In [17]:
```
import pandas as pd

# Load the data
df = pd.read_excel('MTHW Data.xlsx', sheet_name='Sheet1')

# Set 'Meter Location' as the index
df.set_index('Meter Location', inplace=True)

# Drop the first row (index 0)
df = df.iloc[1:]

# Define the correct month order
months = [
    'Jan-22', 'Feb-22', 'Mar-22', 'Apr-22', 'May-22', 'Jun-22',
    'Jul-22', 'Aug-22', 'Sep-22', 'Oct-22', 'Nov-22', 'Dec-22',
    'Jan-23', 'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23',
    'Jul-23', 'Aug-23', 'Sep-23', 'Oct-23', 'Nov-23', 'Dec-23',
    'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-24',
    'Jul-24', 'Aug-24', 'Sep-24', 'Oct-24',
]


# Rename columns to match the desired format
df.columns = months

# Save the cleaned data as CSV
df.to_csv('Cleaned MTHW Data.csv')

print("Cleaned data has been saved as 'Cleaned MTHW Data.csv'")
```

```
Cleaned data has been saved as 'Cleaned MTHW Data.csv'
```

In [19]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all numeric columns
df = df.apply(pd.to_numeric, errors='coerce')
```

In [23]:
```
# Calculate and display summary statistics rounded to 2 decimal places
summary_stats = df.describe().round(2)
print("Summary Statistics:")
display(summary_stats)
```

```
Summary Statistics:
```

	Jan-22	Feb-22	Mar-22	Apr-22	May-22	Jun-22	Jul-22	Au
count	11.00	11.00	11.00	11.00	11.00	11.00	11.00	1
mean	43673.64	42846.36	54698.18	67562.73	105083.64	141117.27	169173.64	1330
std	42489.95	40554.22	51442.10	60547.37	92197.74	127919.05	136519.12	1123
min	1110.00	1090.00	2800.00	3970.00	6770.00	10250.00	10630.00	84
25%	10075.00	10950.00	14625.00	19460.00	31865.00	45400.00	54100.00	376
50%	23000.00	34000.00	28100.00	48000.00	68000.00	110000.00	175000.00	1080
75%	73350.00	65390.00	84240.00	102310.00	170200.00	233150.00	261075.00	2055
max	127500.00	131600.00	157500.00	201800.00	304300.00	422100.00	451200.00	3721

8 rows × 34 columns

In [78]:
```
import pandas as pd

# Load the dataset
file_path = 'Cleaned MTHW Data.csv'  # Replace with your actual file path
df = pd.read_csv(file_path)

# Calculate summary statistics for all columns except 'Meter Location'
summary_stats = df.describe().round(2)

# Display the summary statistics
print("\n📊 Summary Statistics:")
print(summary_stats)
```

```
📊 Summary Statistics:
          Jan-22     Feb-22     Mar-22     Apr-22     May-22     Jun-22  \
count      11.00      11.00      11.00      11.00      11.00      11.00
mean    43673.64   42846.36   54698.18   67562.73  105083.64  141117.27
std     42489.95   40554.22   51442.10   60547.37   92197.74  127919.05
min      1110.00    1090.00    2800.00    3970.00    6770.00   10250.00
25%     10075.00   10950.00   14625.00   19460.00   31865.00   45400.00
50%     23000.00   34000.00   28100.00   48000.00   68000.00  110000.00
75%     73350.00   65390.00   84240.00  102310.00  170200.00  233150.00
max    127500.00  131600.00  157500.00  201800.00  304300.00  422100.00

          Jul-22     Aug-22     Sep-22     Oct-22  ...     Jan-24     Feb-24  \
count      11.00      11.00      11.00      11.00  ...      11.00      11.00
mean   169173.64  133025.45  128355.45  113613.64  ...   43028.18   49595.42
std    136519.12  112334.31  110148.51   96182.78  ...   39903.87   43811.50
min     10630.00    8400.00    7370.00    5780.00  ...     100.00       0.00
25%     54100.00   37650.00   35650.00   30700.00  ...   10650.00   13400.00
50%    175000.00  108000.00   83000.00   82000.00  ...   27000.00   36000.00
75%    261075.00  205535.00  211200.00  181685.00  ...   72980.00   86485.00
max    451200.00  372100.00  350600.00  304500.00  ...  112700.00  119299.60

          Mar-24     Apr-24     May-24     Jun-24     Jul-24     Aug-24  \
count      11.00      11.00      11.00      11.00      11.00      11.00
mean    78665.52   85393.66  131844.56  121910.80  143296.45  130459.95
std     64100.97   66943.20  101292.13  101958.69  105246.63  101895.80
min         0.00    5560.00   10690.00    9340.00   12140.00   10000.00
25%     23600.00   26550.00   40800.00   31700.00   56350.00   38600.00
50%     95450.00   98700.00  144400.00  120000.00  154940.00  130000.00
75%    120725.00  112955.00  194450.00  186439.98  202984.98  197920.01
max    189800.79  216200.20  332200.20  324798.81  348000.99  331899.41

          Sep-24     Oct-24
count      11.00      11.00
mean   106099.06   90899.23
std     83250.25   71278.60
min      7790.00    7320.00
25%     29750.00   24100.00
50%    118040.00  100620.00
75%    157820.01  143200.00
max    260099.60  224101.58

[8 rows x 34 columns]
```

In [94]:

```python
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('Cleaned MTHW Data.csv')

# Set 'Meter Location' as the index
df.set_index('Meter Location', inplace=True)

# Calculate summary statistics for each building
summary_stats = df.T.agg(['min', 'max', 'mean', 'std', lambda x: x.quantile(0.25

# Rename the columns for clarity
summary_stats.columns = df.index
summary_stats.index = ['Min', 'Max', 'Mean', 'Std Dev', '25%', '50%', '75%']

# Transpose the dataframe for better readability
```

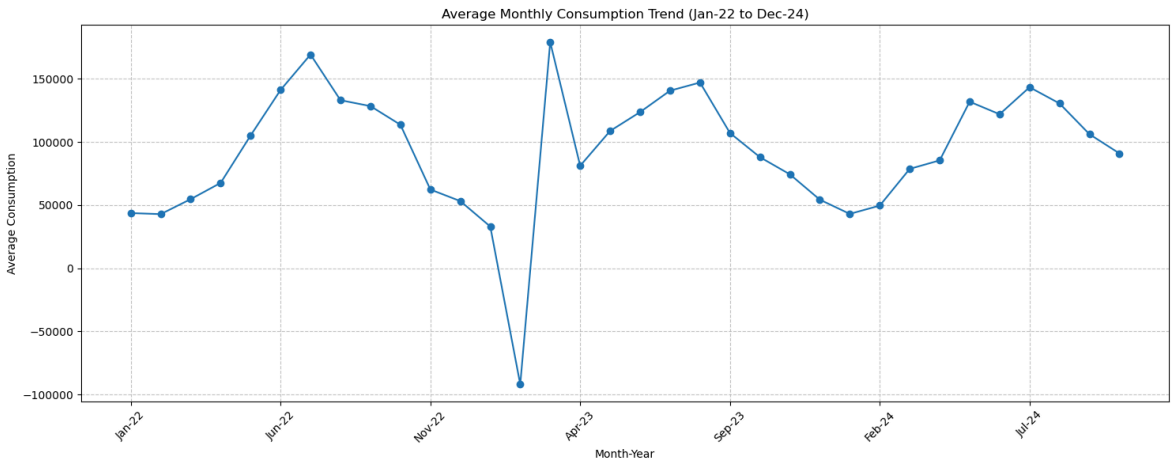
```
summary_stats = summary_stats.T

# Reset index to include Meter Location as a column
summary_stats.reset_index(inplace=True)

# Extract building code from Meter Location
summary_stats['Building Code'] = summary_stats['Meter Location'].str.extract(r'(

# Reorder columns
columns = ['Building Code', 'Min', 'Max', 'Mean', 'Std Dev', '25%', '50%', '75%'
summary_stats = summary_stats[columns]

# Display the results
print("Summary Statistics of MTHW Data")

print(summary_stats.to_string(index=False))
```

```
Summary Statistics of MTHW Data
Building Code        Min       Max          Mean       Std Dev           25%
50%         75%
         F503     3000.0   39300.0  18344.117647  10122.258437  12050.000000  177
00.000  24800.0000
         F505    33310.0  277480.0 153022.058824  71514.736255  84070.000000 1495
15.000 225375.0000
         F518     6900.0  103400.0  53141.176471  28391.720065  28925.000000  531
00.000  77475.0000
         G401    99400.0  451200.0 250332.387647 102827.064527 163075.048828 2333
00.790 332125.0025
         G404    40790.0  198140.0 108071.176471  40355.095113  76570.000000 1134
30.000 136820.0000
         G413    76460.0  263150.0 162261.469000  55764.165186 112822.500000 1680
80.029 214272.5050
         G501      260.0   12140.0   5237.941176   3360.088184   2165.000000   52
00.000   7357.5000
         G405    32900.0  259000.0 127532.352941  55235.033462  85600.000000 1135
50.000 175450.0000
         G411 -1350240.0 1366340.0  23002.941176 334907.880274   6875.000000  238
00.000  36650.0000
         H402     7900.0   65500.0  28322.058824  12478.089592  19050.000000  285
15.000  35000.0000
         F419    19800.0  177000.0  86823.529412  46975.095522  48750.000000  825
00.000 120750.0000
```

In [25]:

```
# Define the correct month order
months = [
    'Jan-22', 'Feb-22', 'Mar-22', 'Apr-22', 'May-22', 'Jun-22',
    'Jul-22', 'Aug-22', 'Sep-22', 'Oct-22', 'Nov-22', 'Dec-22',
    'Jan-23', 'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23',
    'Jul-23', 'Aug-23', 'Sep-23', 'Oct-23', 'Nov-23', 'Dec-23',
    'Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-24',
    'Jul-24', 'Aug-24', 'Sep-24', 'Oct-24', 'Nov-24', 'Dec-24'
]

# Create a new dataframe with reordered columns
df_ordered = pd.DataFrame(index=df.index)
for month in months:
    if month in df.columns:
        df_ordered[month] = df[month]

# Calculate monthly averages
```

```
monthly_avg = df_ordered.mean()

# Plot
plt.figure(figsize=(15, 6))
monthly_avg.plot(kind='line', marker='o')
plt.title('Average Monthly Consumption Trend (Jan-22 to Dec-24)')
plt.xlabel('Month-Year')
plt.ylabel('Average Consumption')
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

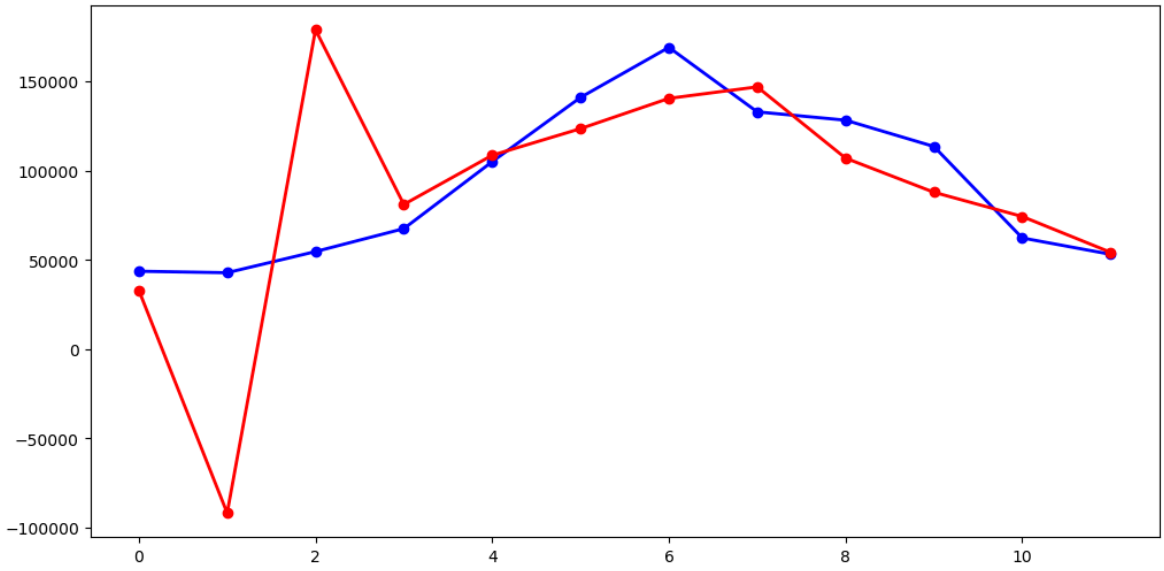


In [ ]:
```
monthly_avg
```

In [38]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Calculate monthly averages for each year
months_2022 = ['Jan-22', 'Feb-22', 'Mar-22', 'Apr-22', 'May-22', 'Jun-22',
               'Jul-22', 'Aug-22', 'Sep-22', 'Oct-22', 'Nov-22', 'Dec-22']
months_2023 = ['Jan-23', 'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23',
               'Jul-23', 'Aug-23', 'Sep-23', 'Oct-23', 'Nov-23', 'Dec-23']
months_2024 = ['Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-24',
               'Jul-24', 'Aug-24', 'Sep-24', 'Oct-24']

avg_2022 = df[months_2022].mean()
avg_2023 = df[months_2023].mean()
avg_2024 = df[months_2024].mean()

# Create the plot
plt.figure(figsize=(12, 6))

# Plot each year with different colors and markers
plt.plot(range(12), avg_2022, 'b-o', label='2022', linewidth=2)
plt.plot(range(12), avg_2023, 'r-o', label='2023', linewidth=2)
plt.plot(range(12), avg_2024, 'g-o', label='2024', linewidth=2)

# Customize the plot
```

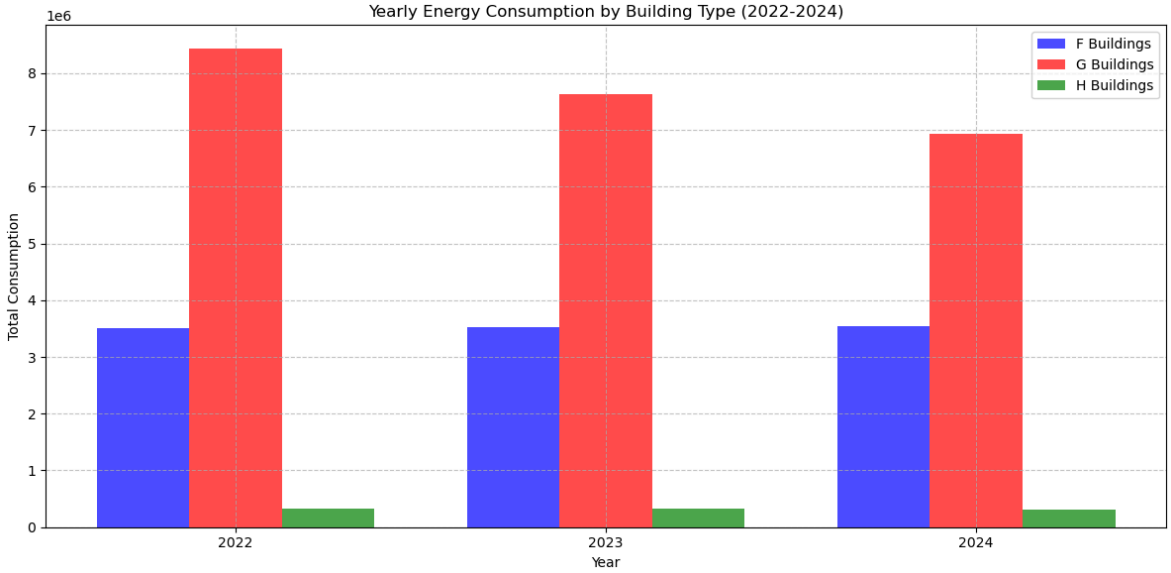
```
plt.title('Average Monthly Energy Consumption Comparison (2022-2024)')
plt.xlabel('Month')
plt.ylabel('Average Consumption')
plt.xticks(range(12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                       'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[38], line 28
     26 plt.plot(range(12), avg_2022, 'b-o', label='2022', linewidth=2)
     27 plt.plot(range(12), avg_2023, 'r-o', label='2023', linewidth=2)
---> 28 plt.plot(range(12), avg_2024, 'g-o', label='2024', linewidth=2)
     30 # Customize the plot
     31 plt.title('Average Monthly Energy Consumption Comparison (2022-2024)')

File ~\anaconda3\Lib\site-packages\matplotlib\pyplot.py:3794, in plot(scalex, sca
ley, data, *args, **kwargs)
   3786 @_copy_docstring_and_deprecators(Axes.plot)
   3787 def plot(
   3788     *args: float | ArrayLike | str,
   (...)
   3792     **kwargs,
   3793 ) -> list[Line2D]:
-> 3794     return gca().plot(
   3795         *args,
   3796         scalex=scalex,
   3797         scaley=scaley,
   3798         **({"data": data} if data is not None else {}),
   3799         **kwargs,
   3800     )

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_axes.py:1779, in Axes.plot(se
lf, scalex, scaley, data, *args, **kwargs)
   1536 """
   1537 Plot y versus x as lines and/or markers.
   1538
   (...)
   1776 (``'green'``) or hex strings (``'#008000'``).
   1777 """
   1778 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1779 lines = [*self._get_lines(self, *args, data=data, **kwargs)]
   1780 for line in lines:
   1781     self.add_line(line)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:296, in _process_plot
_var_args.__call__(self, axes, data, *args, **kwargs)
    294     this += args[0],
    295     args = args[1:]
--> 296 yield from self._plot_args(
    297     axes, this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey)

File ~\anaconda3\Lib\site-packages\matplotlib\axes\_base.py:486, in _process_plot
_var_args._plot_args(self, axes, tup, kwargs, return_kwargs, ambiguous_fmt_datake
y)
    483     axes.yaxis.update_units(y)
    485 if x.shape[0] != y.shape[0]:
--> 486     raise ValueError(f"x and y must have same first dimension, but "
    487                      f"have shapes {x.shape} and {y.shape}")
    488 if x.ndim > 2 or y.ndim > 2:
    489     raise ValueError(f"x and y can be no greater than 2D, but have "
    490                      f"shapes {x.shape} and {y.shape}")

ValueError: x and y must have same first dimension, but have shapes (12,) and (1
0,)
```

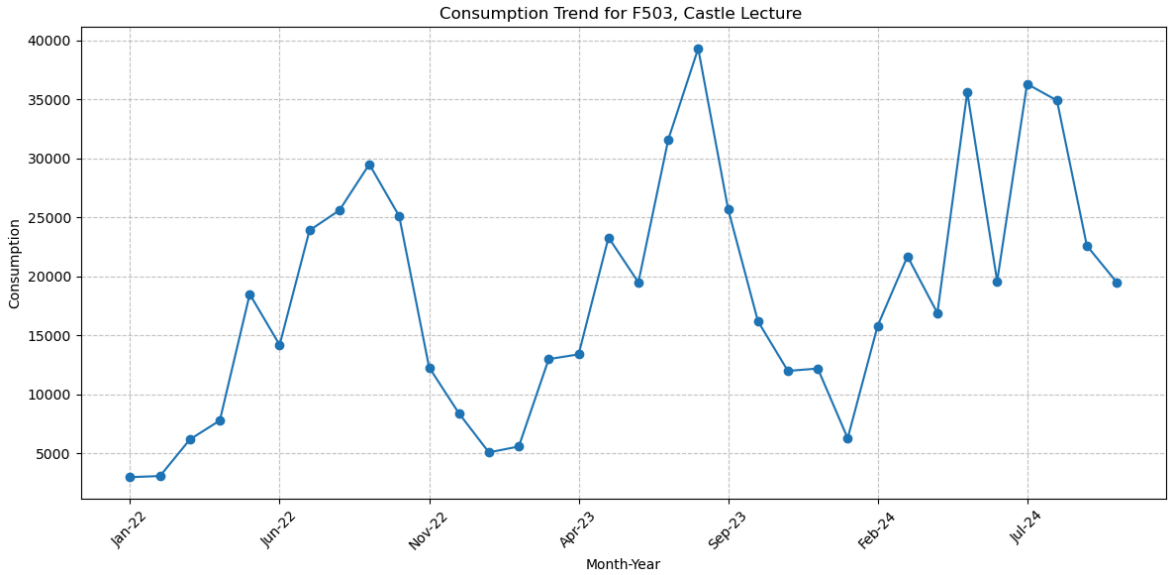


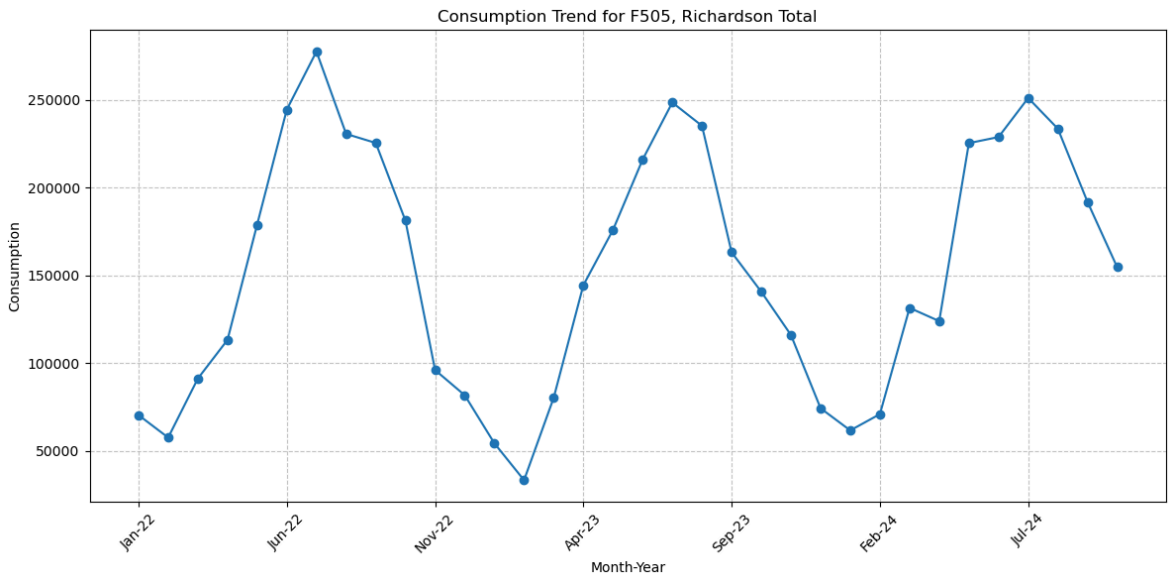
In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Define the months for each year
months_2022 = ['Jan-22', 'Feb-22', 'Mar-22', 'Apr-22', 'May-22', 'Jun-22',
               'Jul-22', 'Aug-22', 'Sep-22', 'Oct-22', 'Nov-22', 'Dec-22']
months_2023 = ['Jan-23', 'Feb-23', 'Mar-23', 'Apr-23', 'May-23', 'Jun-23',
               'Jul-23', 'Aug-23', 'Sep-23', 'Oct-23', 'Nov-23', 'Dec-23']
months_2024 = ['Jan-24', 'Feb-24', 'Mar-24', 'Apr-24', 'May-24', 'Jun-24',
               'Jul-24', 'Aug-24', 'Sep-24', 'Oct-24', 'Nov-24', 'Dec-24']

# Calculate monthly averages for each year
avg_2022 = df[months_2022].mean()
avg_2023 = df[months_2023].mean()
avg_2024 = df[months_2024].mean()

# Set up the bar positions
months = np.arange(12)
width = 0.25

# Create the plot
plt.figure(figsize=(15, 6))

# Create bars for each year
plt.bar(months - width, avg_2022, width, label='2022', color='blue', alpha=0.7)
plt.bar(months, avg_2023, width, label='2023', color='red', alpha=0.7)
plt.bar(months + width, avg_2024, width, label='2024', color='green', alpha=0.7)

# Customize the plot
plt.title('Average Monthly Energy Consumption Comparison (2022-2024)')
plt.xlabel('Month')
plt.ylabel('Average Consumption')
plt.xticks(months, ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True, linestyle='--', alpha=0.3, axis='y')
plt.legend()
plt.tight_layout()
plt.show()
```

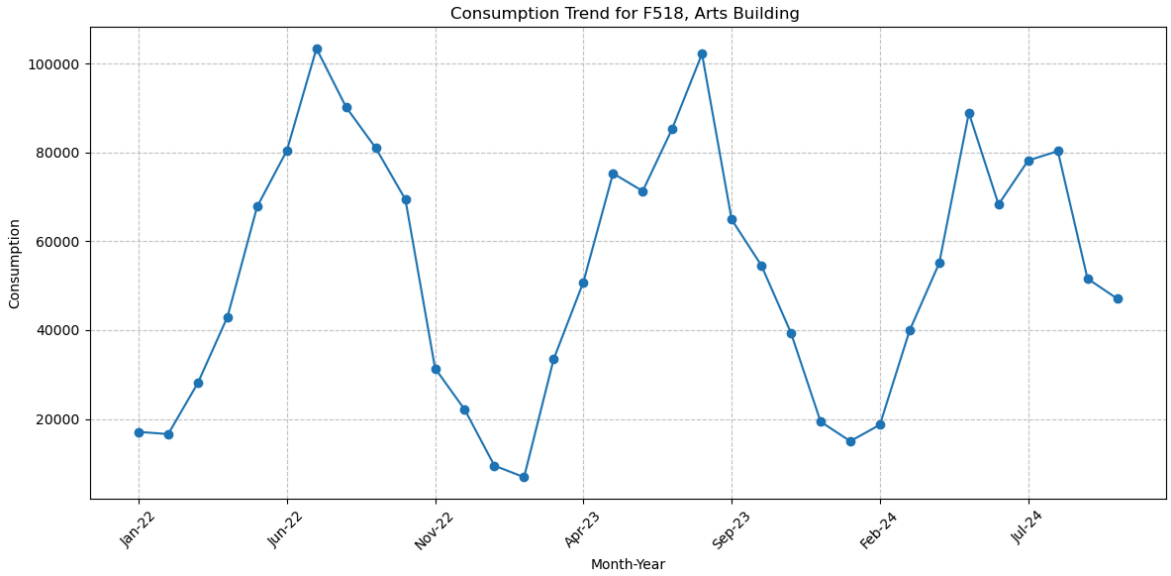
In [30]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Function to calculate yearly sum for building type
def get_yearly_sum(df, building_prefix, year):
    year_cols = [col for col in df.columns if f'-{year}' in col]
    building_rows = df[df.index.str.startswith(building_prefix)]
    return building_rows[year_cols].sum().sum()

