Important Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.ensemble import IsolationForest
```

* 2) Data Preprocessing*

A) Load and Explore the Dataset

---> Load the Dataset and display the first few rows to understand the structure.

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

appliance_data = pd.read_csv('/content/smart_home_dataset.csv')
appliance_data.head()
```

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

B. Data Cleaning

--> Convert the "Unix Timestamp" into a readable data format.

***---> Handling the missing values ***

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

appliance_data = pd.read_csv('/content/smart_home_dataset.csv')

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)

print(appliance_data)
print("size and shape of Data set:")
print(appliance_data.shape)
print("##############################")
print("checking the information of data set:")
appliance_data.info()
print("##############################")
print("checking the null values in the dataset:")
appliance_data.isnull().sum()
```

```
           Unix Timestamp  Transaction_ID  Television  Dryer  Oven  \
0     2020-01-01 00:00:00               1           0      0     0
1     2020-01-01 00:42:02               2           0      1     0
2     2020-01-01 01:24:05               3           0      1     0
3     2020-01-01 02:06:08               4           1      0     1
4     2020-01-01 02:48:11               5           1      0     0
...                   ...             ...         ...    ...   ...
48967 2023-11-30 20:45:35           48968           1      0     1
48968 2023-11-30 21:27:38           48969           0      0     1
48969 2023-11-30 22:09:41           48970           1      0     0
48970 2023-11-30 22:51:44           48971           1      1     1
48971 2023-11-30 23:33:47           48972           0      0     0

       Refrigerator  Microwave  Line Voltage  Voltage  Apparent Power  \
0                 1          0           237      233            1559
1                 0          1           232      230            1970
2                 0          0           223      222            1684
3                 1          0           225      224            1694
4                 1          0           222      214            1889
...             ...        ...           ...      ...             ...
48967             0          0           234      230            1815
48968             0          1           238      235            1692
48969             1          1           235      229            1686
48970             1          1           237      230            1754
48971             0          1           220      215            1660

       Energy Consumption (kWh)     Month Day of the Week  Hour of the Day  \
0                     24.001763   January       Wednesday                0
1                     31.225154   January       Wednesday                0
2                     70.460700   January       Wednesday                1
3                     32.264043   January       Wednesday                2
4                     32.728111   January       Wednesday                2
...                         ...       ...             ...              ...
48967                 20.161006  November        Thursday               20
48968                 91.965343  November        Thursday               21
48969                 40.224097  November        Thursday               22
48970                 37.366360  November        Thursday               22
48971                 66.239065  November        Thursday               23

      Offloading Decision
0                   Local
1                  Remote
2                  Remote
3                  Remote
4                   Local
...                   ...
48967              Remote
48968               Local
48969               Local
48970               Local
```

```
48971                Local

[48972 rows x 15 columns]
size and shape of Data set:
(48972, 15)
##############################
checking the information of data set:
<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  datetime64[ns]
 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: datetime64[ns](1), float64(1), int64(10), object(3)
memory usage: 5.6+ MB
##############################
checking the null values in the dataset:
```

	0
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

--> Splitting the Data before Normalizing and scaling to avoid Data Leak

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split

appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)

appliance_data = pd.DataFrame(appliance_data)
appliance_data.columns = appliance_data.columns.str.strip()
#print(appliance_data
# splitting the data before scaling to avoid Data Leak

x_appliance_data = appliance_data.iloc[:,:-1]
#print(x_appliance_data)
#print("##############################")
y_appliance_data = appliance_data.iloc[:,-1]
#print(y_appliance_data)

x_appliance_data_train,x_appliance_data_test,y_appliance_data_train,y_appliance_data_test = train_test_split(x_appliance_data,y_appliance_data,tes
print("training Dataset X:")
print(x_appliance_data_train)
print("$$$$$$$$$$$$$$$$$$$$$$$$")
print("training Dataset Y:")
print(y_appliance_data_train)

print("Displaying the Test Dataset:")
print("Testing Dataset X:")
print(x_appliance_data_test)
print("$$$$$$$$$$$$$$$$$$$$$$$$")
print("Testing Dataset Y:")
print(y_appliance_data_test)
```

```
training Dataset X:
           Unix Timestamp  Transaction_ID  Television  Dryer  Oven  \
```

```
40485 2023-03-28 04:27:19          40486           1      1     1
45468 2023-08-20 16:36:42          45469           0      1     1
38630 2023-02-02 00:26:43          38631           0      0     0
22381 2021-10-15 12:55:06          22382           1      0     0
29969 2022-05-25 02:41:42          29970           0      0     0
...                   ...            ...         ...    ...   ...
21243 2021-09-12 07:23:31          21244           0      1     0
45891 2023-09-02 01:03:21          45892           1      0     1
42613 2023-05-29 07:47:15          42614           0      1     0
43567 2023-06-26 04:21:51          43568           0      0     0
2732  2020-03-20 18:37:26           2733           1      1     1

       Refrigerator  Microwave  Line Voltage  Voltage  Apparent Power  \
40485             0          0           229      229            1895
45468             0          0           229      222            1576
38630             1          1           222      219            1924
22381             0          1           226      225            1529
29969             1          0           223      219            1964
...             ...        ...           ...      ...             ...
21243             0          1           236      232            1695
45891             1          1           233      228            1554
42613             0          0           236      235            1855
43567             0          0           224      219            1515
2732              1          1           234      228            1635

       Energy Consumption (kWh)      Month Day of the Week  Hour of the Day
40485                 10.102420      March         Tuesday                4
45468                 87.199789     August          Sunday               16
38630                 82.569542   February        Thursday                0
22381                 29.984347    October          Friday               12
29969                 43.646074        May       Wednesday                2
...                         ...        ...             ...              ...
21243                 36.332429  September          Sunday                7
45891                 67.365992  September        Saturday                1
42613                 86.451906        May          Monday                7
43567                 59.434337       June          Monday                4
2732                  21.009772      March          Friday               18

[39177 rows x 14 columns]
$$$$$$$$$$$$$$$$$$$$$$$
training Dataset Y:
40485     Local
45468     Local
38630     Local
22381    Remote
29969     Local
          ...
21243    Remote
45891     Local
42613    Remote
```

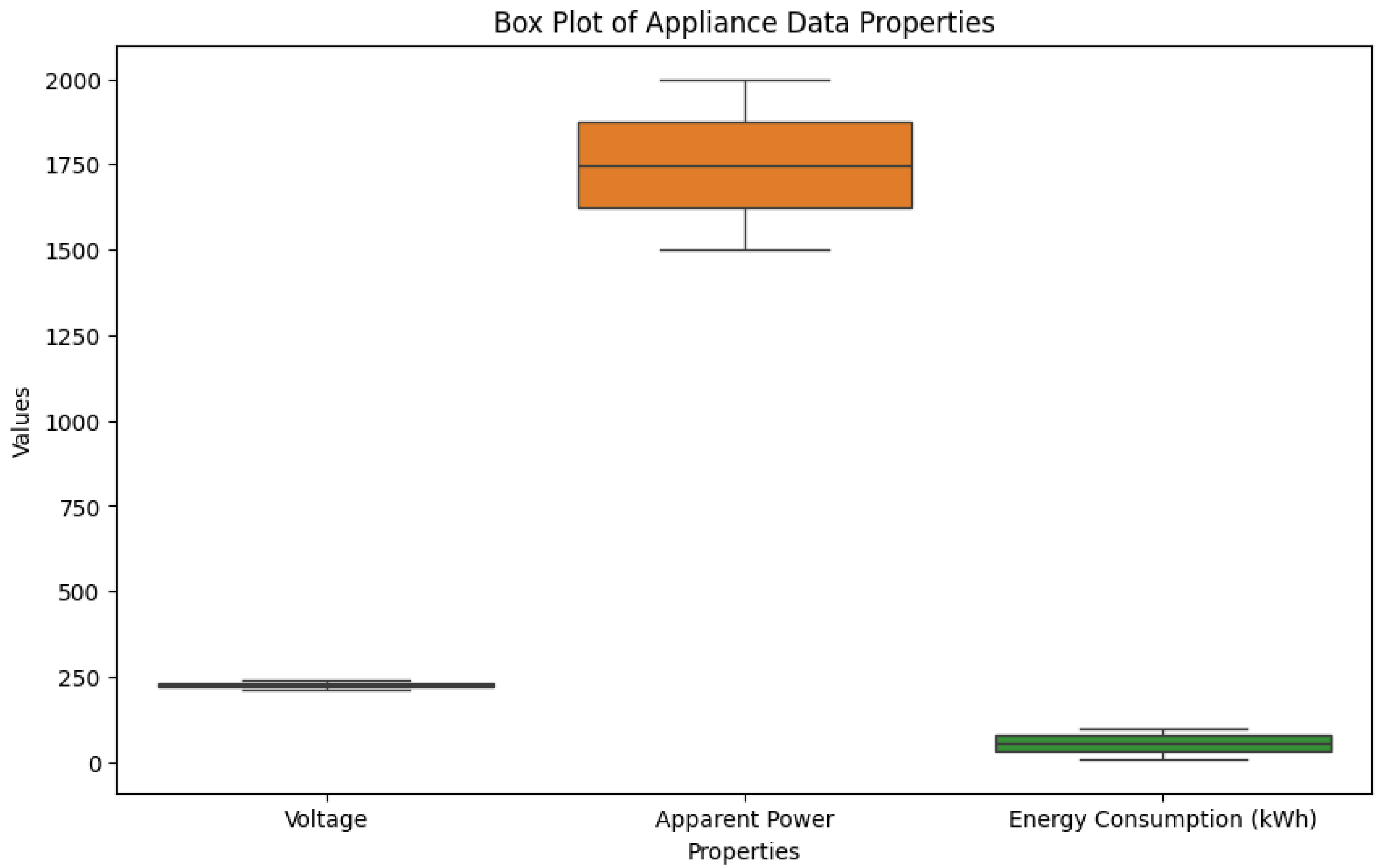
```
43567     Local
2732      Local
Name: Offloading Decision, Length: 39177, dtype: object
Displaying the Test Dataset:
Testing Dataset X:
```

--> Normalize or scale features like Voltage, Apparent Power, and Energy Consumption (kWh) to standardize the data for modeling by identifying the features distribution using Visualized presentation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)

appliance_data = pd.DataFrame(appliance_data)
appliance_data.columns = appliance_data.columns.str.strip()
#print(appliance_data
# splitting the data before scaling to avoid Data Leak

x_appliance_data = appliance_data.iloc[:,:-1]
#print(x_appliance_data)
#print("##############################")
y_appliance_data = appliance_data.iloc[:,-1]
#print(y_appliance_data)

x_appliance_data_train,x_appliance_data_test,y_appliance_data_train,y_appliance_data_test = train_test_split(x_appliance_data,y_appliance_data,tes

appliance_data_properties = ['Voltage', 'Apparent Power', 'Energy Consumption (kWh)']

print("Below Graph (Diagram) depicts the visualization of Distributions for each mentioned Features")
print("\n")
plt.figure(figsize=(10, 6))
sns.boxplot(data=x_appliance_data_train[appliance_data_properties])
plt.title('Box Plot of Appliance Data Properties')
plt.xlabel('Properties')
```

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

    # we will be using "Standard Scaling" method for "Voltage" feature of smart_home_dataset, as they follow normal distribution.
    scalar_standard = StandardScaler()
    x_appliance_data_train['Voltage'] = scalar_standard.fit_transform(x_appliance_data_train[['Voltage']])

    # we will be applying "Power Transformer" scaling method for "ApparantPower" feature of "smart_home_dataset", as they fall into "Skewed_Distributi
    scalar_power = PowerTransformer(method = 'yeo-johnson')
    x_appliance_data_train['Apparent Power'] = scalar_power.fit_transform(x_appliance_data_train[['Apparent Power']])

    # we will be using "Power Transformer" for "EnergyConsumption" feature of "smart_home_dataset", as they are falling under "Skewed_Distribution".
    scalar_power_energy = PowerTransformer(method = 'yeo-johnson')
    x_appliance_data_train['Energy Consumption (kWh)'] = scalar_power_energy.fit_transform(x_appliance_data_train[['Energy Consumption (kWh)']])

    print(" After using scaling methods on data_x:\n")
    print(x_appliance_data_train)

    # we will be scaling the testing data from smart home dataset
    x_appliance_data_test = pd.DataFrame(x_appliance_data_test)
    x_appliance_data_test['Voltage'] = scalar_standard.transform(x_appliance_data_test[['Voltage']])
    x_appliance_data_test['Apparent Power'] = scalar_power.transform(x_appliance_data_test[['Apparent Power']])
    x_appliance_data_test['Energy Consumption (kWh)'] = scalar_power_energy.transform(x_appliance_data_test[['Energy Consumption (kWh)']])

    print("After applying scaling methods on testing data:\n")
    print(x_appliance_data_test)
```

```
Below Graph (Diagram) depicts the visualization of Distributions for each mentioned Features
```




```
After using scaling methods on data_x:

            Unix Timestamp  Transaction_ID  Television  Dryer  Oven  \
40485 2023-03-28 04:27:19           40486           1      1     1
45468 2023-08-20 16:36:42           45469           0      1     1
38630 2023-02-02 00:26:43           38631           0      0     0
22381 2021-10-15 12:55:06           22382           1      0     0
29969 2022-05-25 02:41:42           29970           0      0     0
...                   ...             ...         ...    ...   ...
21243 2021-09-12 07:23:31           21244           0      1     0
45891 2023-09-02 01:03:21           45892           1      0     1
42613 2023-05-29 07:47:15           42614           0      1     0
43567 2023-06-26 04:21:51           43568           0      0     0
2732  2020-03-20 18:37:26            2733           1      1     1

       Refrigerator  Microwave  Line Voltage   Voltage  Apparent Power  \
```

```
       Refrigerator  Microwave  Line Voltage   Voltage  Apparent Power  \
40485             0          0           229  0.624197        1.006945
45468             0          0           229 -0.461662       -1.209199
38630             1          1           222 -0.927030        1.203551
22381             0          1           226  0.003706       -1.544833
29969             1          0           223 -0.927030        1.473516
...             ...        ...           ...       ...             ...
21243             0          1           236  1.089565       -0.370392
45891             1          1           233  0.469074       -1.365987
42613             0          0           236  1.554933        0.734519
43567             0          0           224 -0.927030       -1.645307
2732              1          1           234  0.469074       -0.791404

       Energy Consumption (kWh)      Month Day of the Week  Hour of the Day
40485                 -1.915255      March         Tuesday                4
45468                  1.199030     August          Sunday               16
38630                  1.042463   February        Thursday                0
22381                 -0.948187    October          Friday               12
29969                 -0.380321        May       Wednesday                2
...                         ...        ...             ...              ...
21243                 -0.677880  September          Sunday                7
45891                  0.512127  September        Saturday                1
42613                  1.173883        May          Monday                7
43567                  0.223900       June          Monday                4
2732                  -1.355770      March          Friday               18

[39177 rows x 14 columns]
After applying scaling methods on testing data:

           Unix Timestamp  Transaction_ID  Television  Dryer  Oven  \
33846 2022-09-15 07:45:04           33847           1      0     1
4819  2020-05-20 17:13:21            4820           1      0     0
3114  2020-03-31 22:20:05            3115           1      1     0
39572 2023-03-01 12:36:44           39573           0      1     0
44136 2023-07-12 19:07:38           44137           0      0     0
...                   ...             ...         ...    ...   ...
3715  2020-04-18 11:31:26            3716           1      0     1
8671  2020-09-10 04:45:30            8672           0      1     0
38935 2023-02-10 22:11:37           38936           1      0     1
7887  2020-08-18 07:19:12            7888           0      1     1
27782 2022-03-22 06:00:54           27783           0      0     0

       Refrigerator  Microwave  Line Voltage   Voltage  Apparent Power  \
33846             1          1           229 -0.461662        0.850479
4819              1          1           223 -1.237276        0.918566
3114              1          0           237  0.779320        1.277931
39572             0          1           235  0.934442       -0.573406
44136             0          0           233  0.003706        0.748175
...             ...        ...           ...       ...             ...
3715              1          1           236  1.554933        0.864103
8671              1          0           229 -0.771907       -0.328498
```

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"

```
4819                0.643439      May      Wednesday           17
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split

appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)

# Here we will be extracting meaningful time based features from appliance_data i.e smart_home_dataset

appliance_data['month'] = appliance_data['Unix Timestamp'].dt.month
appliance_data['day of the week'] = appliance_data['Unix Timestamp'].dt.day
appliance_data['hour of the week'] = appliance_data['Unix Timestamp'].dt.weekday * 24 + appliance_data['Unix Timestamp'].dt.hour
appliance_data = appliance_data.drop(columns=['Unix Timestamp'])
print(appliance_data[['month', 'day of the week', 'hour of the week']])
```

```
       month  day of the week  hour of the week
0          1                1                48
1          1                1                48
2          1                1                49
3          1                1                50
4          1                1                50
...      ...              ...               ...
48967     11               30                92
48968     11               30                93
48969     11               30                94
48970     11               30                94
48971     11               30                95

[48972 rows x 3 columns]
```

D). Data Splitting

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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split

appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)
appliance_data['month'] = appliance_data['Unix Timestamp'].dt.month
appliance_data['day of the week'] = appliance_data['Unix Timestamp'].dt.day
appliance_data['hour of the week'] = appliance_data['Unix Timestamp'].dt.weekday * 24 + appliance_data['Unix Timestamp'].dt.hour
appliance_data = appliance_data.drop(columns=['Unix Timestamp'])

appliance_data = pd.DataFrame(appliance_data)
appliance_data.columns = appliance_data.columns.str.strip()
#print(appliance_data
# splitting the data before scaling to avoid Data Leak

x_appliance_data = appliance_data.iloc[:,:-1]
#print(x_appliance_data)
#print("##############################")
y_appliance_data = appliance_data.iloc[:,-1]
#print(y_appliance_data)

x_appliance_data_train,x_appliance_data_test,y_appliance_data_train,y_appliance_data_test = train_test_split(x_appliance_data,y_appliance_data,test_

print("Below Data depicts the Training and Testing Data of smart_home_dataset(appliance_data):\n")

print("Training Dataset of x and y:")
print('Training Dataset x:')
print(x_appliance_data_train)
print("#############################################")
print('Training dataset y:')
print(y_appliance_data_train)
print("##############################")

print("Testing Dataset of x and y:")
print('Testing Dataset x:')
```

```
print(x_appliance_data_test)
print("############################################")
print('Testing dataset y:')
print(y_appliance_data_test)
print("##############################")
```

```
Below Data depicts the Training and Testing Data of smart_home_dataset(appliance_data):

Training Dataset of x and y:
Training Dataset x:
       Transaction_ID  Television  Dryer  Oven  Refrigerator  Microwave  \
40485           40486           1      1     1             0          0
45468           45469           0      1     1             0          0
38630           38631           0      0     0             1          1
22381           22382           1      0     0             0          1
29969           29970           0      0     0             1          0
...               ...         ...    ...   ...           ...        ...
21243           21244           0      1     0             0          1
45891           45892           1      0     1             1          1
42613           42614           0      1     0             0          0
43567           43568           0      0     0             0          0
2732             2733           1      1     1             1          1

       Line Voltage  Voltage  Apparent Power  Energy Consumption (kWh)  \
40485           229      229            1895                 10.102420
45468           229      222            1576                 87.199789
38630           222      219            1924                 82.569542
22381           226      225            1529                 29.984347
29969           223      219            1964                 43.646074
...             ...      ...             ...                       ...
21243           236      232            1695                 36.332429
45891           233      228            1554                 67.365992
42613           236      235            1855                 86.451906
43567           224      219            1515                 59.434337
2732            234      228            1635                 21.009772

           Month Day of the Week  Hour of the Day Offloading Decision  month  \
40485      March         Tuesday                4               Local      3
45468     August          Sunday               16               Local      8
38630   February        Thursday                0               Local      2
22381    October          Friday               12              Remote     10
29969        May       Wednesday                2               Local      5
...          ...             ...              ...                 ...    ...
21243  September          Sunday                7              Remote      9
45891  September        Saturday                1               Local      9
42613        May          Monday                7              Remote      5
43567       June          Monday                4               Local      6
2732       March          Friday               18               Local      3
```

```
         day of the week
40485                 28
45468                 20
38630                  2
22381                 15
29969                 25
...                  ...
21243                 12
45891                  2
42613                 29
43567                 26
2732                  20

[39177 rows x 16 columns]
""""""""""""""""""""""""""""""""""""""""""""""
```

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

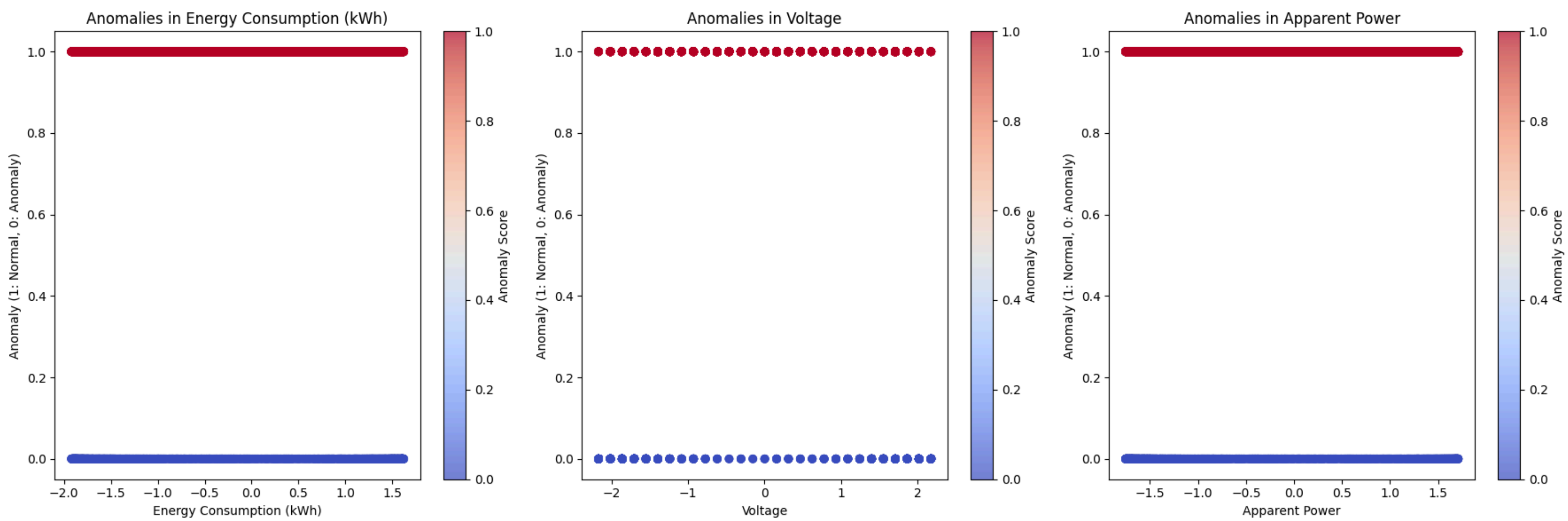
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.ensemble import IsolationForest
appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)


appliance_data = pd.DataFrame(appliance_data)
appliance_data.columns = appliance_data.columns.str.strip()
#print(appliance_data
# splitting the data before scaling to avoid Data Leak

x_appliance_data = appliance_data.iloc[:,:-1]
#print(x_appliance_data)
```

```
#print("##############################")
y_appliance_data = appliance_data.iloc[:,-1]
#print(y_appliance_data)

x_appliance_data_train,x_appliance_data_test,y_appliance_data_train,y_appliance_data_test = train_test_split(x_appliance_data,y_appliance_data,tes

appliance_data_properties = ['Voltage', 'Apparent Power', 'Energy Consumption (kWh)']

#plt.figure(figsize=(10, 6))
#sns.boxplot(data=x_appliance_data_train[appliance_data_properties])
#plt.title('Box Plot of Appliance Data Properties')
#plt.xlabel('Properties')
#plt.ylabel('Values')
#plt.show()

# we will be using "Standard Scaling" method for "Voltage" feature of smart_home_dataset, as they follow normal distribution.
scalar_standard = StandardScaler()
x_appliance_data_train['Voltage'] = scalar_standard.fit_transform(x_appliance_data_train[['Voltage']])

# we will be applying "Power Transformer" scaling method for "ApparantPower" feature of "smart_home_dataset", as they fall into "Skewed_Distributi
scalar_power = PowerTransformer(method = 'yeo-johnson')
x_appliance_data_train['Apparent Power'] = scalar_power.fit_transform(x_appliance_data_train[['Apparent Power']])

# we will be using "Power Transformer" for "EnergyConsumption" feature of "smart_home_dataset", as they are falling under "Skewed_Distribution".
scalar_power_energy = PowerTransformer(method = 'yeo-johnson')
x_appliance_data_train['Energy Consumption (kWh)'] = scalar_power_energy.fit_transform(x_appliance_data_train[['Energy Consumption (kWh)']])

#print(" After using scaling methods on data_x:\n")
#print(x_appliance_data_train)

# we will be scaling the testing data from smart home dataset
x_appliance_data_test = pd.DataFrame(x_appliance_data_test)
x_appliance_data_test['Voltage'] = scalar_standard.transform(x_appliance_data_test[['Voltage']])
x_appliance_data_test['Apparent Power'] = scalar_power.transform(x_appliance_data_test[['Apparent Power']])
x_appliance_data_test['Energy Consumption (kWh)'] = scalar_power_energy.transform(x_appliance_data_test[['Energy Consumption (kWh)']])

# As we have columns with string values in a dataset, we will be transforming them to vaoid their hinderance while training and testing the data.
appliance_data['month'] = appliance_data['Unix Timestamp'].dt.month
appliance_data['day of the week'] = appliance_data['Unix Timestamp'].dt.day
appliance_data['hour of the week'] = appliance_data['Unix Timestamp'].dt.weekday * 24 + appliance_data['Unix Timestamp'].dt.hour
appliance_data = appliance_data.drop(columns=['Unix Timestamp'])


#As we have string values in columns 'Month' and "Day of the Week", we need to map them to respective values to convert them into numeric values.

# Define mappings
```

```
Month_mapping = {
    "January": 1, "February": 2, "March": 3, "April": 4,
    "May": 5, "June": 6, "July": 7, "August": 8,
    "September": 9, "October": 10, "November": 11, "December": 12
}

Day_of_week_mapping = {
    "Monday": 0, "Tuesday": 1, "Wednesday": 2, "Thursday": 3,
    "Friday": 4, "Saturday": 5, "Sunday": 6
}

# Map string values to numbers
x_appliance_data_train['Month'] = x_appliance_data_train['Month'].map(Month_mapping)
x_appliance_data_train['Day of the Week'] = x_appliance_data_train['Day of the Week'].map(Day_of_week_mapping)

#Now we are gonna handle Month and Day columns from the respective training data.
if 'Month'in x_appliance_data_train.columns and 'Day of the Week' in x_appliance_data_train.columns:
  # Transform 'month' and 'day' to cyclical features
    x_appliance_data_train['Month_sin'] = np.sin(2 * np.pi * x_appliance_data_train['Month'] / 12)
    x_appliance_data_train['Month_cos'] = np.cos(2 * np.pi * x_appliance_data_train['Month'] / 12)
    x_appliance_data_train['Day of the Weeky_sin'] = np.sin(2 * np.pi * x_appliance_data_train['Day of the Week'] / 31)
    x_appliance_data_train['Day of the Weeky_cos'] = np.cos(2 * np.pi * x_appliance_data_train['Day of the Week'] / 31)

x_appliance_data_train = x_appliance_data_train.drop(columns=['Month', 'Day of the Week'])
y_appliance_data_train = pd.DataFrame(y_appliance_data_train)

# Now that, we have removed all the options of having Non-Numeric values in the dataset. Let's predict some anomolies in the respective features.
features = ['Energy Consumption (kWh)','Voltage','Apparent Power']
x_appliance_data_train_features = x_appliance_data_train[features]

#Now applying Isolation Forest Model to predict the Anomolies.
isolation_forest_model = IsolationForest(n_estimators =100, contamination = 0.05, max_samples = 'auto', random_state=42, verbose =1 )
isolation_forest_model.fit(x_appliance_data_train_features)
# Now predicting the anomolies only on the training set as mentioned.
x_appliance_data_train_predictions = isolation_forest_model.predict(x_appliance_data_train_features)
x_appliance_data_train_labels = (x_appliance_data_train_predictions == 1).astype(int)

# Step 5: Visualize anomalies for each feature individually

print("Below Diagram charts depicts the Anomolies present in each of the Features:")
print("\n")

# Assuming you have already set up the subplots and scatter plots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Visualize anomalies in Energy Consumption (kWh)
```

```
    scatter1 = axes[0].scatter(
        x_appliance_data_train_features['Energy Consumption (kWh)'],
        x_appliance_data_train_labels,
        c=x_appliance_data_train_labels, cmap='coolwarm', alpha=0.7
    )
    axes[0].set_title('Anomalies in Energy Consumption (kWh)')
    axes[0].set_xlabel('Energy Consumption (kWh)')
    axes[0].set_ylabel('Anomaly (1: Normal, 0: Anomaly)')
    fig.colorbar(scatter1, ax=axes[0], label='Anomaly Score')  # Adding colorbar

    # Visualize anomalies in Voltage
    scatter2 = axes[1].scatter(
        x_appliance_data_train_features['Voltage'],
        x_appliance_data_train_labels,
        c=x_appliance_data_train_labels, cmap='coolwarm', alpha=0.7
    )
    axes[1].set_title('Anomalies in Voltage')
    axes[1].set_xlabel('Voltage')
    axes[1].set_ylabel('Anomaly (1: Normal, 0: Anomaly)')
    fig.colorbar(scatter2, ax=axes[1], label='Anomaly Score')  # Adding colorbar

    # Visualize anomalies in Apparent Power
    scatter3 = axes[2].scatter(
        x_appliance_data_train_features['Apparent Power'],
        x_appliance_data_train_labels,
        c=x_appliance_data_train_labels, cmap='coolwarm', alpha=0.7
    )
    axes[2].set_title('Anomalies in Apparent Power')
    axes[2].set_xlabel('Apparent Power')
    axes[2].set_ylabel('Anomaly (1: Normal, 0: Anomaly)')
    fig.colorbar(scatter3, ax=axes[2], label='Anomaly Score')  # Adding colorbar

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

```
Below Diagram charts depicts the Anomolies present in each of the Features:
```



4. Anomoly Labeling and Evaluation

A). Labeling Anomolies

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

-->Create synthetic labels for anomalies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
```

```
from sklearn.ensemble import IsolationForest
appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

def convert_to_readable_time(unix_timestamp):
  readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
  return readable_time
appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)


appliance_data = pd.DataFrame(appliance_data)
appliance_data.columns = appliance_data.columns.str.strip()
#print(appliance_data
# splitting the data before scaling to avoid Data Leak

x_appliance_data = appliance_data.iloc[:,:-1]
#print(x_appliance_data)
#print("##############################")
y_appliance_data = appliance_data.iloc[:,-1]
#print(y_appliance_data)

x_appliance_data_train,x_appliance_data_test,y_appliance_data_train,y_appliance_data_test = train_test_split(x_appliance_data,y_appliance_data,tes

appliance_data_properties = ['Voltage', 'Apparent Power', 'Energy Consumption (kWh)']

#plt.figure(figsize=(6, 6))
#sns.boxplot(data=x_appliance_data_train[appliance_data_properties])
#plt.title('Box Plot of Appliance Data Properties')
#plt.xlabel('Properties')
#plt.ylabel('Values')
#plt.show()

# we will be using "Standard Scaling" method for "Voltage" feature of smart_home_dataset, as they follow normal distribution.
scalar_standard = StandardScaler()
x_appliance_data_train['Voltage'] = scalar_standard.fit_transform(x_appliance_data_train[['Voltage']])

# we will be applying "Power Transformer" scaling method for "ApparantPower" feature of "smart_home_dataset", as they fall into "Skewed_Distributi
scalar_power = PowerTransformer(method = 'yeo-johnson')
x_appliance_data_train['Apparent Power'] = scalar_power.fit_transform(x_appliance_data_train[['Apparent Power']])

# we will be using "Power Transformer" for "EnergyConsumption" feature of "smart_home_dataset", as they are falling under "Skewed_Distribution".
scalar_power_energy = PowerTransformer(method = 'yeo-johnson')
x_appliance_data_train['Energy Consumption (kWh)'] = scalar_power_energy.fit_transform(x_appliance_data_train[['Energy Consumption (kWh)']])

#print(" After using scaling methods on data_x:\n")
#print(x_appliance_data_train)
```

```
# we will be scaling the testing data from smart home dataset
x_appliance_data_test = pd.DataFrame(x_appliance_data_test)
x_appliance_data_test['Voltage'] = scalar_standard.transform(x_appliance_data_test[['Voltage']])
x_appliance_data_test['Apparent Power'] = scalar_power.transform(x_appliance_data_test[['Apparent Power']])
x_appliance_data_test['Energy Consumption (kWh)'] = scalar_power_energy.transform(x_appliance_data_test[['Energy Consumption (kWh)']])

# As we have columns with string values in a dataset, we will be transforming them to vaoid their hinderance while training and testing the data.
appliance_data['month'] = appliance_data['Unix Timestamp'].dt.month
appliance_data['day of the week'] = appliance_data['Unix Timestamp'].dt.day
appliance_data['hour of the week'] = appliance_data['Unix Timestamp'].dt.weekday * 24 + appliance_data['Unix Timestamp'].dt.hour
appliance_data = appliance_data.drop(columns=['Unix Timestamp'])


#As we have string values in columns 'Month' and "Day of the Week", we need to map them to respective values to convert them into numeric values.

# Define mappings
Month_mapping = {
    "January": 1, "February": 2, "March": 3, "April": 4,
    "May": 5, "June": 6, "July": 7, "August": 8,
    "September": 9, "October": 10, "November": 11, "December": 12
}

Day_of_week_mapping = {
    "Monday": 0, "Tuesday": 1, "Wednesday": 2, "Thursday": 3,
    "Friday": 4, "Saturday": 5, "Sunday": 6
}

# Map string values to numbers
x_appliance_data_train['Month'] = x_appliance_data_train['Month'].map(Month_mapping)
x_appliance_data_train['Day of the Week'] = x_appliance_data_train['Day of the Week'].map(Day_of_week_mapping)

#Now we are gonna handle Month and Day columns from the respective training data.
if 'Month'in x_appliance_data_train.columns and 'Day of the Week' in x_appliance_data_train.columns:
  # Transform 'month' and 'day' to cyclical features
    x_appliance_data_train['Month_sin'] = np.sin(2 * np.pi * x_appliance_data_train['Month'] / 12)
    x_appliance_data_train['Month_cos'] = np.cos(2 * np.pi * x_appliance_data_train['Month'] / 12)
    x_appliance_data_train['Day of the Weeky_sin'] = np.sin(2 * np.pi * x_appliance_data_train['Day of the Week'] / 31)
    x_appliance_data_train['Day of the Weeky_cos'] = np.cos(2 * np.pi * x_appliance_data_train['Day of the Week'] / 31)

x_appliance_data_train = x_appliance_data_train.drop(columns=['Month', 'Day of the Week'])
y_appliance_data_train = pd.DataFrame(y_appliance_data_train)

# Now that, we have removed all the options of having Non-Numeric values in the dataset. Let's predict some anomolies in the respective features.
features = ['Energy Consumption (kWh)','Voltage','Apparent Power']
x_appliance_data_train_features = x_appliance_data_train[features]
```

```
#Now applying Isolation Forest Model to predict the Anomolies.
isolation_forest_model = IsolationForest(n_estimators =100, contamination = 0.05, max_samples = 'auto', random_state=42, verbose =1 )
isolation_forest_model.fit(x_appliance_data_train_features)
# Now predicting the anomolies only on the training set as mentioned.
x_appliance_data_train_predictions = isolation_forest_model.predict(x_appliance_data_train_features)
x_appliance_data_train_labels = (x_appliance_data_train_predictions == 1).astype(int)
```

```
# Assuming x_appliance_data_train is your training dataset
# Defining a synthetic threshold for anomalies in 'Energy Consumption (kWh)'

# 1. Calculate the mean and standard deviation of 'Energy Consumption (kWh)'
mean_energy_train = x_appliance_data_train['Energy Consumption (kWh)'].mean()
std_energy_train = x_appliance_data_train['Energy Consumption (kWh)'].std()

# 2. Define thresholds for anomaly detection (using mean ± 3 * std deviation)
threshold_upper_train = mean_energy_train + 3 * std_energy_train  # upper threshold
threshold_lower_train = mean_energy_train - 3 * std_energy_train  # lower threshold

# 3. Create synthetic labels for anomalies based on these thresholds
# If a value is above the upper threshold or below the lower threshold, it's an anomaly
x_appliance_data_train['Anomaly Label'] = np.where(
    (x_appliance_data_train['Energy Consumption (kWh)'] > threshold_upper_train) |
    (x_appliance_data_train['Energy Consumption (kWh)'] < threshold_lower_train),
    0,  # 0 for anomaly
    1   # 1 for normal
)

# 4. Check the data with anomaly labels
print(x_appliance_data_train[['Energy Consumption (kWh)', 'Anomaly Label']])

# 5. Count the number of anomalies detected
anomalies_count = x_appliance_data_train['Anomaly Label'].value_counts()
print(f"Anomalies detected (including synthetic ones):\n{anomalies_count}")
```

```
       Energy Consumption (kWh)  Anomaly Label
40485                 -1.915255              1
45468                  1.199030              1
38630                  1.042463              1
22381                 -0.948187              1
29969                 -0.380321              1
...                         ...            ...
21243                 -0.677880              1
```

```
45891                0.512127              1
42613                1.173883              1
43567                0.223900              1
2732                -1.355770              1

[39177 rows x 2 columns]
Anomalies detected (including synthetic ones):
Anomaly Label
1    39177
Name: count, dtype: int64
```

B). Evaluate the Model

--> Model Evaluation with Synthetic Labels

--> Calculate specific metrics (Accuracy, Precision, Recall Score, F1 Score)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.ensemble import IsolationForest
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load the dataset
appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

# Convert the 'Unix Timestamp' to readable time
def convert_to_readable_time(unix_timestamp):
    readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
    return readable_time

appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)

# Clean the column names
appliance_data.columns = appliance_data.columns.str.strip()

# Splitting the data before scaling to avoid data leakage
x_appliance_data = appliance_data.iloc[:,:-1]
y_appliance_data = appliance_data.iloc[:,-1]
```

```
# Splitting into training and testing data
x_appliance_data_train, x_appliance_data_test, y_appliance_data_train, y_appliance_data_test = train_test_split(x_appliance_data, y_appliance_data

# Feature selection for scaling
appliance_data_properties = ['Voltage', 'Apparent Power', 'Energy Consumption (kWh)']

# Standard Scaling for 'Voltage'
scalar_standard = StandardScaler()
x_appliance_data_train['Voltage'] = scalar_standard.fit_transform(x_appliance_data_train[['Voltage']])

# Power Transformer for 'Apparent Power'
scalar_power = PowerTransformer(method='yeo-johnson')
x_appliance_data_train['Apparent Power'] = scalar_power.fit_transform(x_appliance_data_train[['Apparent Power']])

# Power Transformer for 'Energy Consumption (kWh)'
scalar_power_energy = PowerTransformer(method='yeo-johnson')
x_appliance_data_train['Energy Consumption (kWh)'] = scalar_power_energy.fit_transform(x_appliance_data_train[['Energy Consumption (kWh)']])

# Scaling the testing data
x_appliance_data_test = pd.DataFrame(x_appliance_data_test)
x_appliance_data_test['Voltage'] = scalar_standard.transform(x_appliance_data_test[['Voltage']])
x_appliance_data_test['Apparent Power'] = scalar_power.transform(x_appliance_data_test[['Apparent Power']])
x_appliance_data_test['Energy Consumption (kWh)'] = scalar_power_energy.transform(x_appliance_data_test[['Energy Consumption (kWh)']])

# Creating features for 'Month' and 'Day of the Week' from 'Unix Timestamp'
appliance_data['month'] = appliance_data['Unix Timestamp'].dt.month
appliance_data['day of the week'] = appliance_data['Unix Timestamp'].dt.day
appliance_data['hour of the week'] = appliance_data['Unix Timestamp'].dt.weekday * 24 + appliance_data['Unix Timestamp'].dt.hour
appliance_data = appliance_data.drop(columns=['Unix Timestamp'])

# Map 'Month' and 'Day of the Week' columns to numeric values
Month_mapping = {
    "January": 1, "February": 2, "March": 3, "April": 4,
    "May": 5, "June": 6, "July": 7, "August": 8,
    "September": 9, "October": 10, "November": 11, "December": 12
}

Day_of_week_mapping = {
    "Monday": 0, "Tuesday": 1, "Wednesday": 2, "Thursday": 3,
    "Friday": 4, "Saturday": 5, "Sunday": 6
}

x_appliance_data_train['Month'] = x_appliance_data_train['Month'].map(Month_mapping)
x_appliance_data_train['Day of the Week'] = x_appliance_data_train['Day of the Week'].map(Day_of_week_mapping)

# Create cyclical features for 'Month' and 'Day of the Week'
```

```
x_appliance_data_train['Month_sin'] = np.sin(2 * np.pi * x_appliance_data_train['Month'] / 12)
x_appliance_data_train['Month_cos'] = np.cos(2 * np.pi * x_appliance_data_train['Month'] / 12)
x_appliance_data_train['Day of the Weeky_sin'] = np.sin(2 * np.pi * x_appliance_data_train['Day of the Week'] / 31)
x_appliance_data_train['Day of the Weeky_cos'] = np.cos(2 * np.pi * x_appliance_data_train['Day of the Week'] / 31)

# Drop original 'Month' and 'Day of the Week' columns
x_appliance_data_train = x_appliance_data_train.drop(columns=['Month', 'Day of the Week'])

# Now, applying Isolation Forest Model for anomaly detection
features = ['Energy Consumption (kWh)', 'Voltage', 'Apparent Power']
x_appliance_data_train_features = x_appliance_data_train[features]
x_appliance_data_test_features = x_appliance_data_test[features]

# Initialize Isolation Forest Model
isolation_forest_model = IsolationForest(n_estimators=100, contamination=0.05, max_samples='auto', random_state=42, verbose=1)
isolation_forest_model.fit(x_appliance_data_train_features)

# Predict anomalies for the test set
x_appliance_data_test_predictions = isolation_forest_model.predict(x_appliance_data_test_features)

# Convert the predictions to binary labels (1 = normal, 0 = anomaly)
predicted_labels = (x_appliance_data_test_predictions == 1).astype(int)

# Create synthetic anomaly labels based on 'Energy Consumption (kWh)' thresholds
mean_energy_test = x_appliance_data_test['Energy Consumption (kWh)'].mean()
std_energy_test = x_appliance_data_test['Energy Consumption (kWh)'].std()
threshold_upper_test = mean_energy_test + 3 * std_energy_test
threshold_lower_test = mean_energy_test - 3 * std_energy_test

# Assign synthetic anomaly labels
x_appliance_data_test['Anomaly Label'] = np.where(
    (x_appliance_data_test['Energy Consumption (kWh)'] > threshold_upper_test) |
    (x_appliance_data_test['Energy Consumption (kWh)'] < threshold_lower_test),
    0,  # 0 for anomaly
    1   # 1 for normal
)

# Model evaluation
accuracy = accuracy_score(x_appliance_data_test['Anomaly Label'], predicted_labels)
precision = precision_score(x_appliance_data_test['Anomaly Label'], predicted_labels)
recall = recall_score(x_appliance_data_test['Anomaly Label'], predicted_labels)
f1 = f1_score(x_appliance_data_test['Anomaly Label'], predicted_labels)

# Print evaluation metrics
print("Model Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
```

```
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Model Evaluation Metrics:
Accuracy: 0.9492
Precision: 1.0000
Recall: 0.9492
F1 Score: 0.9739
```

C). Describing the Anomolies

1.Visualized Approach

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from sklearn.ensemble import IsolationForest

# Load the dataset
appliance_data = pd.read_csv('/content/smart_home_dataset.csv', header=0)

# Convert Unix timestamps to readable time
def convert_to_readable_time(unix_timestamp):
    readable_time = datetime.datetime.utcfromtimestamp(unix_timestamp)
    return readable_time

appliance_data['Unix Timestamp'] = appliance_data['Unix Timestamp'].apply(convert_to_readable_time)

# Clean up column names
appliance_data = pd.DataFrame(appliance_data)
appliance_data.columns = appliance_data.columns.str.strip()

# Splitting the data before scaling to avoid data leakage
x_appliance_data = appliance_data.iloc[:,:-1]
y_appliance_data = appliance_data.iloc[:,-1]

# Train-test split
x_appliance_data_train, x_appliance_data_test, y_appliance_data_train, y_appliance_data_test = train_test_split(
```

```
        x_appliance_data, y_appliance_data, test_size=0.2, random_state=0
    )

    # Standard scaling for "Voltage" feature
    scalar_standard = StandardScaler()
    x_appliance_data_train['Voltage'] = scalar_standard.fit_transform(x_appliance_data_train[['Voltage']])

    # Power Transformer scaling for "Apparent Power" feature
    scalar_power = PowerTransformer(method='yeo-johnson')
    x_appliance_data_train['Apparent Power'] = scalar_power.fit_transform(x_appliance_data_train[['Apparent Power']])

    # Power Transformer scaling for "Energy Consumption (kWh)" feature
    scalar_power_energy = PowerTransformer(method='yeo-johnson')
    x_appliance_data_train['Energy Consumption (kWh)'] = scalar_power_energy.fit_transform(x_appliance_data_train[['Energy Consumption (kWh)']])

    # Scaling the test data
    x_appliance_data_test['Voltage'] = scalar_standard.transform(x_appliance_data_test[['Voltage']])
    x_appliance_data_test['Apparent Power'] = scalar_power.transform(x_appliance_data_test[['Apparent Power']])
    x_appliance_data_test['Energy Consumption (kWh)'] = scalar_power_energy.transform(x_appliance_data_test[['Energy Consumption (kWh)']])

    # Predict anomalies using Isolation Forest
    features = ['Energy Consumption (kWh)', 'Voltage', 'Apparent Power']
    x_appliance_data_train_features = x_appliance_data_train[features]

    # Experiment with different contamination values
    isolation_forest_model = IsolationForest(n_estimators=100, contamination=0.1, max_samples='auto', random_state=42, verbose=1)
    isolation_forest_model.fit(x_appliance_data_train_features)

    # Predict anomalies on the training set
    x_appliance_data_train_predictions = isolation_forest_model.predict(x_appliance_data_train_features)

    # Convert -1 (anomaly) and 1 (normal) to binary labels
    x_appliance_data_train_labels = (x_appliance_data_train_predictions == 1).astype(int)

    # Check how many anomalies were detected
    anomalies_count = pd.Series(x_appliance_data_train_labels).value_counts()
    print(f"Total anomalies detected: {anomalies_count[0]}")
```