1. Import Necessary Libraries

In [48]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
```

In [49]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

2. Data Loading and Preprocessing

A) Load and Explore the Data:

- Load the dataset then display the first few rows to understand the structure.

In [50]:
```
# Set seaborn style for better aesthetics
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 8)

# Load the dataset
df = pd.read_csv('smart_home_dataset.csv')
```

In [51]:
```
from IPython.display import HTML

# Display the first few rows in a scrollable table
HTML(df.head().to_html(classes='table table-striped table-bordered', escape=False, index=False))
```

Out[51]:

Unix Timestamp	Transaction_ID	Television	Dryer	Oven	Refrigerator	Microwave	Line Voltage	Voltage	Apparent Power	Energy Consumption (kWh)	Month	Day of the Week	Hour of the Day	Offloading Decision
1577836800	1	0	0	0	1	0	237	233	1559	24.001763	January	Wednesday	0	Local
1577839322	2	0	1	0	0	1	232	230	1970	31.225154	January	Wednesday	0	Remote
1577841845	3	0	1	0	0	0	223	222	1684	70.460700	January	Wednesday	1	Remote
1577844368	4	1	0	1	1	0	225	224	1694	32.264043	January	Wednesday	2	Remote
1577846891	5	1	0	0	1	0	222	214	1889	32.728111	January	Wednesday	2	Local

In [52]:
```
# Display dataset information
print("\nDataset Information:")
print(df.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48972 entries, 0 to 48971
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Unix Timestamp           48972 non-null  int64  
 1   Transaction_ID           48972 non-null  int64  
 2   Television               48972 non-null  int64  
 3   Dryer                    48972 non-null  int64  
 4   Oven                     48972 non-null  int64  
 5   Refrigerator             48972 non-null  int64  
 6   Microwave                48972 non-null  int64  
 7   Line Voltage             48972 non-null  int64  
 8   Voltage                  48972 non-null  int64  
 9   Apparent Power           48972 non-null  int64  
 10  Energy Consumption (kWh)  48972 non-null  float64
 11  Month                    48972 non-null  object 
 12  Day of the Week          48972 non-null  object 
 13  Hour of the Day          48972 non-null  int64  
 14  Offloading Decision      48972 non-null  object 
dtypes: float64(1), int64(11), object(3)
memory usage: 5.6+ MB
None
```

In [53]:
```
# Display summary statistics
print("\nSummary Statistics:")
HTML(df.describe().to_html(classes='table table-striped table-bordered', escape=False))
```

```
Summary Statistics:
```

Out[53]:

	Unix Timestamp	Transaction_ID	Television	Dryer	Oven	Refrigerator	Microwave	Line Voltage	Voltage	Apparent Power	Energy Consumption (kWh)	Hour of the Day
count	4.897200e+04	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000	48972.000000
mean	1.639612e+09	24486.500000	0.503451	0.500082	0.501981	0.502348	0.501144	229.486625	224.971494	1749.787981	55.101270	11.499224
std	3.566703e+07	14137.143028	0.499993	0.500005	0.500001	0.500000	0.500004	5.762899	6.448629	144.431949	25.922247	6.922248
min	1.577837e+09	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	220.000000	211.000000	1500.000000	10.000160	0.000000
25%	1.608724e+09	12243.750000	0.000000	0.000000	0.000000	0.000000	0.000000	224.000000	220.000000	1625.000000	32.730775	5.000000
50%	1.639612e+09	24486.500000	1.000000	1.000000	1.000000	1.000000	1.000000	229.000000	225.000000	1749.000000	55.153950	11.000000
75%	1.670500e+09	36729.250000	1.000000	1.000000	1.000000	1.000000	1.000000	234.000000	230.000000	1875.000000	77.540494	17.000000
max	1.701387e+09	48972.000000	1.000000	1.000000	1.000000	1.000000	1.000000	239.000000	239.000000	1999.000000	99.997139	23.000000

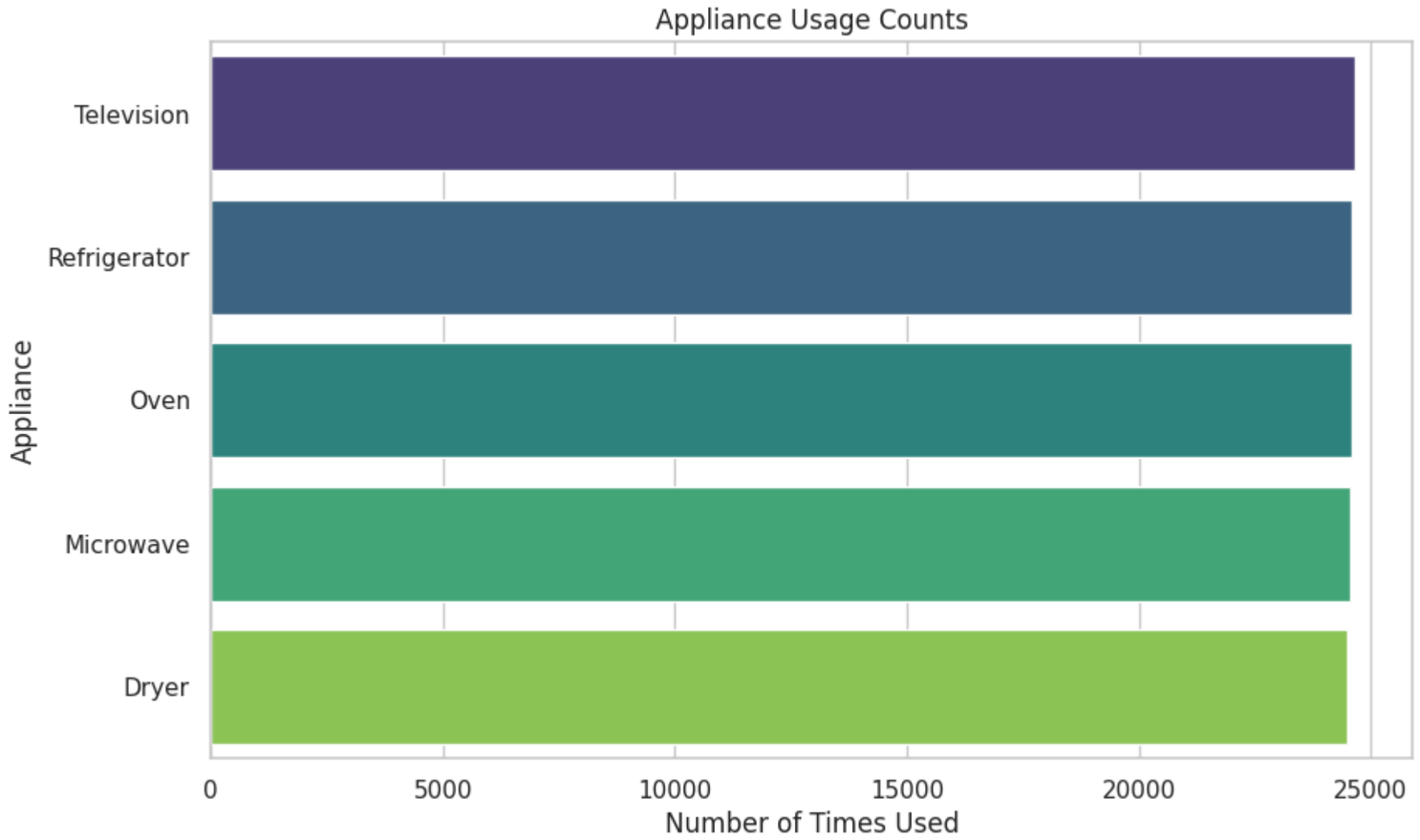
In [54]:
```
# Convert Unix Timestamp to datetime
df['Datetime'] = pd.to_datetime(df['Unix Timestamp'], unit='s')
df.set_index('Datetime', inplace=True)
```

In [55]:
```
# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

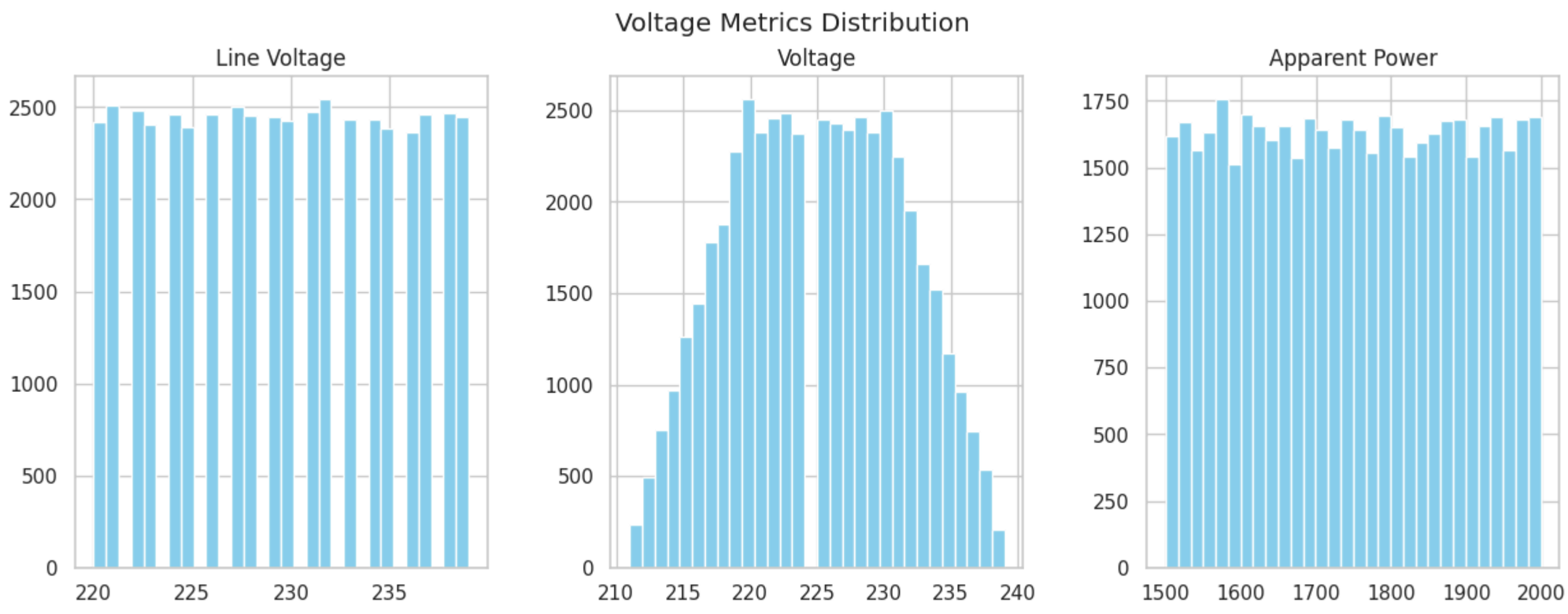
```
Missing Values:
Unix Timestamp              0
Transaction_ID              0
Television                  0
Dryer                       0
Oven                        0
Refrigerator                0
Microwave                   0
Line Voltage                0
Voltage                     0
Apparent Power              0
Energy Consumption (kWh)    0
Month                       0
Day of the Week             0
Hour of the Day             0
Offloading Decision         0
dtype: int64
```

Exploratory Data Analysis

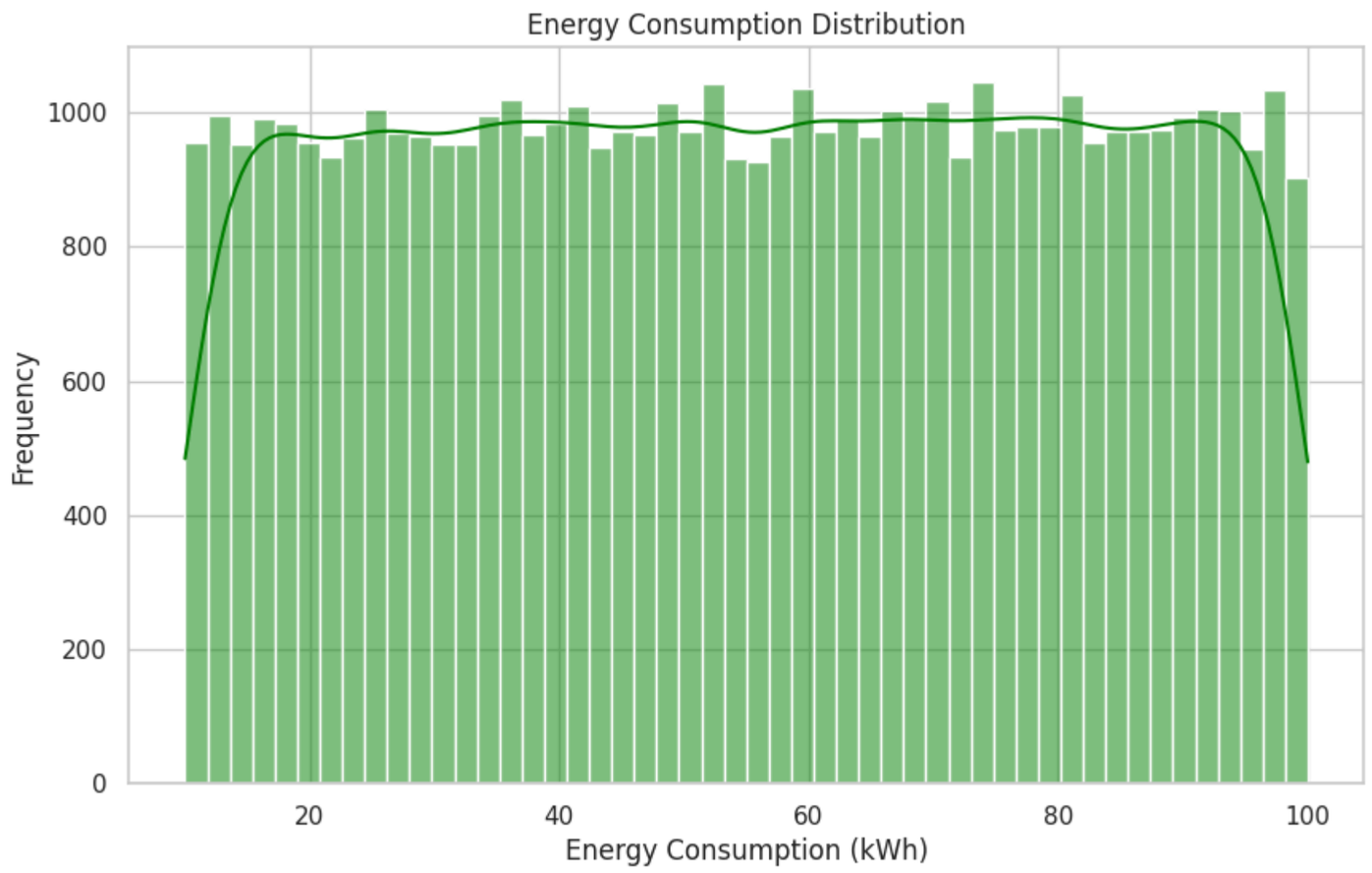
In [56]:
```
# Appliance Usage Distribution
appliances = ['Television', 'Dryer', 'Oven', 'Refrigerator', 'Microwave']
plt.figure(figsize=(10,6))
usage_counts = df[appliances].sum().sort_values(ascending=False)
sns.barplot(x=usage_counts.values, y=usage_counts.index, palette='viridis')
plt.title('Appliance Usage Counts')
plt.xlabel('Number of Times Used')
plt.ylabel('Appliance')
plt.show()
```



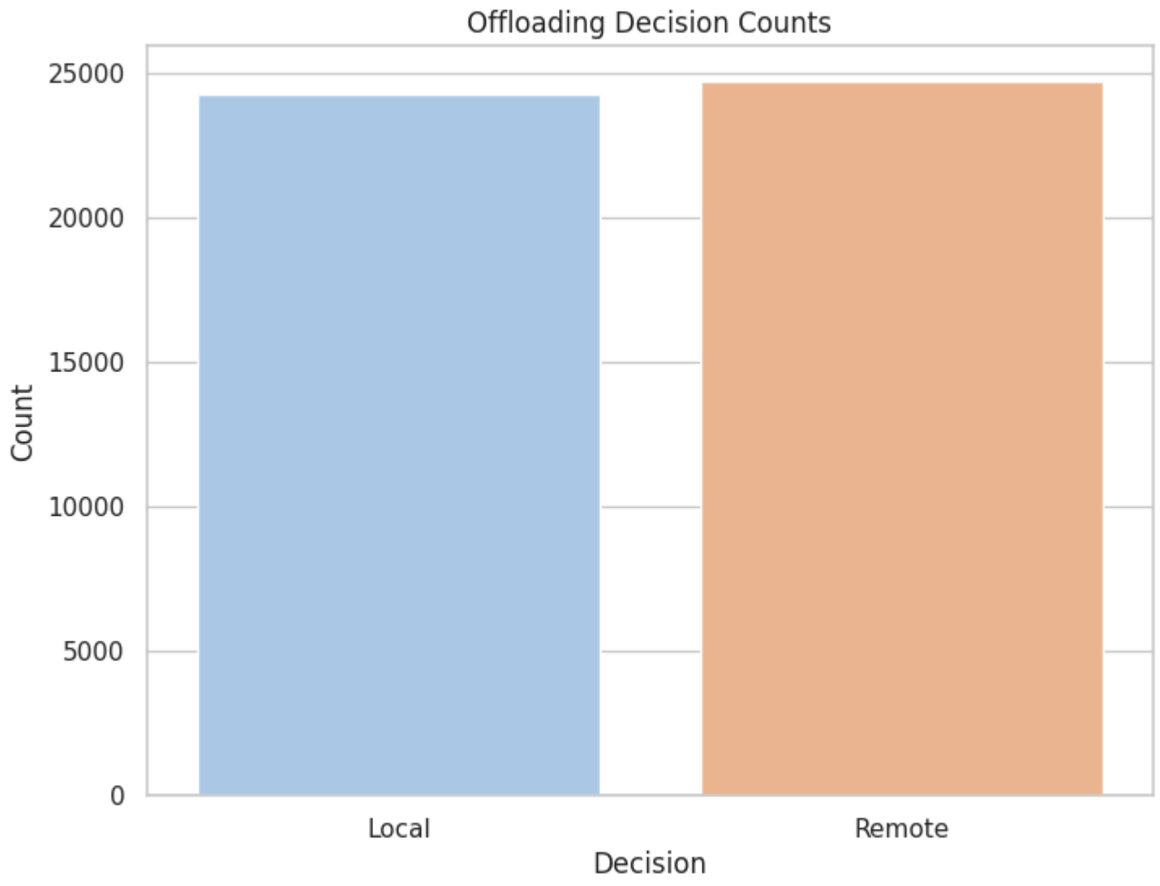
In [57]:
```
# Voltage Metrics Distribution
voltage_metrics = ['Line Voltage', 'Voltage', 'Apparent Power']
df[voltage_metrics].hist(bins=30, figsize=(15,5), layout=(1,3), color='skyblue')
plt.suptitle('Voltage Metrics Distribution')
plt.show()
```



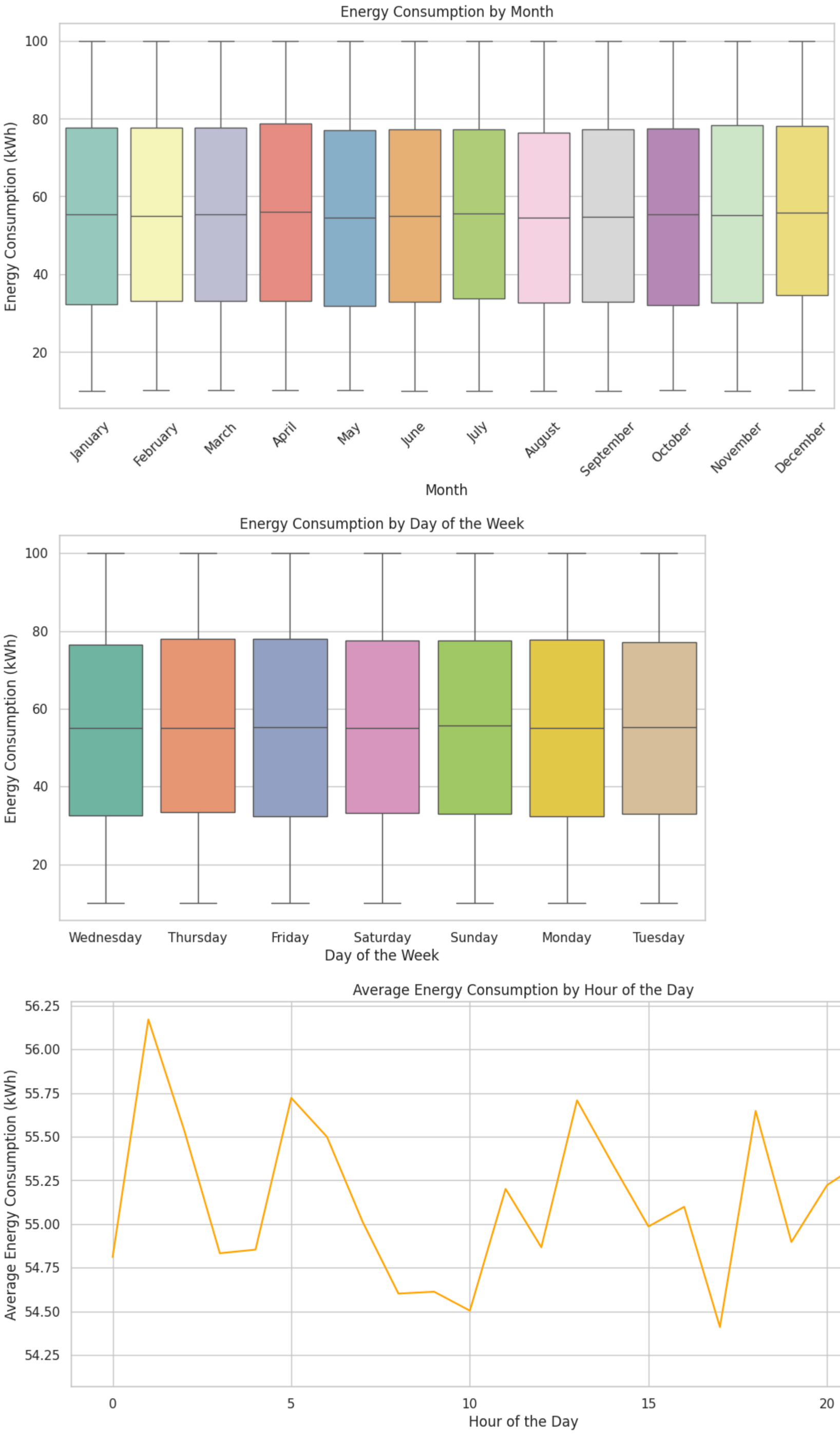
In [58]:
```
# Energy Consumption Distribution
plt.figure(figsize=(10,6))
sns.histplot(df['Energy Consumption (kWh)'], bins=50, kde=True, color='green')
plt.title('Energy Consumption Distribution')
plt.xlabel('Energy Consumption (kWh)')
plt.ylabel('Frequency')
plt.show()
```



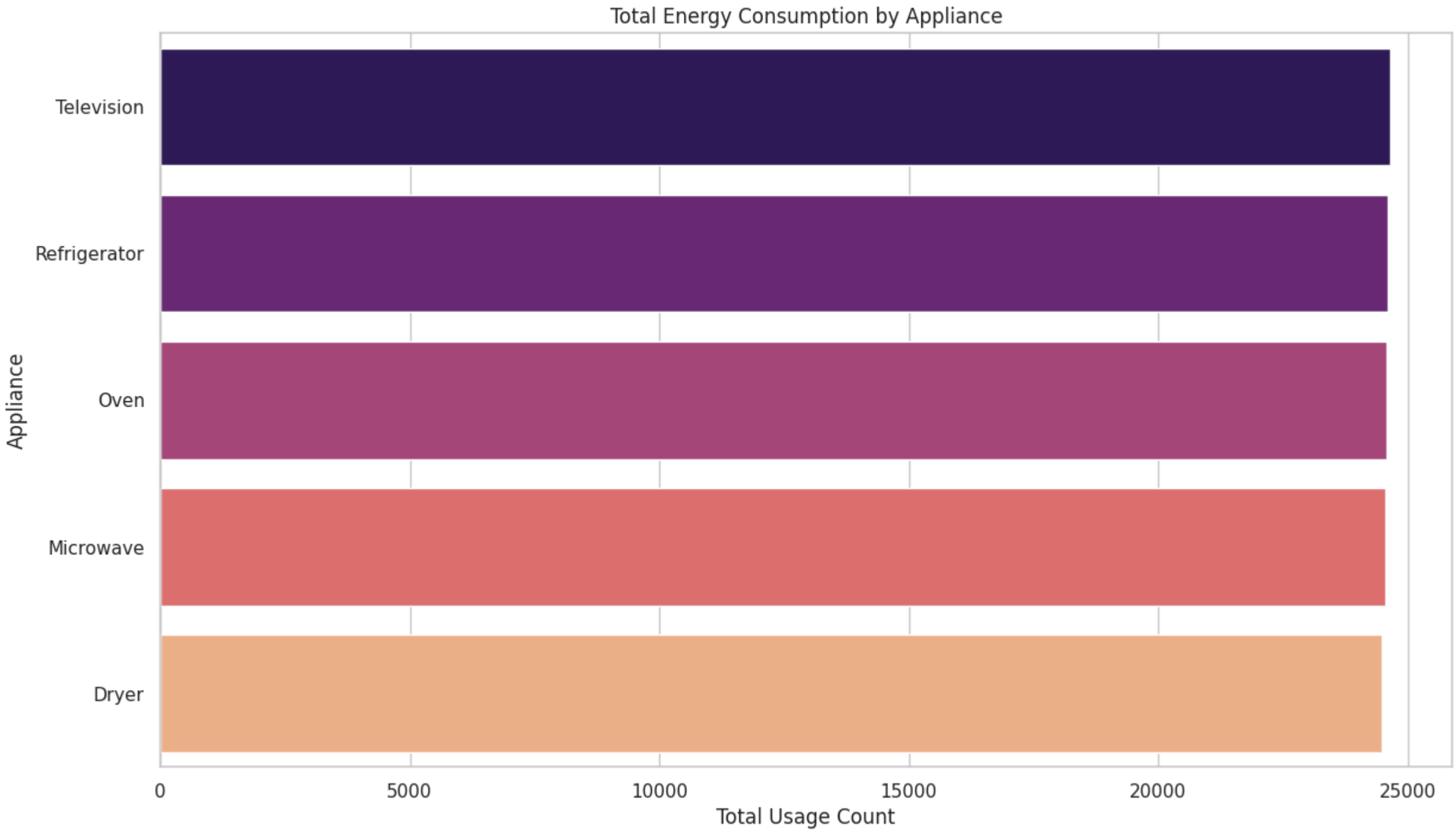
In [59]:
```
# Offloading Decision Distribution
plt.figure(figsize=(8,6))
sns.countplot(x='Offloading Decision', data=df, palette='pastel')
plt.title('Offloading Decision Counts')
plt.xlabel('Decision')
plt.ylabel('Count')
plt.show()
```



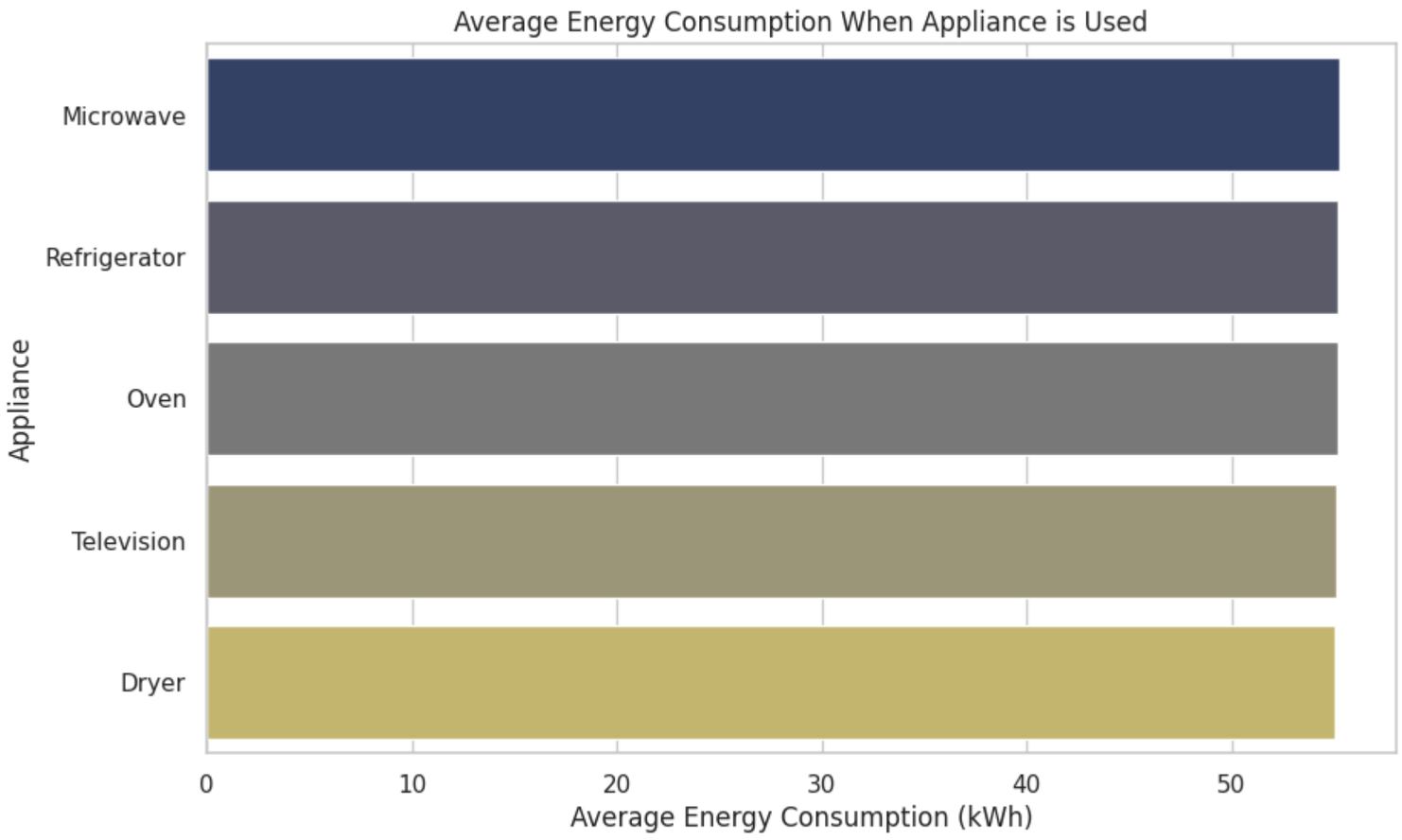
In [60]:
```
# Time-Based Analysis

# a. Energy Consumption by Month
plt.figure(figsize=(12,6))
sns.boxplot(x='Month', y='Energy Consumption (kWh)', data=df, palette='Set3')
plt.title('Energy Consumption by Month')
plt.xlabel('Month')
plt.ylabel('Energy Consumption (kWh)')
plt.xticks(rotation=45)
plt.show()

# b. Energy Consumption by Day of the Week
plt.figure(figsize=(10,6))
sns.boxplot(x='Day of the Week', y='Energy Consumption (kWh)', data=df, palette='Set2')
plt.title('Energy Consumption by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Energy Consumption (kWh)')
plt.show()


# c. Energy Consumption by Hour of the Day
plt.figure(figsize=(14,6))
sns.lineplot(x='Hour of the Day', y='Energy Consumption (kWh)', data=df, ci=None, color='orange')
plt.title('Average Energy Consumption by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Energy Consumption (kWh)')
plt.show()
```



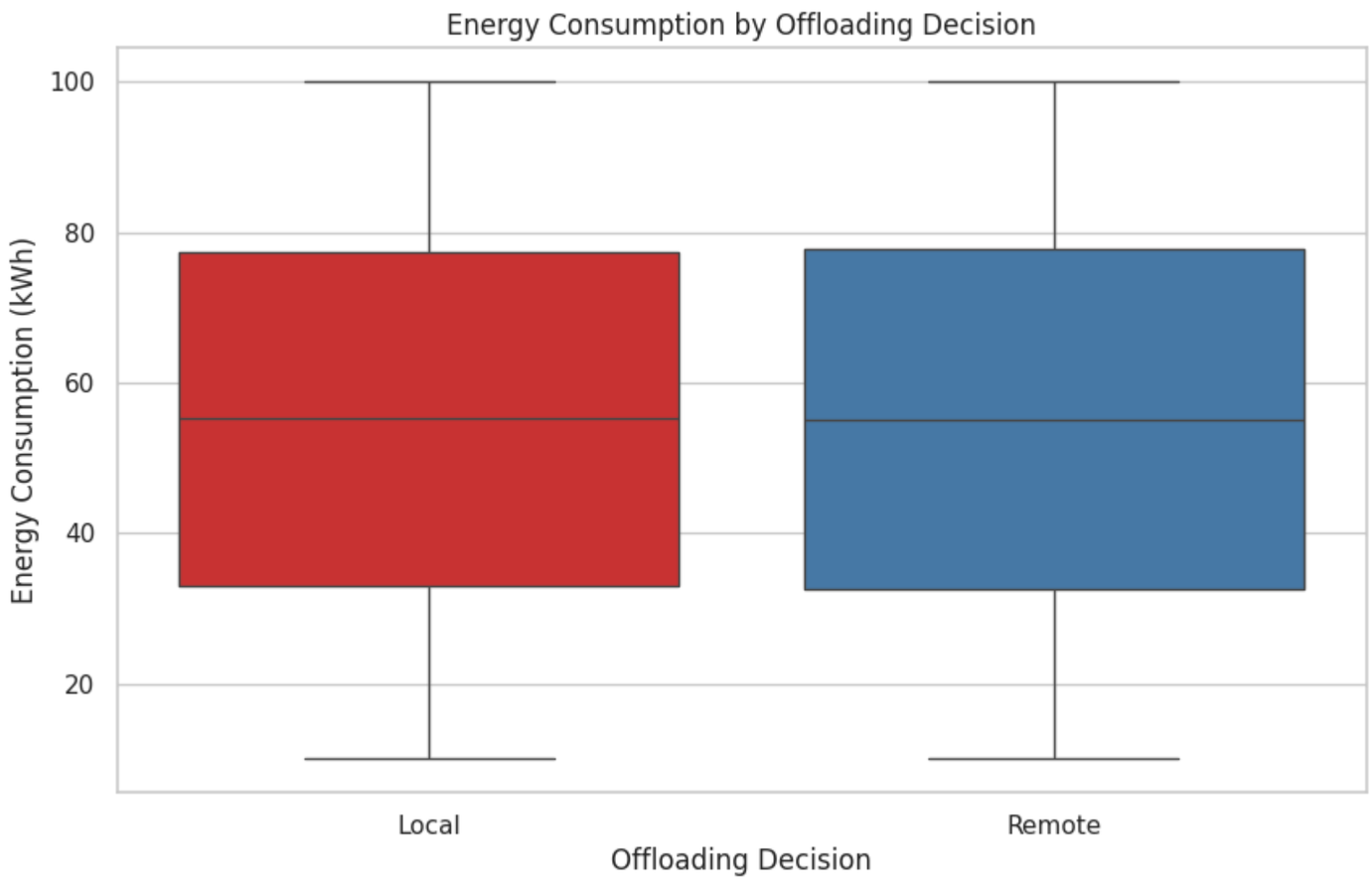
In [61]:
```
# Energy Consumption vs. Appliance Usage
plt.figure(figsize=(14,8))
appliance_usage = df[appliances].sum().sort_values(ascending=False)
sns.barplot(x=appliance_usage.values, y=appliance_usage.index, palette='magma')
plt.title('Total Energy Consumption by Appliance')
plt.xlabel('Total Usage Count')
plt.ylabel('Appliance')
plt.show()
```



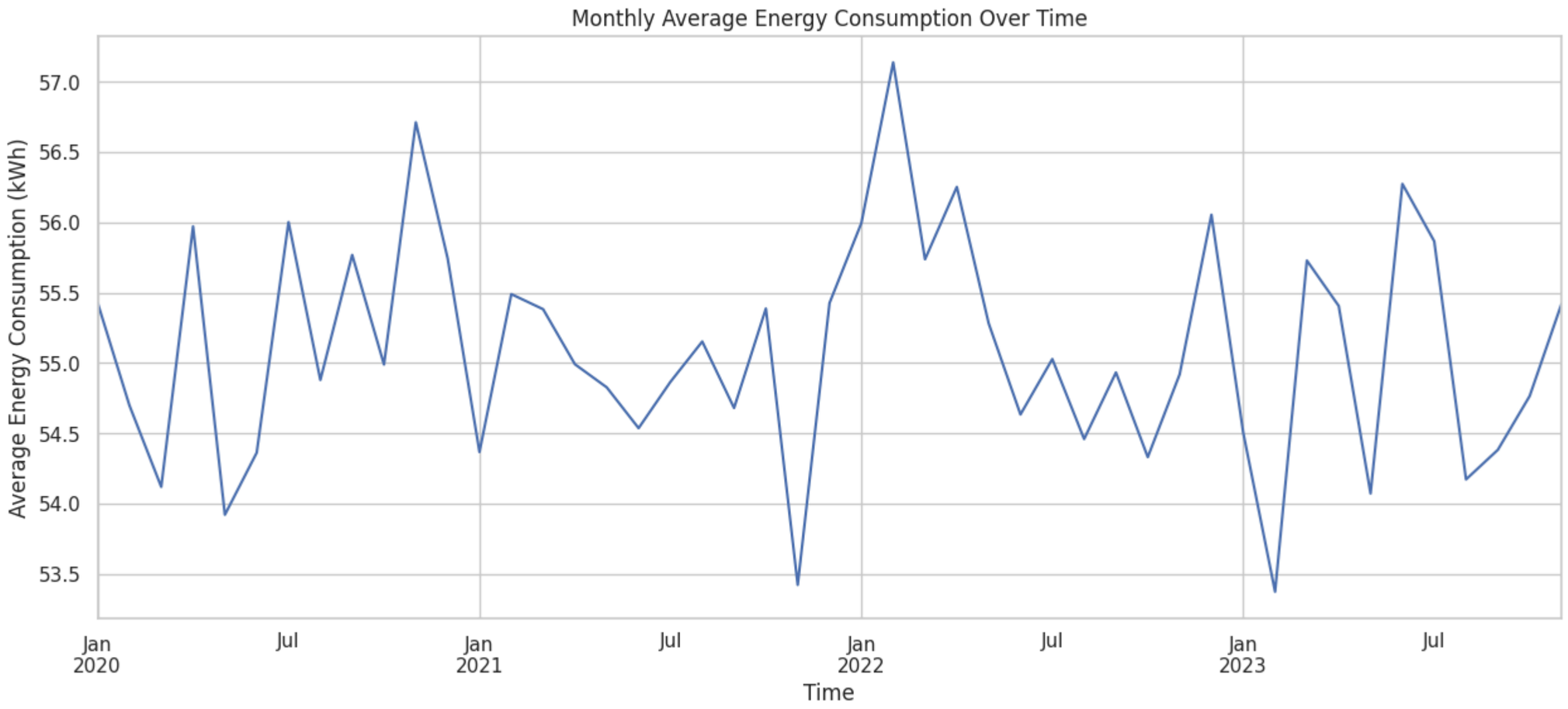
In [62]:
```
# Average energy consumption when each appliance is used
avg_energy = {}
for appliance in appliances:
    avg_energy[appliance] = df[df[appliance] == 1]['Energy Consumption (kWh)'].mean()

avg_energy_series = pd.Series(avg_energy).sort_values(ascending=False)
plt.figure(figsize=(10,6))
sns.barplot(x=avg_energy_series.values, y=avg_energy_series.index, palette='cividis')
plt.title('Average Energy Consumption When Appliance is Used')
plt.xlabel('Average Energy Consumption (kWh)')
plt.ylabel('Appliance')
plt.show()
```



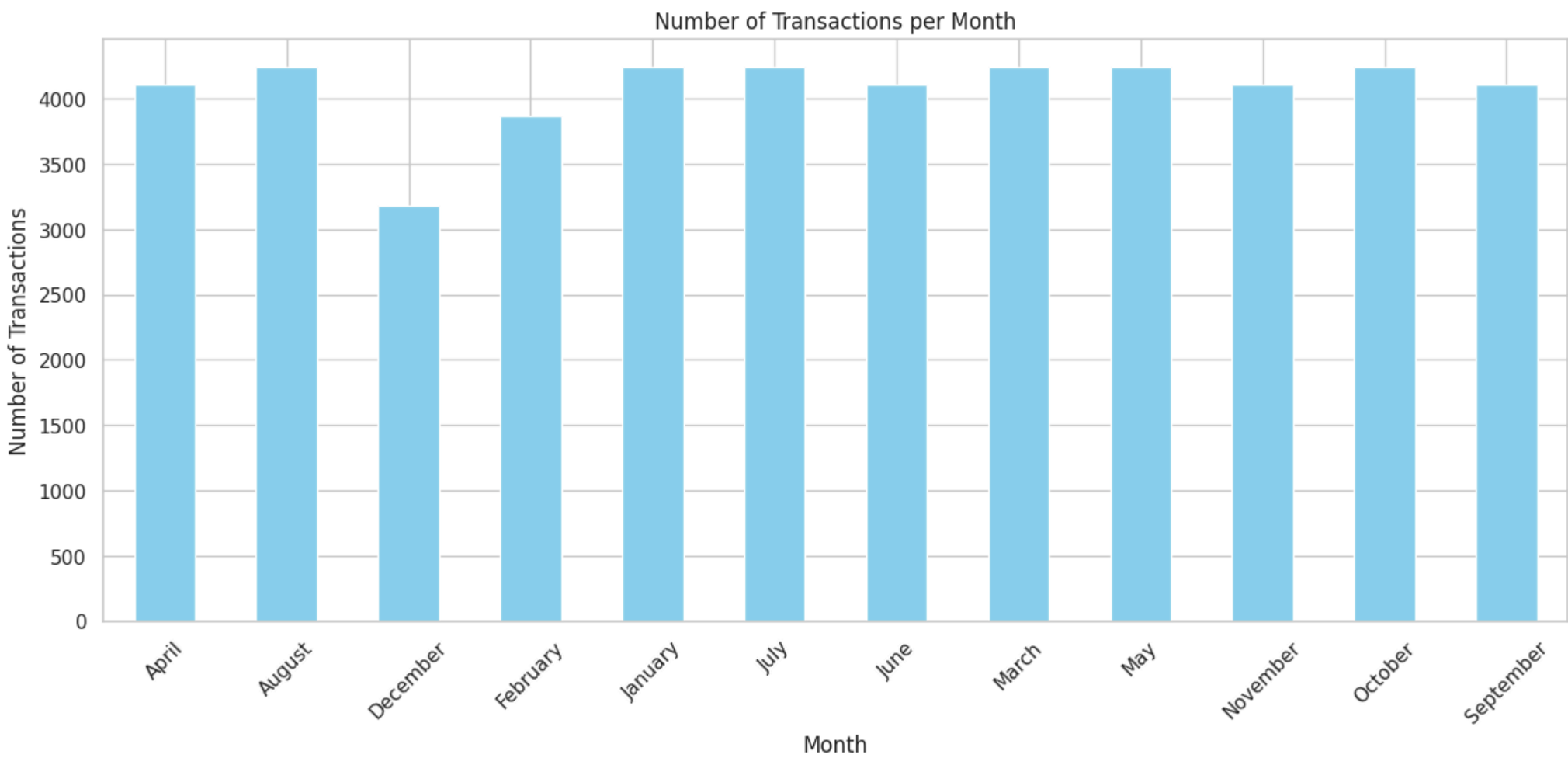
In [63]:
```
# Offloading Decision vs. Energy Consumption
plt.figure(figsize=(10,6))
sns.boxplot(x='Offloading Decision', y='Energy Consumption (kWh)', data=df, palette='Set1')
plt.title('Energy Consumption by Offloading Decision')
plt.xlabel('Offloading Decision')
plt.ylabel('Energy Consumption (kWh)')
plt.show()
```



In [64]:
```
# Time Series Plot of Energy Consumption
plt.figure(figsize=(15,6))
df['Energy Consumption (kWh)'].resample('M').mean().plot()
plt.title('Monthly Average Energy Consumption Over Time')
plt.xlabel('Time')
plt.ylabel('Average Energy Consumption (kWh)')
plt.show()
```



In [65]:
```
# Distribution of Transactions Over Time
plt.figure(figsize=(15,6))
df['Month'].value_counts().sort_index().plot(kind='bar', color='skyblue')
plt.title('Number of Transactions per Month')
plt.xlabel('Month')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



B) Data Cleaning:

- Convert Unix Timestamp to a readable date format
- Handle missing values
- Normalize or scale features like Voltage, Apparent Power, and Energy Consumption (kWh) to standardize the data for modeling.
- **Split before Normalize** (Very Important) **to avoid Machine Data leak**.

In [66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 48972 entries, 2020-01-01 00:00:00 to 2023-11-30 23:33:47
Data columns (total 15 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Unix Timestamp            48972 non-null  int64  
 1   Transaction_ID            48972 non-null  int64  
 2   Television                48972 non-null  int64  
 3   Dryer                     48972 non-null  int64  
 4   Oven                      48972 non-null  int64  
 5   Refrigerator              48972 non-null  int64  
 6   Microwave                 48972 non-null  int64  
 7   Line Voltage              48972 non-null  int64  
 8   Voltage                   48972 non-null  int64  
 9   Apparent Power            48972 non-null  int64  
 10  Energy Consumption (kWh)  48972 non-null  float64
 11  Month                     48972 non-null  object 
 12  Day of the Week           48972 non-null  object 
 13  Hour of the Day           48972 non-null  int64  
 14  Offloading Decision       48972 non-null  object 
dtypes: float64(1), int64(11), object(3)
memory usage: 6.0+ MB
```

In [67]:
```
# Convert Unix Timestamp to datetime
df['Datetime'] = pd.to_datetime(df['Unix Timestamp'], unit='s')
df.set_index('Datetime', inplace=True)
```

In [68]:
```
# Handle missing values
numerical_cols = ['Line Voltage', 'Voltage', 'Apparent Power', 'Energy Consumption (kWh)']
categorical_cols = ['Transaction_ID', 'Television', 'Dryer', 'Oven', 'Refrigerator', 'Microwave',
                    'Month', 'Day of the Week', 'Hour of the Day', 'Offloading Decision']

for col in numerical_cols:
    if df[col].isnull().sum() > 0:
        median = df[col].median()
        df[col].fillna(median, inplace=True)

for col in categorical_cols:
    if df[col].isnull().sum() > 0:
        mode = df[col].mode()[0]
        df[col].fillna(mode, inplace=True)

# Verify that there are no missing values left
print("\nMissing Values After Cleaning:")
print(df.isnull().sum())
```

```
Missing Values After Cleaning:
Unix Timestamp              0
Transaction_ID              0
Television                  0
Dryer                       0
Oven                        0
Refrigerator                0
Microwave                   0
Line Voltage                0
Voltage                     0
Apparent Power              0
Energy Consumption (kWh)    0
Month                       0
Day of the Week             0
Hour of the Day             0
Offloading Decision         0
dtype: int64
```

C) Feature Engineering:

- Extract meaningful time-based features from the timestamp, including Month, Day of the Week, and Hour of the Week
- Drop Unix Timestamp (Not needed anymore)

In [69]:
```
# Extract meaningful time-based features from the datetime index
df['Month_Num'] = df.index.month
df['Day_of_Week_Num'] = df.index.dayofweek  # Monday=0, Sunday=6
df['Hour_of_Day'] = df.index.hour

# Additional Features: Rolling Mean and STD of Energy Consumption
df['Energy_Rolling_Mean_3'] = df['Energy Consumption (kWh)'].rolling(window=3, min_periods=1).mean()
df['Energy_Rolling_STD_3'] = df['Energy Consumption (kWh)'].rolling(window=3, min_periods=1).std().fillna(0)

# Drop original categorical time columns
df.drop(['Unix Timestamp', 'Month', 'Day of the Week', 'Hour of the Day'], axis=1, inplace=True)

# Display the updated DataFrame
print("\nDataFrame After Feature Engineering:")
print(df.head())
```

```
DataFrame After Feature Engineering:
                     Transaction_ID  Television  Dryer  Oven  Refrigerator  \
Datetime
2020-01-01 00:00:00               1           0      0     0             1
2020-01-01 00:42:02               2           0      1     0             0
2020-01-01 01:24:05               3           0      1     0             0
2020-01-01 02:06:08               4           1      0     1             1
2020-01-01 02:48:11               5           1      0     0             1

                     Microwave  Line Voltage  Voltage  Apparent Power  \
Datetime
2020-01-01 00:00:00          0           237      233            1559
2020-01-01 00:42:02          1           232      230            1970
2020-01-01 01:24:05          0           223      222            1684
2020-01-01 02:06:08          0           225      224            1694
2020-01-01 02:48:11          0           222      214            1889

                     Energy Consumption (kWh) Offloading Decision  Month_Num  \
Datetime
2020-01-01 00:00:00                24.001763               Local          1
2020-01-01 00:42:02                31.225154              Remote          1
2020-01-01 01:24:05                70.460700              Remote          1
2020-01-01 02:06:08                32.264043              Remote          1
2020-01-01 02:48:11                32.728111               Local          1

                     Day_of_Week_Num  Hour_of_Day  Energy_Rolling_Mean_3  \
Datetime
2020-01-01 00:00:00                2            0              24.001763
2020-01-01 00:42:02                2            0              27.613458
2020-01-01 01:24:05                2            1              41.895872
2020-01-01 02:06:08                2            2              44.649966
2020-01-01 02:48:11                2            2              45.150951

                     Energy_Rolling_STD_3
Datetime
2020-01-01 00:00:00              0.000000
2020-01-01 00:42:02              5.107709
2020-01-01 01:24:05             25.000128
2020-01-01 02:06:08             22.358786
2020-01-01 02:48:11             21.920114
```

D) Data Splitting:

- Split the dataset into training, and test sets (ex. 80% train, 20%test)

In [70]:
```python
# Define features for modeling
features = ['Energy Consumption (kWh)', 'Voltage', 'Apparent Power',
            'Month_Num', 'Day_of_Week_Num', 'Hour_of_Day',
            'Energy_Rolling_Mean_3', 'Energy_Rolling_STD_3']

# Ensure no missing values in features
print("\nChecking for missing values in features:")
print(df[features].isnull().sum())

# Split the data into training and testing sets (80% train, 20% test)
# Since it's time series data, perform a temporal split
train_size = int(len(df) * 0.8)
train_df = df.iloc[:train_size]
test_df = df.iloc[train_size:]

print(f"\nTraining Set Size: {train_df.shape}")
print(f"Testing Set Size: {test_df.shape}")
```

```
Checking for missing values in features:
Energy Consumption (kWh)    0
Voltage                     0
Apparent Power              0
Month_Num                   0
Day_of_Week_Num             0
Hour_of_Day                 0
Energy_Rolling_Mean_3       0
Energy_Rolling_STD_3        0
dtype: int64

Training Set Size: (39177, 16)
Testing Set Size: (9795, 16)
```

In [71]:
```python
# ==========================
# Data Normalization
# ==========================

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing data
train_scaled = scaler.fit_transform(train_df[features])
test_scaled = scaler.transform(test_df[features])

# Convert the scaled data back to DataFrames for easier handling
train_scaled_df = pd.DataFrame(train_scaled, columns=features, index=train_df.index)
test_scaled_df = pd.DataFrame(test_scaled, columns=features, index=test_df.index)
```

3. Model Selection and Training:

- Consider algorithms suited to anomaly detection, such as Isolation Forest.
- Use the training set to train the model, focusing on detecting unusual patterns in Energy Consumption (kWh), Voltage, and Apparent Power

In [73]:
```python
# Define contamination parameter based on synthetic labels (5%)
contamination_rate = 0.05

# Initialize and train the Isolation Forest model
model_if = IsolationForest(n_estimators=100, contamination=contamination_rate, random_state=42)
model_if.fit(train_scaled_df)

# Predict anomalies using Isolation Forest
test_df.loc[:, 'Anomaly_Prediction_IF'] = model_if.predict(test_scaled_df)
# Convert predictions to binary labels: 0 for normal, 1 for anomaly
test_df.loc[:, 'Anomaly_Prediction_IF'] = test_df['Anomaly_Prediction_IF'].map({1: 0, -1: 1})

# Initialize and train the Local Outlier Factor model
model_lof = LocalOutlierFactor(n_neighbors=20, contamination=contamination_rate, novelty=True)
model_lof.fit(train_scaled_df)

# Predict anomalies using Local Outlier Factor
test_df.loc[:, 'Anomaly_Prediction_LOF'] = model_lof.predict(test_scaled_df)
# Convert predictions to binary labels: 0 for normal, 1 for anomaly
test_df.loc[:, 'Anomaly_Prediction_LOF'] = test_df['Anomaly_Prediction_LOF'].map({1: 0, -1: 1})

# Initialize and train the One-Class SVM model
model_ocsvm = OneClassSVM(kernel='rbf', gamma='scale', nu=contamination_rate)
model_ocsvm.fit(train_scaled_df)
```

```
# Predict anomalies using One-Class SVM
test_df.loc[:, 'Anomaly_Prediction_OCSVM'] = model_ocsvm.predict(test_scaled_df)
# Convert predictions to binary labels: 0 for normal, 1 for anomaly
test_df.loc[:, 'Anomaly_Prediction_OCSVM'] = test_df['Anomaly_Prediction_OCSVM'].map({1: 0, -1: 1})
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with feature names
  warnings.warn(
```

4. Anomaly Labeling and Evaluation:

A) Label Anomalies:

- Define a synthetic threshold for anomalies in 'Energy Consumption (kWh)'
- Create synthetic labels for anomalies

In [75]:
```
# Define a synthetic threshold for anomalies in 'Energy Consumption (kWh)'
# For example, data points above the 95th percentile are considered anomalies
threshold = df['Energy Consumption (kWh)'].quantile(0.95)
print(f"\nSynthetic Anomaly Threshold for Energy Consumption: {threshold:.2f} kWh")

# Create synthetic labels based on the threshold
test_df.loc[:, 'Anomaly_Label'] = test_df['Energy Consumption (kWh)'].apply(lambda x: 1 if x > threshold else 0)
```

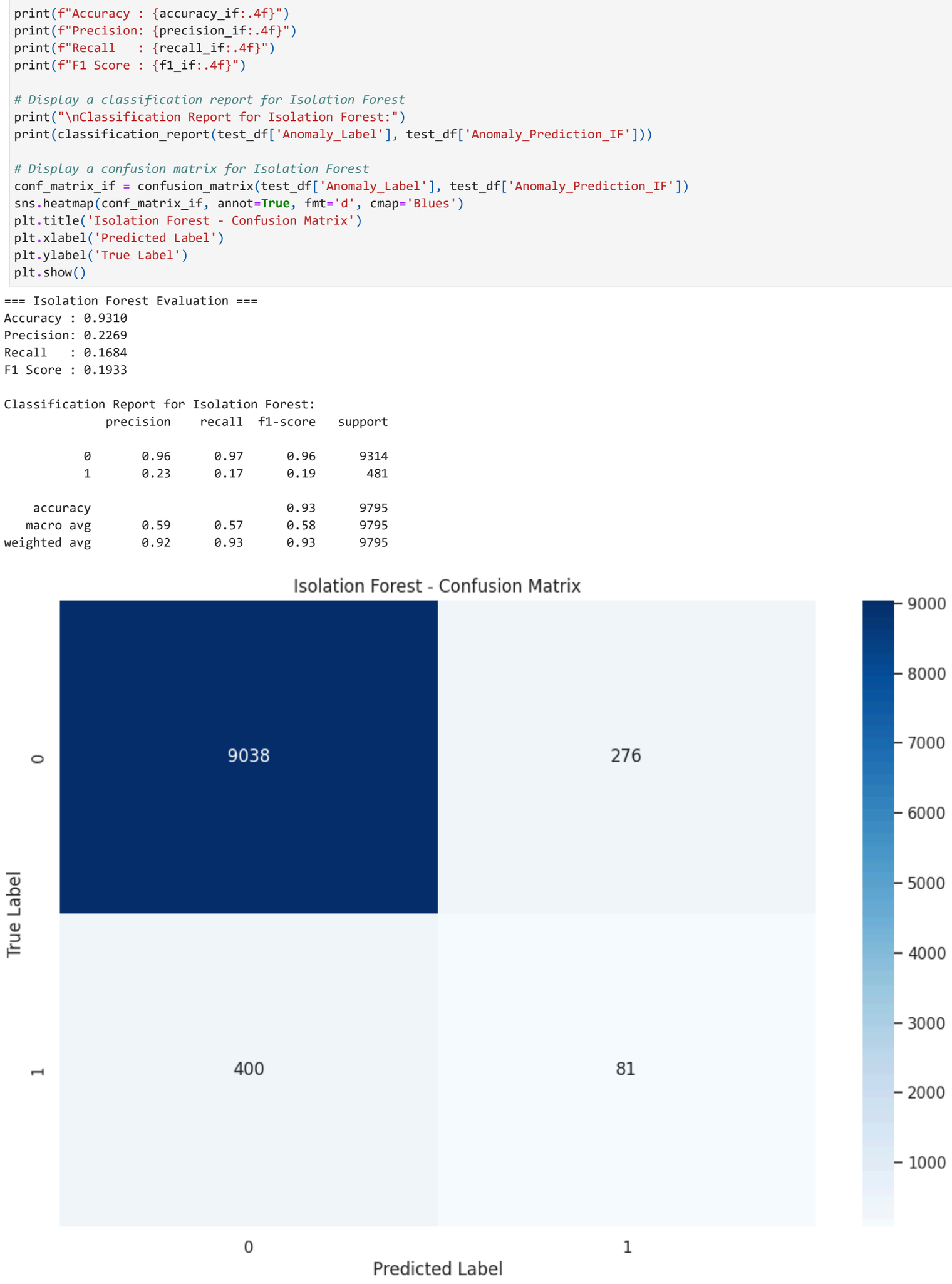
```
Synthetic Anomaly Threshold for Energy Consumption: 95.41 kWh
```

B) Evaluate the Model:

- Model Evaluation with Synthetic Labels
- Calculate specific metrics (Accuracy, Precision, Recall Score, F1 Score)

In [76]:
```
# Evaluate Isolation Forest
print("\n=== Isolation Forest Evaluation ===")
accuracy_if = accuracy_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_IF'])
precision_if = precision_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_IF'])
recall_if = recall_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_IF'])
f1_if = f1_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_IF'])

print(f"Accuracy : {accuracy_if:.4f}")
print(f"Precision: {precision_if:.4f}")
print(f"Recall   : {recall_if:.4f}")
print(f"F1 Score : {f1_if:.4f}")

# Display a classification report for Isolation Forest
print("\nClassification Report for Isolation Forest:")
print(classification_report(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_IF']))

# Display a confusion matrix for Isolation Forest
conf_matrix_if = confusion_matrix(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_IF'])
sns.heatmap(conf_matrix_if, annot=True, fmt='d', cmap='Blues')
plt.title('Isolation Forest - Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

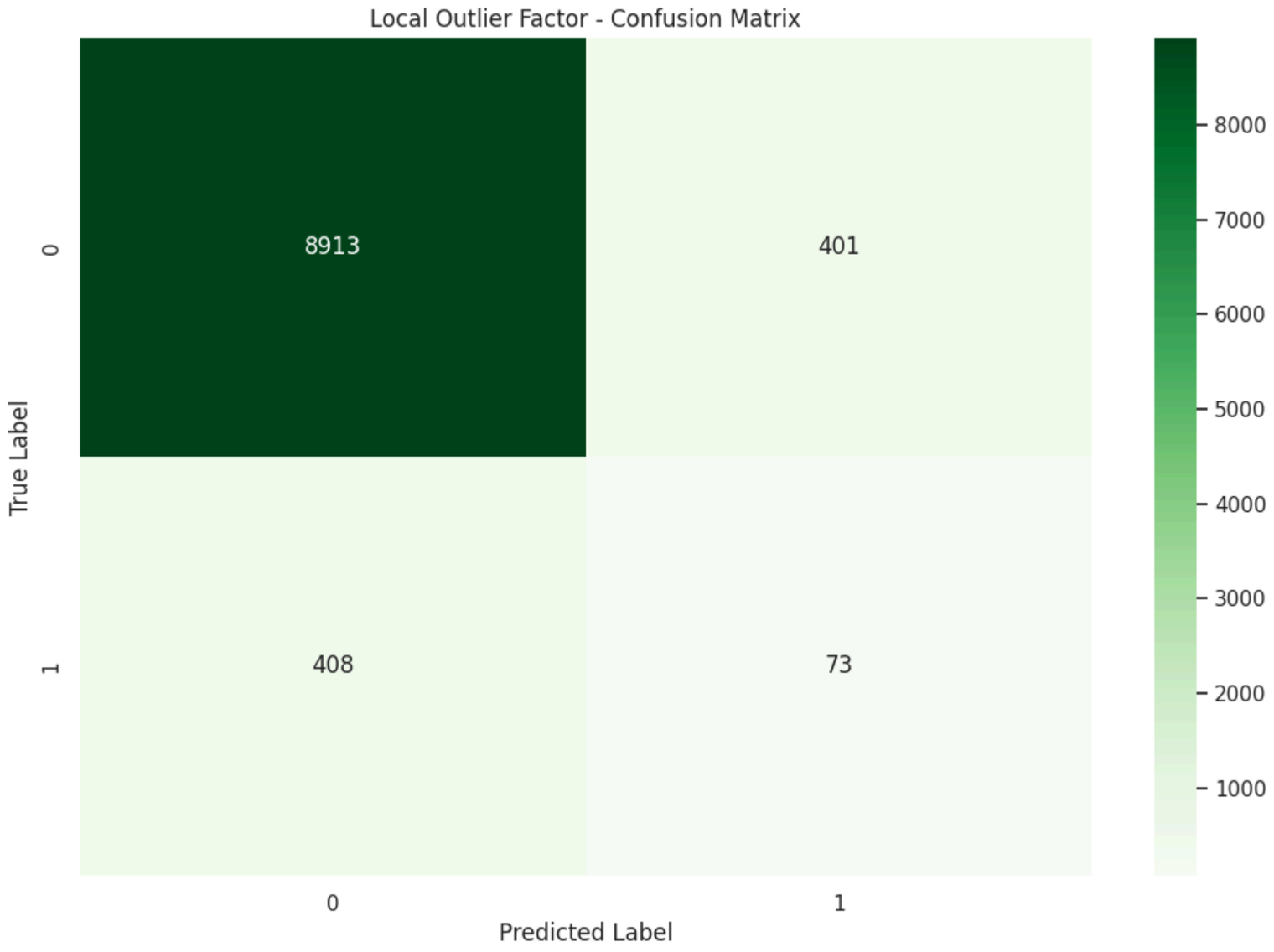
```
=== Isolation Forest Evaluation ===
Accuracy : 0.9310
Precision: 0.2269
Recall   : 0.1684
F1 Score : 0.1933

Classification Report for Isolation Forest:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      9314
           1       0.23      0.17      0.19       481

    accuracy                           0.93      9795
   macro avg       0.59      0.57      0.58      9795
weighted avg       0.92      0.93      0.93      9795
```



In [77]:
```
# Evaluate Local Outlier Factor
print("\n=== Local Outlier Factor Evaluation ===")
accuracy_lof = accuracy_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_LOF'])
precision_lof = precision_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_LOF'])
recall_lof = recall_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_LOF'])
f1_lof = f1_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_LOF'])

print(f"Accuracy : {accuracy_lof:.4f}")
```

```
print(f"Precision: {precision_lof:.4f}")
print(f"Recall   : {recall_lof:.4f}")
print(f"F1 Score : {f1_lof:.4f}")

# Display a classification report for LOF
print("\nClassification Report for Local Outlier Factor:")
print(classification_report(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_LOF']))

# Display a confusion matrix for LOF
conf_matrix_lof = confusion_matrix(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_LOF'])
sns.heatmap(conf_matrix_lof, annot=True, fmt='d', cmap='Greens')
plt.title('Local Outlier Factor - Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

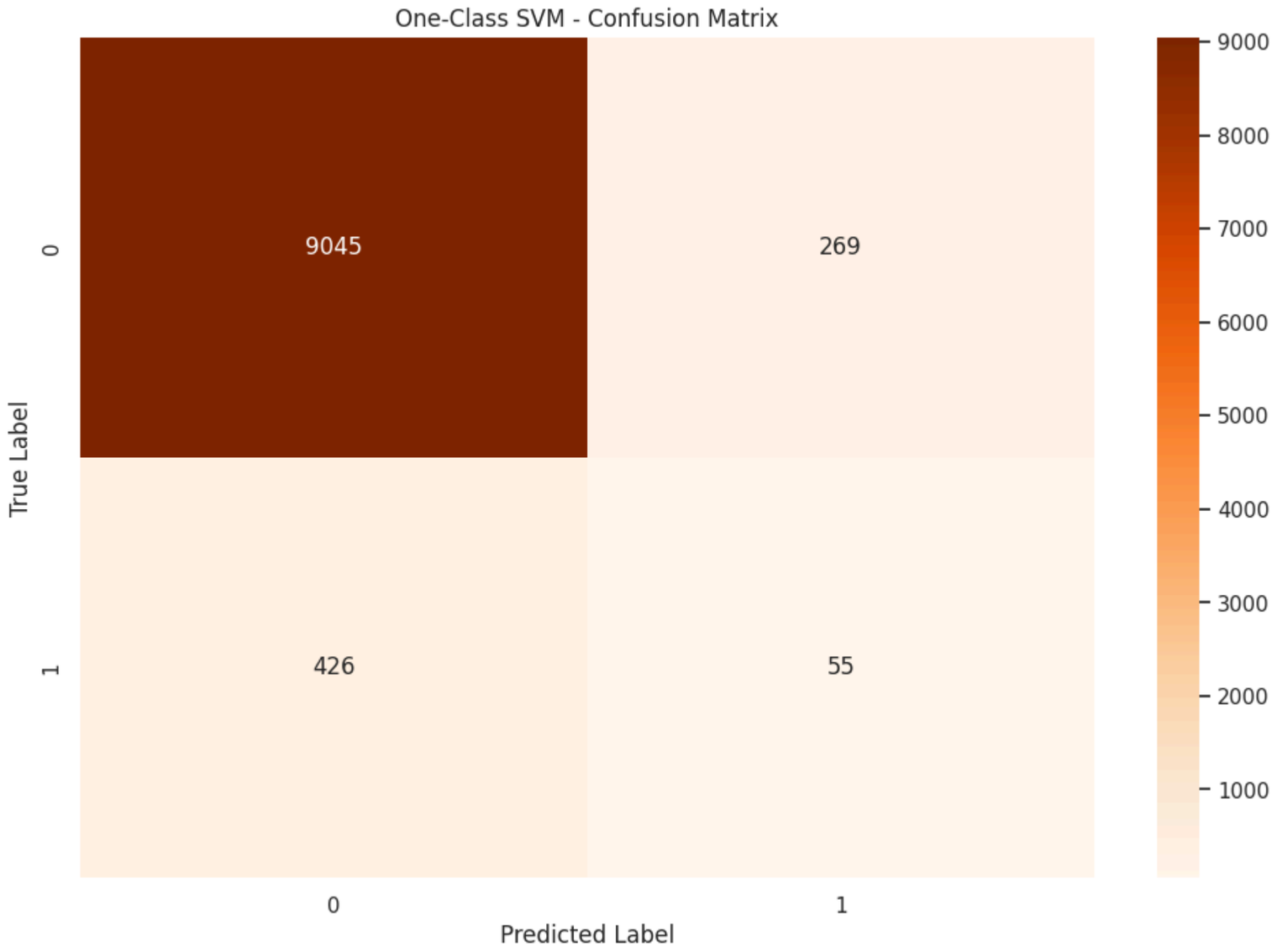
```
=== Local Outlier Factor Evaluation ===
Accuracy : 0.9174
Precision: 0.1540
Recall   : 0.1518
F1 Score : 0.1529

Classification Report for Local Outlier Factor:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      9314
           1       0.15      0.15      0.15       481

    accuracy                           0.92      9795
   macro avg       0.56      0.55      0.55      9795
weighted avg       0.92      0.92      0.92      9795
```



In [78]:

```
# Evaluate One-Class SVM
print("\n=== One-Class SVM Evaluation ===")
accuracy_ocsvm = accuracy_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_OCSVM'])
precision_ocsvm = precision_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_OCSVM'])
recall_ocsvm = recall_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_OCSVM'])
f1_ocsvm = f1_score(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_OCSVM'])

print(f"Accuracy : {accuracy_ocsvm:.4f}")
print(f"Precision: {precision_ocsvm:.4f}")
print(f"Recall   : {recall_ocsvm:.4f}")
print(f"F1 Score : {f1_ocsvm:.4f}")

# Display a classification report for One-Class SVM
print("\nClassification Report for One-Class SVM:")
print(classification_report(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_OCSVM']))

# Display a confusion matrix for One-Class SVM
conf_matrix_ocsvm = confusion_matrix(test_df['Anomaly_Label'], test_df['Anomaly_Prediction_OCSVM'])
sns.heatmap(conf_matrix_ocsvm, annot=True, fmt='d', cmap='Oranges')
plt.title('One-Class SVM - Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

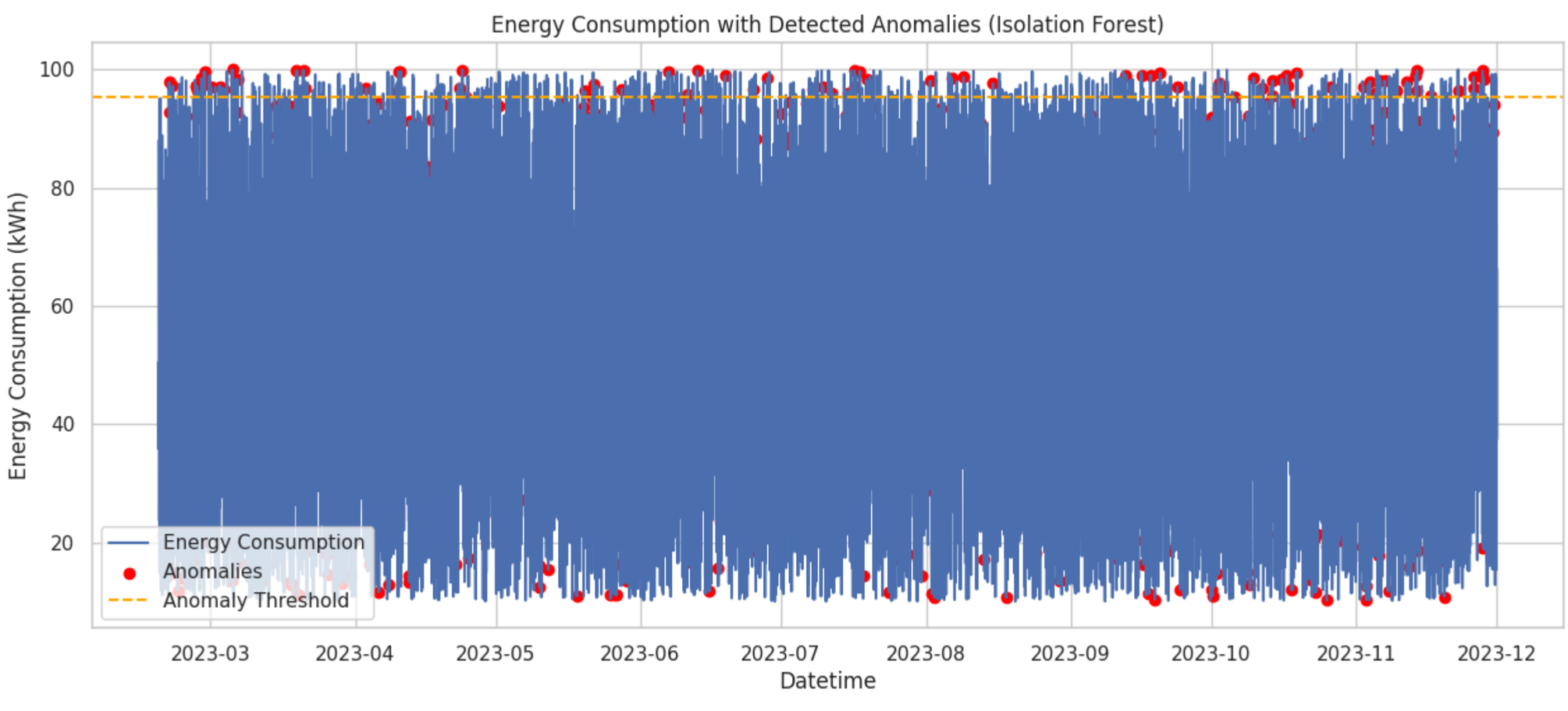
```
=== One-Class SVM Evaluation ===
Accuracy : 0.9290
Precision: 0.1698
Recall   : 0.1143
F1 Score : 0.1366

Classification Report for One-Class SVM:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      9314
           1       0.17      0.11      0.14       481

    accuracy                           0.93      9795
   macro avg       0.56      0.54      0.55      9795
weighted avg       0.92      0.93      0.92      9795
```



C) Describe the Anomalies

- Describe the anomalies that are detected either by text or by visualizing it. It must contain useful data to show where anomalies were detected.

In [79]:
```
# Choosing the best performing model based on F1 score

# Plot Energy Consumption over time with anomalies highlighted (Isolation Forest)
plt.figure(figsize=(15,6))
plt.plot(test_df.index, test_df['Energy Consumption (kWh)'], label='Energy Consumption')
plt.scatter(test_df[test_df['Anomaly_Prediction_IF'] == 1].index,
            test_df[test_df['Anomaly_Prediction_IF'] == 1]['Energy Consumption (kWh)'],
            color='red', label='Anomalies')
plt.axhline(y=threshold, color='orange', linestyle='--', label='Anomaly Threshold')
plt.title('Energy Consumption with Detected Anomalies (Isolation Forest)')
plt.xlabel('Datetime')
plt.ylabel('Energy Consumption (kWh)')
plt.legend()
plt.show()
```



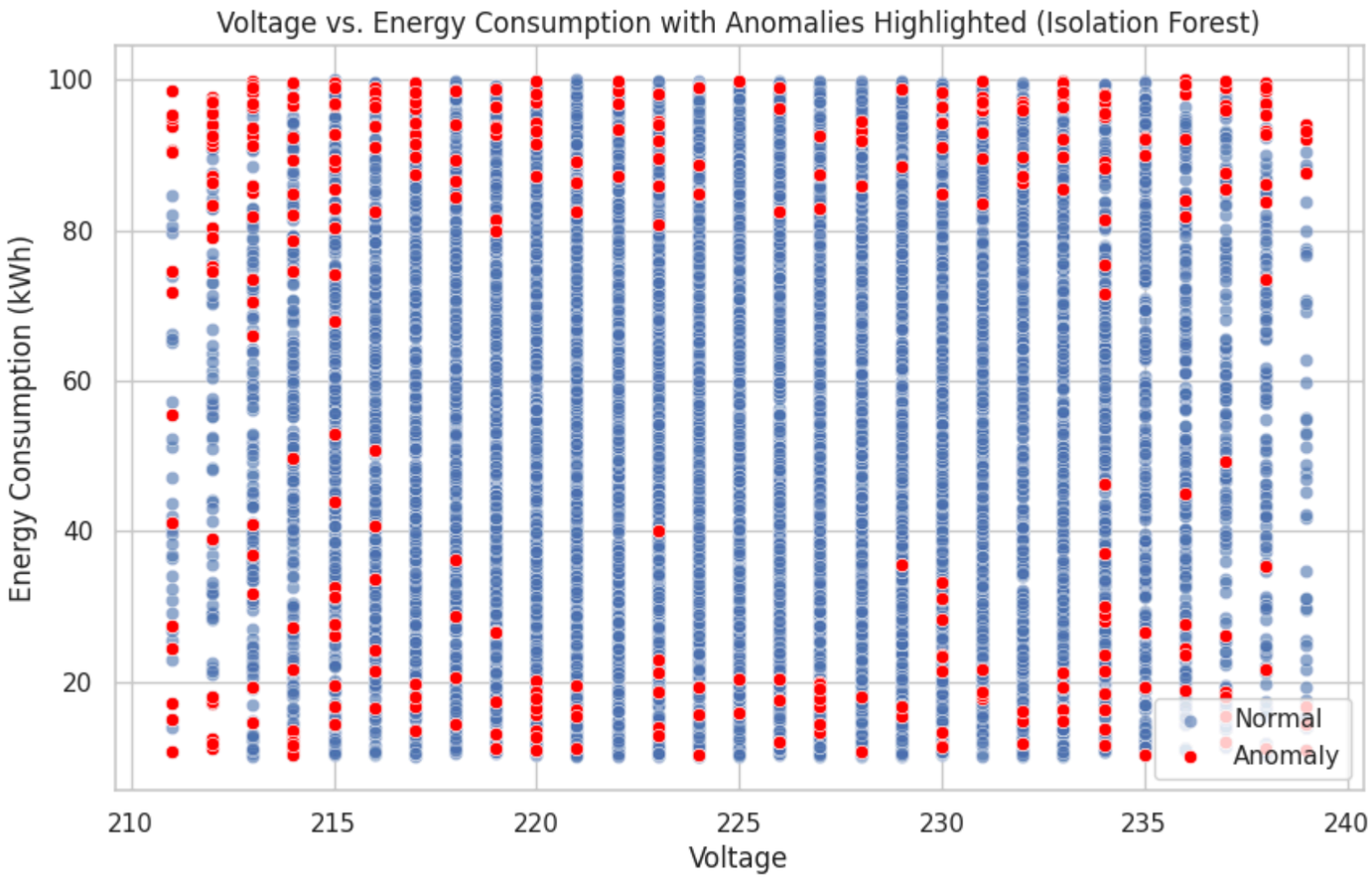
In [80]:
```
# Describe anomalies detected by Isolation Forest
anomalies_if = test_df[test_df['Anomaly_Prediction_IF'] == 1]
print(f"\nNumber of Anomalies Detected by Isolation Forest: {anomalies_if.shape[0]}")
```

```
Number of Anomalies Detected by Isolation Forest: 357
```

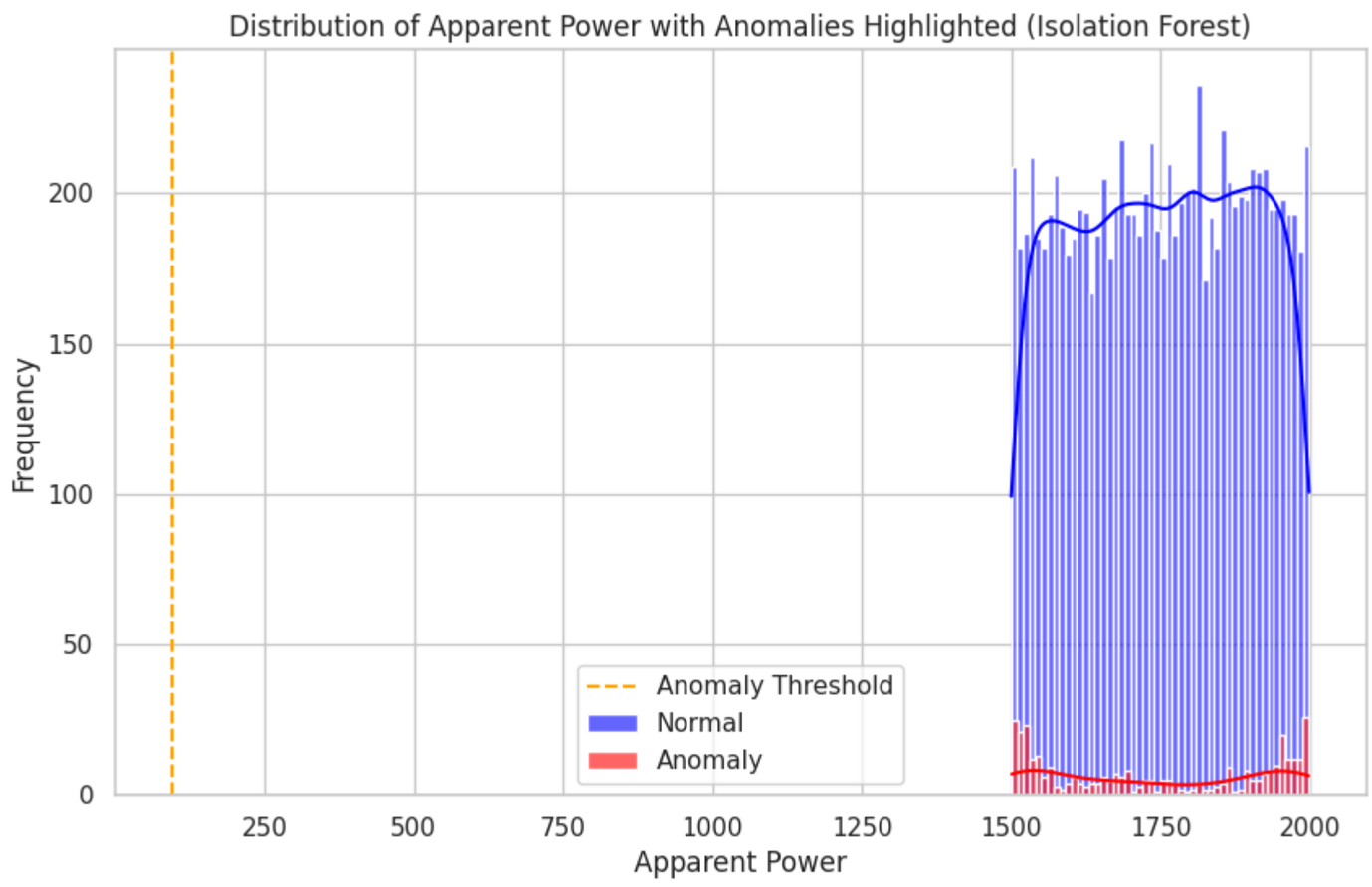
In [81]:
```
# Display some details about the anomalies
print("\nSample Anomalies Detected by Isolation Forest:")
print(anomalies_if[['Energy Consumption (kWh)', 'Voltage', 'Apparent Power']].head())
```

```
Sample Anomalies Detected by Isolation Forest:
                     Energy Consumption (kWh)  Voltage  Apparent Power
Datetime
2023-02-18 22:55:03                 21.201238      233            1754
2023-02-19 19:56:31                 15.526713      220            1519
2023-02-20 05:03:09                 97.856203      220            1620
2023-02-20 05:45:12                 92.783923      219            1816
2023-02-20 23:16:25                 97.240021      212            1512
```

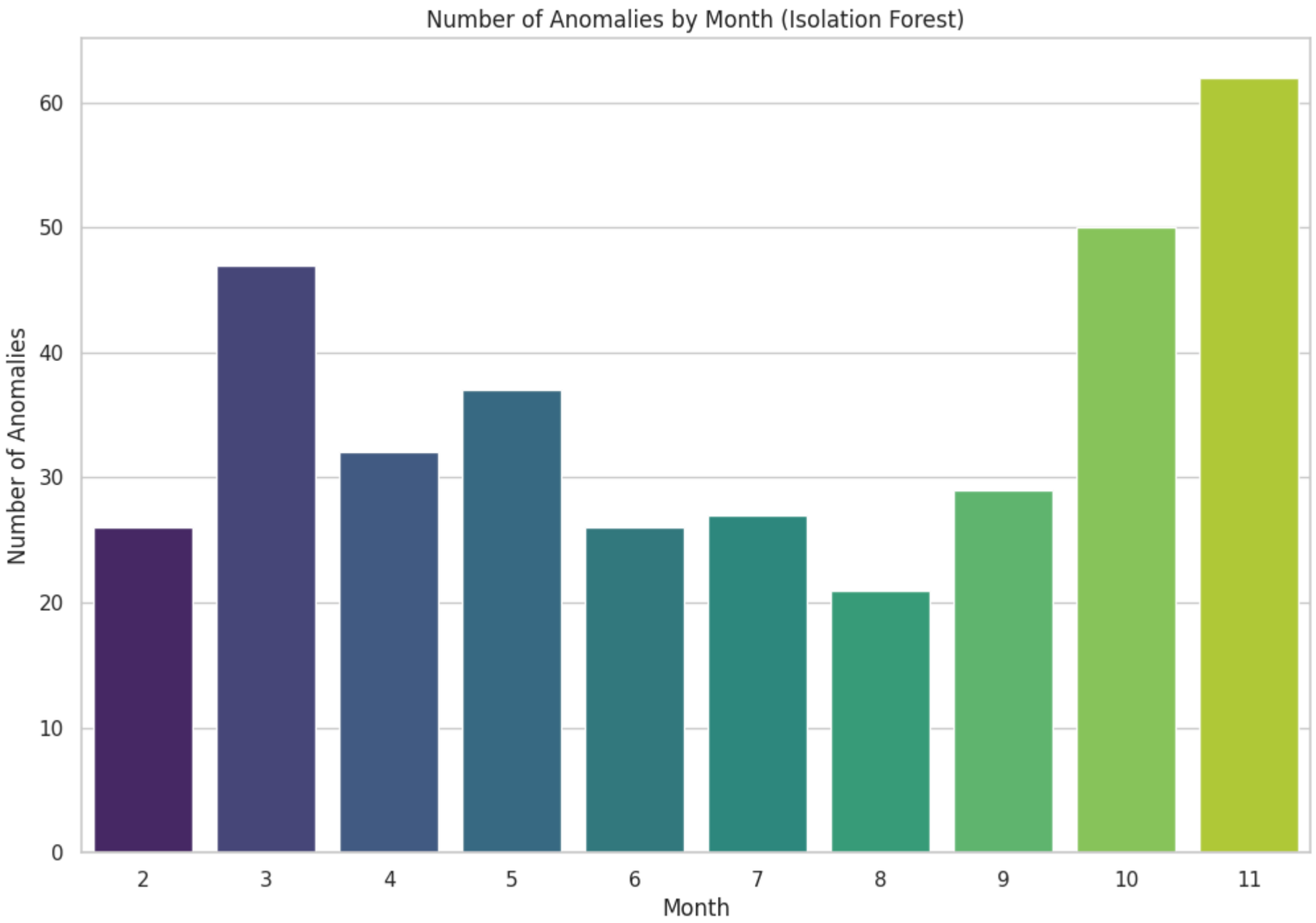
In [82]:
```
# Additional visualization: Voltage vs. Energy Consumption with anomalies highlighted (Isolation Forest)
plt.figure(figsize=(10,6))
sns.scatterplot(x='Voltage', y='Energy Consumption (kWh)', data=test_df, label='Normal', alpha=0.6)
sns.scatterplot(x='Voltage', y='Energy Consumption (kWh)',
                data=anomalies_if, color='red', label='Anomaly')
plt.title('Voltage vs. Energy Consumption with Anomalies Highlighted (Isolation Forest)')
plt.xlabel('Voltage')
plt.ylabel('Energy Consumption (kWh)')
plt.legend()
plt.show()
```

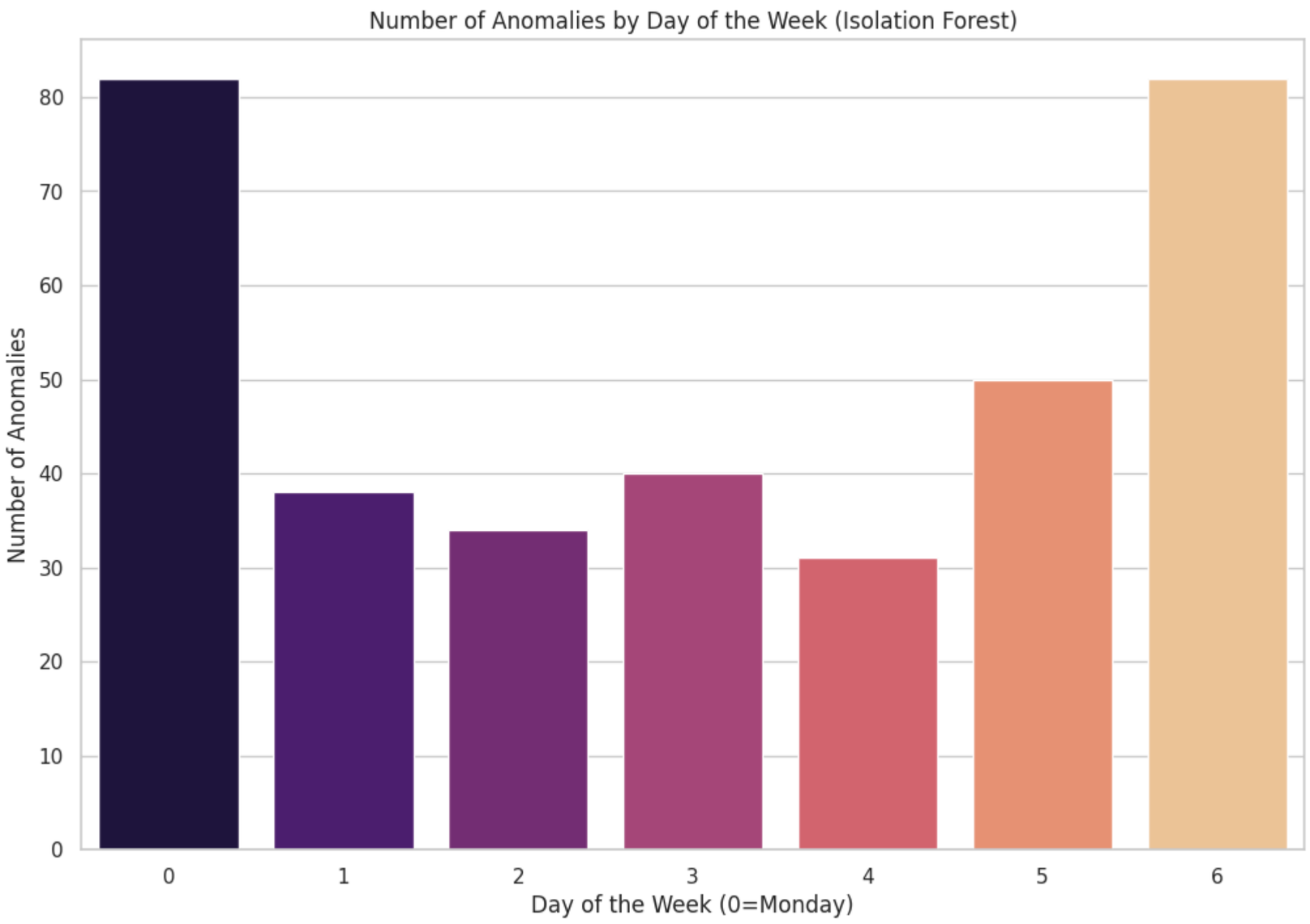


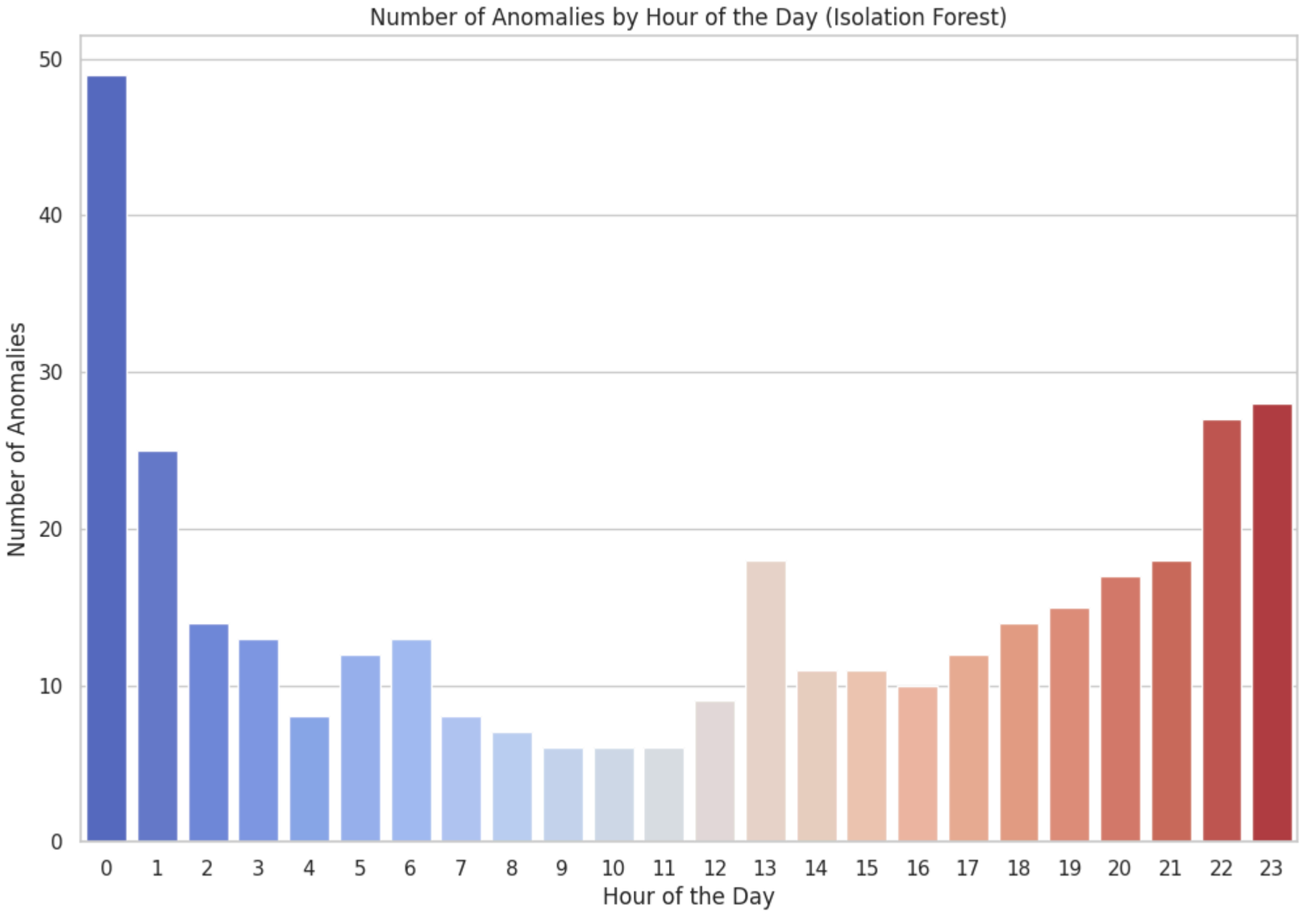
In [83]:
```
# Additional visualization: Apparent Power distribution with anomalies (Isolation Forest)
plt.figure(figsize=(10,6))
sns.histplot(test_df['Apparent Power'], bins=50, kde=True, label='Normal', color='blue', alpha=0.6)
sns.histplot(anomalies_if['Apparent Power'], bins=50, kde=True, label='Anomaly', color='red', alpha=0.6)
plt.axvline(x=threshold, color='orange', linestyle='--', label='Anomaly Threshold')
plt.title('Distribution of Apparent Power with Anomalies Highlighted (Isolation Forest)')
plt.xlabel('Apparent Power')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



In [84]:
```
# Analyze anomalies by Month (Isolation Forest)
anomalies_month_if = anomalies_if['Month_Num'].value_counts().sort_index()
sns.barplot(x=anomalies_month_if.index, y=anomalies_month_if.values, palette='viridis')
plt.title('Number of Anomalies by Month (Isolation Forest)')
plt.xlabel('Month')
plt.ylabel('Number of Anomalies')
plt.show()

# Analyze anomalies by Day of the Week (Isolation Forest)
anomalies_day_if = anomalies_if['Day_of_Week_Num'].value_counts().sort_index()
sns.barplot(x=anomalies_day_if.index, y=anomalies_day_if.values, palette='magma')
plt.title('Number of Anomalies by Day of the Week (Isolation Forest)')
plt.xlabel('Day of the Week (0=Monday)')
plt.ylabel('Number of Anomalies')
plt.show()

# Analyze anomalies by Hour of the Day (Isolation Forest)
anomalies_hour_if = anomalies_if['Hour_of_Day'].value_counts().sort_index()
sns.barplot(x=anomalies_hour_if.index, y=anomalies_hour_if.values, palette='coolwarm')
plt.title('Number of Anomalies by Hour of the Day (Isolation Forest)')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Anomalies')
plt.show()
```


Number of Anomalies by Month (Isolation Forest)
Number of Anomalies
0
10
20
30
40
50
60
2
3
4
5
6
7
8
9
10
11
Month


Number of Anomalies by Day of the Week (Isolation Forest)
Number of Anomalies
0
10
20
30
40
50
60
70
80
0
1
2
3
4
5
6
Day of the Week (0=Monday)


Number of Anomalies by Hour of the Day (Isolation Forest)
Number of Anomalies
0
10
20
30
40
50
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Hour of the Day