# Calculate yearly sums for each building type
years = ['22', '23', '24']
f_sums = [get_yearly_sum(df, 'F', year) for year in years]
g_sums = [get_yearly_sum(df, 'G', year) for year in years]
h_sums = [get_yearly_sum(df, 'H', year) for year in years]

# Set up the bar positions
x = np.arange(3)
width = 0.25

# Create the plot
plt.figure(figsize=(12, 6))

# Create bars
plt.bar(x - width, f_sums, width, label='F Buildings', color='blue', alpha=0.7)
plt.bar(x, g_sums, width, label='G Buildings', color='red', alpha=0.7)
plt.bar(x + width, h_sums, width, label='H Buildings', color='green', alpha=0.7)

# Customize the plot
plt.title('Yearly Energy Consumption by Building Type (2022-2024)')
plt.xlabel('Year')
plt.ylabel('Total Consumption')
plt.xticks(x, ['2022', '2023', '2024'])
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

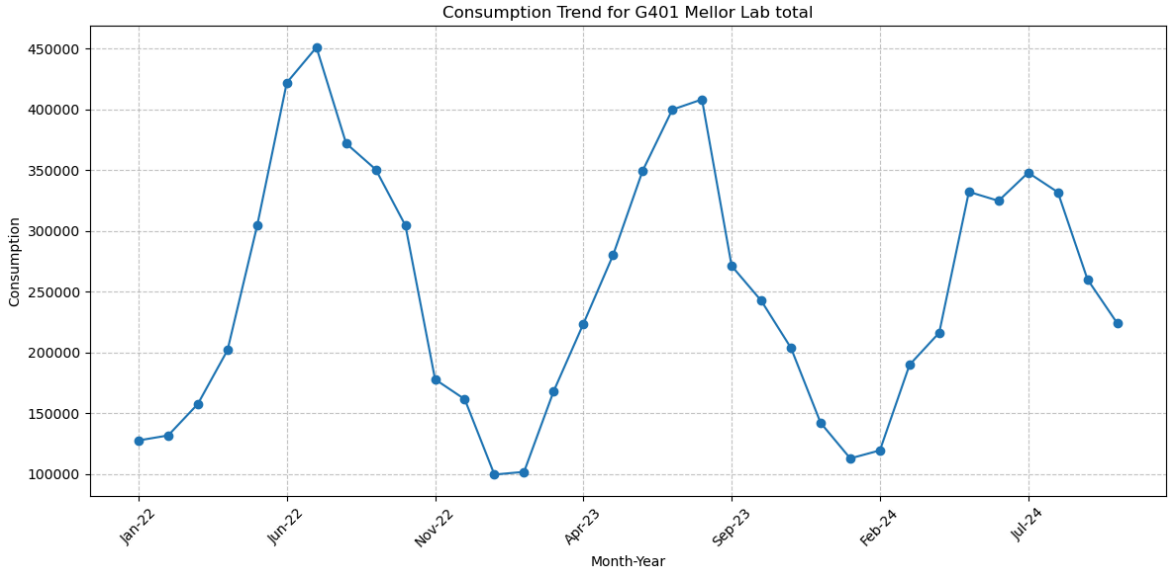


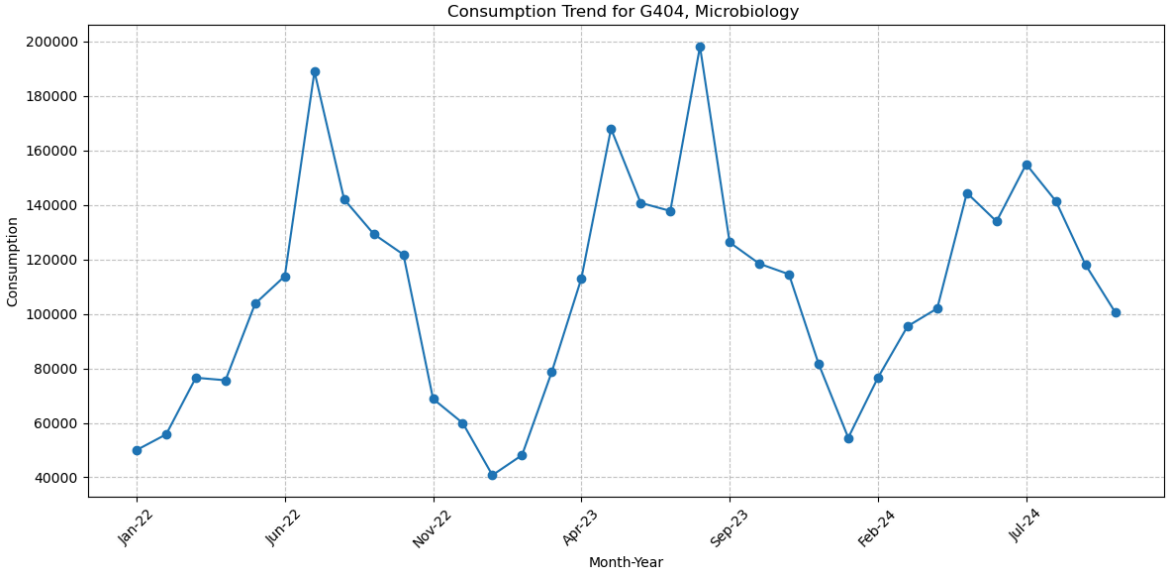
In [32]:
```
# Plot individual consumption trends
def plot_meter_location(location_name, data):
    plt.figure(figsize=(12, 6))
    data.loc[location_name].plot(kind='line', marker='o')
    plt.title(f'Consumption Trend for {location_name}')
    plt.xlabel('Month-Year')
    plt.ylabel('Consumption')
    plt.xticks(rotation=45)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()

# Plot for each location
for location in df.index:
    plot_meter_location(location, df)
```

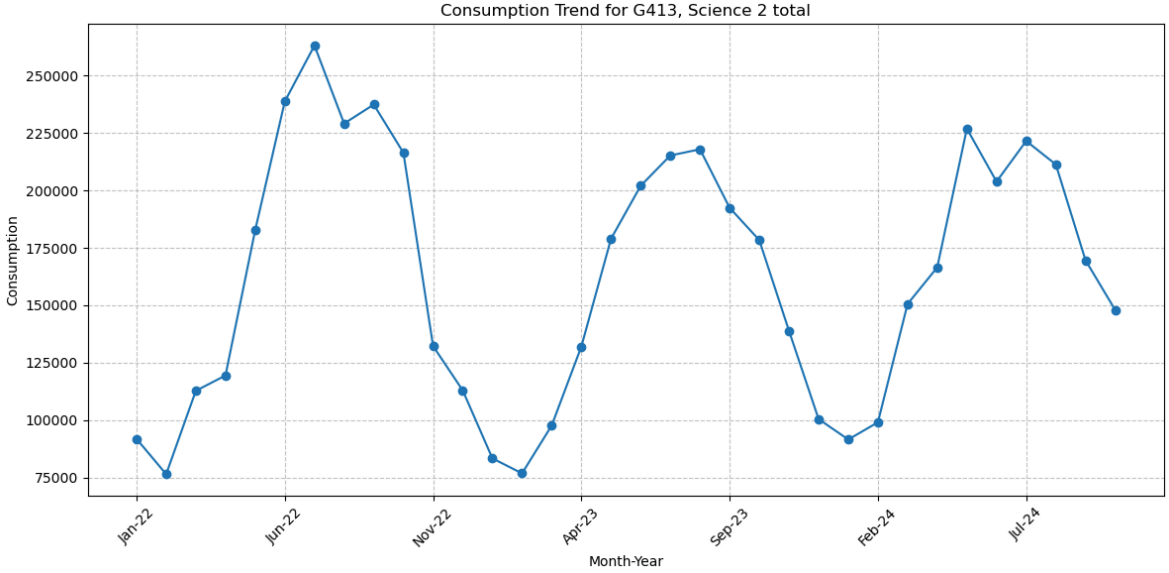


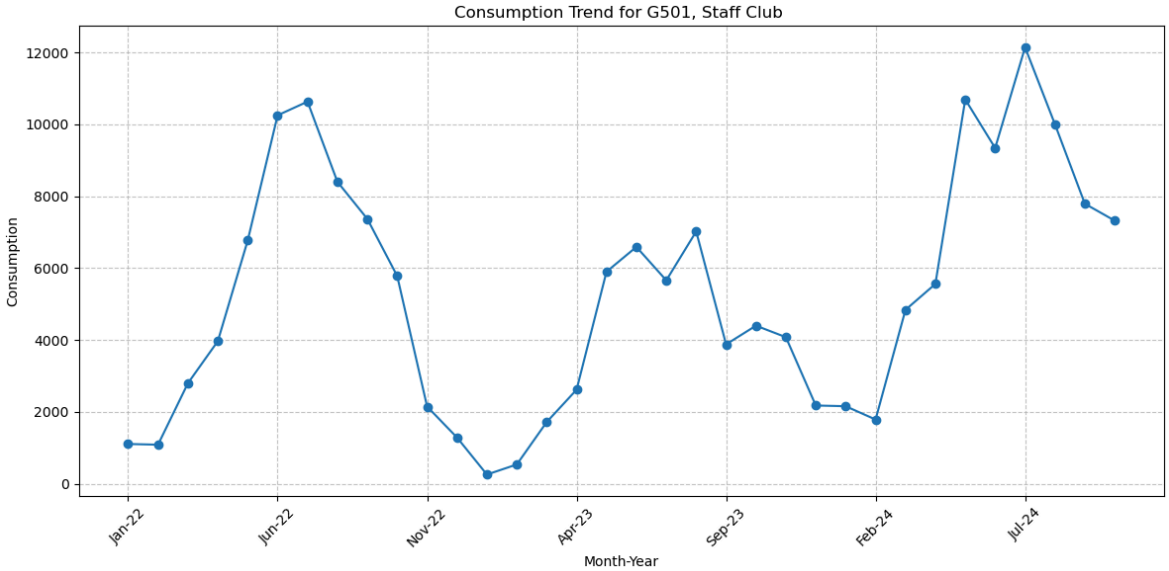

Consumption Trend for F505, Richardson Total
Consumption
250000
200000
150000
100000
50000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year

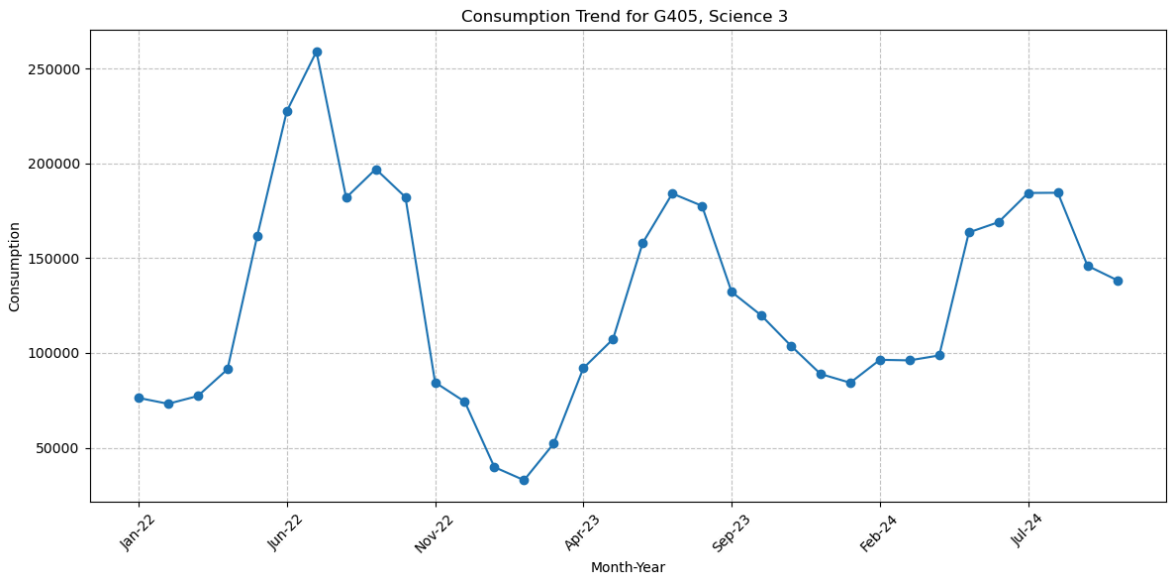

Consumption Trend for F518, Arts Building
Consumption
100000
80000
60000
40000
20000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year

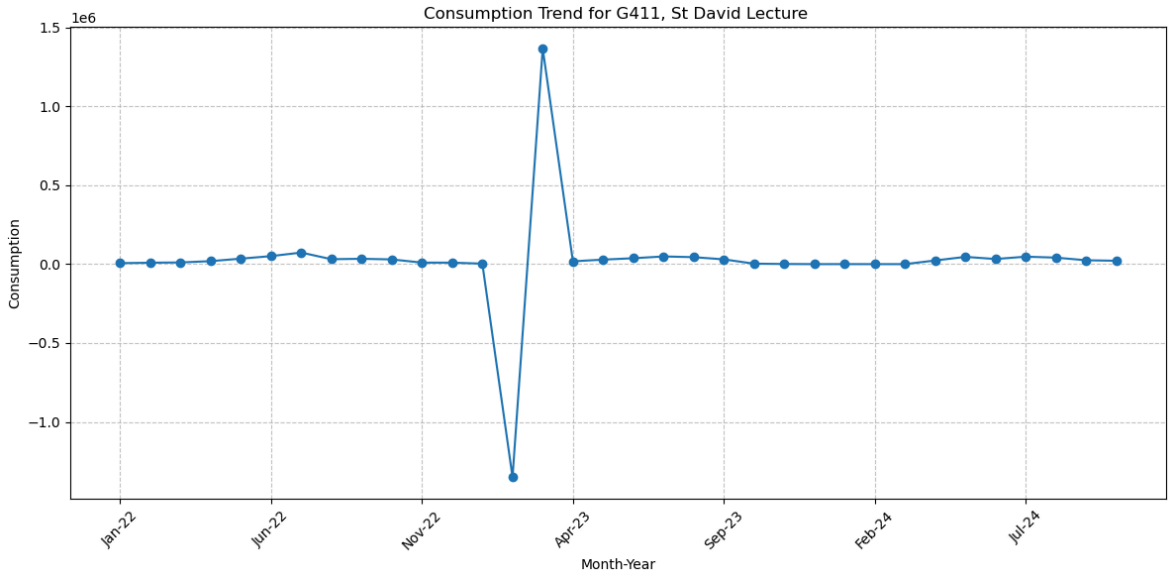

Consumption Trend for G401 Mellor Lab total
Consumption
450000
400000
350000
300000
250000
200000
150000
100000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year

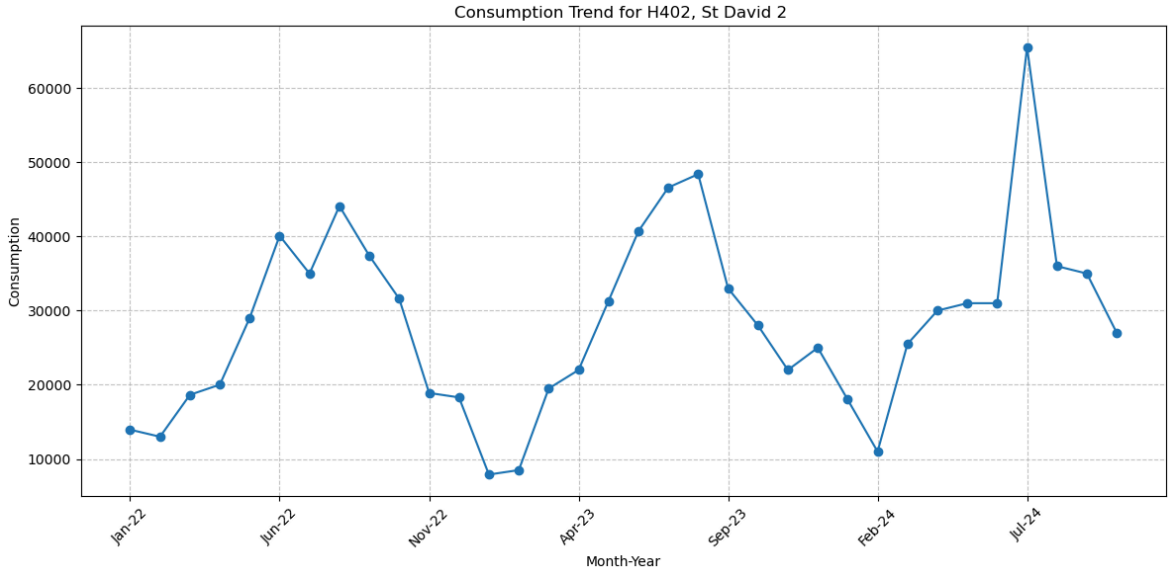

Consumption Trend for G404, Microbiology
Consumption
200000
180000
160000
140000
120000
100000
80000
60000
40000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year

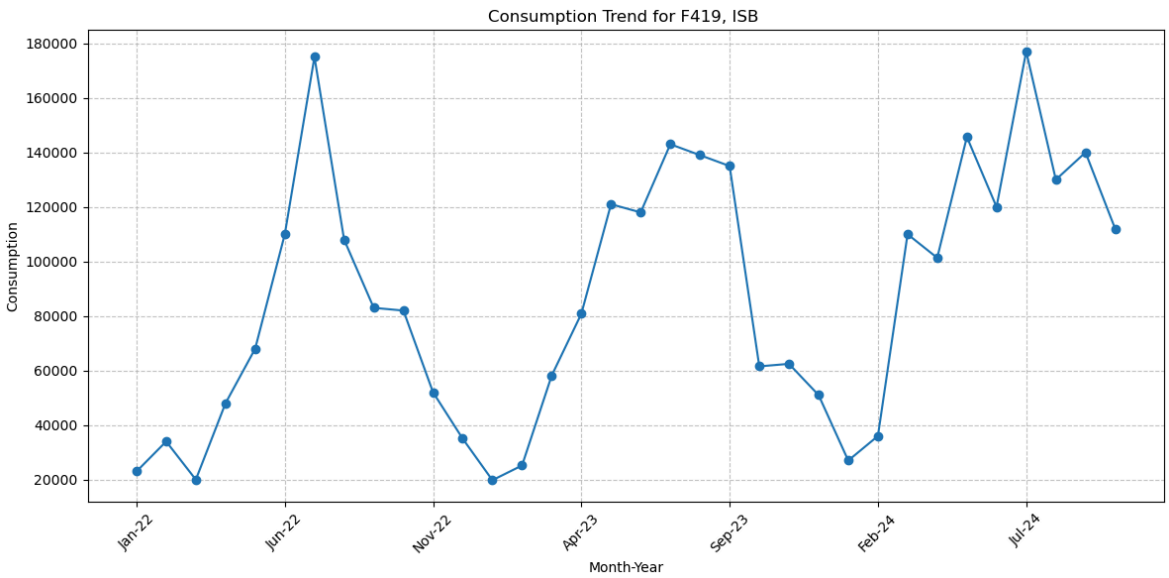

Consumption Trend for G413, Science 2 total
Consumption
250000
225000
200000
175000
150000
125000
100000
75000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year


Consumption Trend for G501, Staff Club
Consumption
12000
10000
8000
6000
4000
2000
0
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year

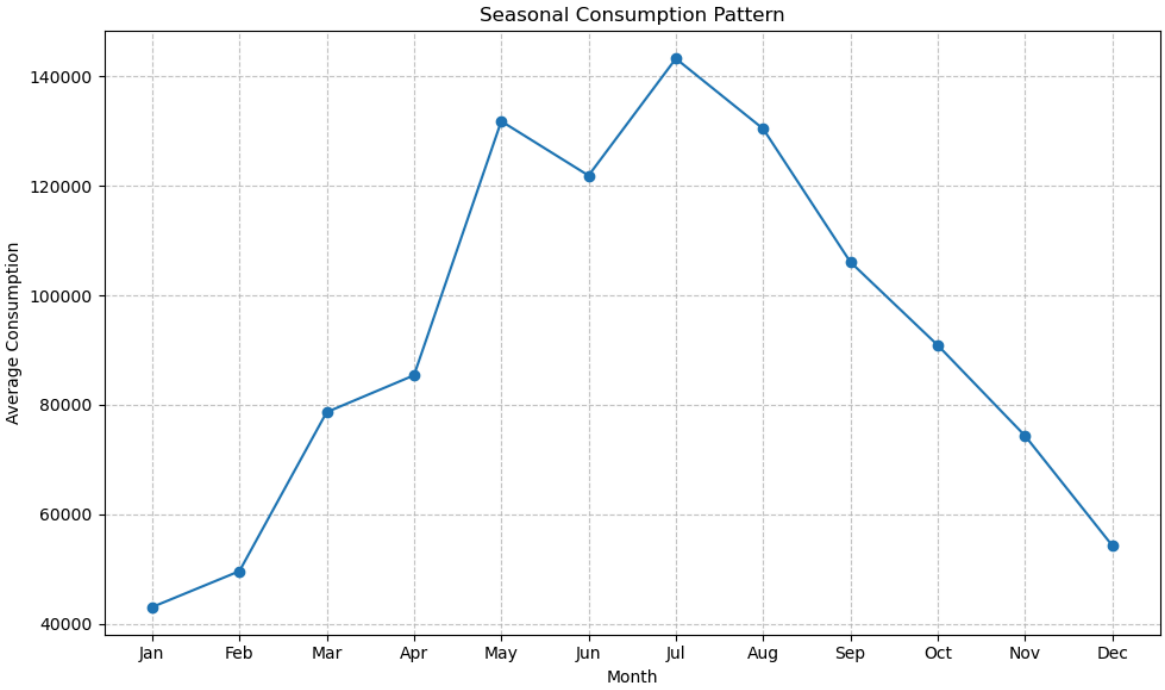

Consumption Trend for G405, Science 3
Consumption
250000
200000
150000
100000
50000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year


Consumption Trend for G411, St David Lecture
1e6
Consumption
1.5
1.0
0.5
0.0
−0.5
−1.0
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year

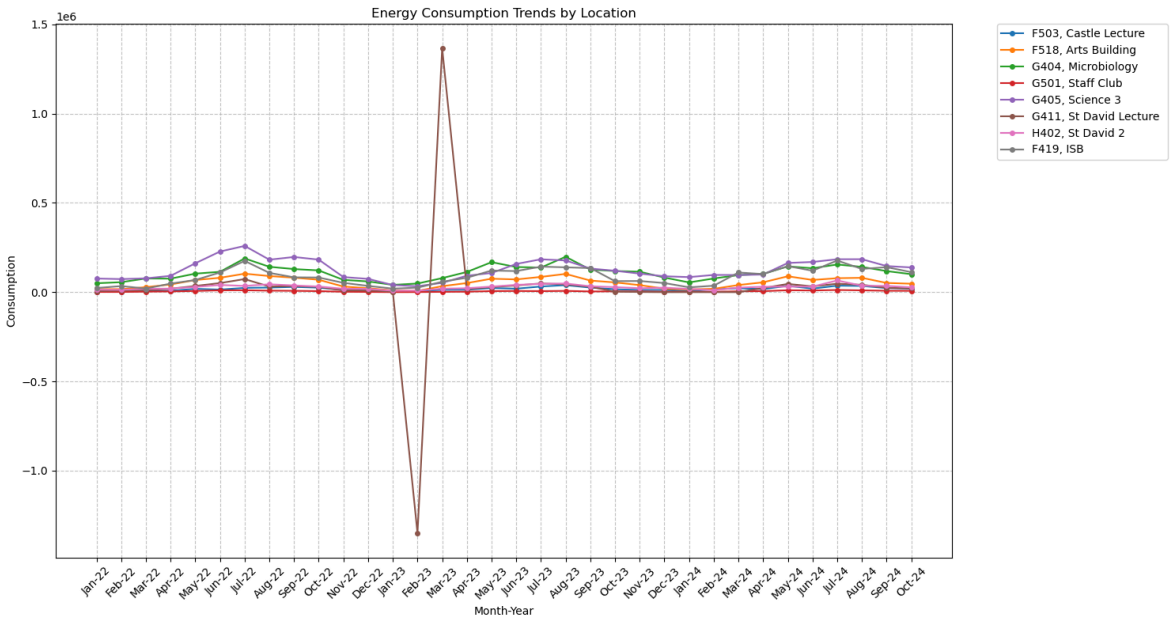

Consumption Trend for H402, St David 2
Consumption
60000
50000
40000
30000
20000
10000
Jan-22
Jun-22
Nov-22
Apr-23
Sep-23
Feb-24
Jul-24
Month-Year



In [34]:
```
# Create seasonal averages by month
def extract_month(col):
    return pd.to_datetime(col, format='%b-%y').month

monthly_pattern = pd.DataFrame()
for col in df.columns:
    if '-' in col:
        month = extract_month(col)
        monthly_pattern[month] = df[col]

seasonal_avg = monthly_pattern.mean()

# Plot seasonal pattern
plt.figure(figsize=(10, 6))
seasonal_avg.plot(kind='line', marker='o')
plt.title('Seasonal Consumption Pattern')
plt.xlabel('Month')
plt.ylabel('Average Consumption')
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.tight_layout()
plt.show()
```

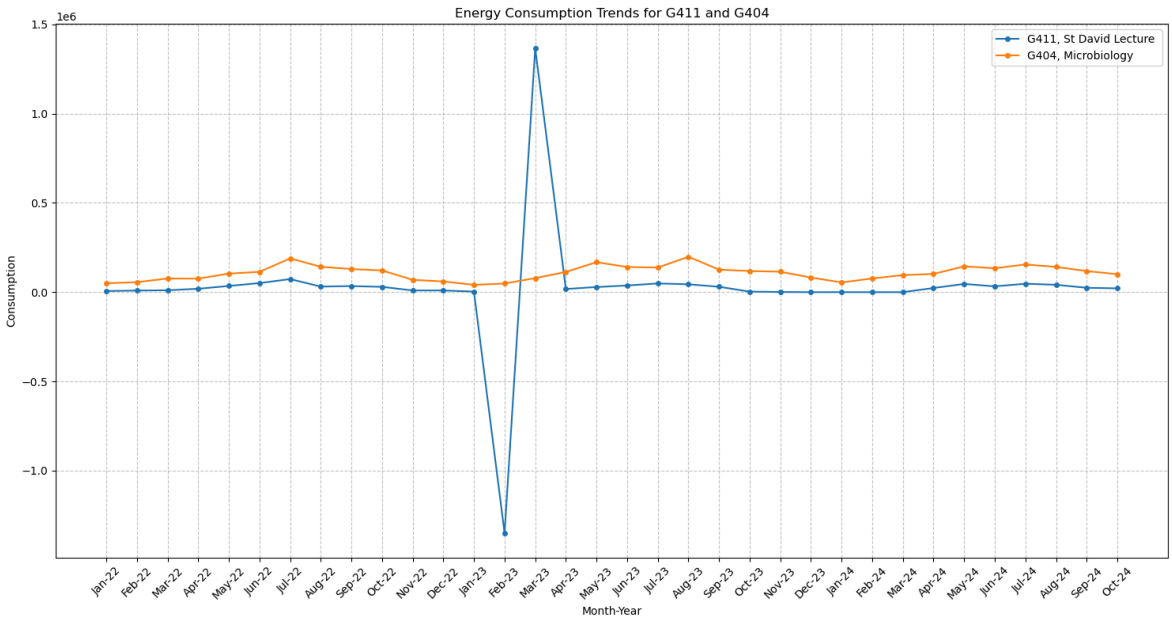


In [36]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all numeric columns
df = df.apply(pd.to_numeric, errors='coerce')

# Remove rows with all zeros or NaN
df = df[(df != 0).any(axis=1)]
df = df.dropna(how='all')

# Create the plot
plt.figure(figsize=(15, 8))

# Plot each location with a different color
for location in df.index:
    # Skip rows that contain 'total' or 'Steam' to make plot clearer
    if 'total' not in location.lower() and 'steam' not in location.lower():
        plt.plot(df.columns, df.loc[location], label=location, marker='o', marke

plt.title('Energy Consumption Trends by Location')
plt.xlabel('Month-Year')
plt.ylabel('Consumption')
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.tight_layout()
plt.show()
```



In [42]:

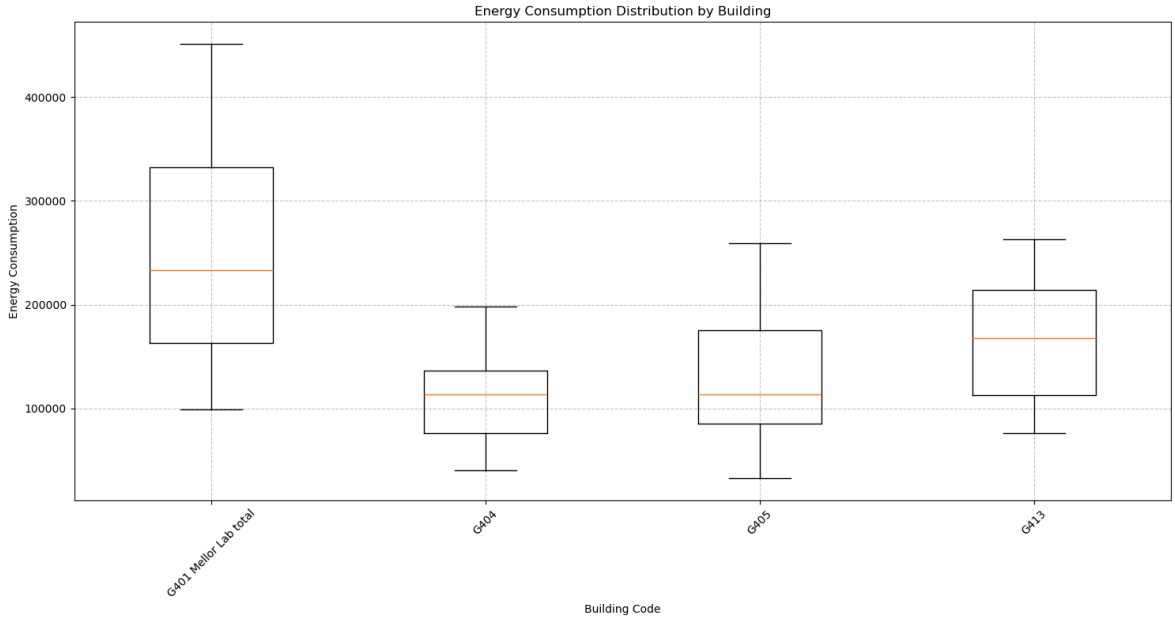
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all numeric columns
df = df.apply(pd.to_numeric, errors='coerce')

# Filter for G401 and G404
buildings_to_plot = ['G411, St David Lecture ','G404, Microbiology ']
df_filtered = df.loc[buildings_to_plot]

# Create the plot
plt.figure(figsize=(15, 8))

# Plot each location with a different color
for location in df_filtered.index:
    plt.plot(df_filtered.columns, df_filtered.loc[location], label=location, mar

plt.title('Energy Consumption Trends for G411 and G404')
plt.xlabel('Month-Year')
plt.ylabel('Consumption')
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



In [46]:

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

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric, replacing non-numeric values with NaN
df = df.apply(pd.to_numeric, errors='coerce')

# Select the buildings with high fluctuations
buildings = [
    'G401 Mellor Lab total',
    'G404, Microbiology ',
    'G405, Science 3 ',
    'G413, Science 2 total'
]

# Create a figure with larger size
plt.figure(figsize=(15, 8))

# Create the box plot
data_to_plot = [df.loc[building].dropna() for building in buildings]
plt.boxplot(data_to_plot, labels=[b.split(',')[0] for b in buildings])

# Customize the plot
plt.title('Energy Consumption Distribution by Building')
plt.xlabel('Building Code')
plt.ylabel('Energy Consumption')
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)

# Adjust layout to prevent label cutoff
plt.tight_layout()

plt.show()
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_13440\3603856193.py:24: MatplotlibDep
recationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_label
s' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  plt.boxplot(data_to_plot, labels=[b.split(',')[0] for b in buildings])
```

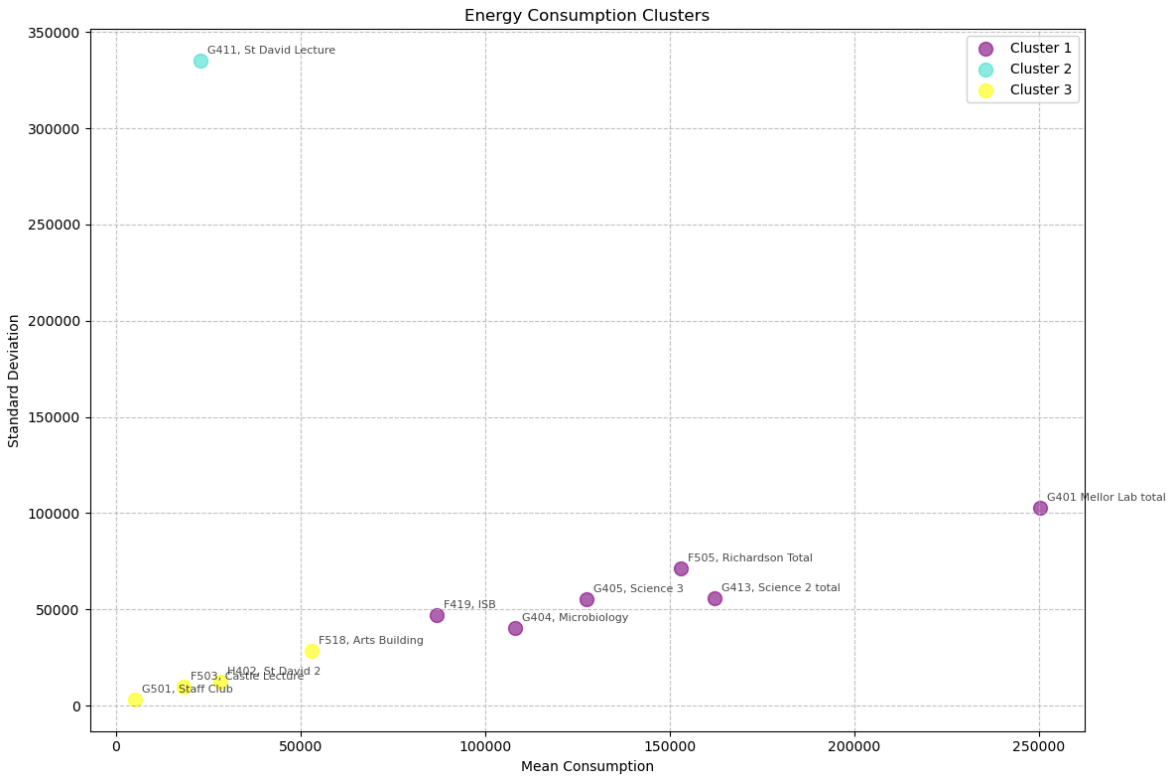


In [48]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all numeric columns and fill NaN with 0
df = df.apply(pd.to_numeric, errors='coerce').fillna(0)

# Remove rows where all values are 0
df = df[(df != 0).any(axis=1)]

# Calculate mean and standard deviation for each building
mean_consumption = df.mean(axis=1)
std_consumption = df.std(axis=1)

# Create feature matrix for clustering
X = np.column_stack((mean_consumption, std_consumption))

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform K-means clustering
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Create scatter plot
plt.figure(figsize=(12, 8))
colors = ['purple', 'turquoise', 'yellow']

# Plot each cluster
```

```
for i in range(n_clusters):
    mask = clusters == i
    plt.scatter(mean_consumption[mask], std_consumption[mask],
                c=colors[i], label=f'Cluster {i+1}', alpha=0.6, s=100)

# Add labels for each point
for i, txt in enumerate(df.index):
    plt.annotate(txt, (mean_consumption[i], std_consumption[i]),
                xytext=(5, 5), textcoords='offset points',
                fontsize=8, alpha=0.7)

plt.xlabel('Mean Consumption')
plt.ylabel('Standard Deviation')
plt.title('Energy Consumption Clusters')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Print buildings in each cluster
print("\nCluster Analysis Results:")
for i in range(n_clusters):
    print(f"\nCluster {i+1} ({colors[i]}):")
    cluster_buildings = df.index[clusters == i].tolist()
    for building in cluster_buildings:
        print(f"- {building}")
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_13440\1031647495.py:44: FutureWarnin
g: Series.__getitem__ treating keys as positions is deprecated. In a future versi
on, integer keys will always be treated as labels (consistent with DataFrame beha
vior). To access a value by position, use `ser.iloc[pos]`
  plt.annotate(txt, (mean_consumption[i], std_consumption[i]),
```



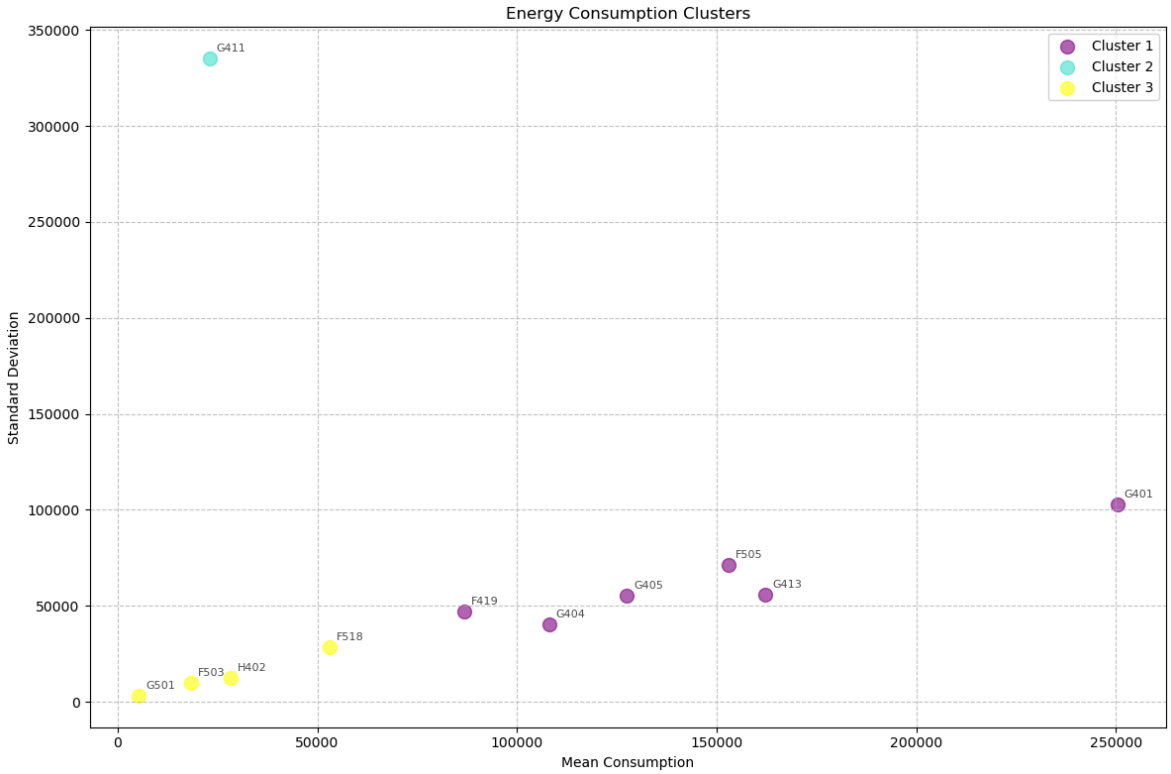
```
Cluster Analysis Results:

Cluster 1 (purple):
- F505, Richardson Total
- G401 Mellor Lab total
- G404, Microbiology
- G413, Science 2 total
- G405, Science 3
- F419, ISB

Cluster 2 (turquoise):
- G411, St David Lecture

Cluster 3 (yellow):
- F503, Castle Lecture
- F518, Arts Building
- G501, Staff Club
- H402, St David 2
```

In [50]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all numeric columns
numeric_df = df.select_dtypes(include=[np.number])

# Calculate mean and standard deviation for each building
mean_consumption = numeric_df.mean(axis=1)
std_consumption = numeric_df.std(axis=1)

# Create feature matrix for clustering
X = np.column_stack((mean_consumption, std_consumption))

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform K-means clustering
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# Create scatter plot
plt.figure(figsize=(12, 8))
colors = ['purple', 'turquoise', 'yellow']

# Plot each cluster
for i in range(n_clusters):
    mask = clusters == i
    plt.scatter(mean_consumption[mask], std_consumption[mask],
                c=colors[i], label=f'Cluster {i+1}', alpha=0.6, s=100)

# Add labels with only first 4 letters of building names
for i, txt in enumerate(df.index):
    # Extract first 4 letters, removing spaces and commas
```

```
    label = ''.join(c for c in txt if c.isalnum())[:4]
    plt.annotate(label, (mean_consumption[i], std_consumption[i]),
                 xytext=(5, 5), textcoords='offset points',
                 fontsize=8, alpha=0.7)

plt.xlabel('Mean Consumption')
plt.ylabel('Standard Deviation')
plt.title('Energy Consumption Clusters')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

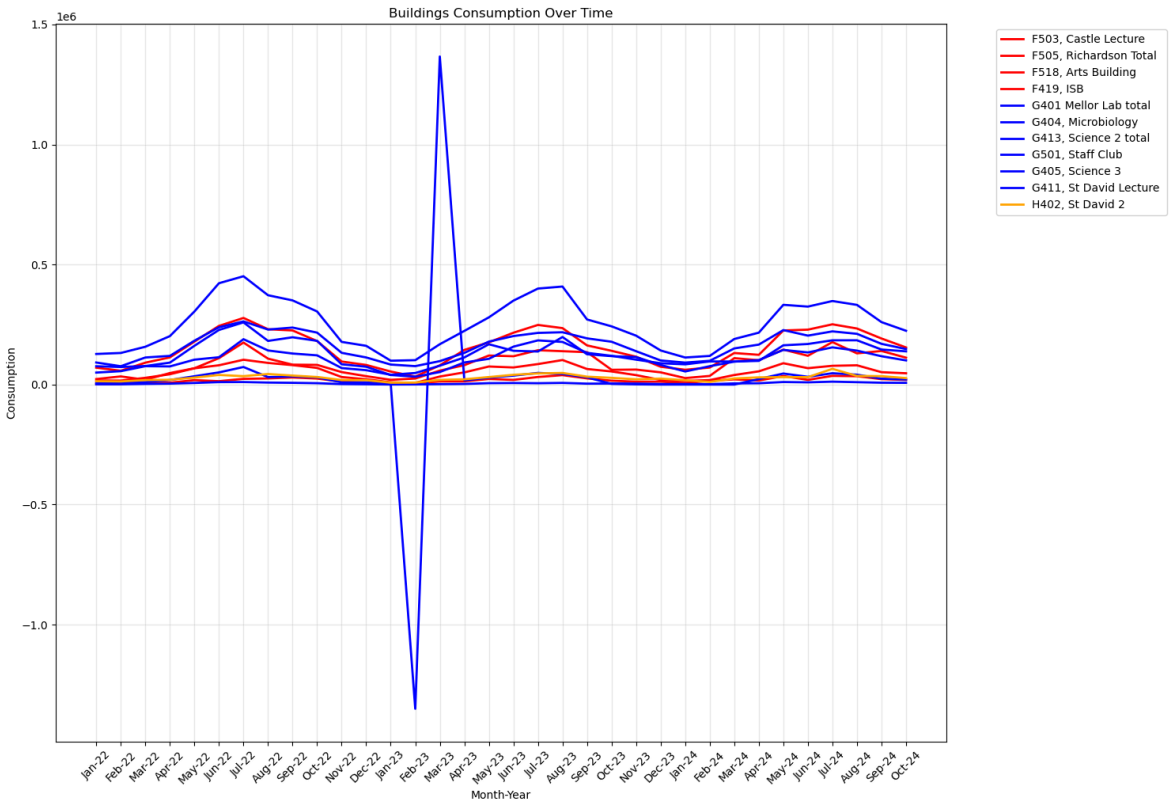
```
C:\Users\sugan\AppData\Local\Temp\ipykernel_13440\808600369.py:43: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  plt.annotate(label, (mean_consumption[i], std_consumption[i]),
```



In [52]:

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

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv')

# Set the 'Meter Location' as the index
df.set_index('Meter Location', inplace=True)

# Convert all columns except 'Meter Location' to numeric, coercing errors to NaN
for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Function to get the building code
def get_building_code(location):
    return location.split(',')[0][0]
```

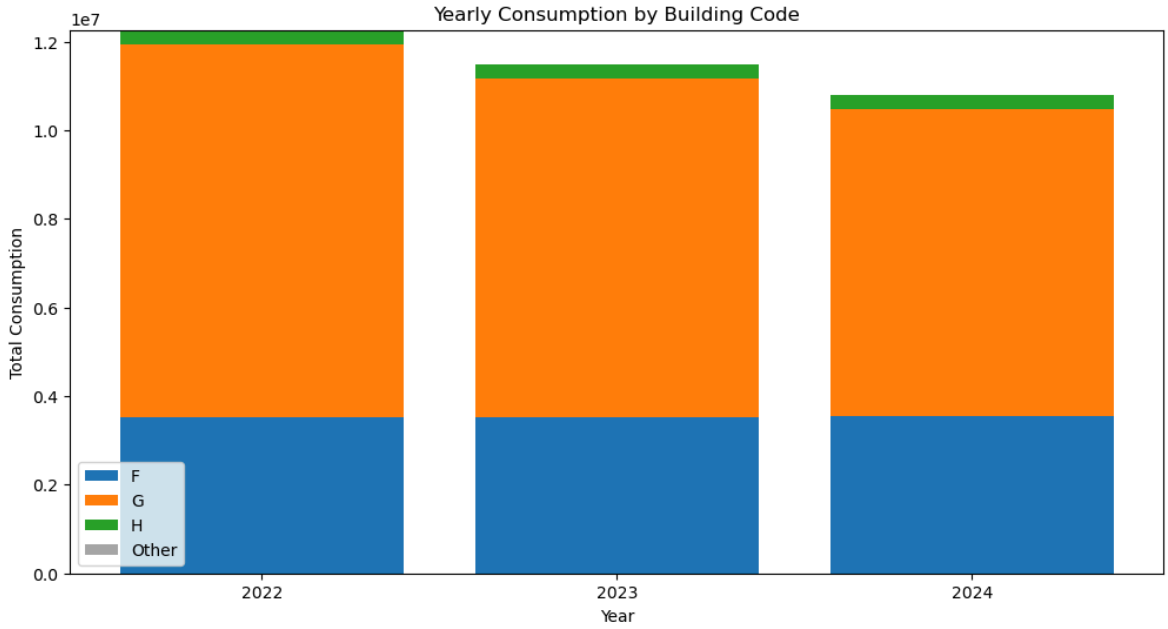
```
# Add a new column for building code
df['Building Code'] = df.index.map(get_building_code)

# Define color mapping for building codes - corrected syntax
color_map = {
    'F': 'red',
    'G': 'blue',
    'H': 'orange',
    'D': 'darkgrey',
    'E': 'darkgrey'
}

# Modified plotting function with color coding
def plot_individual_buildings(df):
    plt.figure(figsize=(15, 10))

    # First plot non-F,G,H buildings in dark grey
    other_buildings = df[~df['Building Code'].isin(['F', 'G', 'H'])]
    for building in other_buildings.index:
        data = other_buildings.loc[building, other_buildings.columns != 'Buildin
        plt.plot(data, color='darkgrey', alpha=0.5, linewidth=1, label='_nolegen

    # Then plot F, G, H buildings with their specific colors
    for code in ['F', 'G', 'H']:
        buildings = df[df['Building Code'] == code]
        for building in buildings.index:
            data = buildings.loc[building, buildings.columns != 'Building Code']
            plt.plot(data, color=color_map[code], label=building, linewidth=2)

    plt.title('Buildings Consumption Over Time')
    plt.xlabel('Month-Year')
    plt.ylabel('Consumption')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.xticks(rotation=45)
    plt.grid(True, alpha=0.3)
    plt.show()

# Create single plot with all buildings
plot_individual_buildings(df)
```



In [54]:

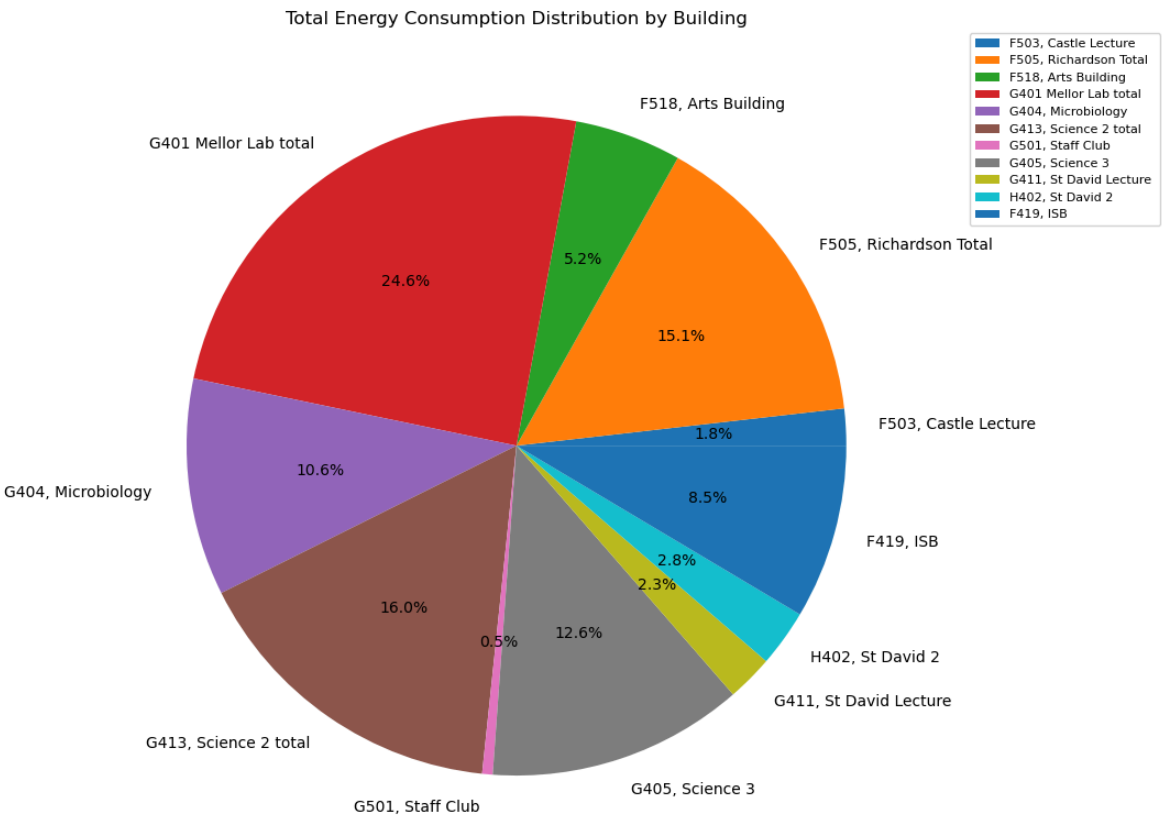
```
# 2) Sum of F, G, H, and rest of the buildings for each year
def sum_by_building_code(df):
    # Create a copy of the dataframe without the 'Building Code' column
    df_numeric = df.drop('Building Code', axis=1)

    # Group by Building Code and sum
    yearly_sums = df.groupby('Building Code')[df_numeric.columns].sum()

    # Create year groups (assuming column names are in 'MMM-YY' format)
    year_groups = {}
    for col in df_numeric.columns:
        year = '20' + col.split('-')[1]
        if year not in year_groups:
            year_groups[year] = []
        year_groups[year].append(col)

    # Sum by year
    yearly_data = {}
    for year, cols in year_groups.items():
        yearly_data[year] = yearly_sums[cols].sum(axis=1)

    return pd.DataFrame(yearly_data)

yearly_sums = sum_by_building_code(df)

# Plot stacked bar chart for yearly comparison
plt.figure(figsize=(12, 6))
bottom = np.zeros(len(yearly_sums.columns))

for code in ['F', 'G', 'H']:
    if code in yearly_sums.index:
        plt.bar(yearly_sums.columns, yearly_sums.loc[code], bottom=bottom, label
        bottom += yearly_sums.loc[code]

other_sum = yearly_sums.drop(['F', 'G', 'H'], errors='ignore').sum()
```

```
plt.bar(yearly_sums.columns, other_sum, bottom=bottom, label='Other', color='dar

plt.title('Yearly Consumption by Building Code')
plt.xlabel('Year')
plt.ylabel('Total Consumption')
plt.legend()
plt.show()
```



In [56]:

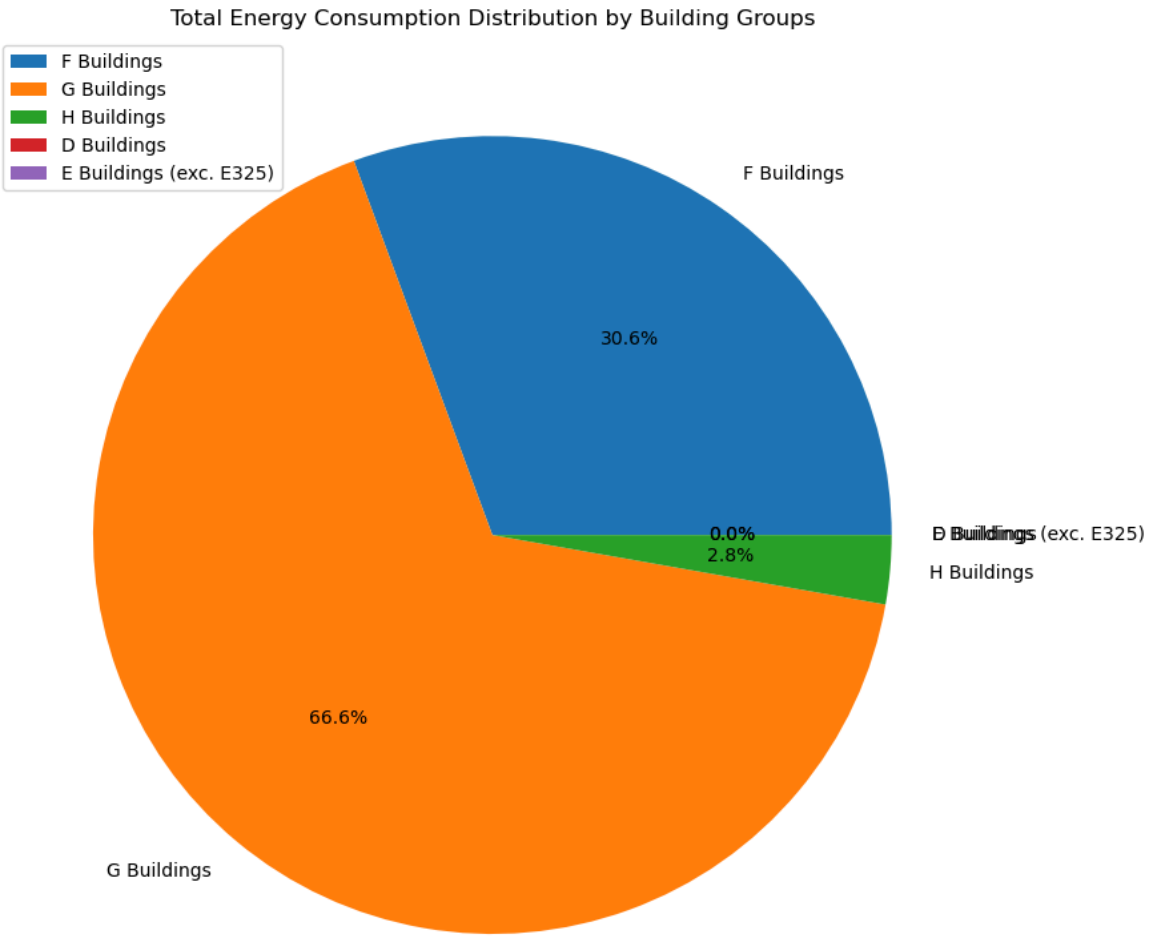
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Calculate total consumption for each building
total_consumption = df.sum(axis=1)

# Remove buildings with zero or negligible consumption
total_consumption = total_consumption[total_consumption > 1000]

# Create a pie chart
plt.figure(figsize=(12, 8))
plt.pie(total_consumption, labels=total_consumption.index, autopct='%1.1f%%')
plt.title('Total Energy Consumption Distribution by Building')

# Add legend
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0., fontsiz

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



In [58]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Calculate total consumption for each building
total_consumption = df.sum(axis=1)

# Create groups based on building codes
F_buildings = total_consumption[total_consumption.index.str.startswith('F')].sum
G_buildings = total_consumption[total_consumption.index.str.startswith('G')].sum
H_buildings = total_consumption[total_consumption.index.str.startswith('H')].sum
D_buildings = total_consumption[total_consumption.index.str.startswith('D')].sum
E_buildings = total_consumption[(total_consumption.index.str.startswith('E')) &
                                (~total_consumption.index.str.contains('E325'))].s

# Create data for pie chart
building_groups = {
    'F Buildings': F_buildings,
    'G Buildings': G_buildings,
    'H Buildings': H_buildings,
    'D Buildings': D_buildings,
    'E Buildings (exc. E325)': E_buildings
}

# Create pie chart
plt.figure(figsize=(10, 8))
plt.pie(building_groups.values(), labels=building_groups.keys(), autopct='%1.1f%
plt.title('Total Energy Consumption Distribution by Building Groups')
```

```
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

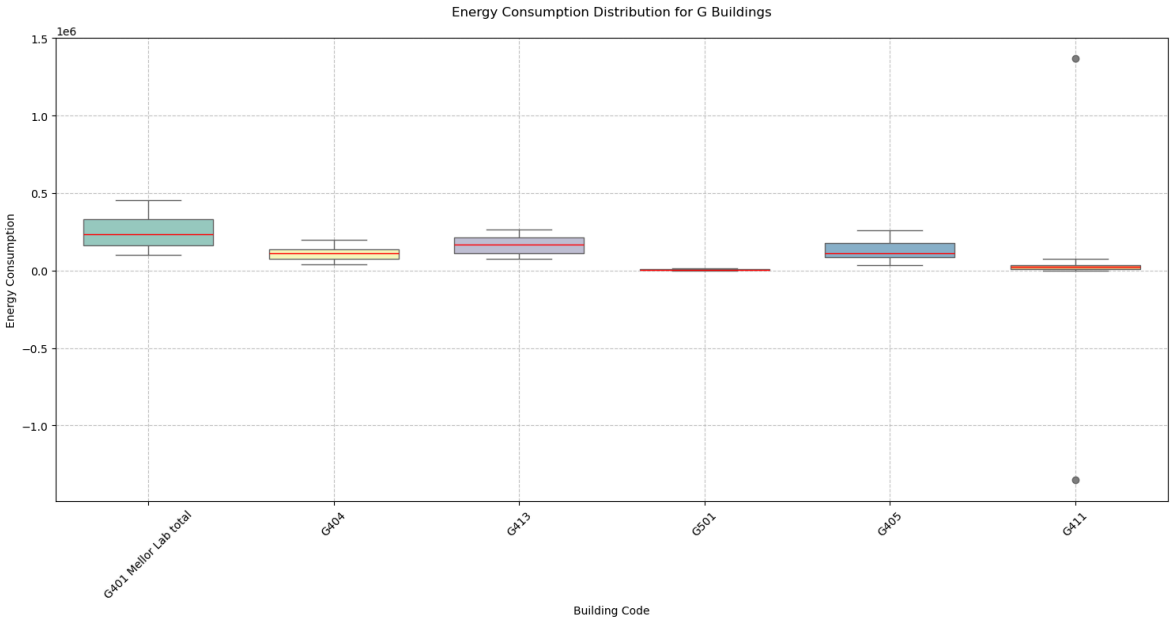


In [60]:

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

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Filter only G buildings
g_buildings = df[df.index.str.startswith('G')]

# Reshape the data for seaborn
data_list = []
for building in g_buildings.index:
    building_data = g_buildings.loc[building].dropna()
    data_list.extend([(building.split(',')[0], value) for value in building_data

df_plot = pd.DataFrame(data_list, columns=['Building', 'Consumption'])

# Create figure with larger size
plt.figure(figsize=(15, 8))
```

```
# Create box plot using seaborn
sns.boxplot(data=df_plot, x='Building', y='Consumption',
            palette='Set3',  # Use a colorful palette
            width=0.7,
            medianprops={"color": "red"},  # Make median line red
            flierprops={"marker": "o", "markerfacecolor": "gray"})  # Style outl

# Customize the plot
plt.title('Energy Consumption Distribution for G Buildings', pad=20)
plt.xlabel('Building Code')
plt.ylabel('Energy Consumption')
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)

# Adjust layout to prevent label cutoff
plt.tight_layout()
plt.show()
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_13440\639224237.py:26: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(data=df_plot, x='Building', y='Consumption',
```

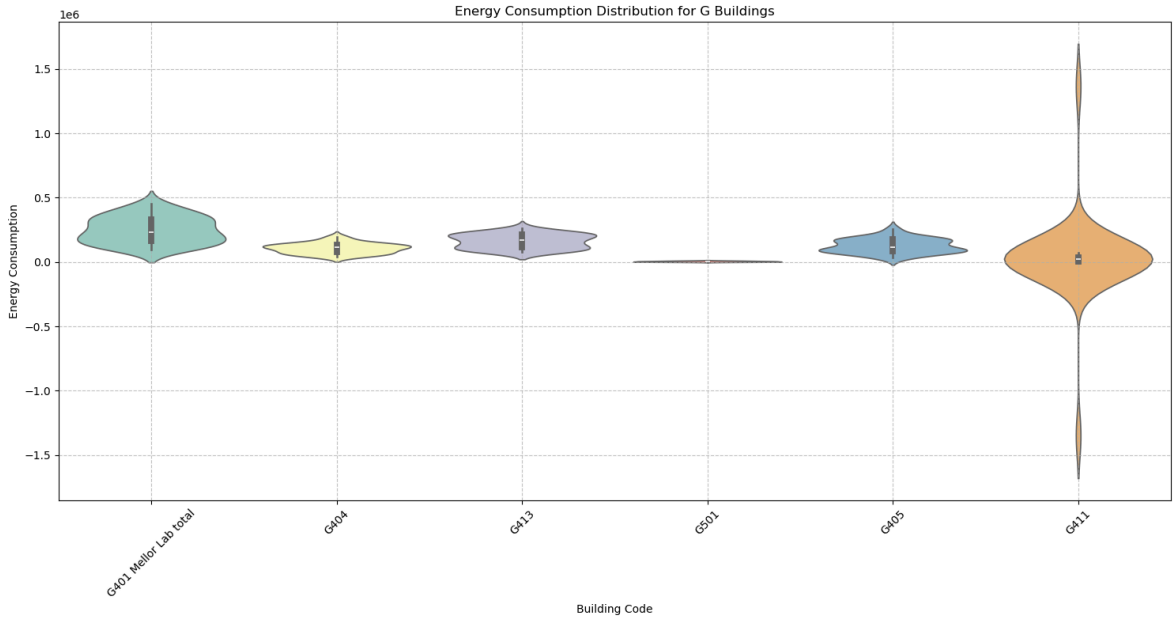


In [62]:

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

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Filter only G buildings
g_buildings = df[df.index.str.startswith('G')]

# Reshape the data for seaborn
data_list = []
```

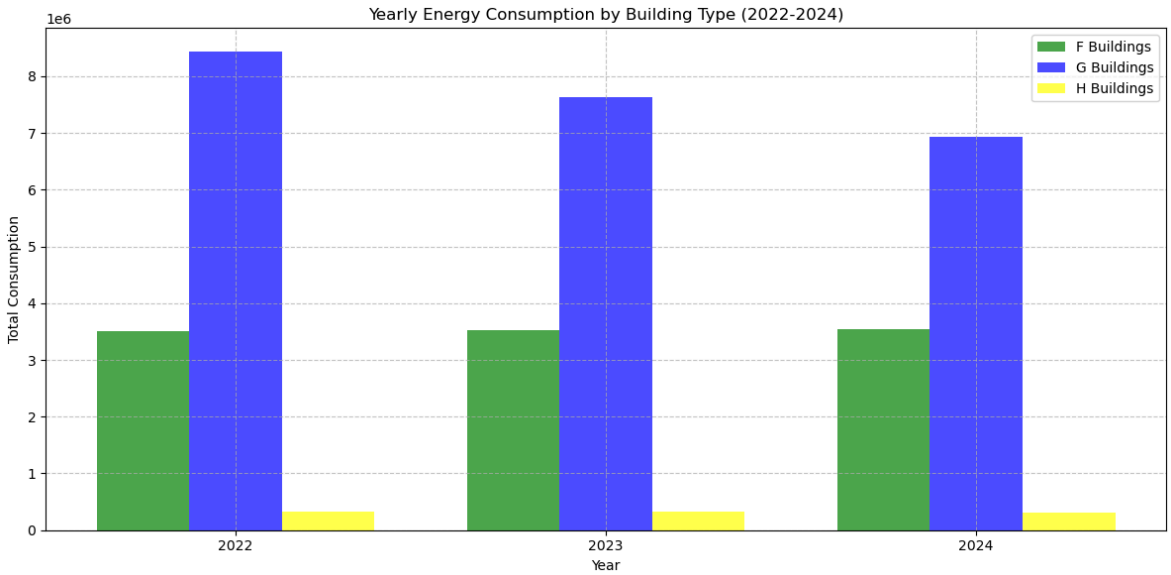
```
for building in g_buildings.index:
    for column in g_buildings.columns:
        value = g_buildings.loc[building, column]
        if pd.notna(value):  # Only include non-NA values
            data_list.append({
                'Building': building.split(',')[0],
                'Consumption': value
            })

df_plot = pd.DataFrame(data_list)

# Create figure with larger size
plt.figure(figsize=(15, 8))

# Create violin plot using seaborn
sns.violinplot(data=df_plot, x='Building', y='Consumption',
               palette='Set3',
               inner='box')

# Customize the plot
plt.title('Energy Consumption Distribution for G Buildings')
plt.xlabel('Building Code')
plt.ylabel('Energy Consumption')
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.7)

# Adjust layout to prevent label cutoff
plt.tight_layout()
plt.show()
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_13440\2032909002.py:31: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.violinplot(data=df_plot, x='Building', y='Consumption',
```



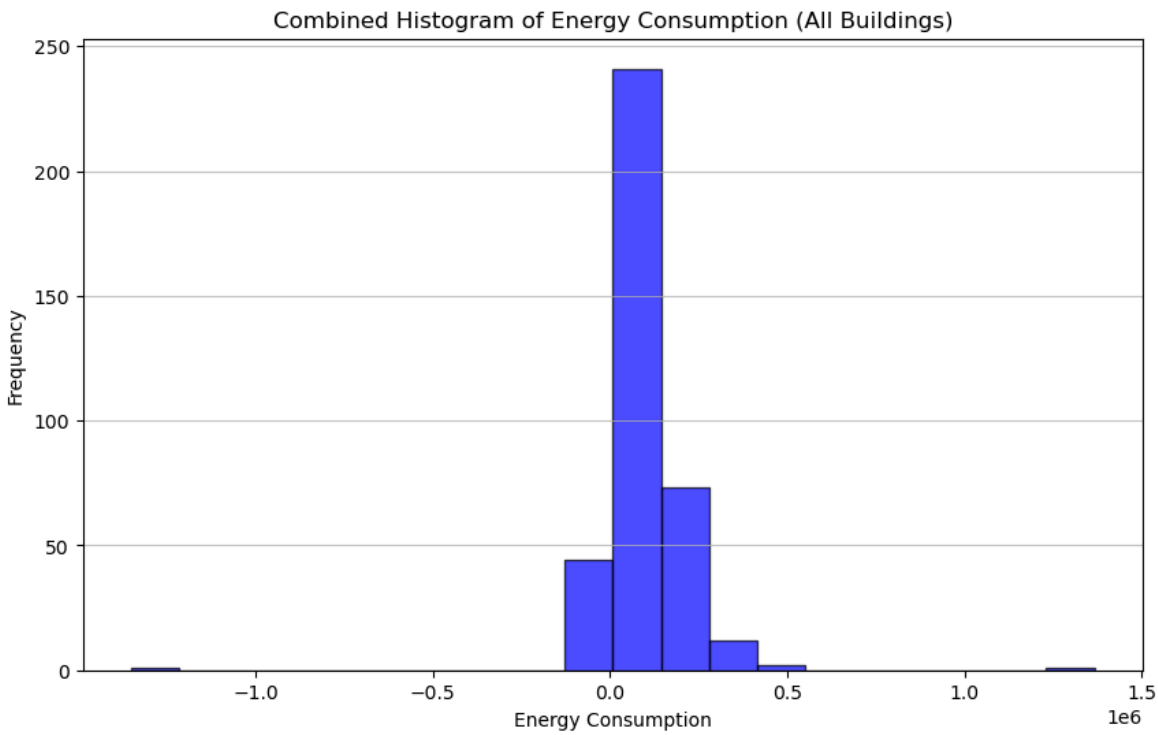
In [64]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import numpy as np

# Load the data
df = pd.read_csv('Cleaned MTHW Data.csv', index_col='Meter Location')

# Convert all columns to numeric
df = df.apply(pd.to_numeric, errors='coerce')

# Function to calculate yearly sum for building type
def get_yearly_sum(df, building_prefix, year):
    year_cols = [col for col in df.columns if f'-{year}' in col]
    building_rows = df[df.index.str.startswith(building_prefix)]
    return building_rows[year_cols].sum().sum()

# Calculate yearly sums for each building type
years = ['22', '23', '24']
building_types = ['F', 'G', 'H']
colors = ['green', 'blue', 'yellow']

# Set up the bar positions
x = np.arange(3)
width = 0.25

# Create the plot
plt.figure(figsize=(12, 6))

# Create bars for each building type with specified colors
for i, (building, color) in enumerate(zip(building_types, colors)):
    yearly_sums = [get_yearly_sum(df, building, year) for year in years]
    plt.bar(x + (i-1)*width, yearly_sums, width, label=f'{building} Buildings',

# Customize the plot
plt.title('Yearly Energy Consumption by Building Type (2022-2024)')
plt.xlabel('Year')
plt.ylabel('Total Consumption')
plt.xticks(x, ['2022', '2023', '2024'])
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



In [155…

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Cleaned MTHW Data.csv')

# Combine all energy consumption data into a single series
combined_data = df.set_index('Meter Location').T.stack()

# Plot the combined histogram
plt.figure(figsize=(10, 6))
plt.hist(combined_data, bins=20, color='blue', alpha=0.7, edgecolor='black')
plt.title('Combined Histogram of Energy Consumption (All Buildings)')
plt.xlabel('Energy Consumption')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.show()
```



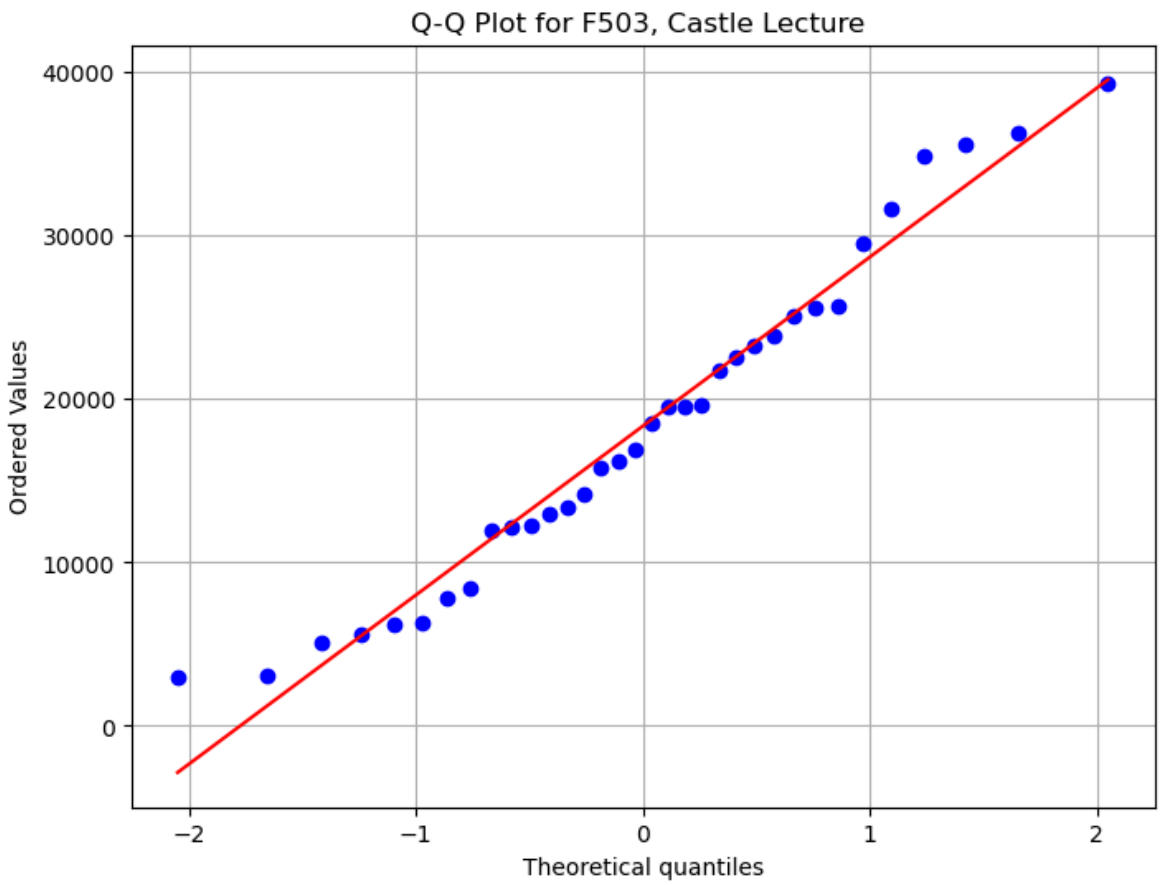
In [157…

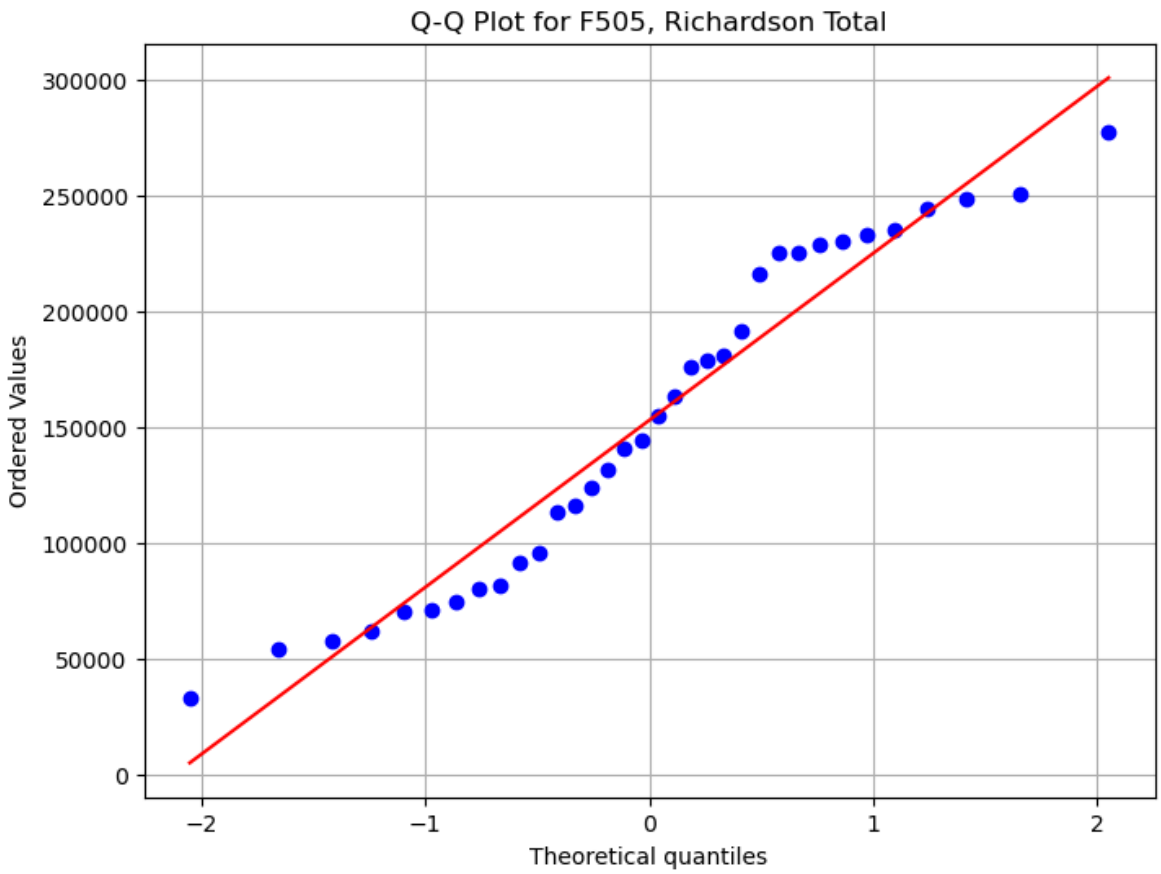
```
from scipy.stats import skew, kurtosis

# Calculate skewness and kurtosis for each building
skewness_kurtosis = []
for building in building_data.columns:
    data = building_data[building].dropna()
    skewness = skew(data)
    kurt = kurtosis(data)
    skewness_kurtosis.append({'Building': building, 'Skewness': skewness, 'Kurto

# Convert results to a DataFrame
skewness_kurtosis_df = pd.DataFrame(skewness_kurtosis)

# Display the results
print(skewness_kurtosis_df)
```

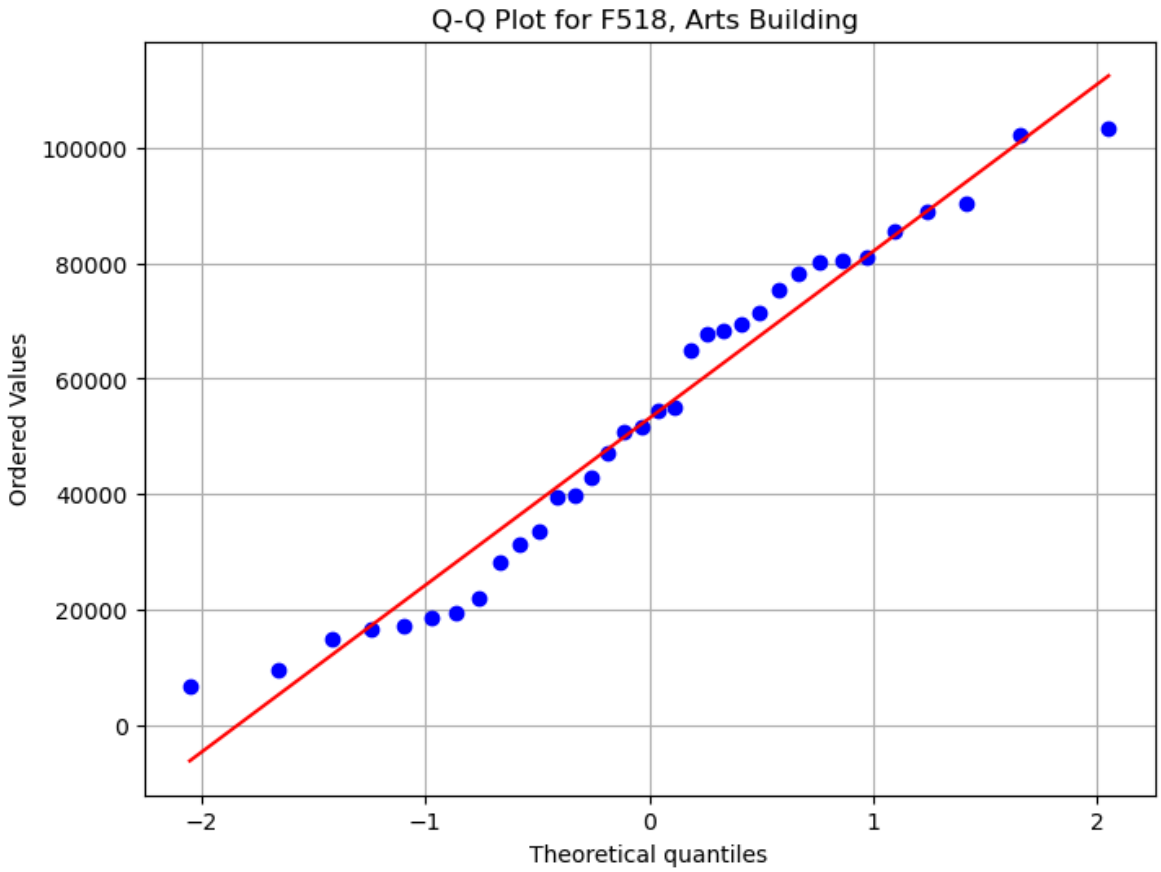
```
                    Building  Skewness   Kurtosis
0     F503, Castle Lecture   0.342946  -0.756328
1    F505, Richardson Total  0.030092  -1.349186
2      F518, Arts Building   0.000374  -1.190042
3     G401 Mellor Lab total  0.217844  -1.120271
4       G404, Microbiology   0.221831  -0.576232
5     G413, Science 2 total -0.024955  -1.322901
6         G501, Staff Club   0.315089  -0.988717
7          G405, Science 3   0.339333  -0.647913
8   G411, St David Lecture  -0.135440  13.906975
9         H402, St David 2   0.653960   0.699564
10               F419, ISB   0.151730  -1.139935
```

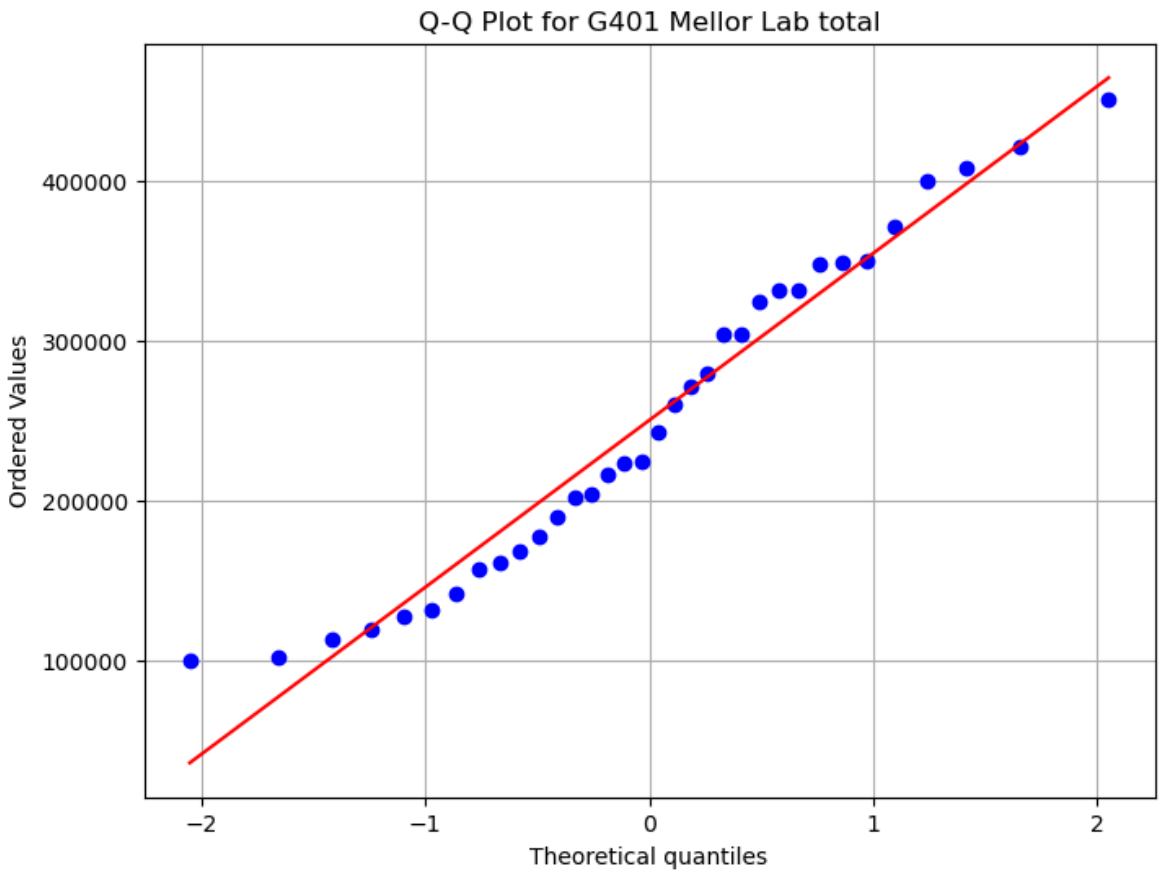
In [159…

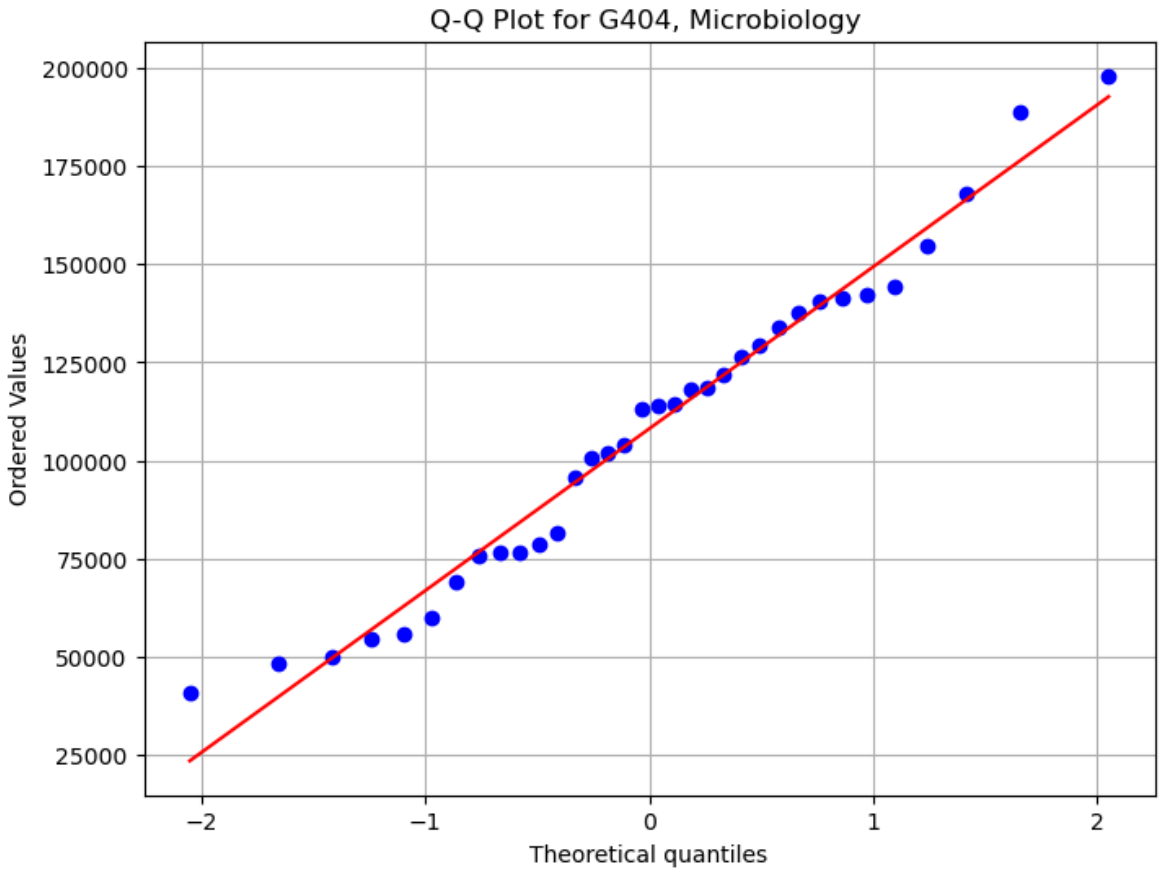
```
import scipy.stats as stats

# Create Q-Q plots for each building
for building in building_data.columns:
    data = building_data[building].dropna()
    plt.figure(figsize=(8, 6))
    stats.probplot(data, dist="norm", plot=plt)
    plt.title(f'Q-Q Plot for {building}')
    plt.grid()
    plt.show()
```

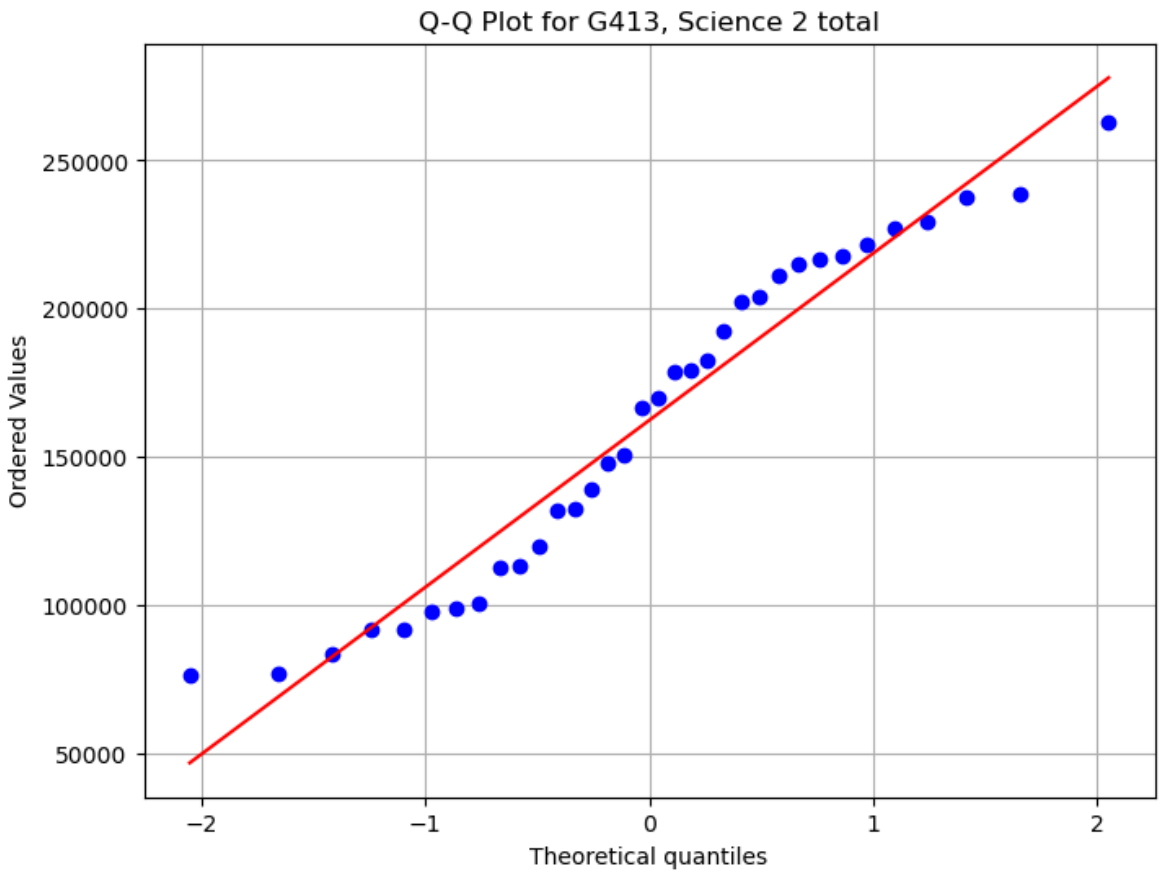


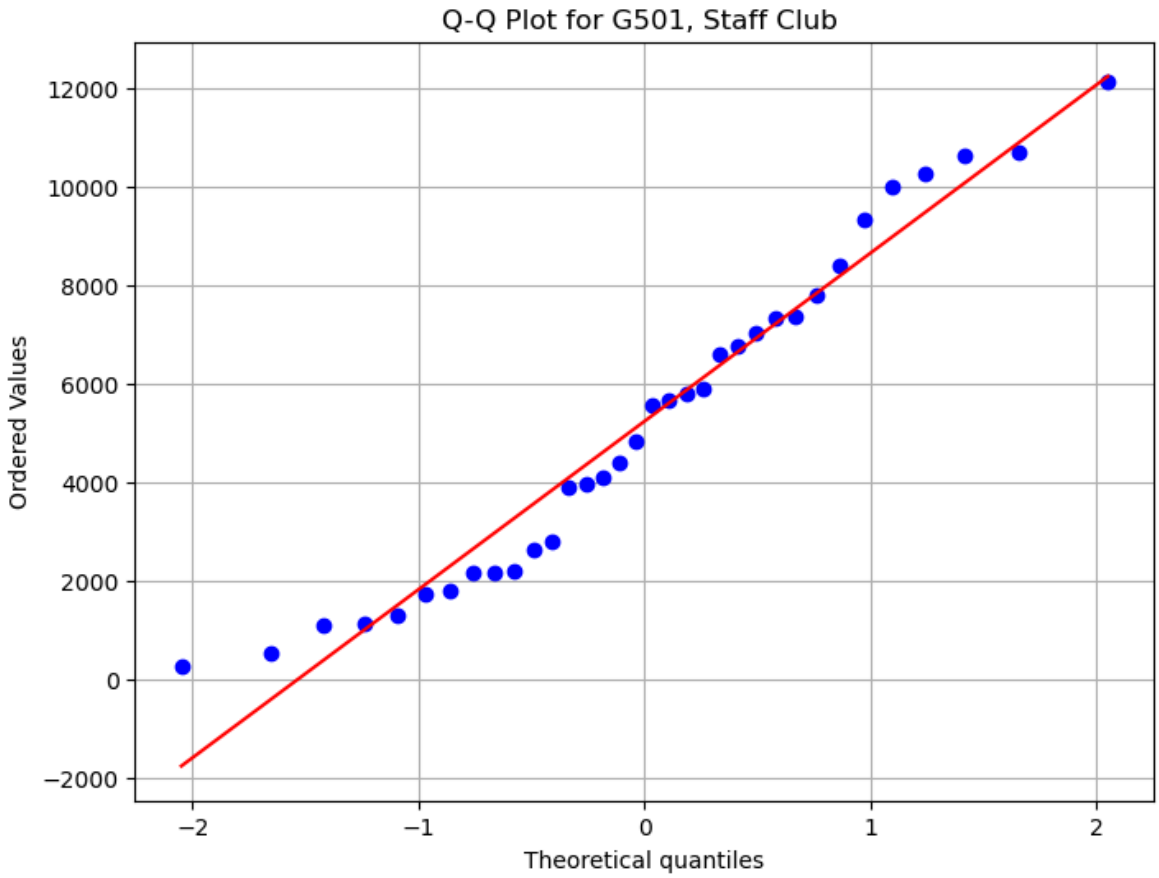

Q-Q Plot for F505, Richardson Total
Ordered Values
Theoretical quantiles
300000
250000
200000
150000
100000
50000
0
−2
−1
0
1
2

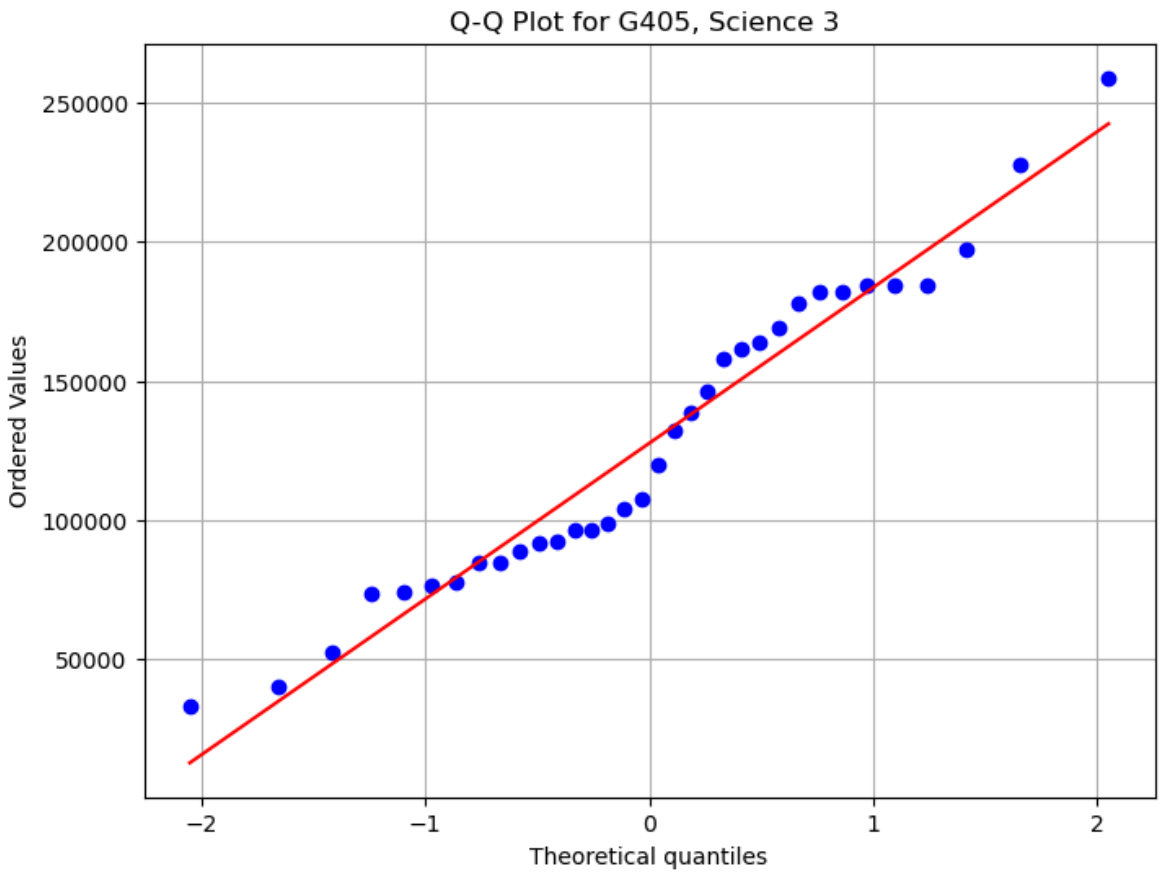

Q-Q Plot for F518, Arts Building
Ordered Values
Theoretical quantiles
100000
80000
60000
40000
20000
0
−2
−1
0
1
2

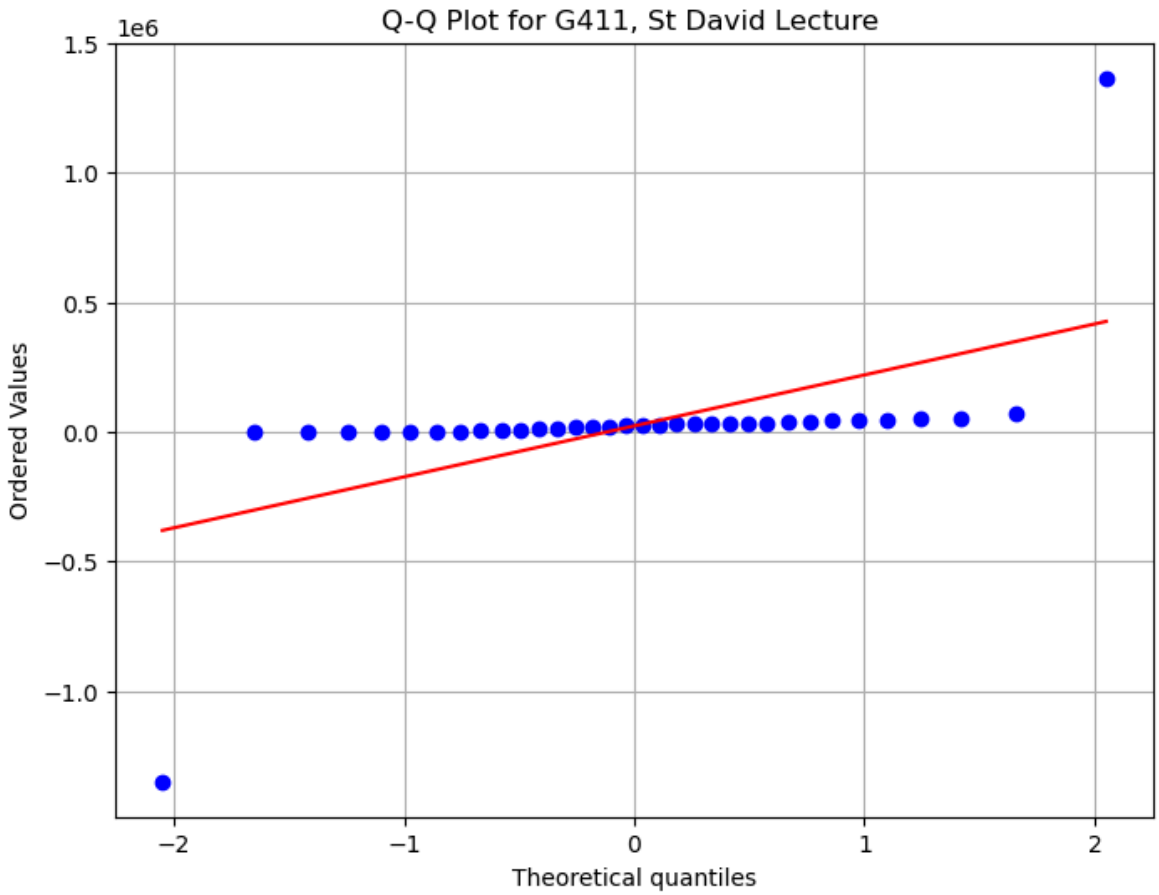

Q-Q Plot for G401 Mellor Lab total
Ordered Values
400000
300000
200000
100000
−2
−1
0
1
2
Theoretical quantiles

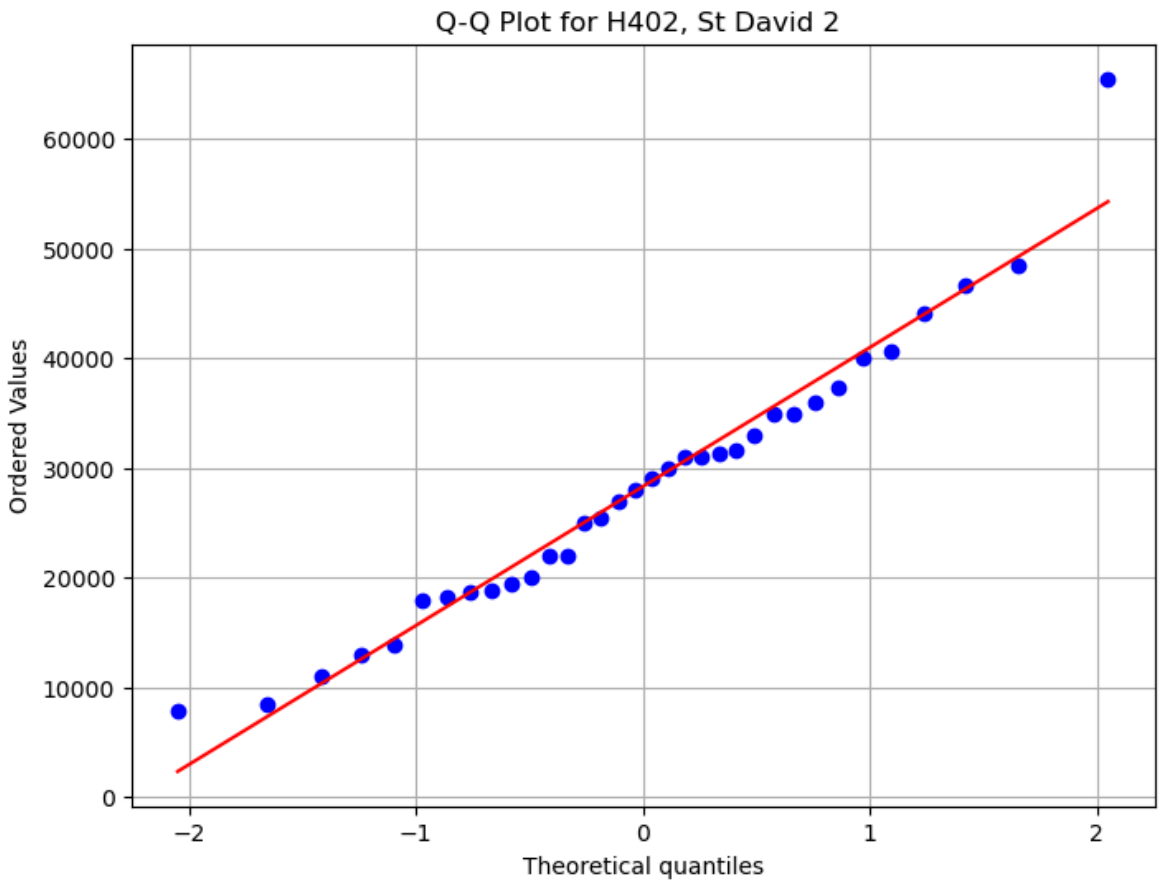

Q-Q Plot for G404, Microbiology
Ordered Values
200000
175000
150000
125000
100000
75000
50000
25000
−2
−1
0
1
2
Theoretical quantiles

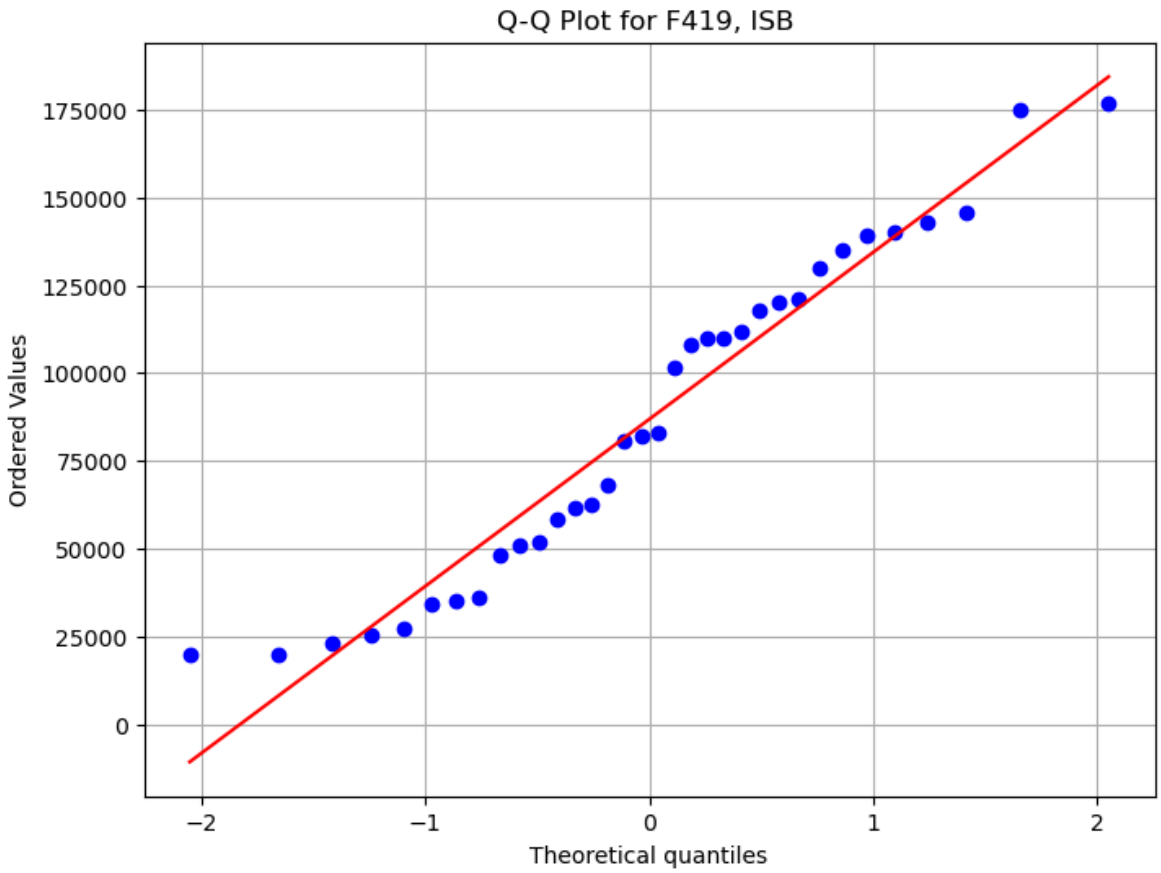

Q-Q Plot for G413, Science 2 total
Ordered Values
250000
200000
150000
100000
50000
−2
−1
0
1
2
Theoretical quantiles


Q-Q Plot for G501, Staff Club
Ordered Values
12000
10000
8000
6000
4000
2000
0
−2000
−2
−1
0
1
2
Theoretical quantiles


Q-Q Plot for G405, Science 3
Ordered Values
Theoretical quantiles
250000
200000
150000
100000
50000
−2
−1
0
1
2


Q-Q Plot for G411, St David Lecture
1e6
Ordered Values
Theoretical quantiles
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2

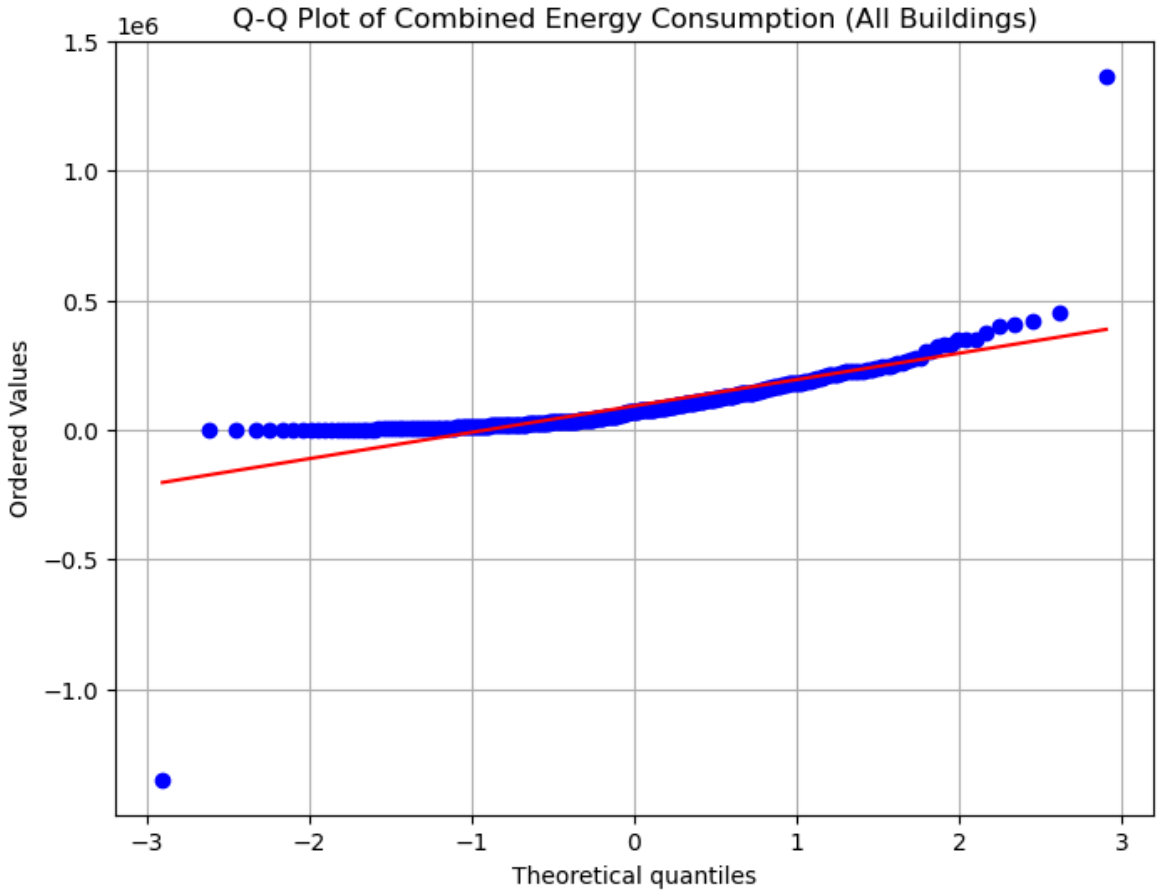




In [106…

```
import scipy.stats as stats

# Generate Q-Q plot for combined data
plt.figure(figsize=(8, 6))
stats.probplot(combined_data, dist="norm", plot=plt)
plt.title('Q-Q Plot of Combined Energy Consumption (All Buildings)')
```

```
plt.grid()
plt.show()
```

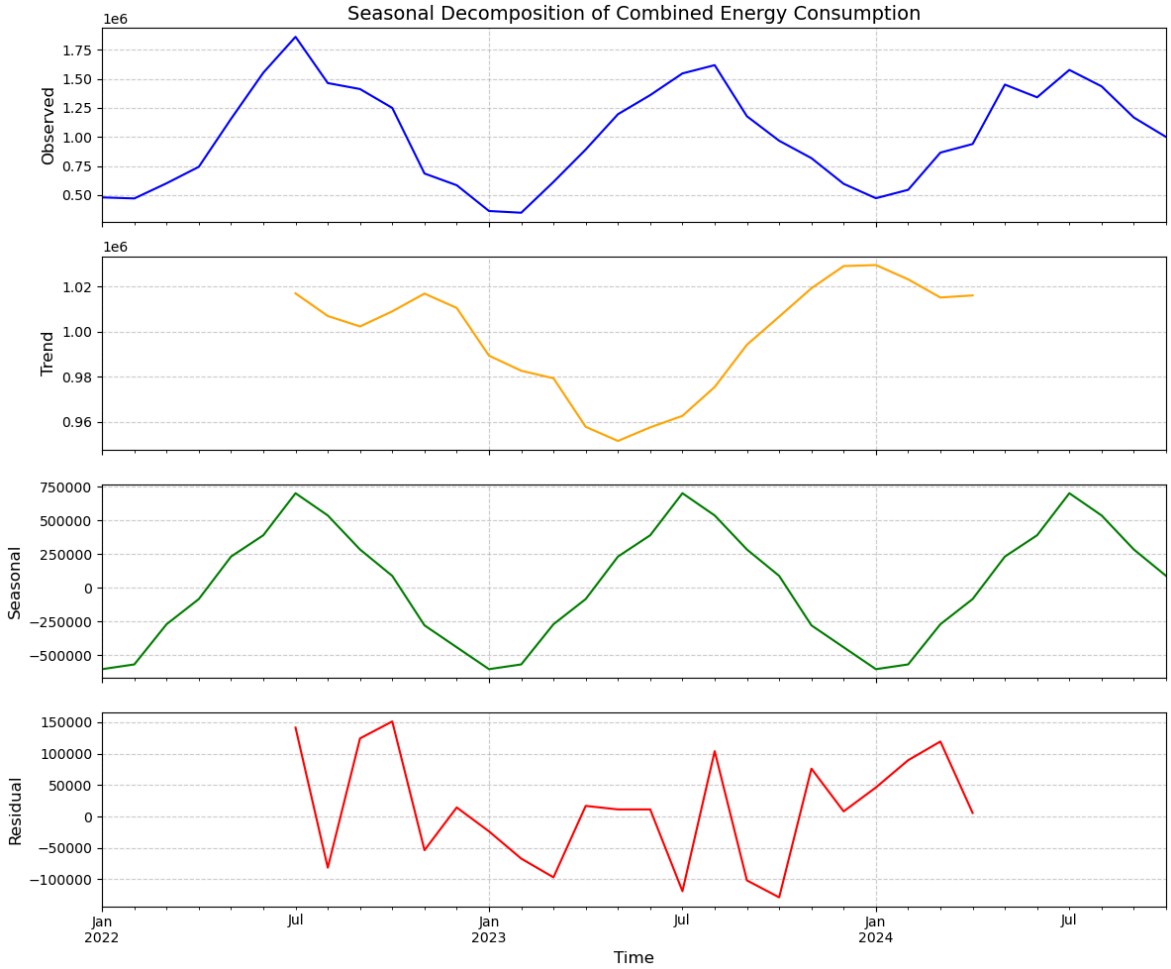


In [161…

```
import pandas as pd
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Imp MTHW Data.csv')

# Preprocess: Set 'Meter Location' as index and transpose to get time-series for
time_series_data = df.set_index('Meter Location').T

# Convert index to datetime (assuming data starts from January 2022)
time_series_data.index = pd.date_range(start='2022-01', periods=len(time_series_

# Aggregate energy consumption across all buildings
combined_data = time_series_data.sum(axis=1)

# Perform seasonal decomposition on the aggregated data
decomposition = seasonal_decompose(combined_data, model='additive', period=12)

# Plot seasonal decomposition
fig, axes = plt.subplots(4, 1, figsize=(12, 10), sharex=True)

decomposition.observed.plot(ax=axes[0], color='blue')
axes[0].set_ylabel('Observed', fontsize=12)
axes[0].set_title('Seasonal Decomposition of Combined Energy Consumption', fonts

decomposition.trend.plot(ax=axes[1], color='orange')
axes[1].set_ylabel('Trend', fontsize=12)
```

```
decomposition.seasonal.plot(ax=axes[2], color='green')
axes[2].set_ylabel('Seasonal', fontsize=12)

decomposition.resid.plot(ax=axes[3], color='red')
axes[3].set_ylabel('Residual', fontsize=12)
axes[3].set_xlabel('Time', fontsize=12)

# Add gridlines for better readability
for ax in axes:
    ax.grid(True, linestyle='--', alpha=0.6)

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

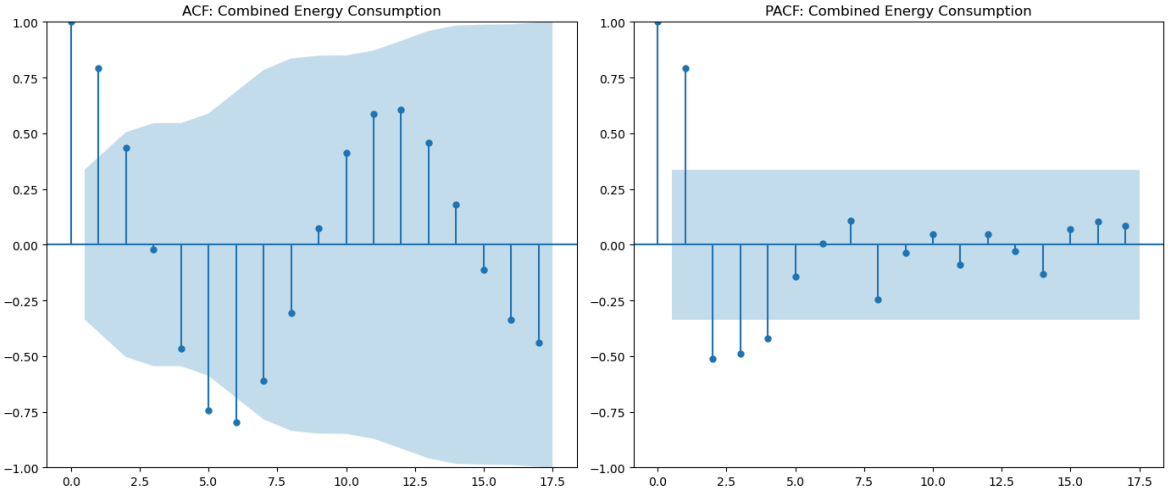
```
C:\Users\sugan\AppData\Local\Temp\ipykernel_13440\1993073577.py:12: FutureWarnin
g: 'M' is deprecated and will be removed in a future version, please use 'ME' ins
tead.
  time_series_data.index = pd.date_range(start='2022-01', periods=len(time_series
_data), freq='M')
```



In [142…

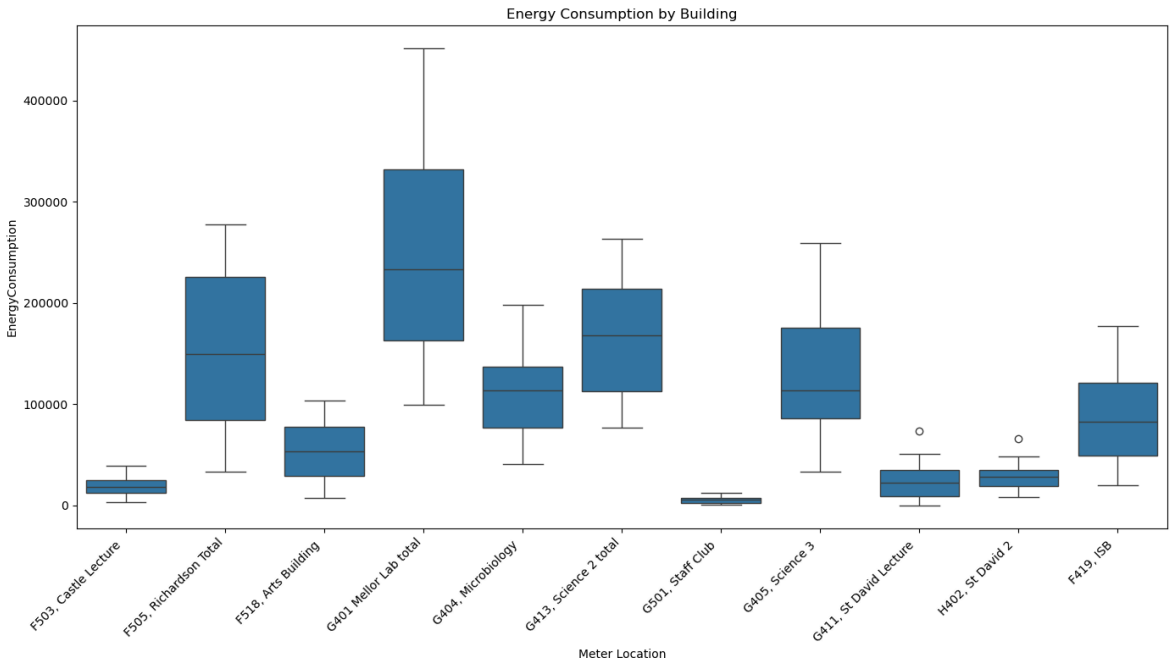
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

# Calculate the maximum allowable number of lags
max_lags = len(combined_data) // 2

# Plot ACF and PACF with corrected lags
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Autocorrelation Function (ACF)
plot_acf(combined_data, lags=max_lags, ax=axes[0])
```

```
axes[0].set_title('ACF: Combined Energy Consumption')

# Partial Autocorrelation Function (PACF)
plot_pacf(combined_data, lags=max_lags, ax=axes[1])
axes[1].set_title('PACF: Combined Energy Consumption')

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



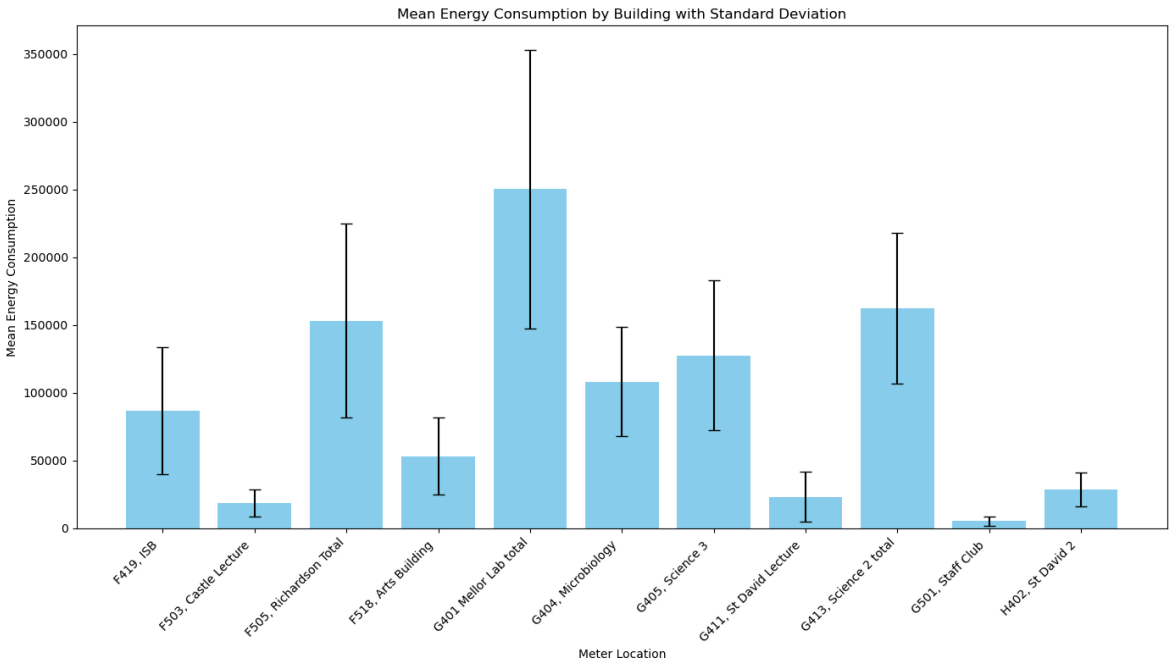
In [138…

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

# Load the dataset
df = pd.read_csv('Imp MTHW Data.csv')

# Data Preprocessing: Reshape data from wide to long format
df_long = pd.melt(df, id_vars=['Meter Location'], var_name='Month', value_name='

# Remove any rows with missing values
df_long = df_long.dropna()

# Remove the year part of the month
df_long['Month'] = df_long['Month'].str.split('-').str[0]

# Box Plots
plt.figure(figsize=(14, 8))
sns.boxplot(x='Meter Location', y='EnergyConsumption', data=df_long)
plt.title('Energy Consumption by Building')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()
```



In [136…

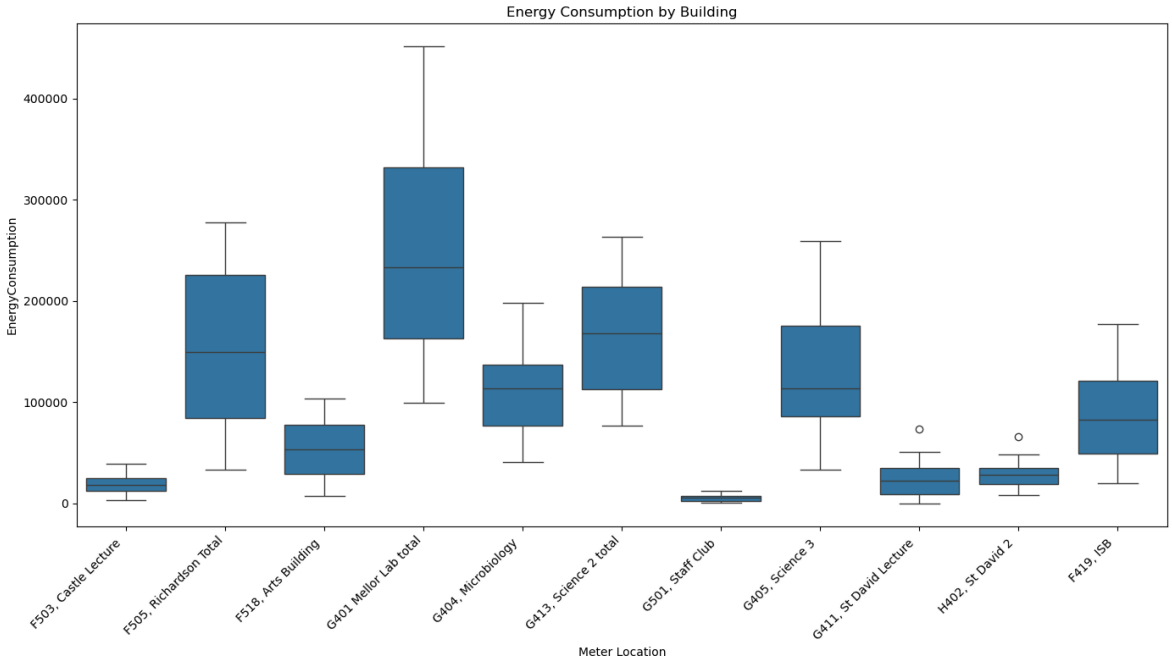
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Load the dataset
df = pd.read_csv('Imp MTHW Data.csv')

# Data Preprocessing: Reshape data from wide to long format
df_long = pd.melt(df, id_vars=['Meter Location'], var_name='Month', value_name='

# Remove any rows with missing values
df_long = df_long.dropna()

# Remove the year part of the month
df_long['Month'] = df_long['Month'].str.split('-').str[0]

# Calculate mean and standard deviation
summary_stats = df_long.groupby('Meter Location')['EnergyConsumption'].agg(['mea

# Bar Chart with Error Bars
plt.figure(figsize=(14, 8))
plt.bar(summary_stats['Meter Location'], summary_stats['mean'], yerr=summary_sta
plt.xlabel('Meter Location')
plt.ylabel('Mean Energy Consumption')
plt.title('Mean Energy Consumption by Building with Standard Deviation')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [146…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Load the dataset
df = pd.read_csv('Imp MTHW Data.csv')

# Data Preprocessing: Reshape data from wide to long format
df_long = pd.melt(df, id_vars=['Meter Location'], var_name='Month', value_name='

# Remove any rows with missing values
df_long = df_long.dropna()

# Remove the year part of the month
df_long['Month'] = df_long['Month'].str.split('-').str[0]

# Box Plots
plt.figure(figsize=(14, 8))
sns.boxplot(x='Meter Location', y='EnergyConsumption', data=df_long)
plt.title('Energy Consumption by Building')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()
plt.show()

# ANOVA
# Perform ANOVA
model = ols('EnergyConsumption ~ C(df_long["Meter Location"])', data=df_long).fi
anova_table = sm.stats.anova_lm(model, typ=2)
print("\nANOVA Table:")
print(anova_table)

# Tukey's HSD Test
# Perform Tukey's HSD post-hoc test
tukey = pairwise_tukeyhsd(endog=df_long['EnergyConsumption'],
                          groups=df_long['Meter Location'],
```

```
                                alpha=0.05)
print("\nTukey's HSD Test Results:")
print(tukey)
```



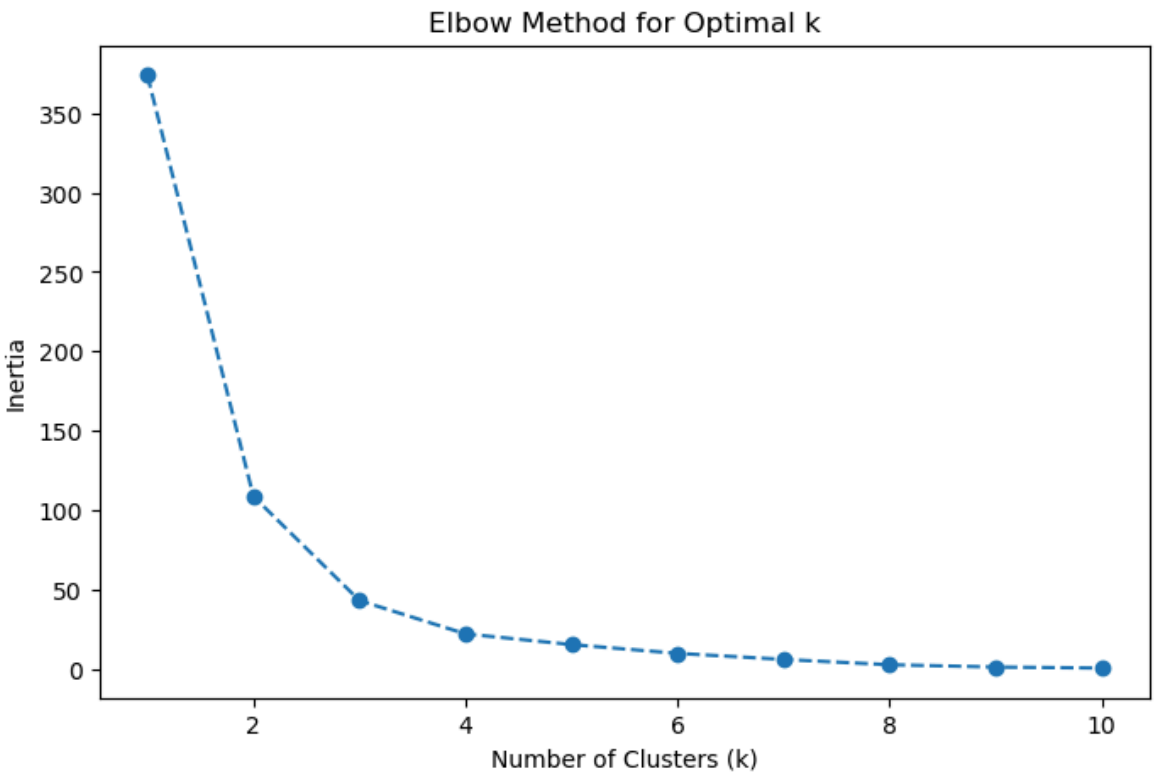
```
ANOVA Table:
                                   sum_sq     df         F        PR(>F)
C(df_long["Meter Location"])  1.990363e+12   10.0  80.77708  5.419847e-86
Residual                      8.944389e+11  363.0       NaN           NaN

Tukey's HSD Test Results:
                        Multiple Comparison of Means - Tukey HSD, FWER=0.05
================================================================================
===================
         group1                   group2           meandiff   p-adj     lower
upper     reject
--------------------------------------------------------------------------------
-------------------
             F419, ISB    F503, Castle Lecture   -68479.4118    0.0 -107479.2387
-29479.5848   True
             F419, ISB   F505, Richardson Total   66198.5294    0.0   27198.7025
105198.3564   True
             F419, ISB     F518, Arts Building   -33682.3529 0.1631  -72682.1799
5317.474  False
             F419, ISB    G401 Mellor Lab total  163508.8582    0.0  124509.0313
202508.6852   True
             F419, ISB      G404, Microbiology    21247.6471 0.7999  -17752.1799
60247.474  False
             F419, ISB         G405, Science 3    40708.8235 0.0324    1708.9966
79708.6505   True
             F419, ISB  G411, St David Lecture   -63720.5882    0.0 -102720.4152
-24720.7613   True
             F419, ISB    G413, Science 2 total   75437.9396    0.0   36438.1126
114437.7665   True
             F419, ISB        G501, Staff Club   -81585.5882    0.0 -120585.4152
-42585.7613   True
             F419, ISB        H402, St David 2   -58501.4706 0.0001  -97501.2975
-19501.6436   True
  F503, Castle Lecture   F505, Richardson Total  134677.9412    0.0   95678.1142
173677.7681   True
  F503, Castle Lecture     F518, Arts Building    34797.0588 0.1302   -4202.7681
73796.8858  False
  F503, Castle Lecture    G401 Mellor Lab total    231988.27    0.0  192988.4431
270988.0969   True
  F503, Castle Lecture      G404, Microbiology    89727.0588    0.0   50727.2319
128726.8858   True
  F503, Castle Lecture         G405, Science 3   109188.2353    0.0   70188.4084
148188.0622   True
  F503, Castle Lecture  G411, St David Lecture     4758.8235    1.0  -34241.0034
43758.6505  False
  F503, Castle Lecture    G413, Science 2 total  143917.3514    0.0  104917.5244
182917.1783   True
  F503, Castle Lecture        G501, Staff Club   -13106.1765 0.9915  -52106.0034
25893.6505  False
  F503, Castle Lecture        H402, St David 2     9977.9412 0.9991  -29021.8858
48977.7681  False
 F505, Richardson Total    F518, Arts Building   -99880.8824    0.0 -138880.7093
-60881.0554   True
 F505, Richardson Total   G401 Mellor Lab total   97310.3288    0.0   58310.5019
136310.1558   True
 F505, Richardson Total     G404, Microbiology   -44950.8824 0.0099  -83950.7093
-5951.0554   True
 F505, Richardson Total        G405, Science 3   -25489.7059 0.5657  -64489.5328
13510.1211  False
 F505, Richardson Total G411, St David Lecture  -129919.1176    0.0 -168918.9446
```

```
-90919.2907   True
 F505, Richardson Total   G413, Science 2 total    9239.4102 0.9995  -29760.4168
48239.2371  False
 F505, Richardson Total       G501, Staff Club  -147784.1176    0.0 -186783.9446
-108784.2907   True
 F505, Richardson Total       H402, St David 2     -124700.0    0.0 -163699.8269
-85700.1731   True
   F518, Arts Building    G401 Mellor Lab total  197191.2112    0.0  158191.3842
236191.0381   True
   F518, Arts Building      G404, Microbiology       54930.0 0.0004   15930.1731
93929.8269   True
   F518, Arts Building         G405, Science 3    74391.1765    0.0   35391.3495
113391.0034   True
   F518, Arts Building  G411, St David Lecture   -30038.2353 0.3109  -69038.0622
8961.5916  False
   F518, Arts Building    G413, Science 2 total  109120.2925    0.0   70120.4656
148120.1195   True
   F518, Arts Building        G501, Staff Club   -47903.2353  0.004  -86903.0622
-8903.4084   True
   F518, Arts Building        H402, St David 2   -24819.1176 0.6056  -63818.9446
14180.7093  False
  G401 Mellor Lab total     G404, Microbiology  -142261.2112    0.0 -181261.0381
-103261.3842   True
  G401 Mellor Lab total        G405, Science 3  -122800.0347    0.0 -161799.8616
-83800.2078   True
  G401 Mellor Lab total G411, St David Lecture  -227229.4465    0.0 -266229.2734
-188229.6195   True
  G401 Mellor Lab total   G413, Science 2 total  -88070.9186    0.0 -127070.7456
-49071.0917   True
  G401 Mellor Lab total       G501, Staff Club  -245094.4465    0.0 -284094.2734
-206094.6195   True
  G401 Mellor Lab total       H402, St David 2  -222010.3288    0.0 -261010.1558
-183010.5019   True
    G404, Microbiology         G405, Science 3    19461.1765 0.8746  -19538.6505
58461.0034  False
    G404, Microbiology  G411, St David Lecture   -84968.2353    0.0 -123968.0622
-45968.4084   True
    G404, Microbiology    G413, Science 2 total   54190.2925 0.0005   15190.4656
93190.1195   True
    G404, Microbiology        G501, Staff Club  -102833.2353    0.0 -141833.0622
-63833.4084   True
    G404, Microbiology        H402, St David 2   -79749.1176    0.0 -118748.9446
-40749.2907   True
       G405, Science 3  G411, St David Lecture  -104429.4118    0.0 -143429.2387
-65429.5848   True
       G405, Science 3    G413, Science 2 total   34729.1161 0.1321   -4270.7109
73728.943  False
       G405, Science 3        G501, Staff Club  -122294.4118    0.0 -161294.2387
-83294.5848   True
       G405, Science 3        H402, St David 2   -99210.2941    0.0 -138210.1211
-60210.4672   True
G411, St David Lecture    G413, Science 2 total  139158.5278    0.0  100158.7009
178158.3548   True
G411, St David Lecture        G501, Staff Club      -17865.0 0.9244  -56864.8269
21134.8269  False
G411, St David Lecture        H402, St David 2     5219.1176    1.0  -33780.7093
44218.9446  False
  G413, Science 2 total       G501, Staff Club  -157023.5278    0.0 -196023.3548
-118023.7009   True
  G413, Science 2 total       H402, St David 2  -133939.4102    0.0 -172939.2371
```

```
-94939.5832   True
      G501, Staff Club        H402, St David 2    23084.1176 0.7056  -15915.7093
62083.9446  False
--------------------------------------------------------------------------------
-------------------
```

In [163…

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load dataset
file_path = "Imp MTHW Data.csv"
df = pd.read_csv(file_path)

# Drop non-numeric column ('Meter Location')
data = df.drop(columns=['Meter Location'])

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Finding the optimal number of clusters using the Elbow Method
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

# Plot Elbow Method
def plot_elbow():
    plt.figure(figsize=(8, 5))
    plt.plot(k_range, inertia, marker='o', linestyle='--')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.show()

plot_elbow()

# Apply K-Means with optimal k (assuming k=3 based on the elbow plot)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(data_scaled)

# Display cluster assignment
import ace_tools as tools
tools.display_dataframe_to_user(name="K-Means Clustering Results", dataframe=df)
```

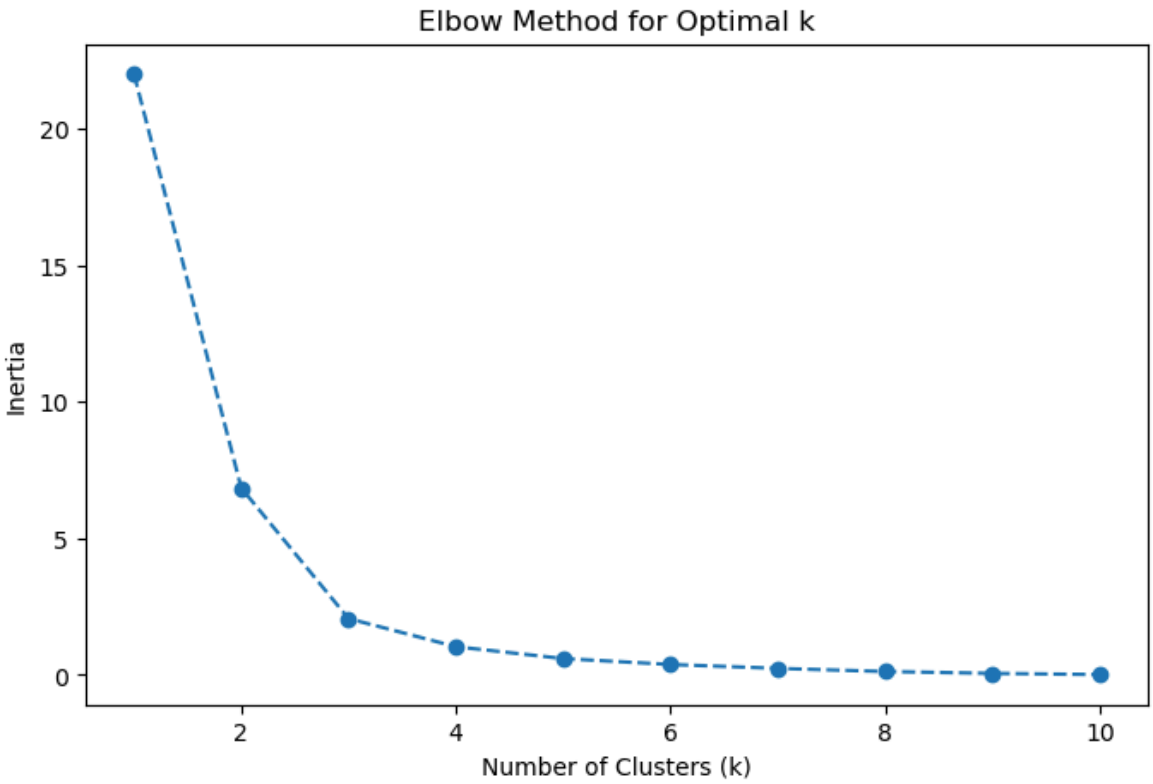



```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[163], line 43
     40 df['Cluster'] = kmeans.fit_predict(data_scaled)
     42 # Display cluster assignment
---> 43 import ace_tools as tools
     44 tools.display_dataframe_to_user(name="K-Means Clustering Results", datafr
ame=df)

ModuleNotFoundError: No module named 'ace_tools'
```

In [167…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load dataset
file_path = "Imp MTHW Data.csv"
df = pd.read_csv(file_path)

# Drop non-numeric column ('Meter Location')
data = df.drop(columns=['Meter Location'])

# Compute mean and standard deviation for each building
mean_consumption = data.mean(axis=1)
std_consumption = data.std(axis=1)

# Create a new dataframe with mean and standard deviation
features = pd.DataFrame({'Mean Consumption': mean_consumption, 'Std Consumption'

# Standardize the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

```
# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
pca_features = pca.fit_transform(features_scaled)

# Finding the optimal number of clusters using the Elbow Method
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pca_features)
    inertia.append(kmeans.inertia_)

# Plot Elbow Method
def plot_elbow():
    plt.figure(figsize=(8, 5))
    plt.plot(k_range, inertia, marker='o', linestyle='--')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.show()

plot_elbow()

# Apply K-Means with optimal k (assuming k=3 based on the elbow plot)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(pca_features)

# Display cluster assignment
import ace_tools as tools
tools.display_dataframe_to_user(name="PCA-based Clustering Results", dataframe=d
```

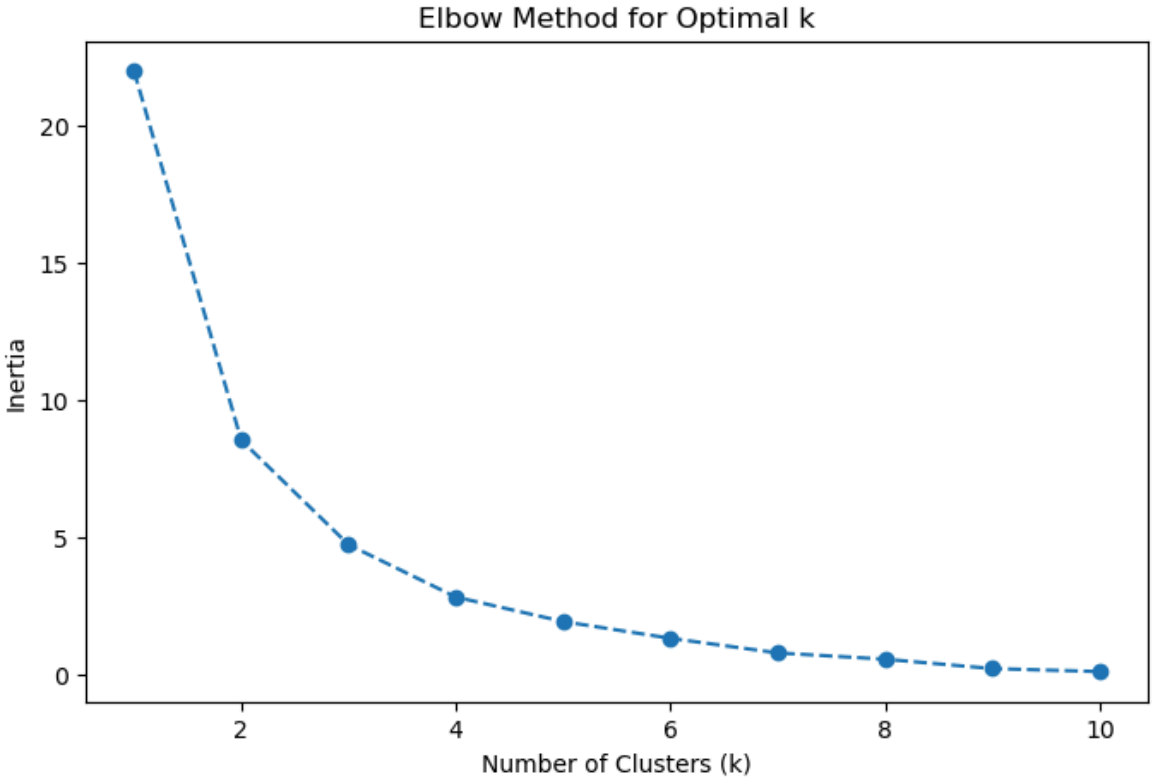


```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[167], line 55
     52 df['Cluster'] = kmeans.fit_predict(pca_features)
     54 # Display cluster assignment
---> 55 import ace_tools as tools
     56 tools.display_dataframe_to_user(name="PCA-based Clustering Results", data
frame=df)

ModuleNotFoundError: No module named 'ace_tools'
```

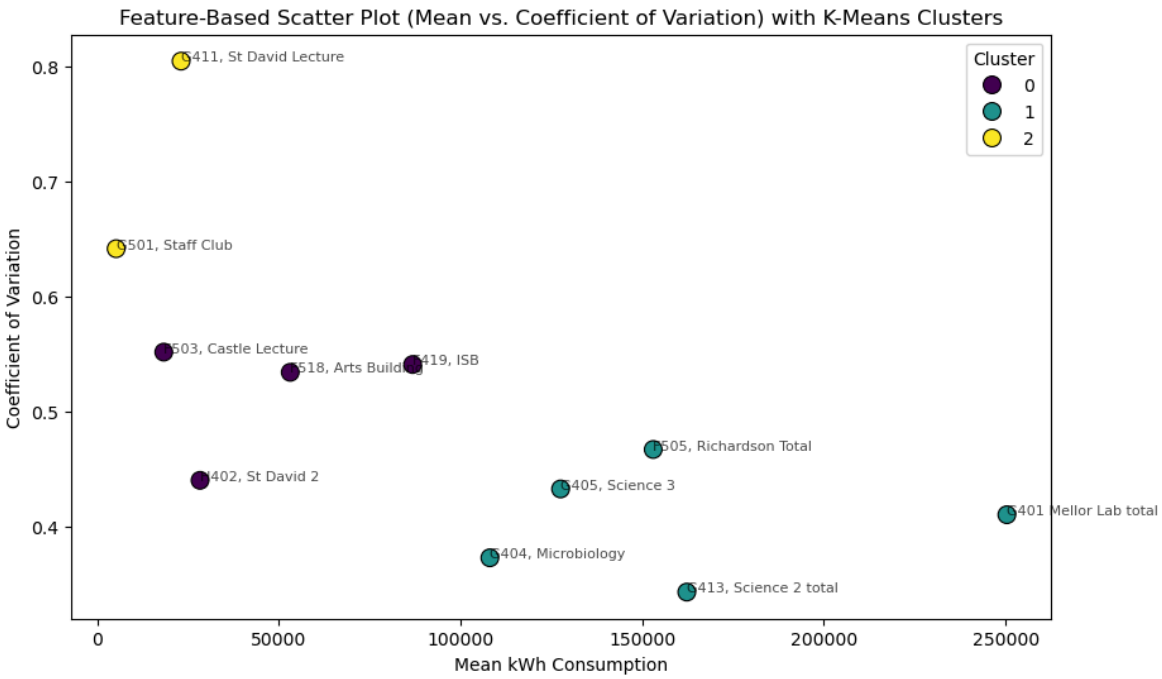
In [169…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load dataset
file_path = "Imp MTHW Data.csv"
df = pd.read_csv(file_path)

# Drop non-numeric column ('Meter Location')
data = df.drop(columns=['Meter Location'])

# Compute mean and coefficient of variation (CV)
mean_consumption = data.mean(axis=1)
cv_consumption = data.std(axis=1) / mean_consumption

# Create a new dataframe with mean and CV
features = pd.DataFrame({'Mean Consumption': mean_consumption, 'Coefficient of V

# Standardize the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
pca_features = pca.fit_transform(features_scaled)

# Finding the optimal number of clusters using the Elbow Method
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pca_features)
    inertia.append(kmeans.inertia_)

# Plot Elbow Method
def plot_elbow():
    plt.figure(figsize=(8, 5))
    plt.plot(k_range, inertia, marker='o', linestyle='--')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.show()

plot_elbow()
```

```
# Apply K-Means with optimal k (assuming k=3 based on the elbow plot)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(pca_features)

# Scatter plot for Mean vs Coefficient of Variation with clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x=features['Mean Consumption'], y=features['Coefficient of Varia
for i, txt in enumerate(df['Meter Location']):
    plt.annotate(txt, (features['Mean Consumption'][i], features['Coefficient of
plt.xlabel('Mean kWh Consumption')
plt.ylabel('Coefficient of Variation')
plt.title('Feature-Based Scatter Plot (Mean vs. Coefficient of Variation) with K
plt.colorbar(label='Cluster')
plt.show()

# Display cluster assignment
import ace_tools as tools
tools.display_dataframe_to_user(name="Feature-Based Clustering Results", datafra
```



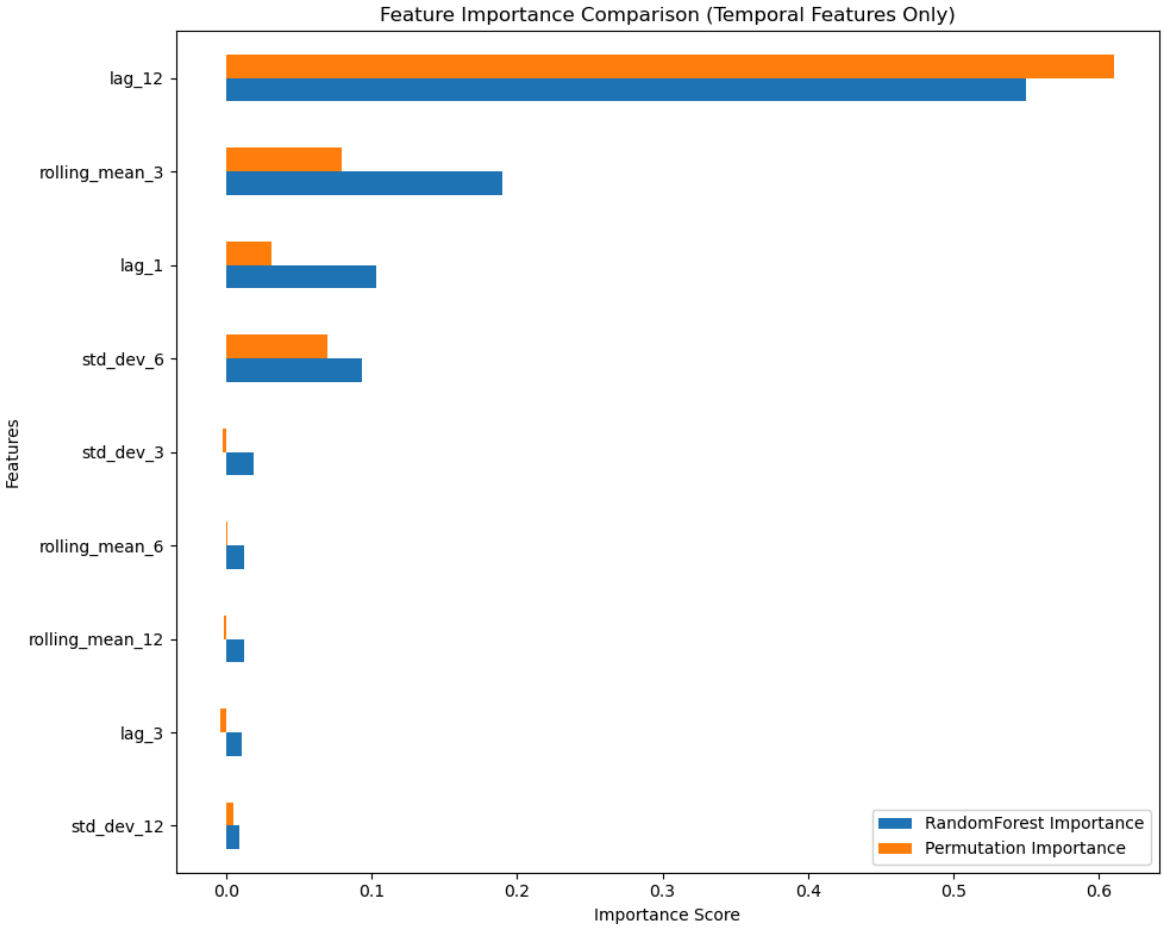
```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[169], line 63
     61 plt.ylabel('Coefficient of Variation')
     62 plt.title('Feature-Based Scatter Plot (Mean vs. Coefficient of Variation)
with K-Means Clusters')
---> 63 plt.colorbar(label='Cluster')
     64 plt.show()
     66 # Display cluster assignment

File ~\anaconda3\Lib\site-packages\matplotlib\pyplot.py:2512, in colorbar(mappabl
e, cax, ax, **kwargs)
   2510     mappable = gci()
   2511     if mappable is None:
-> 2512         raise RuntimeError('No mappable was found to use for colorbar '
   2513                            'creation. First define a mappable such as '
   2514                            'an image (with imshow) or a contour set ('
   2515                            'with contourf).')
   2516 ret = gcf().colorbar(mappable, cax=cax, ax=ax, **kwargs)
   2517 return ret

RuntimeError: No mappable was found to use for colorbar creation. First define a
mappable such as an image (with imshow) or a contour set (with contourf).
```



In [185…

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('Imp MTHW Data.csv')

# Transpose the data to have time as rows and ignore building names
df.set_index("Meter Location", inplace=True)
df = df.T  # Transpose to make time the rows
df = df.apply(pd.to_numeric, errors='coerce')  # Ensure numeric values
```

```
# Create temporal features from the aggregated data (e.g., mean across all build
df['rolling_mean_3'] = df.mean(axis=1).rolling(window=3).mean()
df['rolling_mean_6'] = df.mean(axis=1).rolling(window=6).mean()
df['rolling_mean_12'] = df.mean(axis=1).rolling(window=12).mean()
df['lag_1'] = df.mean(axis=1).shift(1)
df['lag_3'] = df.mean(axis=1).shift(3)
df['lag_12'] = df.mean(axis=1).shift(12)
df['std_dev_3'] = df.mean(axis=1).rolling(window=3).std()
df['std_dev_6'] = df.mean(axis=1).rolling(window=6).std()
df['std_dev_12'] = df.mean(axis=1).rolling(window=12).std()

# Drop NaN rows caused by rolling and lag operations
features = df[['rolling_mean_3', 'rolling_mean_6', 'rolling_mean_12',
               'lag_1', 'lag_3', 'lag_12',
               'std_dev_3', 'std_dev_6', 'std_dev_12']].dropna()

# Define target variable (e.g., overall average energy consumption across all bu
target = df.mean(axis=1).loc[features.index]  # Use the same index as features f

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=

# Train a Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Calculate feature importance using Random Forest
rf_importances = rf_model.feature_importances_

# Calculate permutation importance for interpretability
perm_importance = permutation_importance(rf_model, X_test, y_test, n_repeats=10,
perm_importances = perm_importance.importances_mean

# Combine results for plotting
importance_df = pd.DataFrame({
    'Feature': features.columns,
    'RandomForest Importance': rf_importances,
    'Permutation Importance': perm_importances
}).set_index('Feature')

# Sort by Random Forest importance for better visualization
importance_df.sort_values('RandomForest Importance', ascending=True, inplace=Tru

# Plot feature importance comparison
plt.figure(figsize=(10, 8))
importance_df.plot(kind='barh', ax=plt.gca())
plt.title('Feature Importance Comparison (Temporal Features Only)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

# Print top 10 important features for each method
print("Top 10 Features by Random Forest:")
print(importance_df['RandomForest Importance'].sort_values(ascending=False).head

print("\nTop 10 Features by Permutation Importance:")
print(importance_df['Permutation Importance'].sort_values(ascending=False).head(
```



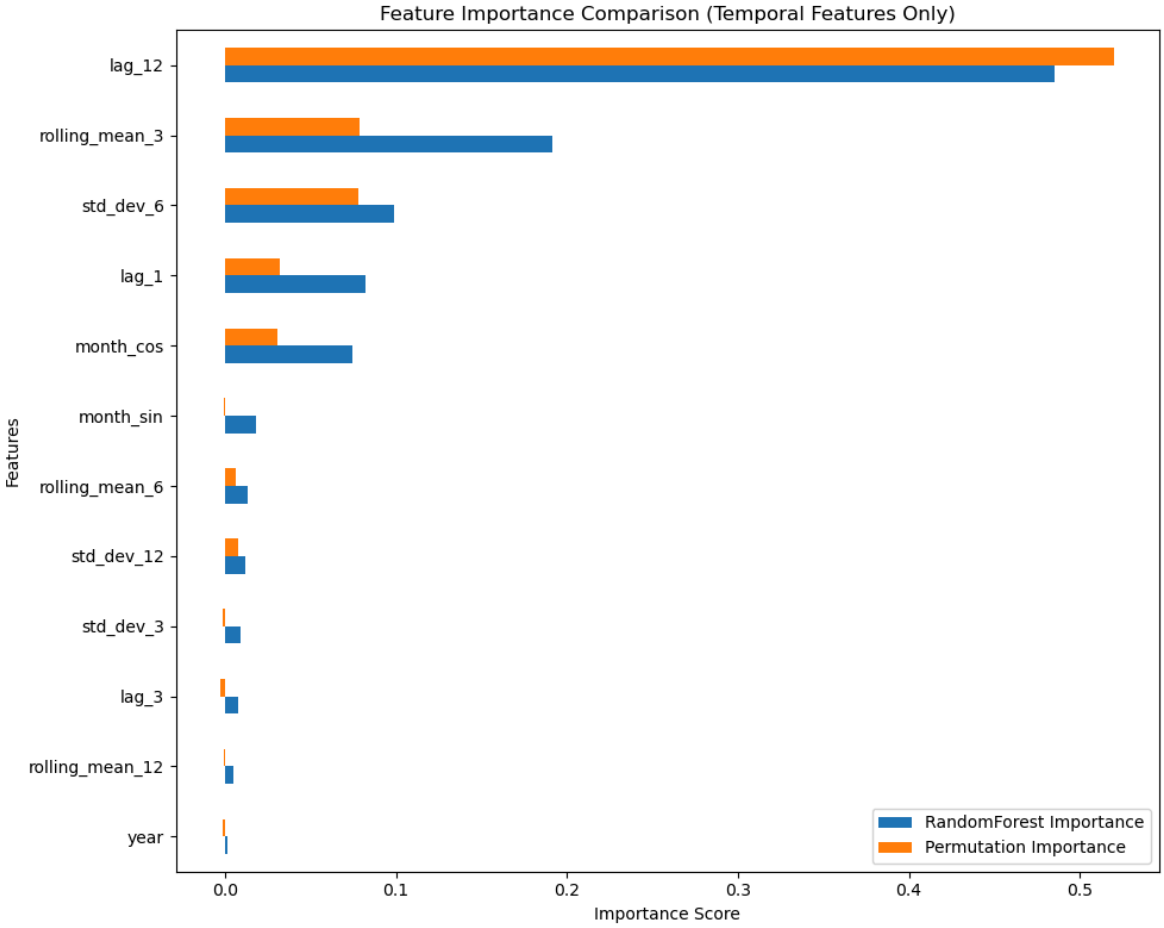

```
Top 10 Features by Random Forest:
Feature
lag_12             0.549553
rolling_mean_3     0.190081
lag_1              0.103628
std_dev_6          0.093248
std_dev_3          0.018799
rolling_mean_6     0.012467
rolling_mean_12    0.012279
lag_3              0.010735
std_dev_12         0.009209
Name: RandomForest Importance, dtype: float64

Top 10 Features by Permutation Importance:
Feature
lag_12             0.610684
rolling_mean_3     0.079812
std_dev_6          0.069745
lag_1              0.031652
std_dev_12         0.005443
rolling_mean_6     0.001308
rolling_mean_12   -0.001042
std_dev_3         -0.002373
lag_3             -0.003662
Name: Permutation Importance, dtype: float64
```

In [354…

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
```

```
# Load the data
df = pd.read_csv('Imp MTHW Data.csv')

# Transpose the data to have time as rows and ignore building names
df.set_index("Meter Location", inplace=True)
df = df.T  # Transpose to make time the rows
df = df.apply(pd.to_numeric, errors='coerce')  # Ensure numeric values

# Convert index to datetime
df.index = pd.to_datetime(df.index, format='%b-%y')

# Create temporal features from the aggregated data (e.g., mean across all build
df['rolling_mean_3'] = df.mean(axis=1).rolling(window=3).mean()
df['rolling_mean_6'] = df.mean(axis=1).rolling(window=6).mean()
df['rolling_mean_12'] = df.mean(axis=1).rolling(window=12).mean()
df['lag_1'] = df.mean(axis=1).shift(1)
df['lag_3'] = df.mean(axis=1).shift(3)
df['lag_12'] = df.mean(axis=1).shift(12)
df['std_dev_3'] = df.mean(axis=1).rolling(window=3).std()
df['std_dev_6'] = df.mean(axis=1).rolling(window=6).std()
df['std_dev_12'] = df.mean(axis=1).rolling(window=12).std()

# Add month-based features (sine and cosine transformations)
df['month_sin'] = np.sin(2 * np.pi * df.index.month / 12)
df['month_cos'] = np.cos(2 * np.pi * df.index.month / 12)

# Add year
df['year'] = df.index.year

# Drop NaN rows caused by rolling and lag operations
features = df[['rolling_mean_3', 'rolling_mean_6', 'rolling_mean_12',
               'lag_1', 'lag_3', 'lag_12',
               'std_dev_3', 'std_dev_6', 'std_dev_12',
               'month_sin', 'month_cos', 'year']].dropna()

# Define target variable (e.g., overall average energy consumption across all bu
target = df.mean(axis=1).loc[features.index]  # Use the same index as features f

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=

# Train a Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Calculate feature importance using Random Forest
rf_importances = rf_model.feature_importances_

# Calculate permutation importance for interpretability
perm_importance = permutation_importance(rf_model, X_test, y_test, n_repeats=10,
perm_importances = perm_importance.importances_mean

# Combine results for plotting
importance_df = pd.DataFrame({
    'Feature': features.columns,
    'RandomForest Importance': rf_importances,
    'Permutation Importance': perm_importances
}).set_index('Feature')
```

```
# Sort by Random Forest importance for better visualization
importance_df.sort_values('RandomForest Importance', ascending=True, inplace=Tru

# Plot feature importance comparison
plt.figure(figsize=(10, 8))
importance_df.plot(kind='barh', ax=plt.gca())
plt.title('Feature Importance Comparison (Temporal Features Only)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

# Print top 10 important features for each method
print("Top 10 Features by Random Forest:")
print(importance_df['RandomForest Importance'].sort_values(ascending=False).head

print("\nTop 10 Features by Permutation Importance:")
print(importance_df['Permutation Importance'].sort_values(ascending=False).head(
```



```
Top 10 Features by Random Forest:
Feature
lag_12            0.485567
rolling_mean_3    0.191637
std_dev_6         0.098874
lag_1             0.082562
month_cos         0.074285
month_sin         0.018190
rolling_mean_6    0.013519
std_dev_12        0.011604
std_dev_3         0.009103
lag_3             0.007870
Name: RandomForest Importance, dtype: float64

Top 10 Features by Permutation Importance:
Feature
lag_12             0.520325
rolling_mean_3     0.078767
std_dev_6          0.078185
lag_1              0.032237
month_cos          0.030627
std_dev_12         0.007954
rolling_mean_6     0.006630
rolling_mean_12   -0.000291
month_sin         -0.000346
std_dev_3         -0.001065
Name: Permutation Importance, dtype: float64
```

In [189…

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, Randomize
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

# Load the data
data = pd.read_csv('Imp MTHW Data.csv')
data.set_index("Meter Location", inplace=True)
data = data.T  # Transpose to make time the rows
data = data.apply(pd.to_numeric, errors='coerce')  # Ensure numeric values

# Create temporal features
data['rolling_mean_3'] = data.mean(axis=1).rolling(window=3).mean()
#data['rolling_mean_6'] = data.mean(axis=1).rolling(window=6).mean()
data['rolling_mean_12'] = data.mean(axis=1).rolling(window=12).mean()
data['lag_1'] = data.mean(axis=1).shift(1)
#data['lag_3'] = data.mean(axis=1).shift(3)
#data['lag_12'] = data.mean(axis=1).shift(12)
#data['std_dev_3'] = data.mean(axis=1).rolling(window=3).std()
#data['std_dev_6'] = data.mean(axis=1).rolling(window=6).std()
#data['std_dev_12'] = data.mean(axis=1).rolling(window=12).std()

# Select top features based on importance
top_features = [#'lag_12',
    'rolling_mean_3', 'lag_1',
    #'std_dev_6', 'std_dev_3', 'rolling_mean_6',
    'rolling_mean_12',
    #'lag_3', 'std_dev_12'
]
features = data[top_features].dropna()
```

```
# Define target variable (overall average energy consumption across all building
target = data.mean(axis=1).loc[features.index]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define hyperparameter search space
param_dist = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Perform RandomizedSearchCV
rf = RandomForestRegressor(random_state=42)
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist,
                                   n_iter=100, cv=3, random_state=42, n_jobs=-1)
random_search.fit(X_train_scaled, y_train)

# Get best model
best_rf = random_search.best_estimator_

# Evaluate the model on training and test sets
y_train_pred = best_rf.predict(X_train_scaled)
y_test_pred = best_rf.predict(X_test_scaled)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

# Perform cross-validation
cv_scores = cross_val_score(best_rf, X_train_scaled, y_train, cv=3, scoring='r2'

print(f"Best parameters: {random_search.best_params_}")
print(f"Training R2: {train_r2:.4f}")
print(f"Test R2: {test_r2:.4f}")
print(f"Cross-validation R2: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4

# Feature importance of the best model
feature_importance = pd.DataFrame({'feature': top_features,
                                   'importance': best_rf.feature_importances_})
feature_importance = feature_importance.sort_values('importance', ascending=Fals
print("\nFeature Importance:")
print(feature_importance)
```

```
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.p
y:540: FitFailedWarning:
120 fits failed out of a total of 300.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sc
ore='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
63 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\model_selection\_valid
ation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\base.py", line 1466, i
n wrapper
    estimator._validate_params()
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\base.py", line 666, in
_validate_params
    validate_parameter_constraints(
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\utils\_param_validatio
n.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er of RandomForestRegressor must be an int in the range [1, inf), a float in the
range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

--------------------------------------------------------------------------------
57 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\model_selection\_valid
ation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\base.py", line 1466, i
n wrapper
    estimator._validate_params()
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\base.py", line 666, in
_validate_params
    validate_parameter_constraints(
  File "C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\utils\_param_validatio
n.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er of RandomForestRegressor must be an int in the range [1, inf), a float in the
range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:110
2: UserWarning: One or more of the test scores are non-finite: [-0.52400117  0.34
241844  0.27771392         nan         nan -0.07507254
  0.29295521 -0.46897884         nan  0.32360589         nan         nan
  0.34686465 -0.46897884         nan  0.31690836 -0.43032874 -0.43344497
         nan         nan -0.46897884  0.27771392 -0.0827977          nan
         nan -0.0827977  -0.04635802 -0.52400117  0.32360589 -0.0827977
         nan -0.43344497         nan         nan  0.29295521         nan
  0.31193998 -0.52400117         nan         nan         nan -0.52400117
 -0.46897884         nan -0.07507254 -0.52400117         nan -0.0827977
  0.29295521         nan -0.0827977   0.31193998         nan -0.07507254
 -0.04635802 -0.07507254  0.31229529         nan         nan  0.33784381
         nan -0.52400117 -0.52400117         nan         nan -0.0827977
```

```
         nan         nan         nan  0.33784381         nan  0.32360589
 -0.43344497 -0.04635802  0.2942298  -0.52400117 -0.52400117         nan
 -0.07507254         nan  0.31285455  0.27771392         nan  0.32926992
 -0.52400117         nan  0.31690836  0.31690836 -0.0827977          nan
  0.34686465  0.29295521         nan  0.31690836         nan  0.27606251
         nan  0.32360589         nan         nan]
  warnings.warn(
Best parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_lea
f': 2, 'max_features': 'log2', 'max_depth': 20}
Training R2: 0.8765
Test R2: 0.6068
Cross-validation R2: 0.3469 (+/- 0.1094)

Feature Importance:
           feature  importance
1            lag_1    0.408629
0   rolling_mean_3    0.386366
2  rolling_mean_12    0.205005
```

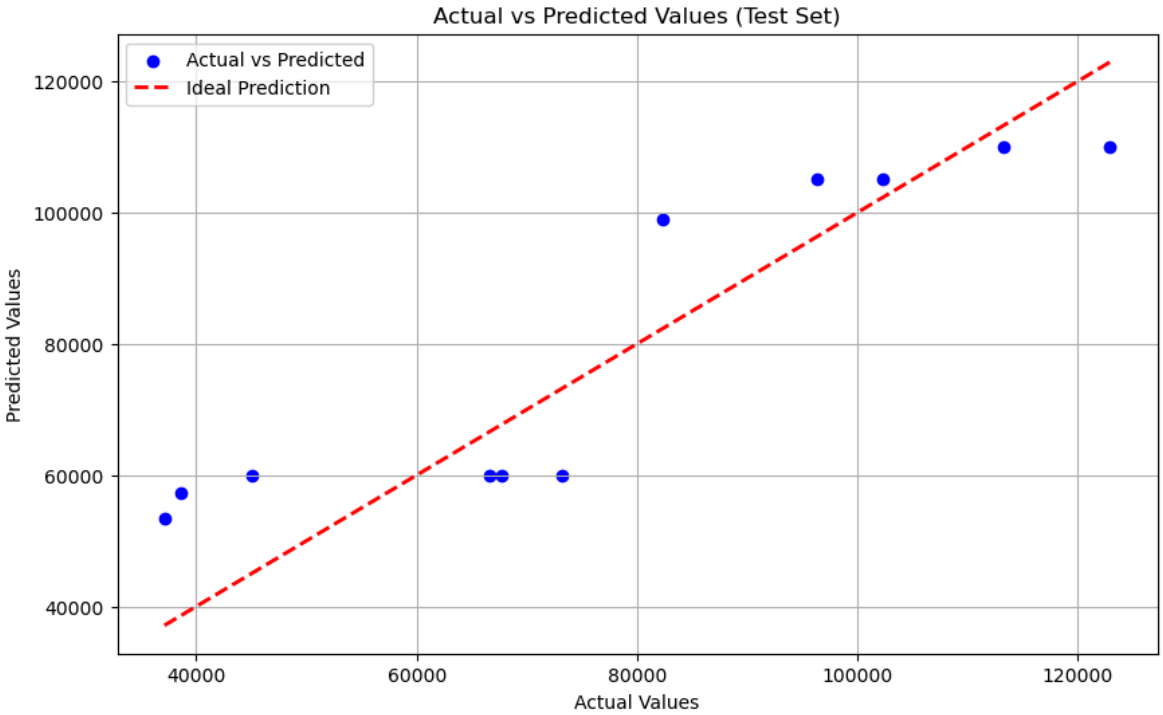
In [352…

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler

# Load the data
data = pd.read_csv('Imp MTHW Data.csv')
data.set_index("Meter Location", inplace=True)
data = data.T  # Transpose to make time the rows
data.index = pd.to_datetime(data.index, format='%b-%y')  # Convert index to date
data = data.apply(pd.to_numeric, errors='coerce')  # Ensure numeric values

# Create temporal features
data['rolling_mean_3'] = data.mean(axis=1).rolling(window=3).mean()
data['rolling_mean_6'] = data.mean(axis=1).rolling(window=6).mean()
data['lag_1'] = data.mean(axis=1).shift(1)
data['rolling_mean_12'] = data.mean(axis=1).rolling(window=12).mean()
data['month'] = data.index.month
data['year'] = data.index.year

# Select features
features = ['rolling_mean_3',
            'lag_1',
            'rolling_mean_12',
            #'rolling_mean_6',
            #'month',
            #'year'
           ]
#X = data[features].dropna()

#X = data[features].fillna(data[features].mean())

X = data[features].ffill().bfill()

# Define target variable
y = data.mean(axis=1).loc[X.index]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

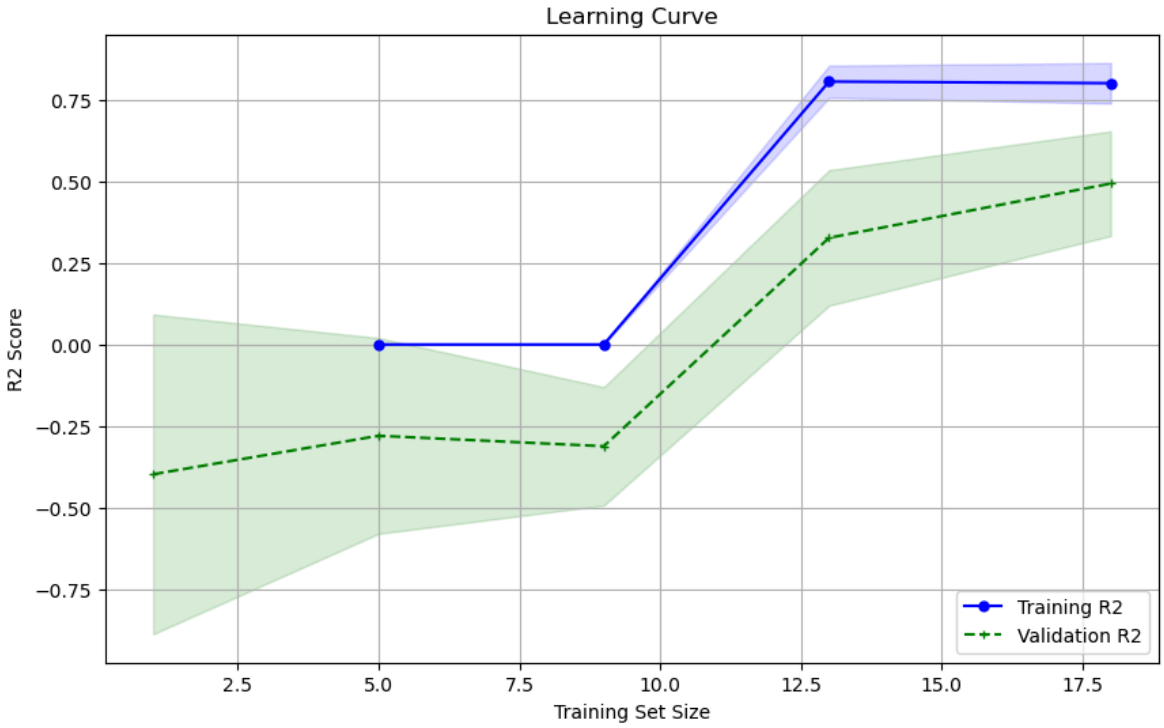
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define hyperparameter search space
param_dist = {
    #'n_estimators': [20, 30, 50,100,150],
    'n_estimators': [20, 30, 50,100],
    'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
    'max_depth': [1, 2, 3,4,5],
    'min_samples_split': [3, 9, 12],
    #'min_samples_split': [1,2, 3],
    'min_samples_leaf': [5, 10, 15],
    'subsample': [0.8, 0.9,1.0]
}

# Perform RandomizedSearchCV
gb = GradientBoostingRegressor(random_state=42)
random_search = RandomizedSearchCV(estimator=gb, param_distributions=param_dist,
                                   n_iter=50, cv=5, random_state=42, n_jobs=-1)
random_search.fit(X_train_scaled, y_train)

# Get best model
best_gb = random_search.best_estimator_

# Evaluate the model
y_train_pred = best_gb.predict(X_train_scaled)
y_test_pred = best_gb.predict(X_test_scaled)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Best parameters: {random_search.best_params_}")
print(f"Training R2: {train_r2:.4f}")
print(f"Test R2: {test_r2:.4f}")

# Feature importance
feature_importance = pd.DataFrame({'feature': features,
                                   'importance': best_gb.feature_importances_})
feature_importance = feature_importance.sort_values('importance', ascending=Fals
print("\nFeature Importance:")
print(feature_importance)

# Plotting Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_test_pred, color='blue', label='Actual vs Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--',
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (Test Set)')
plt.legend()
plt.grid(True)
plt.show()

# Calculate Learning Curve
train_sizes, train_scores, test_scores = learning_curve(
    best_gb, X_train_scaled, y_train, cv=5, scoring='r2', train_sizes=np.linspac

train_mean = np.mean(train_scores, axis=1)
```

```
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Plot Learning Curve
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label=
plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, al
plt.plot(train_sizes, test_mean, color='green', marker='+', markersize=5, linest
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha=
plt.title('Learning Curve')
plt.xlabel('Training Set Size')
plt.ylabel('R2 Score')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
Best parameters: {'subsample': 1.0, 'n_estimators': 30, 'min_samples_split': 9,
'min_samples_leaf': 5, 'max_depth': 4, 'learning_rate': 0.05}
Training R2: 0.8063
Test R2: 0.8102

Feature Importance:
           feature  importance
0   rolling_mean_3    0.978545
1            lag_1    0.021455
2  rolling_mean_12    0.000000
```



```
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:1211: U
ndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:1211: U
ndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:1211: U
ndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:1211: U
ndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
C:\Users\sugan\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:1211: U
ndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```



In [ ]: