelta
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_
from sklearn.ensemble import RandomForestRegressor
import numpy as np
```

### Dados Sintéticos

In [2]:
```
data = []
start_date = datetime(2023, 1, 1)
trend_coefficients = [random.uniform(0.95, 1.05) for _ in range(10)]

for home_id in range(10):
    current_date = start_date
    for _ in range(1000):
        timestamp = current_date.strftime("%Y-%m-%d %H:%M:%S")

        forward_active_energy = round(random.uniform(0.8, 1.2) * trend_coefficients
        reverse_active_energy = round(random.uniform(0.8, 1.2) * trend_coefficients

        home_data = {
            "Home_ID": home_id,
            "Timestamp": timestamp,
            "Forward_Active_Energy": forward_active_energy,
            "Reverse_Active_Energy": reverse_active_energy,
        }
        data.append(home_data)

        current_date += timedelta(minutes=5)

df = pd.DataFrame(data)
df
```

Out[2]:

| | Home_ID | Timestamp | Forward_Active_Energy | Reverse_Active_Energy |
|---|---|---|---|---|
| **0** | 0 | 2023-01-01 00:00:00 | 1.06 | 1.13 |
| **1** | 0 | 2023-01-01 00:05:00 | 0.86 | 1.10 |
| **2** | 0 | 2023-01-01 00:10:00 | 1.08 | 0.87 |
| **3** | 0 | 2023-01-01 00:15:00 | 1.17 | 0.90 |
| **4** | 0 | 2023-01-01 00:20:00 | 1.17 | 1.17 |
| **...** | ... | ... | ... | ... |
| **9995** | 9 | 2023-01-04 10:55:00 | 1.23 | 1.20 |
| **9996** | 9 | 2023-01-04 11:00:00 | 0.87 | 0.83 |
| **9997** | 9 | 2023-01-04 11:05:00 | 1.20 | 1.00 |
| **9998** | 9 | 2023-01-04 11:10:00 | 1.11 | 1.06 |
| **9999** | 9 | 2023-01-04 11:15:00 | 1.23 | 1.02 |

10000 rows × 4 columns

## Qualidade de Dados

In [3]:

```
# Função lambda para identificação de dados duplicados

df2 = df.apply(lambda df: df.duplicated(), axis=1)
df2.sum()
```

Out[3]:

```
Home_ID                    0
Timestamp                  0
Forward_Active_Energy     29
Reverse_Active_Energy    274
dtype: int64
```

In [4]:

```
# Verificação de dados nulos

df.isna().sum()
```

Out[4]:

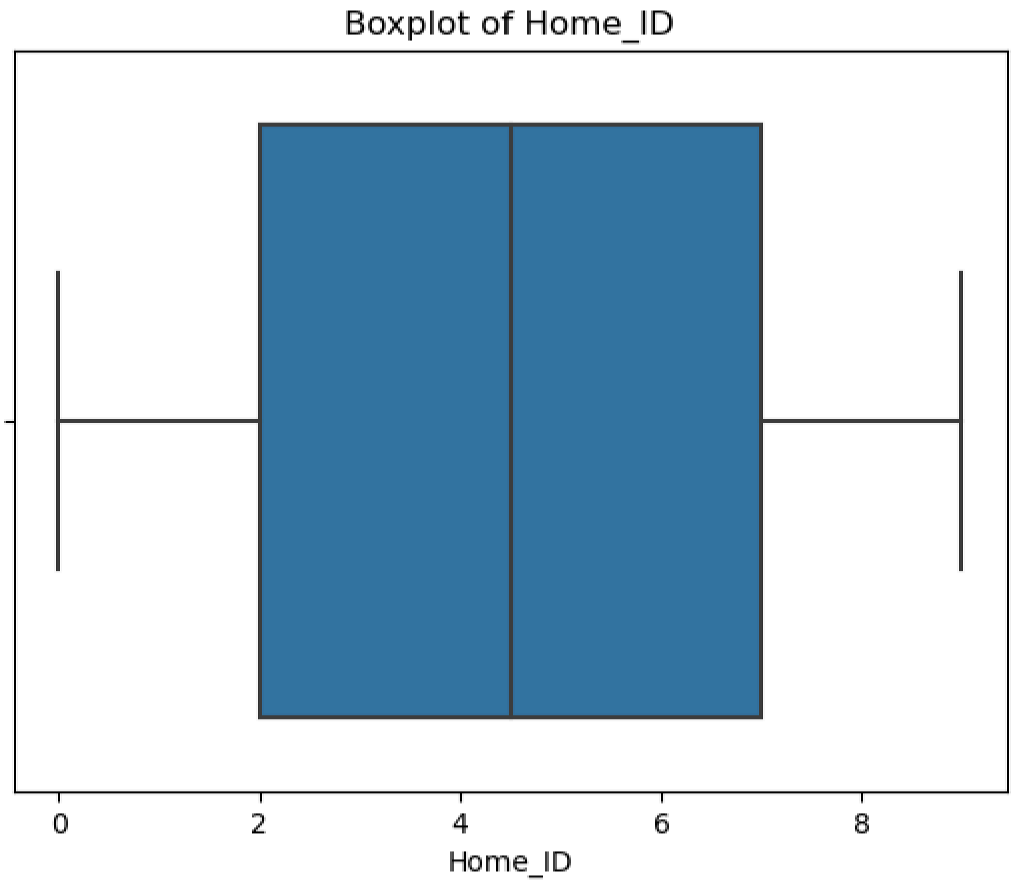
```
Home_ID                  0
Timestamp                0
Forward_Active_Energy    0
Reverse_Active_Energy    0
dtype: int64
```

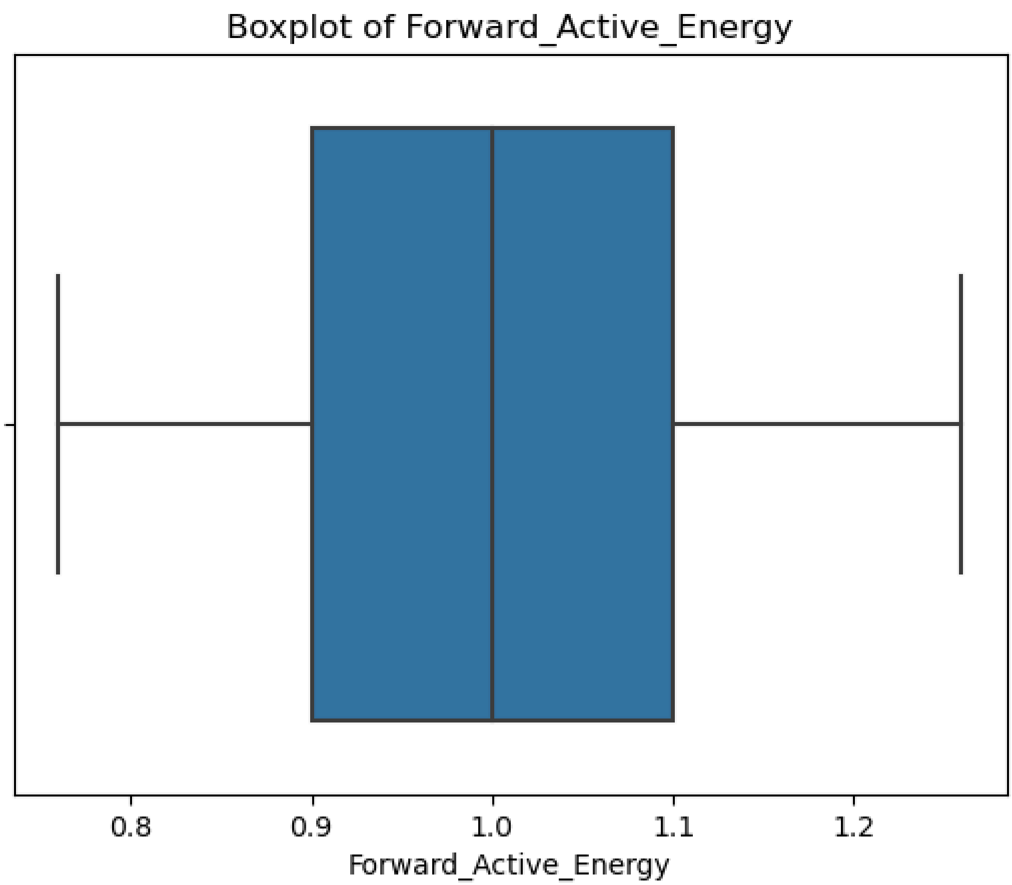
## Boxplot

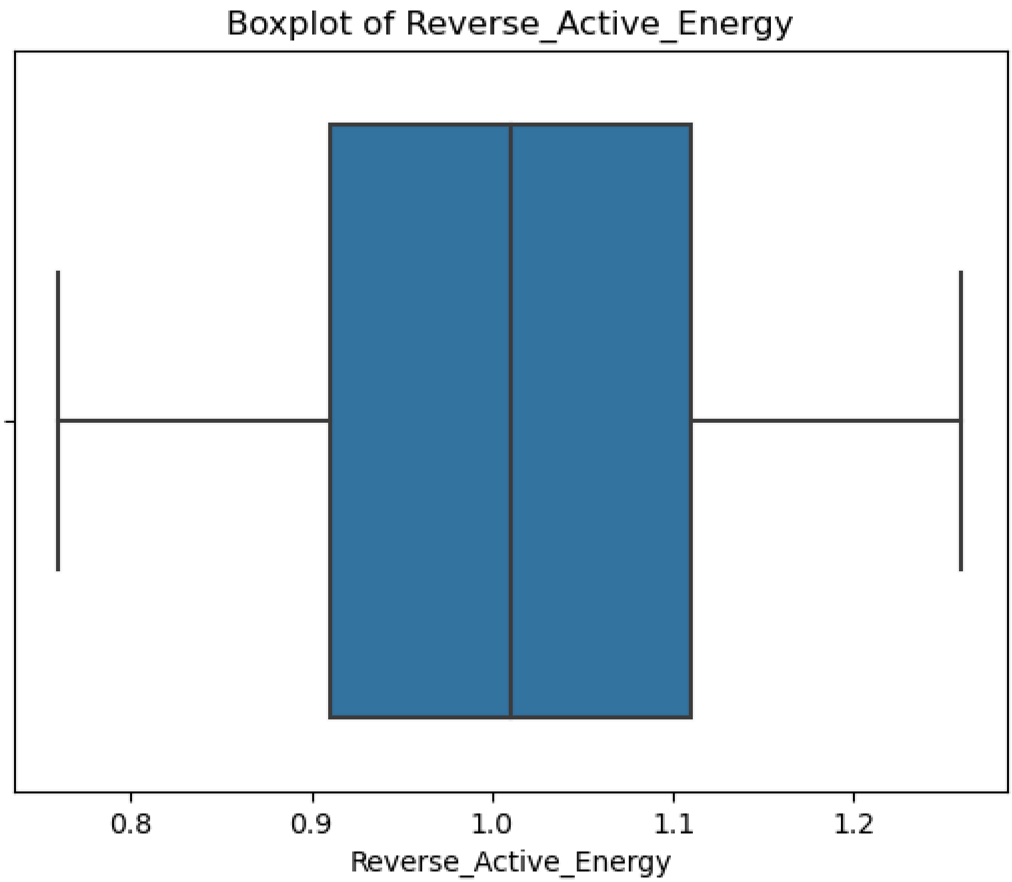
É um método para demonstrar graficamente os grupos de variação de dados numéricos através de seus quartis. Os gráficos Boxplot são úteis para identificar dospersão de dados, simetria, outliers e posições.

In [5]:

```
for c in df.columns:
    if df[c].dtype != 'object':  # Check if the column is numeric
        sns.boxplot(data=df, x=c)
        plt.title(f'Boxplot of {c}')
        plt.show()
        plt.close()
```


Boxplot of Home_ID
0
2
4
6
8
Home_ID


Boxplot of Forward_Active_Energy
0.8
0.9
1.0
1.1
1.2
Forward_Active_Energy

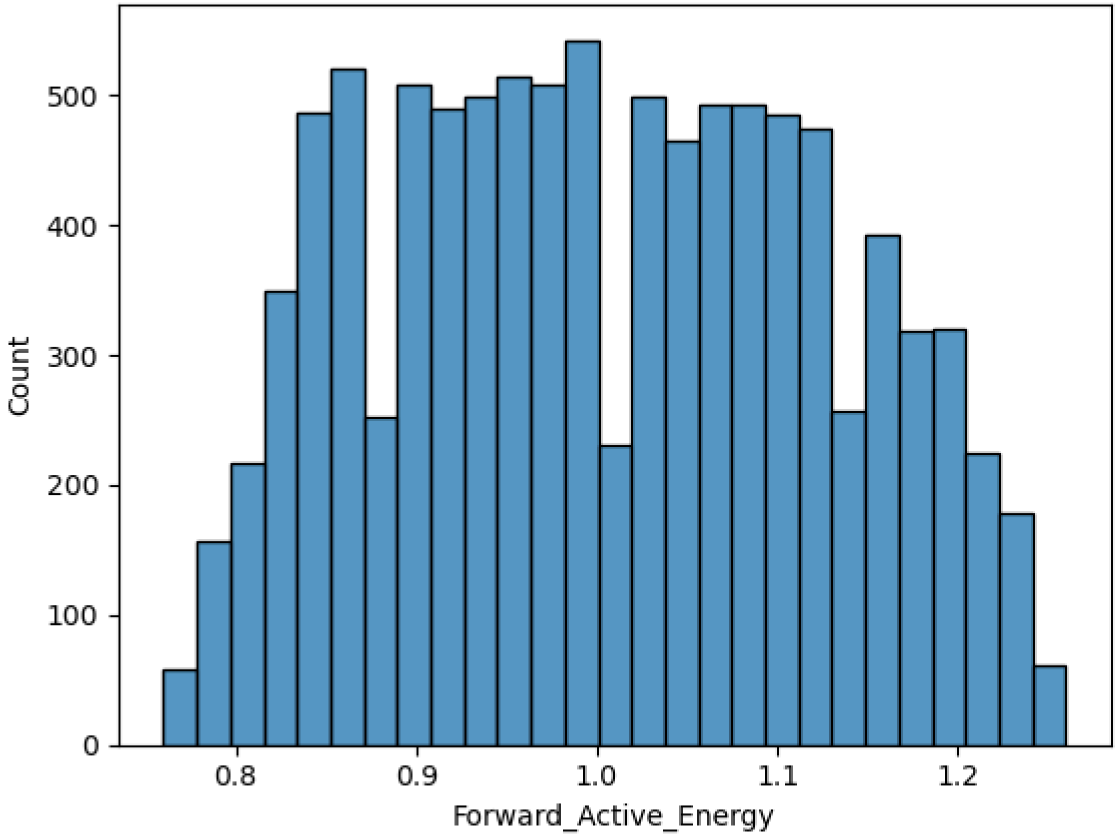


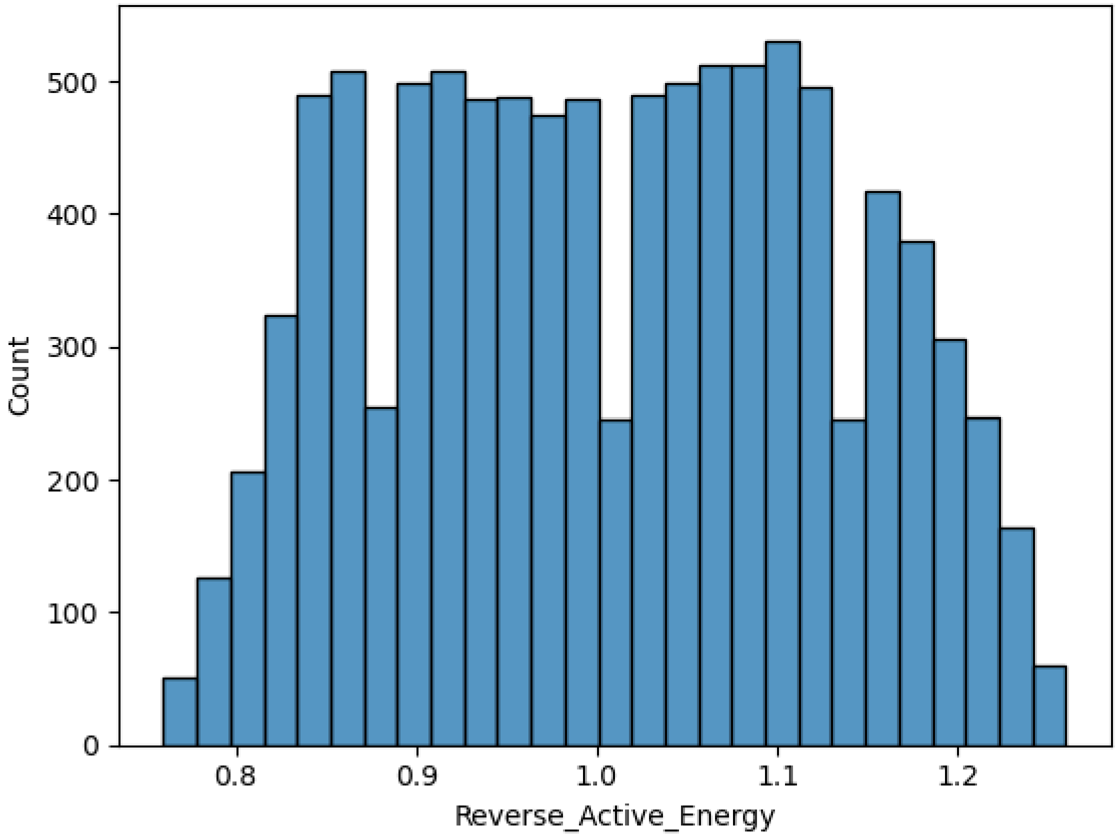

### Histogramas

Este gráfico mostrará as distribuições de frequência, será possível identificar se cada característica é uma distribuição gaussiana ou não

In [6]:
```
df2 = df[['Forward_Active_Energy', 'Reverse_Active_Energy']]

for c in df2:
    ax = sns.histplot(df, x=c)
    plt.show()
    plt.close()
```




## Isolation Forest

In [7]:
```
selected_columns = ["Forward_Active_Energy", "Reverse_Active_Energy"]

X = df[selected_columns]
```

```python
isolation_forest = IsolationForest(contamination=0.054)
isolation_forest.fit(X)
anomaly_scores = isolation_forest.predict(X)

df["Anomaly_Score"] = anomaly_scores

anomalies = df[df["Anomaly_Score"] == -1]
anomalies
```

Out[7]:

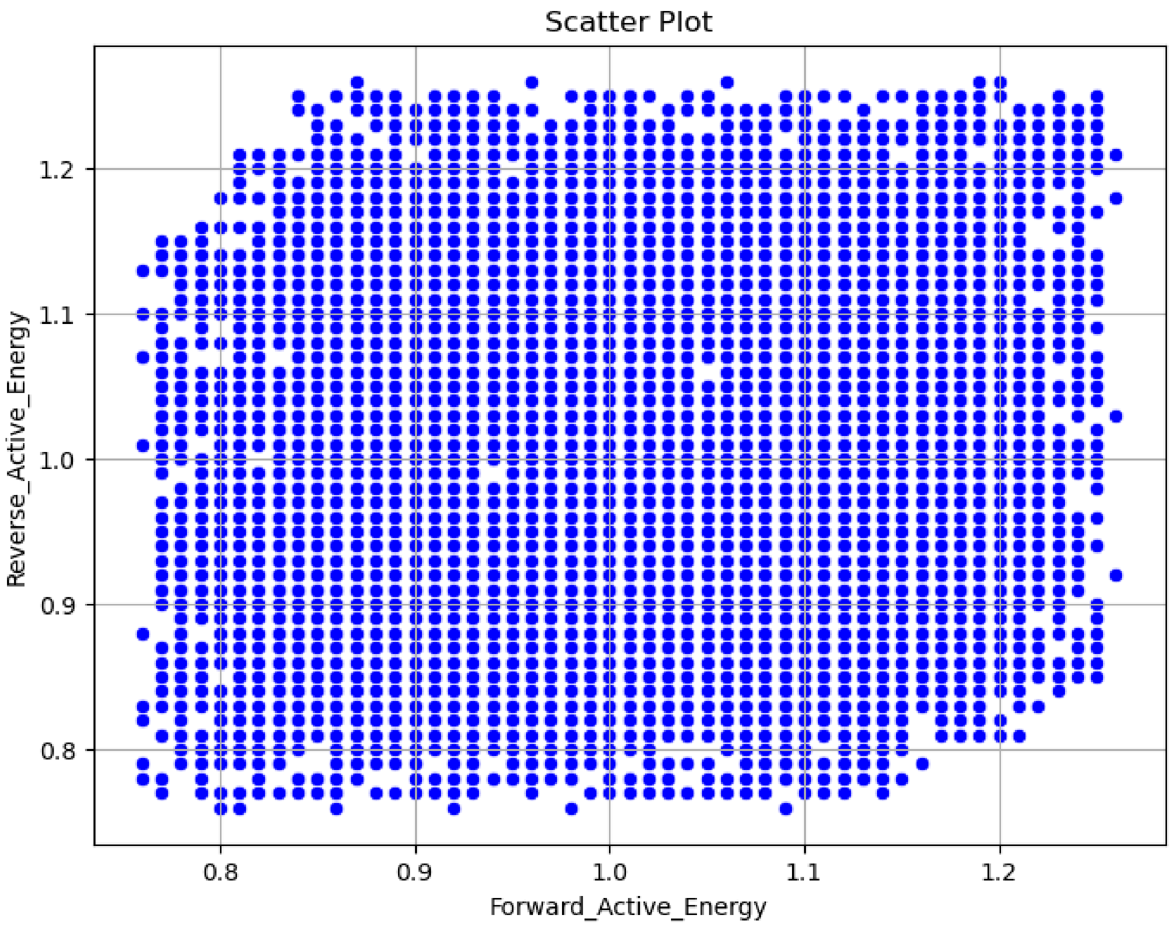
| | Home_ID | Timestamp | Forward_Active_Energy | Reverse_Active_Energy | Anomaly_Score |
|---|---|---|---|---|---|
| **23** | 0 | 2023-01-01 01:55:00 | 0.95 | 1.24 | -1 |
| **47** | 0 | 2023-01-01 03:55:00 | 0.89 | 1.25 | -1 |
| **54** | 0 | 2023-01-01 04:30:00 | 0.88 | 1.25 | -1 |
| **55** | 0 | 2023-01-01 04:35:00 | 1.24 | 1.21 | -1 |
| **76** | 0 | 2023-01-01 06:20:00 | 1.25 | 1.02 | -1 |
| **...** | ... | ... | ... | ... | ... |
| **9904** | 9 | 2023-01-04 03:20:00 | 1.24 | 1.19 | -1 |
| **9908** | 9 | 2023-01-04 03:40:00 | 0.85 | 1.24 | -1 |
| **9924** | 9 | 2023-01-04 05:00:00 | 1.23 | 0.85 | -1 |
| **9969** | 9 | 2023-01-04 08:45:00 | 1.19 | 0.83 | -1 |
| **9970** | 9 | 2023-01-04 08:50:00 | 1.16 | 1.24 | -1 |

538 rows × 5 columns

In [8]:

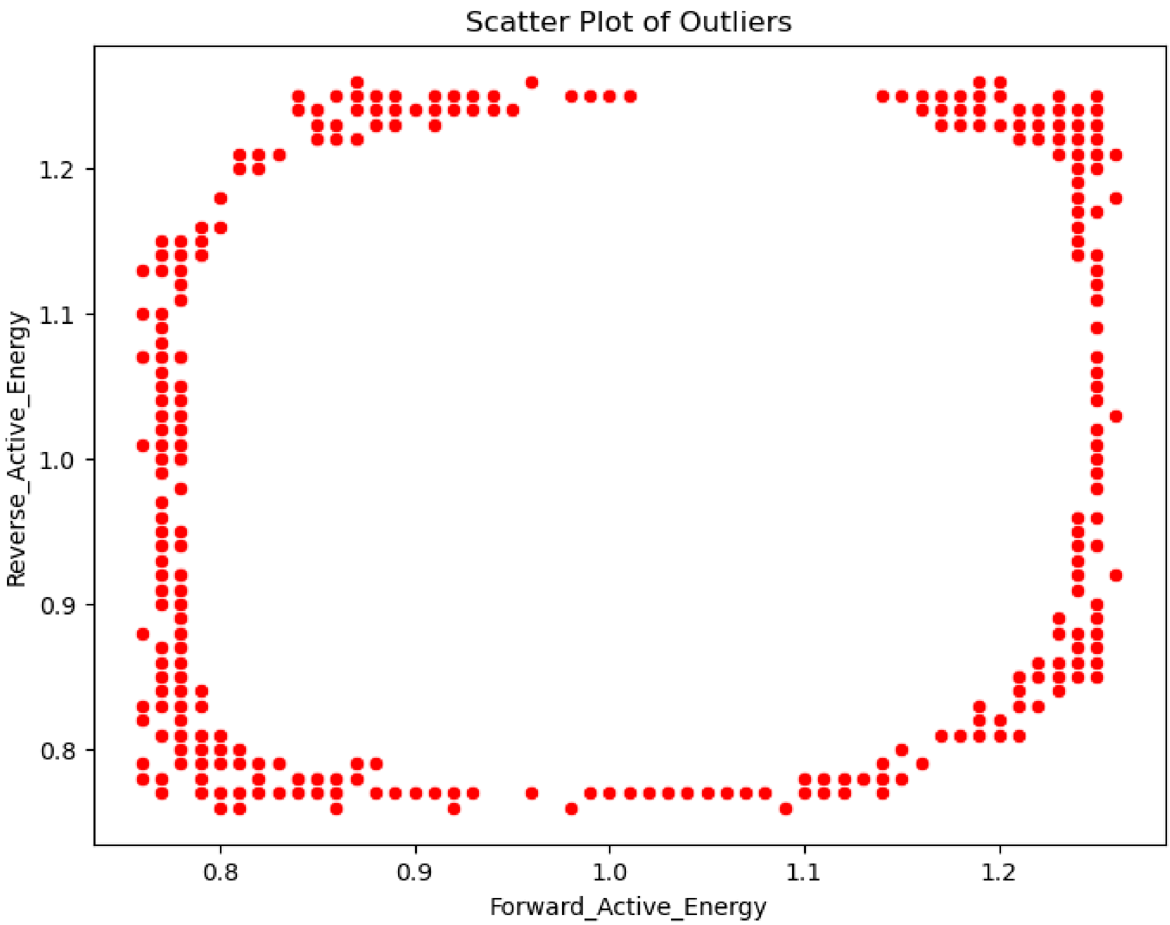
```python
# Scatter Plot serve para verificar a distribuição de dados totais

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x="Forward_Active_Energy", y="Reverse_Active_Energy", mark
plt.xlabel("Forward_Active_Energy")
plt.ylabel("Reverse_Active_Energy")
plt.title("Scatter Plot")
plt.grid(True)
plt.show()
```

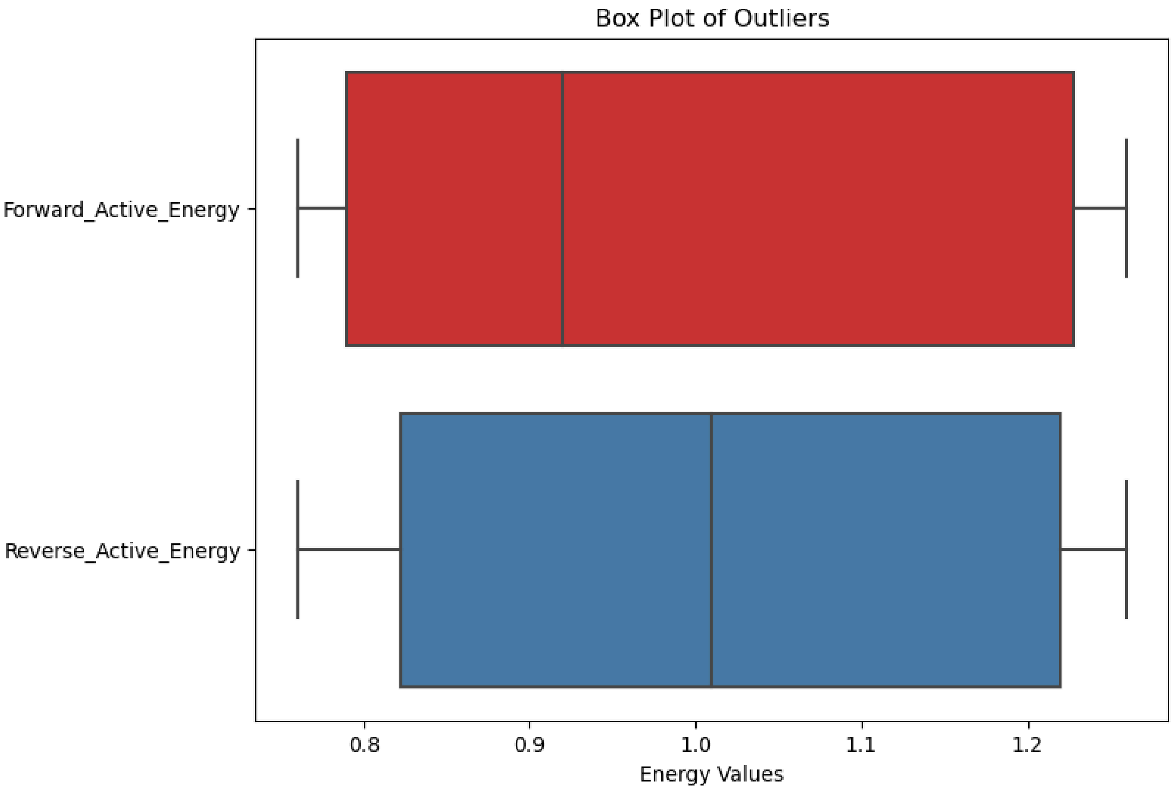



In [9]:
```
# Este Scatter Plot mostra a distribuição de anomalias

plt.figure(figsize=(8, 6))
sns.scatterplot(x='Forward_Active_Energy', y='Reverse_Active_Energy', data=anomalie
plt.xlabel('Forward_Active_Energy')
plt.ylabel('Reverse_Active_Energy')
plt.title('Scatter Plot of Outliers')
plt.show()
```

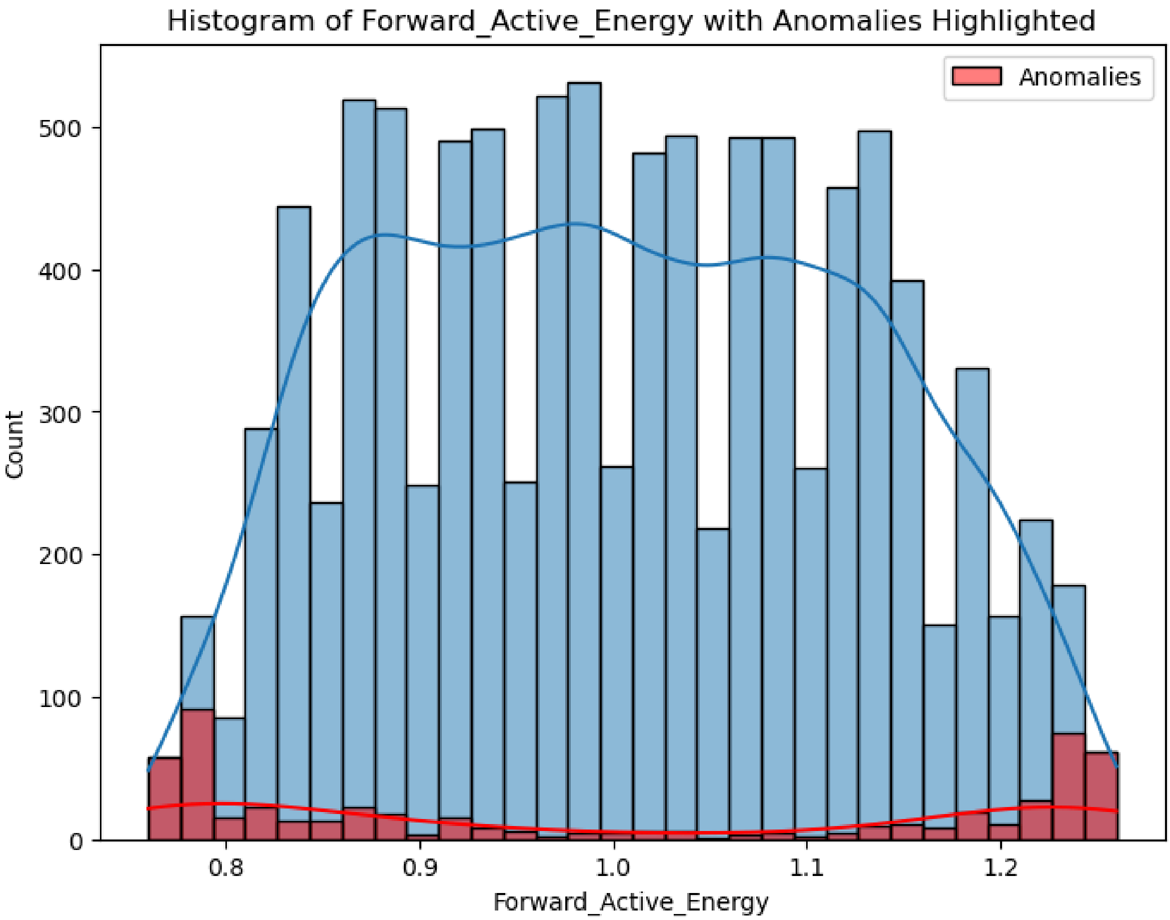



In [10]:
```
plt.figure(figsize=(8, 6))
sns.boxplot(data=anomalies[['Forward_Active_Energy', 'Reverse_Active_Energy']], ori
plt.xlabel('Energy Values')
plt.title('Box Plot of Outliers')
plt.show()
```




## Este histograma faz a verificação entre o comportamento de consumo normal e comportamento anômalo

In [11]:
```
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='Forward_Active_Energy', bins=30, kde=True)
sns.histplot(data=anomalies, x='Forward_Active_Energy', bins=30, kde=True, color='r
plt.xlabel('Forward_Active_Energy')
plt.ylabel('Count')
plt.title('Histogram of Forward_Active_Energy with Anomalies Highlighted')
plt.legend()
plt.show()
```




# Clustering e Identificação de Padrões de Consumo

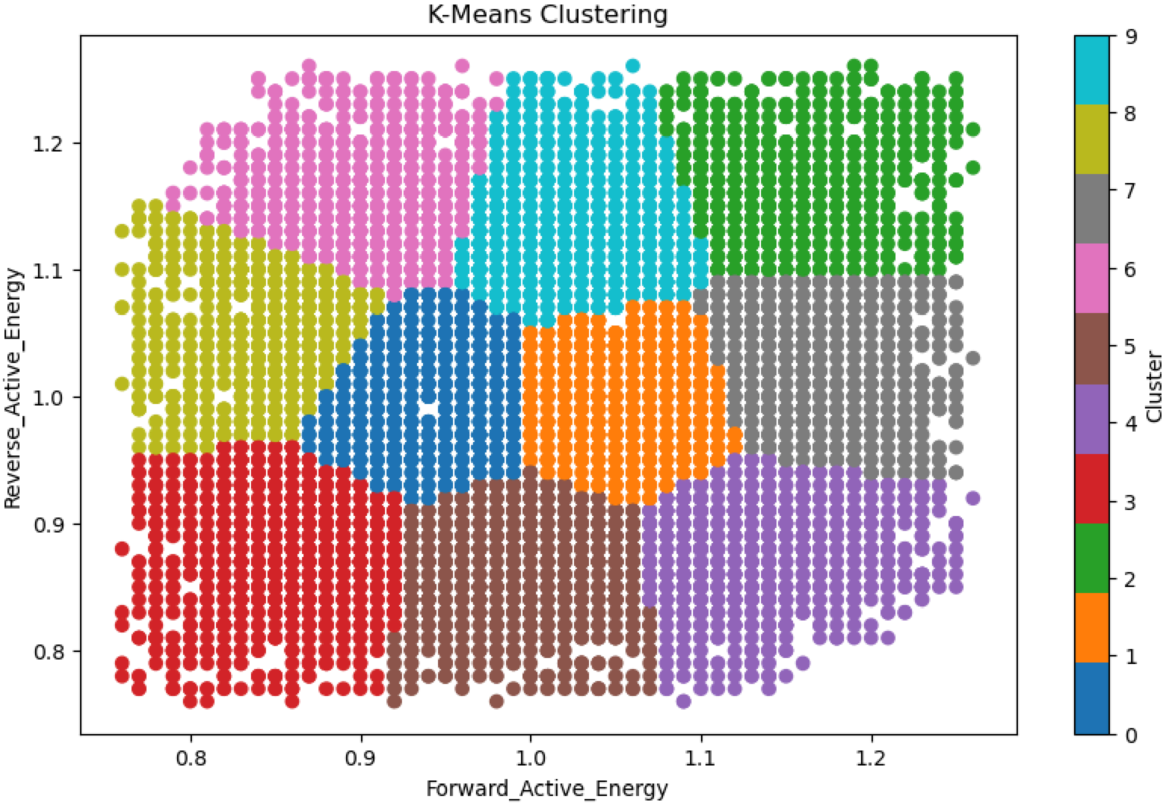
In [12]:
```
X = df[["Forward_Active_Energy", "Reverse_Active_Energy"]]
k = 10
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(X)

df["Cluster_Label"] = kmeans.labels_

custom_cmap = plt.cm.get_cmap('tab10', k)

plt.figure(figsize=(10, 6))
scatter = plt.scatter(df["Forward_Active_Energy"], df["Reverse_Active_Energy"], c=c
plt.xlabel("Forward_Active_Energy")
plt.ylabel("Reverse_Active_Energy")
plt.title("K-Means Clustering")
plt.colorbar(scatter, label="Cluster")
plt.show()
```

```
C:\Users\giuli\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: Future
Warning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set t
he value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\giuli\AppData\Local\Temp\ipykernel_12176\1634787301.py:8: MatplotlibDepre
cationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be
removed two minor releases later. Use ``matplotlib.colormaps[name]`` or ``matplotl
ib.colormaps.get_cmap(obj)`` instead.
  custom_cmap = plt.cm.get_cmap('tab10', k)
```




## Modelo de Train-Test

In [13]:
```
X = df[['Forward_Active_Energy']]
y = df['Reverse_Active_Energy']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_st
```

In [14]:
```
model = IsolationForest()
model.fit(X_train, y_train)
```

Out[14]:
```
▾ IsolationForest
IsolationForest()
```

In [15]:
```
preds = model.predict(X_test)
```

# Métricas de avaliação de Modelo Proposto

In [16]:
```
mae = mean_absolute_error(y_test, preds)
mse = mean_squared_error(y_test, preds)
rmse = np.sqrt(mean_squared_error(y_test, preds))
mape = mean_absolute_percentage_error(y_test, preds)
print(f'MAE:  {mae}')
```

```
print(f'MSE:  {mse}')
print(f'RMSE: {rmse}')
print(f'MAPE: {mape}')
```

```
MAE:  1.5748818181818183
MSE:  3.134373242424242
RMSE: 1.7704161212619598
MAPE: 1.5705519338998966
```

In [17]:
```
y_train.mean()
```

Out[17]:
```
1.007786567164179
```

In [18]:
```
baseline = np.arange(len(y_test))
baseline.fill(y_train.mean())
```

In [19]:
```
mae_baseline = mean_absolute_error(y_test, baseline)
mse_baseline = mean_squared_error(y_test, baseline)
rmse_baseline = np.sqrt(mean_squared_error(y_test, baseline))
mape_baseline = mean_absolute_percentage_error(y_test, baseline)
```

In [20]:
```
print(f'MAE:  {mae_baseline}')
print(f'MSE:  {mse_baseline}')
print(f'RMSE: {rmse_baseline}')
print(f'MAPE: {mape_baseline}')
```

```
MAE:  0.10532424242424242
MSE:  0.014991424242424242
RMSE: 0.1224394717500212
MAPE: 0.10547198137493363
```

In [21]:
```
print(f'MAE / MAE_BASELINE:   {mae_baseline/mae}')
print(f'MSE / MSE_BASELINE:   {mse_baseline/mse}')
print(f'RMSE / RMSE_BASELINE: {rmse_baseline/rmse}')
print(f'MAPE / MAPE_BASELINE: {mape_baseline/mape}')
```

```
MAE / MAE_BASELINE:   0.06687755310162763
MSE / MSE_BASELINE:   0.0047829097184447986
RMSE / RMSE_BASELINE: 0.06915858383776231
MAPE / MAPE_BASELINE: 0.06715599726335199
```

In [ ]: