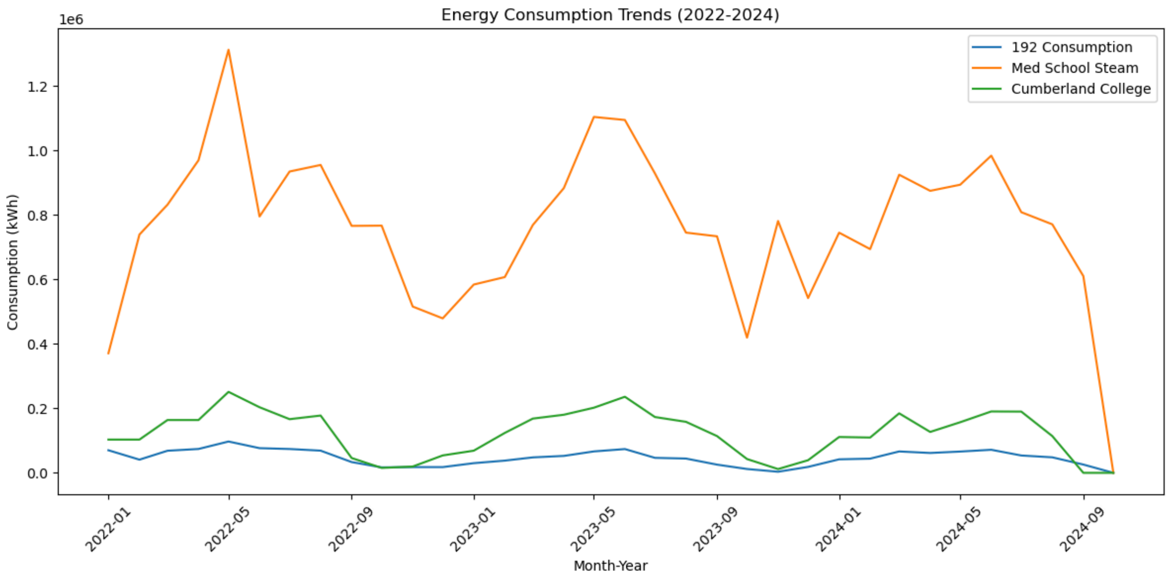
In [1]:
```
import pandas as pd
from datetime import datetime, timedelta

# Read the Excel file
df = pd.read_excel('192 Castle street.xlsx', sheet_name='Steam and MTHW', skiprc

# Create MonthYear column
start_date = datetime(2013, 10, 1)
df['MonthYear'] = [start_date + timedelta(days=30*i) for i in range(len(df))]
df['MonthYear'] = df['MonthYear'].dt.strftime('%b_%Y')

# Rename columns
df = df.rename(columns={
    'MTHW Consumption\nkWh': 'MTHW Consumption kWh',
    '192 Consumption\nKWh': '192 Consumption KWh',
    '  A + B\nkWh': 'Med School Steam',
    'D401& D404  Consumption - kWh': 'Cumberland College'
})

# Select and reorder columns
df = df[['MonthYear', 'MTHW Consumption kWh', '192 Consumption KWh', 'Med School

# First convert columns to numeric and handle null/empty values
df['192 Consumption KWh'] = pd.to_numeric(df['192 Consumption KWh'], errors='coe
df['Med School Steam'] = pd.to_numeric(df['Med School Steam'], errors='coerce').
df['Cumberland College'] = pd.to_numeric(df['Cumberland College'], errors='coerc

# Calculate Total Consumption
df['Total Consumption'] = df['192 Consumption KWh'] + df['Med School Steam'] + d

# Fill NaN values with 0
df = df.fillna(0)

# Display the data types of the columns
print(df.dtypes)

# Display the first few rows of the DataFrame
print(df.head())

# Save the DataFrame to a CSV file
df.to_csv('processed_steam_mthw_data.csv', index=False)
```

```
MonthYear                  object
MTHW Consumption kWh     float64
192 Consumption KWh      float64
Med School Steam         float64
Cumberland College       float64
Total Consumption        float64
dtype: object
  MonthYear  MTHW Consumption kWh  192 Consumption KWh  Med School Steam  \
0  Oct_2013             1461660.0                  0.0        898028.326
1  Oct_2013              890533.0                  0.0        670143.674
2  Nov_2013              698871.0                  0.0        658405.000
3  Dec_2013              742527.0                  0.0        624331.000
4  Jan_2014              768501.0                  0.0        630391.000

   Cumberland College  Total Consumption
0                 0.0         898028.326
1                 0.0         670143.674
2                 0.0         658405.000
3                 0.0         624331.000
4                 0.0         630391.000
```

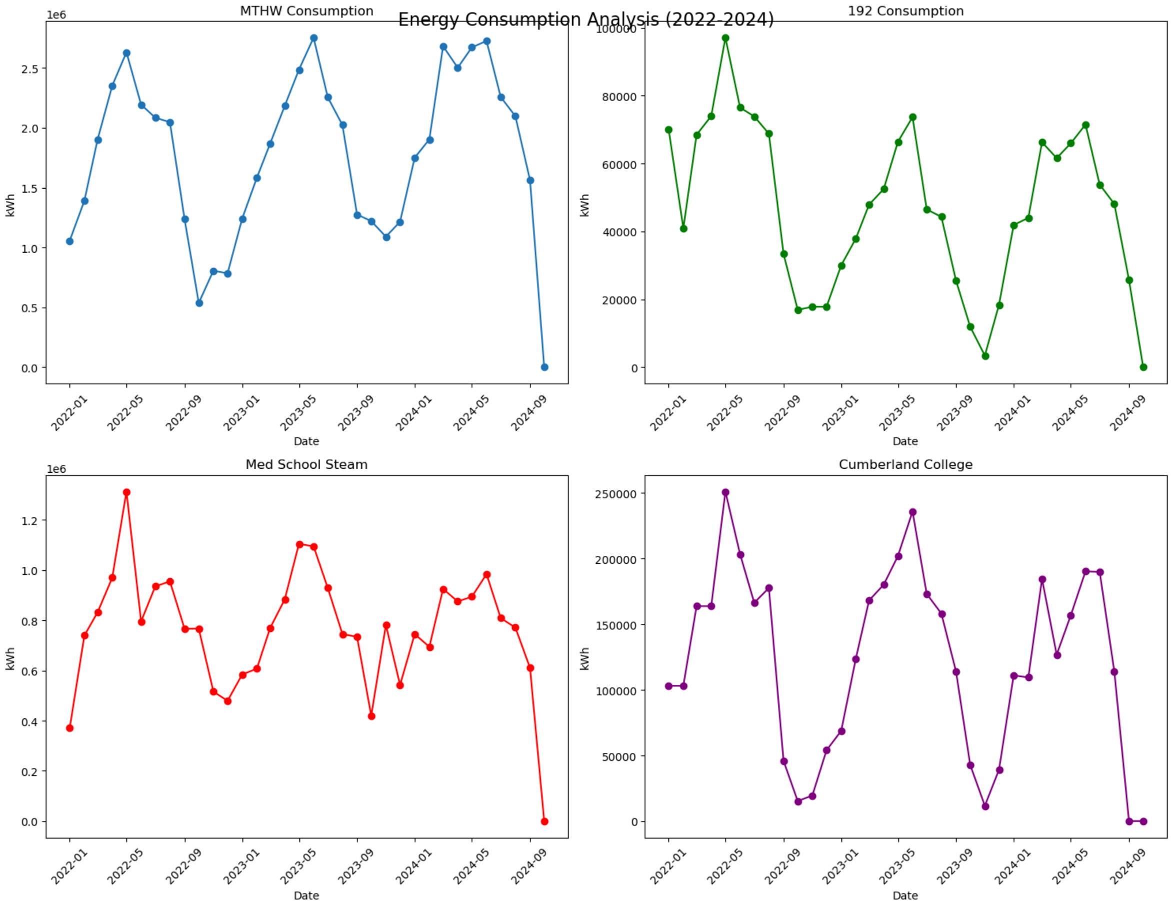
In [2]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data (make sure the CSV file is in the correct directory)
data = pd.read_csv('processed_steam_mthw_data.csv')

# Convert MonthYear to datetime
data['MonthYear'] = pd.to_datetime(data['MonthYear'], format='%b_%Y')

# Filter data for 2022-2024
filtered_data = data[(data['MonthYear'] >= '2022-01-01') & (data['MonthYear'] <=

# Create a line plot for all three consumption types
plt.figure(figsize=(12, 6))
plt.plot(filtered_data['MonthYear'], filtered_data['192 Consumption KWh'], label
plt.plot(filtered_data['MonthYear'], filtered_data['Med School Steam'], label='M
plt.plot(filtered_data['MonthYear'], filtered_data['Cumberland College'], label=

plt.title('Energy Consumption Trends (2022-2024)')
plt.xlabel('Month-Year')
plt.ylabel('Consumption (kWh)')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()

# Calculate summary statistics
summary_stats = filtered_data[['192 Consumption KWh', 'Med School Steam', 'Cumbe
print("\nSummary Statistics:")
print(summary_stats)
```




```
Summary Statistics:
       192 Consumption KWh  Med School Steam  Cumberland College
count            34.000000      3.400000e+01           34.000000
mean          46850.000000      7.625814e+05       122563.010967
std           24206.776537      2.413148e+05        71010.523806
min               0.000000      0.000000e+00            0.000000
25%           26775.000000      6.311175e+05        57668.220339
50%           47200.000000      7.695745e+05       125282.627119
75%           67975.000000      9.167350e+05       176484.322034
max           97100.000000      1.312107e+06       251062.711864
```

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Read the CSV file
df = pd.read_csv('processed_steam_mthw_data.csv')

# Convert MonthYear to datetime
df['MonthYear'] = pd.to_datetime(df['MonthYear'], format='%b_%Y')

# Filter data from Jan 2022 to Dec 2024
mask = (df['MonthYear'] >= '2022-01-01') & (df['MonthYear'] <= '2024-12-31')
filtered_df = df.loc[mask]

# Create subplots for each consumption type
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
fig.suptitle('Energy Consumption Analysis (2022-2024)', fontsize=16, y=0.95)

# Plot MTHW Consumption
axes[0,0].plot(filtered_df['MonthYear'], filtered_df['MTHW Consumption kWh'], ma
axes[0,0].set_title('MTHW Consumption')
axes[0,0].set_xlabel('Date')
axes[0,0].set_ylabel('kWh')
axes[0,0].tick_params(axis='x', rotation=45)

# Plot 192 Consumption
axes[0,1].plot(filtered_df['MonthYear'], filtered_df['192 Consumption KWh'], mar
axes[0,1].set_title('192 Consumption')
axes[0,1].set_xlabel('Date')
axes[0,1].set_ylabel('kWh')
axes[0,1].tick_params(axis='x', rotation=45)
```

```
# Plot Med School Steam
axes[1,0].plot(filtered_df['MonthYear'], filtered_df['Med School Steam'], marker
axes[1,0].set_title('Med School Steam')
axes[1,0].set_xlabel('Date')
axes[1,0].set_ylabel('kWh')
axes[1,0].tick_params(axis='x', rotation=45)

# Plot Cumberland College
axes[1,1].plot(filtered_df['MonthYear'], filtered_df['Cumberland College'], mark
axes[1,1].set_title('Cumberland College')
axes[1,1].set_xlabel('Date')
axes[1,1].set_ylabel('kWh')
axes[1,1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

# Calculate and display summary statistics
print("\nSummary Statistics (2022-2024):")
columns_to_analyze = ['MTHW Consumption kWh', '192 Consumption KWh', 'Med School
print(filtered_df[columns_to_analyze].describe())

# Calculate yearly averages
yearly_avg = filtered_df.groupby(filtered_df['MonthYear'].dt.year)[columns_to_an
print("\nYearly Averages:")
print(yearly_avg)
```

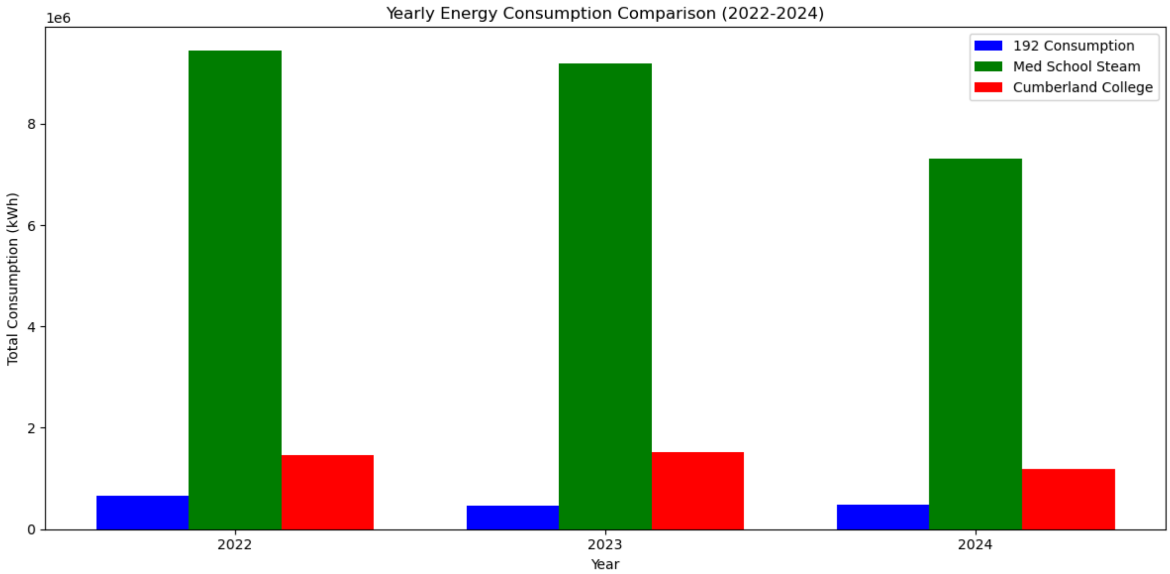


```
Summary Statistics (2022-2024):
       MTHW Consumption kWh  192 Consumption KWh  Med School Steam  \
count          3.400000e+01            34.000000      3.400000e+01
mean           1.775532e+06         46850.000000      7.625814e+05
std            6.970408e+05         24206.776537      2.413148e+05
min            0.000000e+00             0.000000      0.000000e+00
25%            1.238167e+06         26775.000000      6.311175e+05
50%            1.902885e+06         47200.000000      7.695745e+05
75%            2.256635e+06         67975.000000      9.167350e+05
max            2.752910e+06         97100.000000      1.312107e+06

       Cumberland College
count           34.000000
mean        122563.010967
std          71010.523806
min              0.000000
25%          57668.220339
50%         125282.627119
75%         176484.322034
max         251062.711864

Yearly Averages:
           MTHW Consumption kWh  192 Consumption KWh  Med School Steam  \
MonthYear
2022               1.585168e+06         54633.333333     786177.416667
2023               1.766175e+06         38200.000000     765809.333333
2024               2.015197e+06         47890.000000     730392.800000

           Cumberland College
MonthYear
2022            122237.641243
2023            126448.375706
2024            118291.016949
```

In [4]:

```python
import numpy as np

# Read the CSV file
df = pd.read_csv('processed_steam_mthw_data.csv')

# Convert MonthYear to datetime
df['MonthYear'] = pd.to_datetime(df['MonthYear'], format='%b_%Y')

# Calculate yearly sums for each consumption type
yearly_sums = df.groupby(df['MonthYear'].dt.year)[['192 Consumption KWh', 'Med S

# Filter for years 2022-2024
yearly_sums = yearly_sums.loc[2022:2024]

# Set up the bar chart
fig, ax = plt.subplots(figsize=(12, 6))

# Set position of bar on X axis
bar_width = 0.25
r1 = np.arange(len(yearly_sums.index))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]

# Create bars
plt.bar(r1, yearly_sums['192 Consumption KWh'], width=bar_width, label='192 Cons
plt.bar(r2, yearly_sums['Med School Steam'], width=bar_width, label='Med School
```

```
plt.bar(r3, yearly_sums['Cumberland College'], width=bar_width, label='Cumberlan

# Add labels and title
plt.xlabel('Year')
plt.ylabel('Total Consumption (kWh)')
plt.title('Yearly Energy Consumption Comparison (2022-2024)')
plt.xticks([r + bar_width for r in range(len(yearly_sums.index))], yearly_sums.i

# Add legend
plt.legend()

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()

# Print the yearly sums
print("\nYearly Consumption Totals (kWh):")
print(yearly_sums)
```

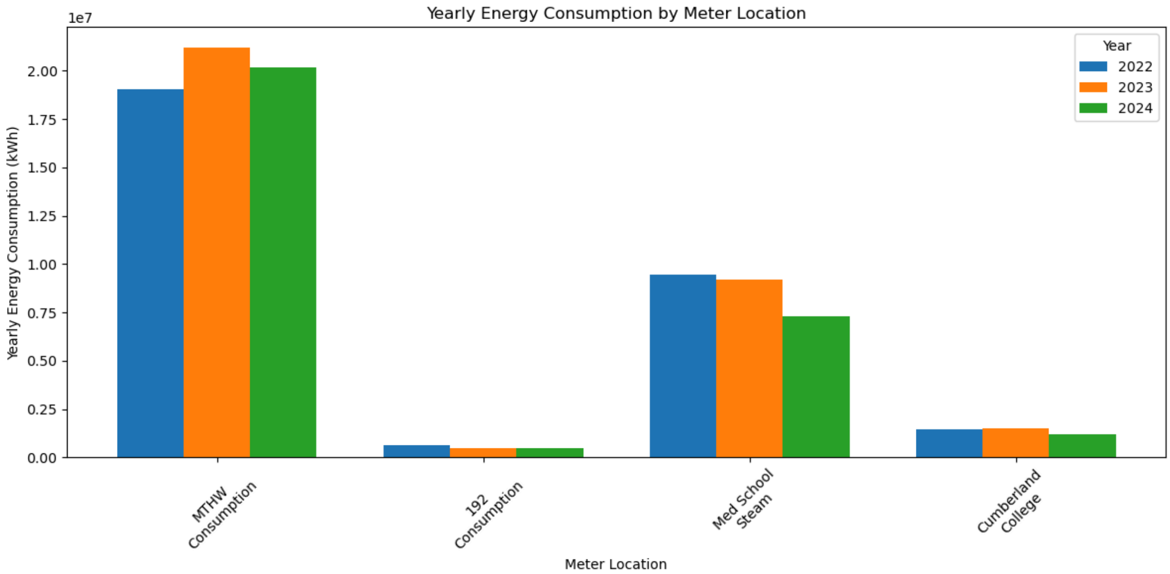



```
Yearly Consumption Totals (kWh):
           192 Consumption KWh  Med School Steam  Cumberland College
MonthYear
2022                  655600.0         9434129.0        1.466852e+06
2023                  458400.0         9189712.0        1.517381e+06
2024                  478900.0         7303928.0        1.182910e+06
```

In [5]:

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

# Read the CSV file
df = pd.read_csv('processed_steam_mthw_data.csv')

# Convert MonthYear to datetime
df['MonthYear'] = pd.to_datetime(df['MonthYear'], format='%b_%Y')

# Add Year column
df['Year'] = df['MonthYear'].dt.year

# Calculate yearly sums for each consumption type
yearly_sums = df.groupby('Year')[['MTHW Consumption kWh', '192 Consumption KWh',
```

```
                                  'Med School Steam', 'Cumberland College']].sum(

# Filter for years 2022-2024
yearly_sums = yearly_sums.loc[2022:2024]

# Set up the plot
fig, ax = plt.subplots(figsize=(12, 6))

# Set width of bars and positions
bar_width = 0.25
x = np.arange(len(yearly_sums.columns))

# Create bars for each year with different colors
plt.bar(x - bar_width, yearly_sums.loc[2022], bar_width, label='2022', color='#1
plt.bar(x, yearly_sums.loc[2023], bar_width, label='2023', color='#ff7f0e')
plt.bar(x + bar_width, yearly_sums.loc[2024], bar_width, label='2024', color='#2

# Customize the plot
plt.xlabel('Meter Location')
plt.ylabel('Yearly Energy Consumption (kWh)')
plt.title('Yearly Energy Consumption by Meter Location')
plt.xticks(x, ['MTHW\nConsumption', '192\nConsumption', 'Med School\nSteam', 'Cu
           rotation=45)
plt.legend(title='Year')

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()

# Print yearly totals
print("\nYearly Consumption Totals (kWh):")
print(yearly_sums)
```



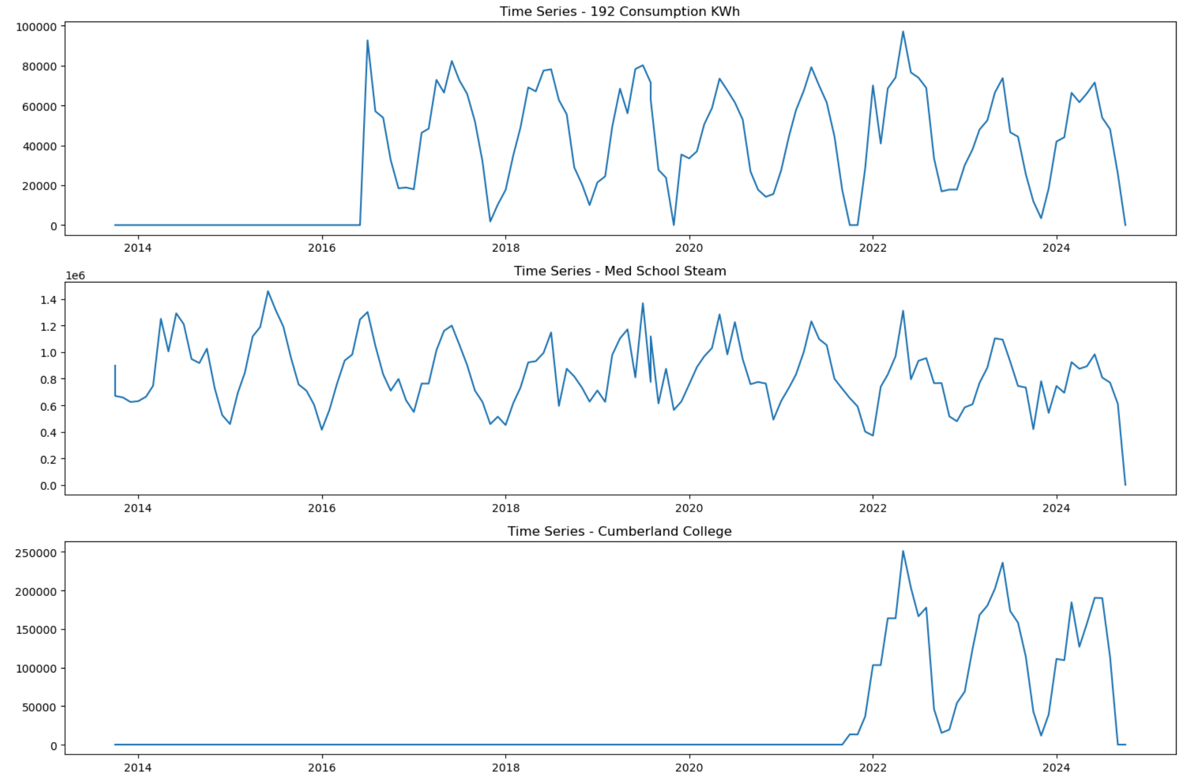
```
Yearly Consumption Totals (kWh):
      MTHW Consumption kWh  192 Consumption KWh  Med School Steam  \
Year
2022          19022018.0             655600.0         9434129.0
2023          21194104.0             458400.0         9189712.0
2024          20151971.0             478900.0         7303928.0

      Cumberland College
Year
2022        1.466852e+06
2023        1.517381e+06
2024        1.182910e+06
```

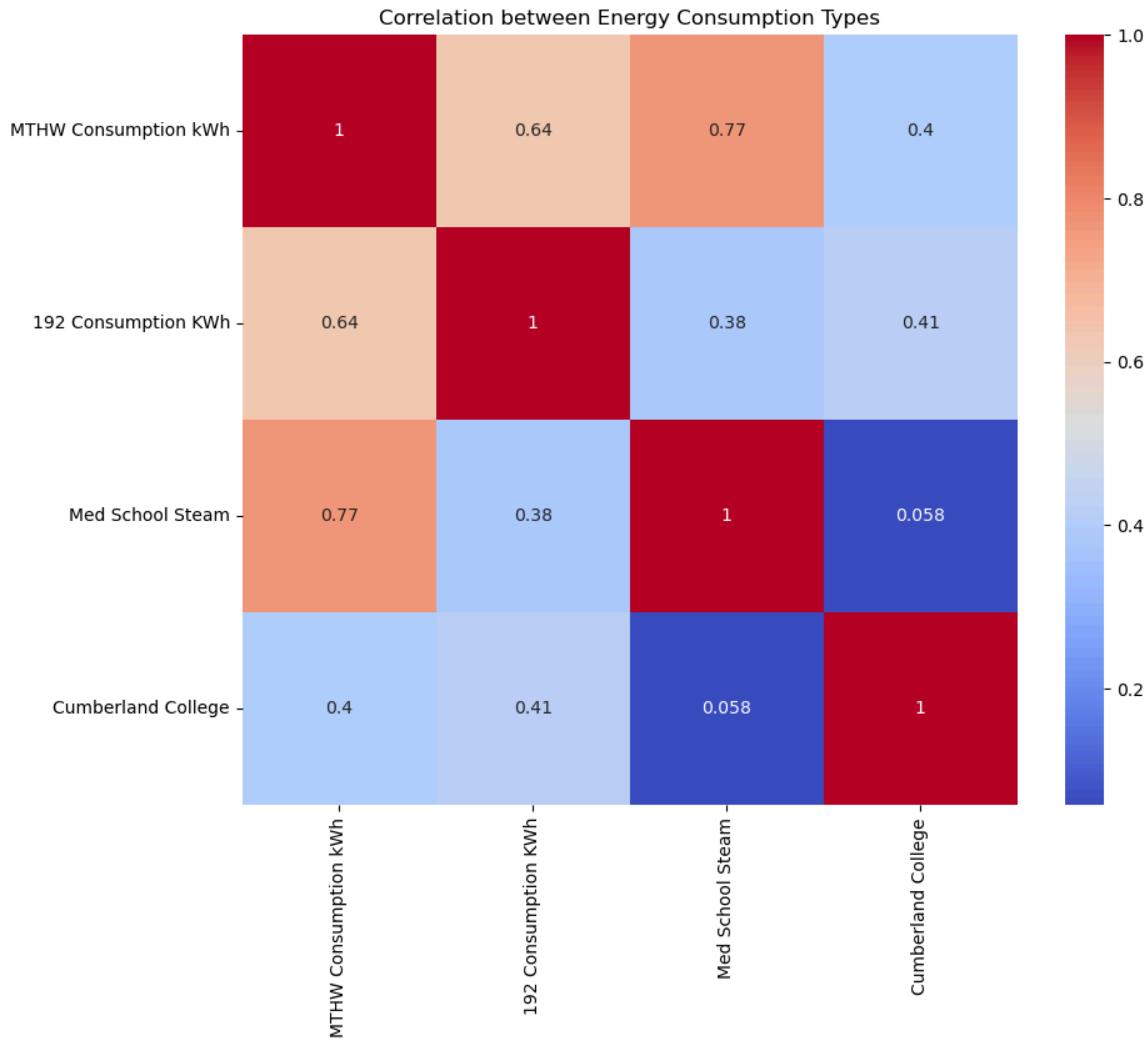
In [16]:

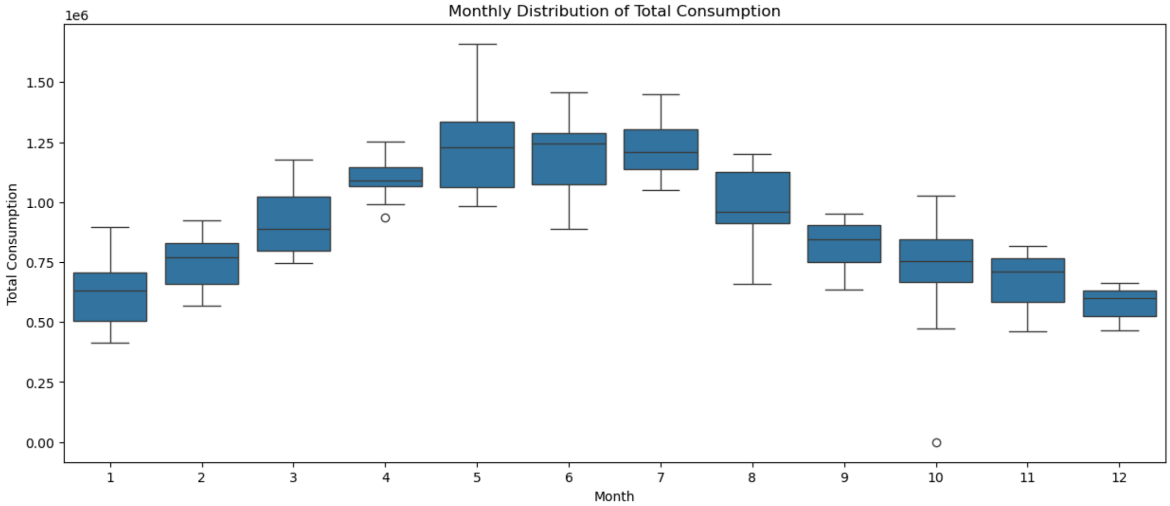
```
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy import stats

# Read the data
df = pd.read_csv('processed_steam_mthw_data.csv')
df['MonthYear'] = pd.to_datetime(df['MonthYear'], format='%b_%Y')

# 1. Time Series Analysis
def plot_seasonal_decomposition():
    plt.figure(figsize=(15, 10))
    for column in ['192 Consumption KWh', 'Med School Steam', 'Cumberland Colleg
        result = seasonal_decompose(df[column].fillna(0), period=12)
        plt.subplot(3, 1, ['192 Consumption KWh', 'Med School Steam', 'Cumberlan
        plt.plot(df['MonthYear'], df[column])
        plt.title(f'Time Series - {column}')
    plt.tight_layout()
    plt.show()

# 2. Correlation Analysis
def plot_correlation():
    correlation_matrix = df[['MTHW Consumption kWh', '192 Consumption KWh',
                            'Med School Steam', 'Cumberland College']].corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
    plt.title('Correlation between Energy Consumption Types')
    plt.show()

# 3. Seasonal Analysis
def plot_seasonal_patterns():
    df['Month'] = df['MonthYear'].dt.month
    df['Season'] = pd.cut(df['MonthYear'].dt.month,
                         bins=[0,3,6,9,12],
                         labels=['Winter','Spring','Summer','Autumn'])

    plt.figure(figsize=(15, 6))
    sns.boxplot(x='Month', y='Total Consumption', data=df)
    plt.title('Monthly Distribution of Total Consumption')
    plt.show()

# 4. Year-over-Year Analysis
def plot_yearly_comparison():
    df['Year'] = df['MonthYear'].dt.year
    yearly_data = df.groupby('Year')[['192 Consumption KWh', 'Med School Steam',
                                     'Cumberland College']].sum()

    yearly_data.plot(kind='bar', figsize=(12, 6))
```

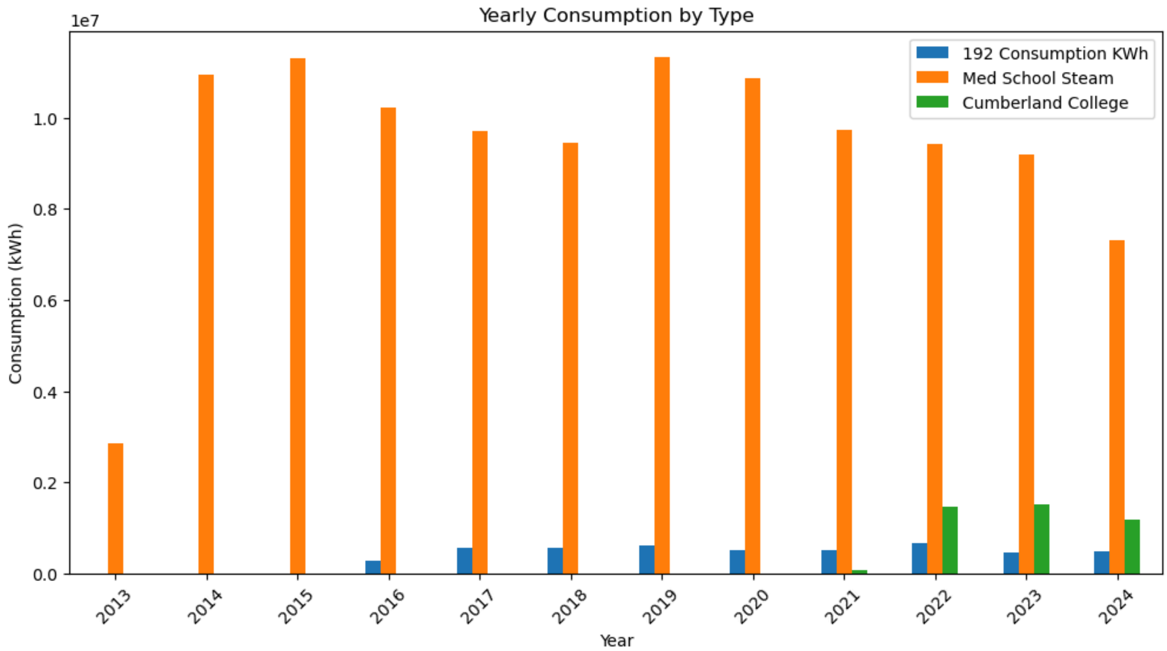
```
    plt.title('Yearly Consumption by Type')
    plt.ylabel('Consumption (kWh)')
    plt.xticks(rotation=45)
    plt.show()

# 5. Anomaly Detection
def detect_anomalies(threshold=3):
    anomalies = {}
    for column in ['192 Consumption KWh', 'Med School Steam', 'Cumberland Colleg
        z_scores = np.abs(stats.zscore(df[column].fillna(0)))
        anomalies[column] = df[z_scores > threshold]

    return anomalies

# 6. Consumption Pattern Analysis
def plot_consumption_distribution():
    plt.figure(figsize=(15, 8))
    plt.stackplot(df['MonthYear'],
                  df['192 Consumption KWh'],
                  df['Med School Steam'],
                  df['Cumberland College'],
                  labels=['192 Consumption', 'Med School Steam', 'Cumberland Coll
    plt.title('Energy Consumption Distribution Over Time')
    plt.legend(loc='upper left')
    plt.xlabel('Date')
    plt.ylabel('Consumption (kWh)')
    plt.show()

# Execute all analyses
if __name__ == "__main__":
    print("Generating all visualizations...")
    plot_seasonal_decomposition()
    plot_correlation()
    plot_seasonal_patterns()
    plot_yearly_comparison()
    plot_consumption_distribution()

    # Print anomalies
    anomalies = detect_anomalies()
    for meter, anomaly_data in anomalies.items():
        print(f"\nAnomalies detected for {meter}:")
        print(anomaly_data[['MonthYear', meter]])
```

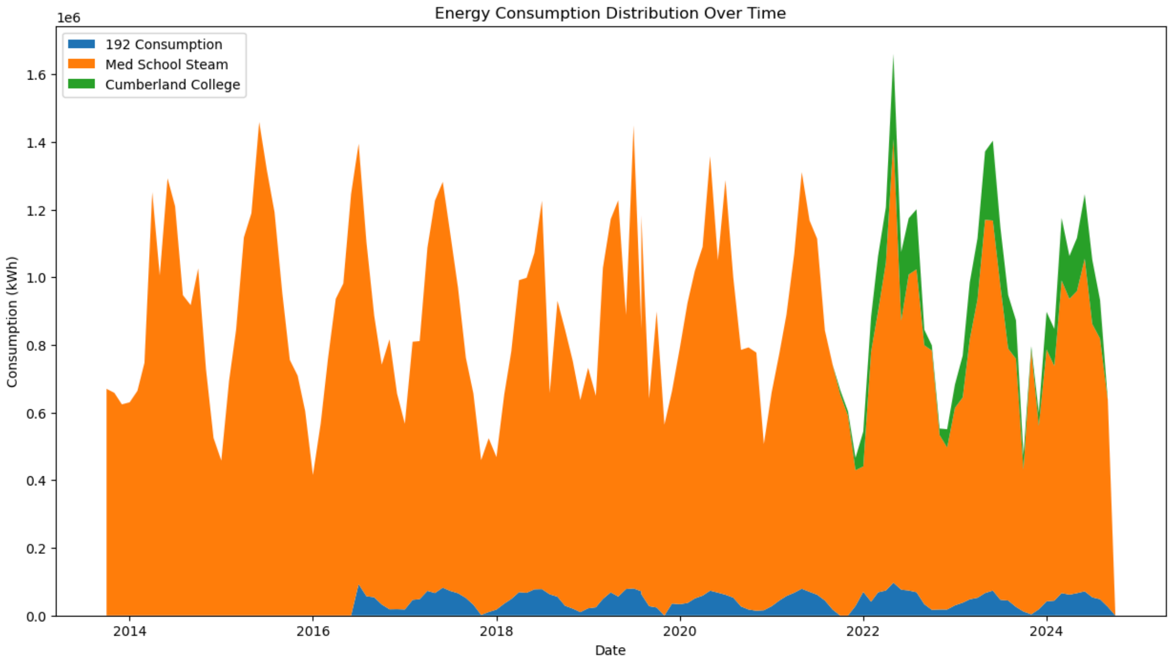
```
Generating all visualizations...
```


Time Series - 192 Consumption KWh
Time Series - Med School Steam
1e6
Time Series - Cumberland College
2014
2016
2018
2020
2022
2024


Correlation between Energy Consumption Types
MTHW Consumption kWh
192 Consumption KWh
Med School Steam
Cumberland College
1
0.64
0.77
0.4
0.38
0.41
0.058


Monthly Distribution of Total Consumption
1e6
Total Consumption
Month



Yearly Consumption by Type
1e7
192 Consumption KWh
Med School Steam
Cumberland College
Consumption (kWh)
Year

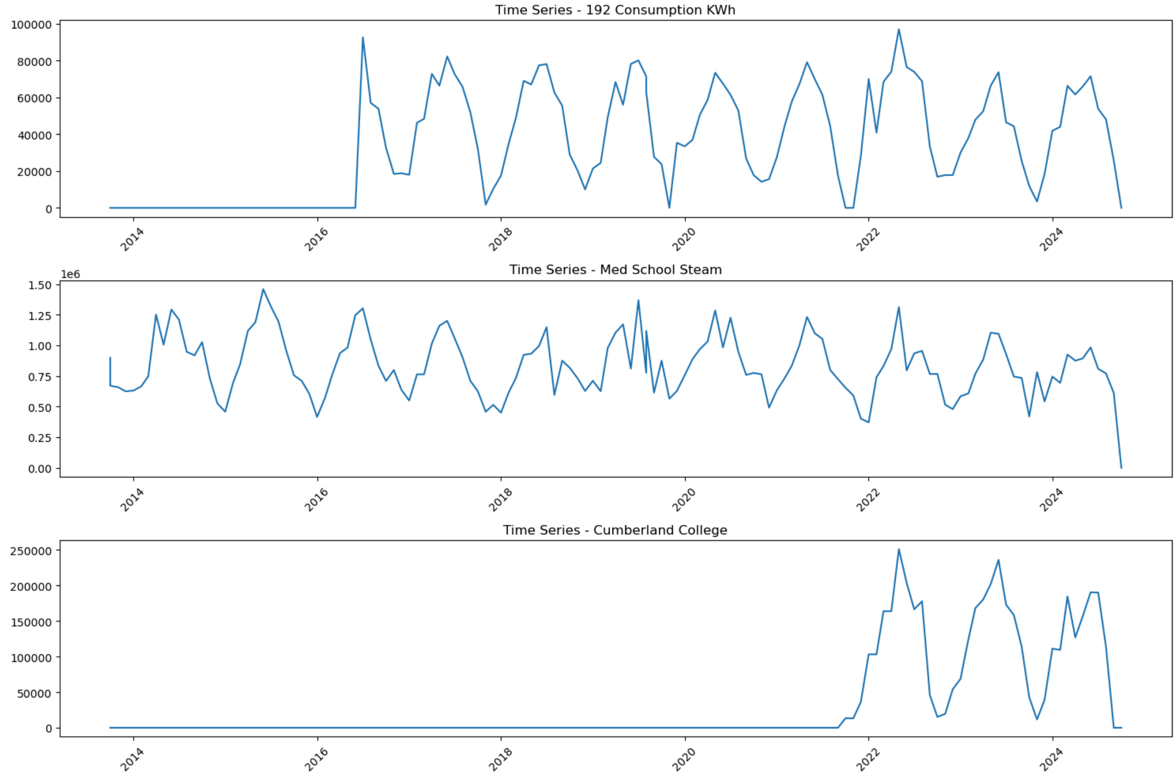


Energy Consumption Distribution Over Time
1e6
192 Consumption
Med School Steam
Cumberland College
Consumption (kWh)
Date

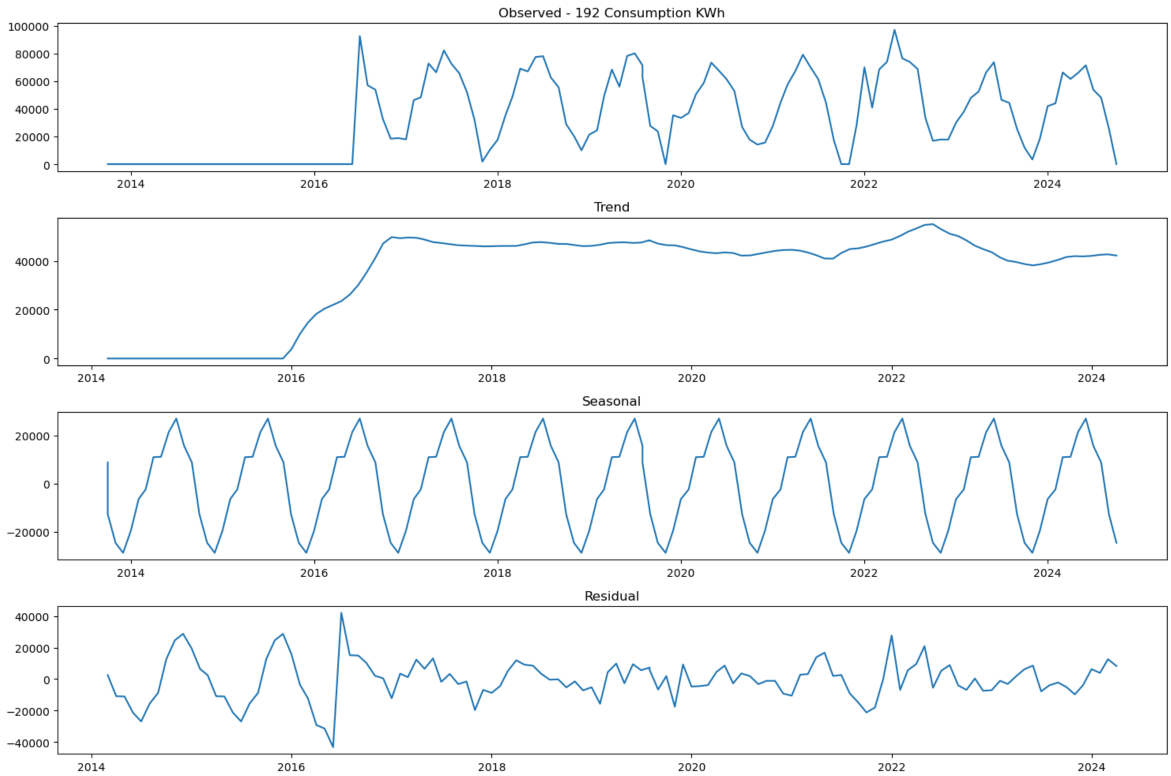
```
Anomalies detected for 192 Consumption KWh:
Empty DataFrame
Columns: [MonthYear, 192 Consumption KWh]
Index: []

Anomalies detected for Med School Steam:
     MonthYear  Med School Steam
134 2024-10-01               0.0

Anomalies detected for Cumberland College:
     MonthYear  Cumberland College
105 2022-05-01       251062.711864
118 2023-06-01       235916.101695
```

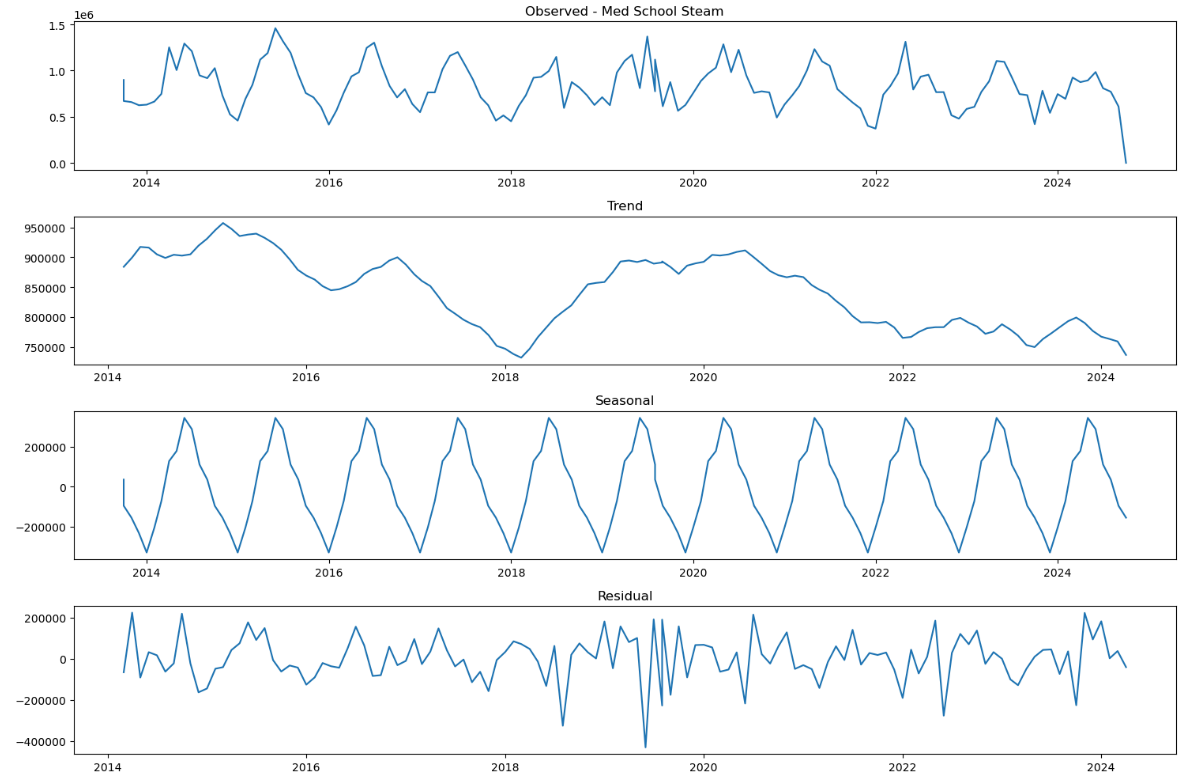
In [23]:

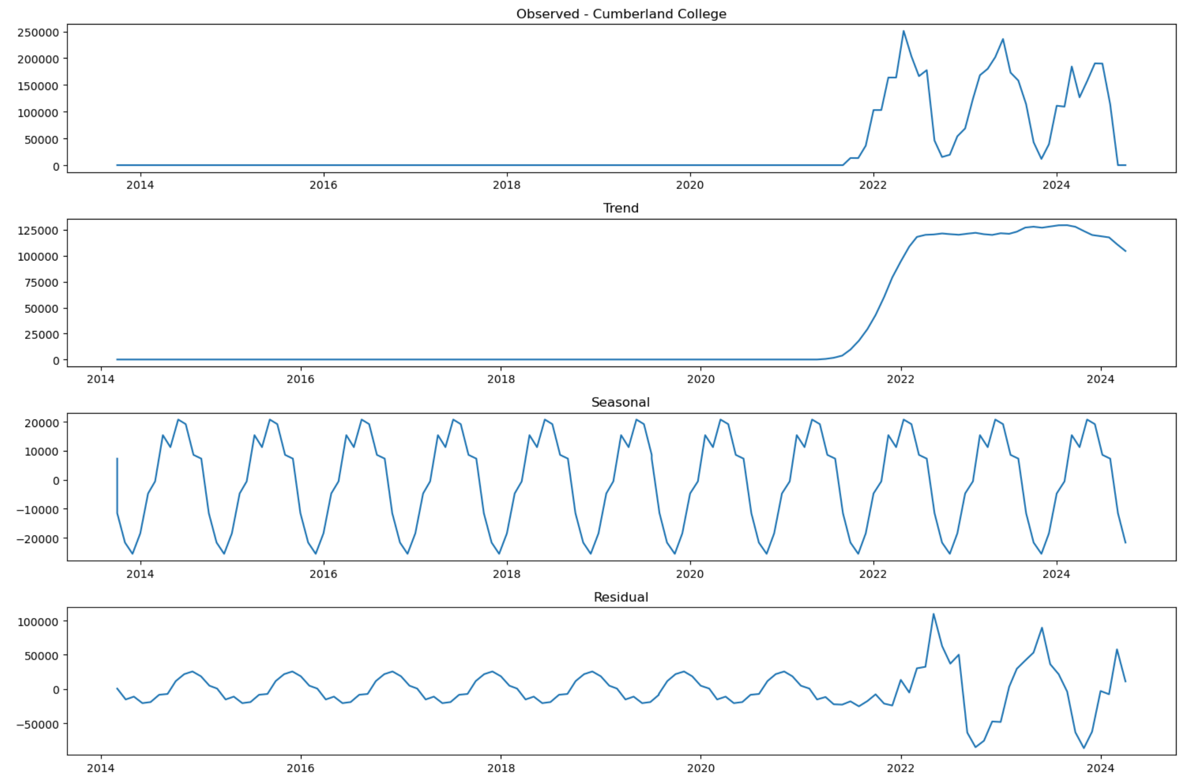
```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Read the data
df = pd.read_csv('processed_steam_mthw_data.csv')
df['MonthYear'] = pd.to_datetime(df['MonthYear'], format='%b_%Y')

# Create individual plots for each consumption type
plt.figure(figsize=(15, 10))

# Plot for 192 Consumption
plt.subplot(3, 1, 1)
plt.plot(df['MonthYear'], df['192 Consumption KWh'])
plt.title('Time Series - 192 Consumption KWh')
plt.xticks(rotation=45)

# Plot for Med School Steam
plt.subplot(3, 1, 2)
plt.plot(df['MonthYear'], df['Med School Steam'])
plt.title('Time Series - Med School Steam')
plt.xticks(rotation=45)

# Plot for Cumberland College
plt.subplot(3, 1, 3)
plt.plot(df['MonthYear'], df['Cumberland College'])
plt.title('Time Series - Cumberland College')
plt.xticks(rotation=45)

# Adjust layout to prevent overlap
plt.tight_layout()

# Display the plot
plt.show()

# For seasonal decomposition of each column
for column in ['192 Consumption KWh', 'Med School Steam', 'Cumberland College']:
    plt.figure(figsize=(15, 10))
    result = seasonal_decompose(df[column].fillna(0), period=12)

    # Plot decomposition
    plt.subplot(4, 1, 1)
    plt.plot(df['MonthYear'], result.observed)
    plt.title(f'Observed - {column}')

    plt.subplot(4, 1, 2)
```

```python
    plt.plot(df['MonthYear'], result.trend)
    plt.title('Trend')

    plt.subplot(4, 1, 3)
    plt.plot(df['MonthYear'], result.seasonal)
    plt.title('Seasonal')

    plt.subplot(4, 1, 4)
    plt.plot(df['MonthYear'], result.resid)
    plt.title('Residual')

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





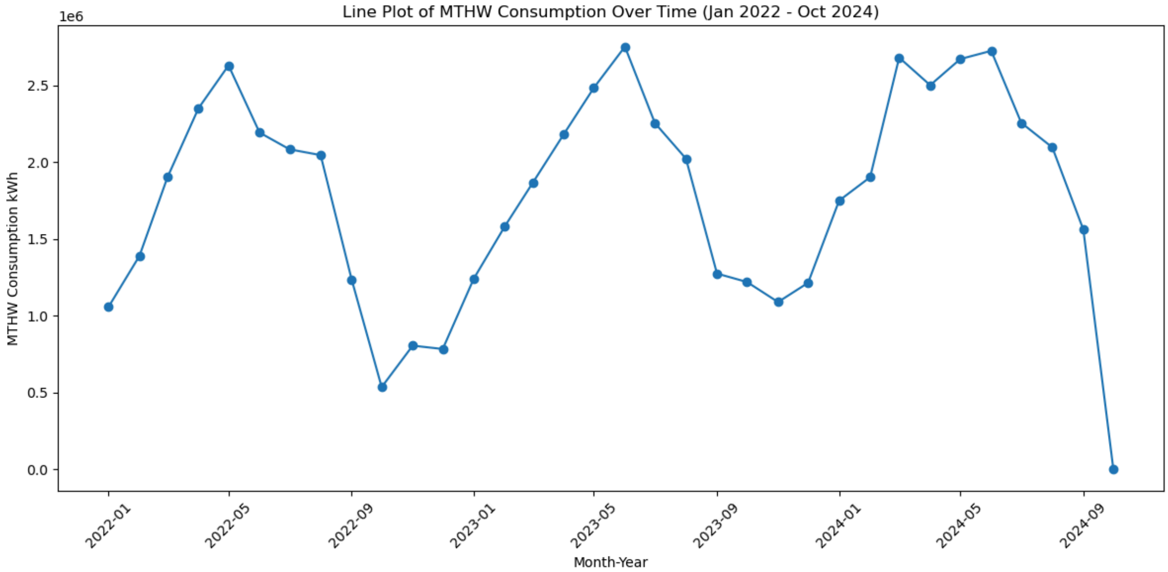

In [ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

# Load the dataset
data = pd.read_csv("processed_steam_mthw_data.csv")

# Filter data for the required time range (Jan 2022 - Oct 2024)
data['MonthYear'] = pd.to_datetime(data['MonthYear'], format='%b_%Y')
filtered_data = data[(data['MonthYear'] >= '2022-01-01') & (data['MonthYear'] <=

# Extract relevant columns
```

```
filtered_data = filtered_data[['MonthYear', 'MTHW Consumption kWh']]
filtered_data.set_index('MonthYear', inplace=True)

# Line Plot
plt.figure(figsize=(12, 6))
plt.plot(filtered_data.index, filtered_data['MTHW Consumption kWh'], marker='o')
plt.title('Line Plot of MTHW Consumption Over Time (Jan 2022 - Oct 2024)')
plt.xlabel('Month-Year')
plt.ylabel('MTHW Consumption kWh')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# QQ Plot
plt.figure(figsize=(8, 6))
stats.probplot(filtered_data['MTHW Consumption kWh'], dist="norm", plot=plt)
plt.title('QQ Plot of MTHW Consumption (Jan 2022 - Oct 2024)')
plt.tight_layout()
plt.show()

# Bar Chart with Whiskers (Standard Deviation as Whiskers)
plt.figure(figsize=(12, 6))
filtered_data['MTHW Consumption kWh'].plot(kind='bar', yerr=filtered_data['MTHW
plt.title('Bar Chart with Whiskers for MTHW Consumption (Jan 2022 - Oct 2024)')
plt.xlabel('Month-Year')
plt.ylabel('MTHW Consumption kWh')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Violin Plot
plt.figure(figsize=(8, 6))
sns.violinplot(data=filtered_data, y='MTHW Consumption kWh')
plt.title('Violin Plot of MTHW Consumption (Jan 2022 - Oct 2024)')
plt.ylabel('MTHW Consumption kWh')
plt.tight_layout()
plt.show()

# Density Curve
plt.figure(figsize=(8, 6))
sns.kdeplot(filtered_data['MTHW Consumption kWh'], shade=True)
plt.title('Density Curve of MTHW Consumption (Jan 2022 - Oct 2024)')
plt.xlabel('MTHW Consumption kWh')
plt.tight_layout()
plt.show()
```

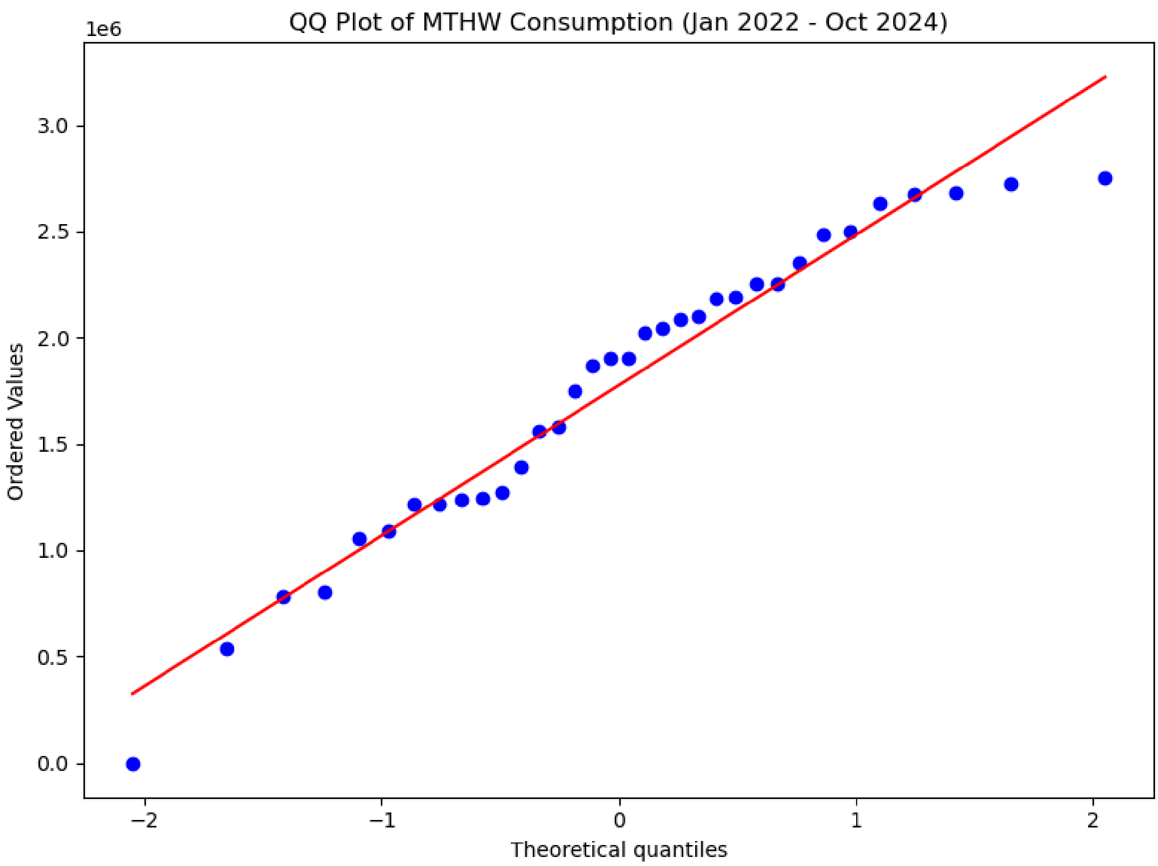
In [25]:

```
# Load the dataset
data = pd.read_csv("processed_steam_mthw_data.csv")

# Filter data for the required time range (Jan 2022 - Oct 2024)
data['MonthYear'] = pd.to_datetime(data['MonthYear'], format='%b_%Y')
filtered_data = data[(data['MonthYear'] >= '2022-01-01') & (data['MonthYear'] <=

# Extract relevant columns
filtered_data = filtered_data[['MonthYear', 'MTHW Consumption kWh']]
filtered_data.set_index('MonthYear', inplace=True)

# Line Plot
plt.figure(figsize=(12, 6))
plt.plot(filtered_data.index, filtered_data['MTHW Consumption kWh'], marker='o')
```

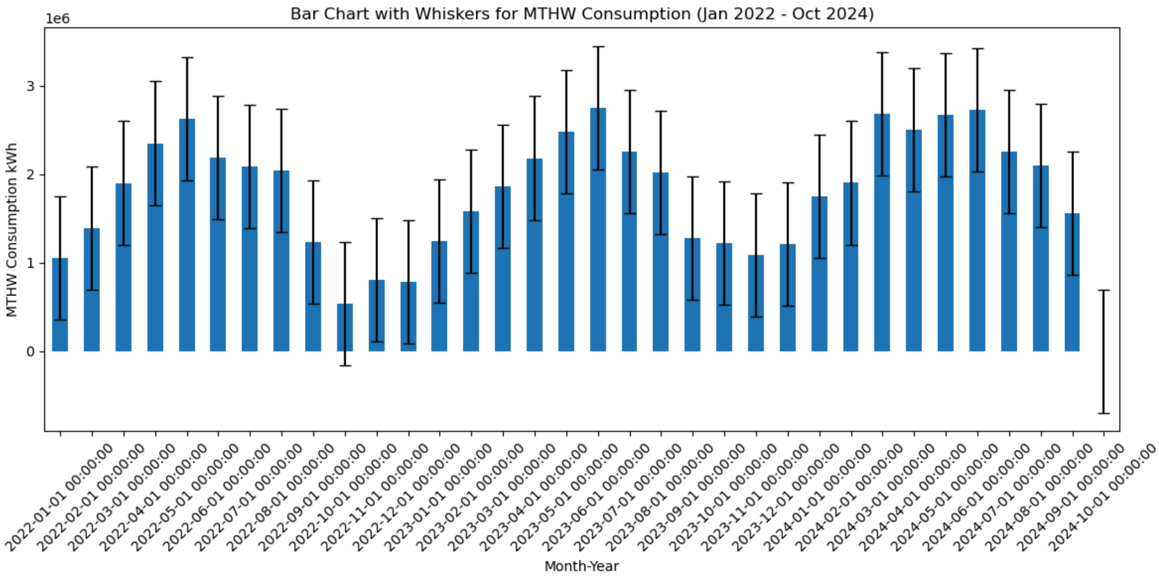
```
plt.title('Line Plot of MTHW Consumption Over Time (Jan 2022 - Oct 2024)')
plt.xlabel('Month-Year')
plt.ylabel('MTHW Consumption kWh')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



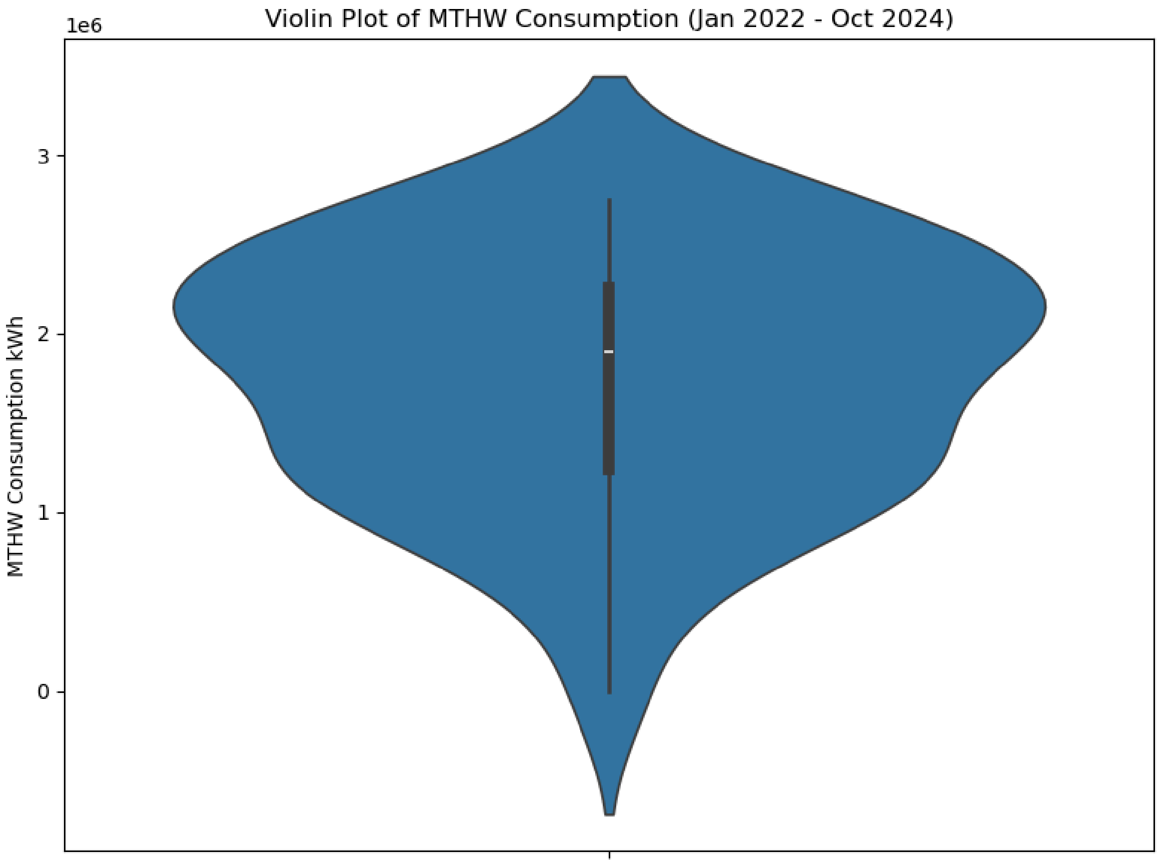

In [29]:
```
# QQ Plot
plt.figure(figsize=(8, 6))
stats.probplot(filtered_data['MTHW Consumption kWh'], dist="norm", plot=plt)
plt.title('QQ Plot of MTHW Consumption (Jan 2022 - Oct 2024)')
plt.tight_layout()
plt.show()
```



In [31]:
```
# Bar Chart with Whiskers (Standard Deviation as Whiskers)
plt.figure(figsize=(12, 6))
filtered_data['MTHW Consumption kWh'].plot(kind='bar', yerr=filtered_data['MTHW
plt.title('Bar Chart with Whiskers for MTHW Consumption (Jan 2022 - Oct 2024)')
plt.xlabel('Month-Year')
plt.ylabel('MTHW Consumption kWh')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

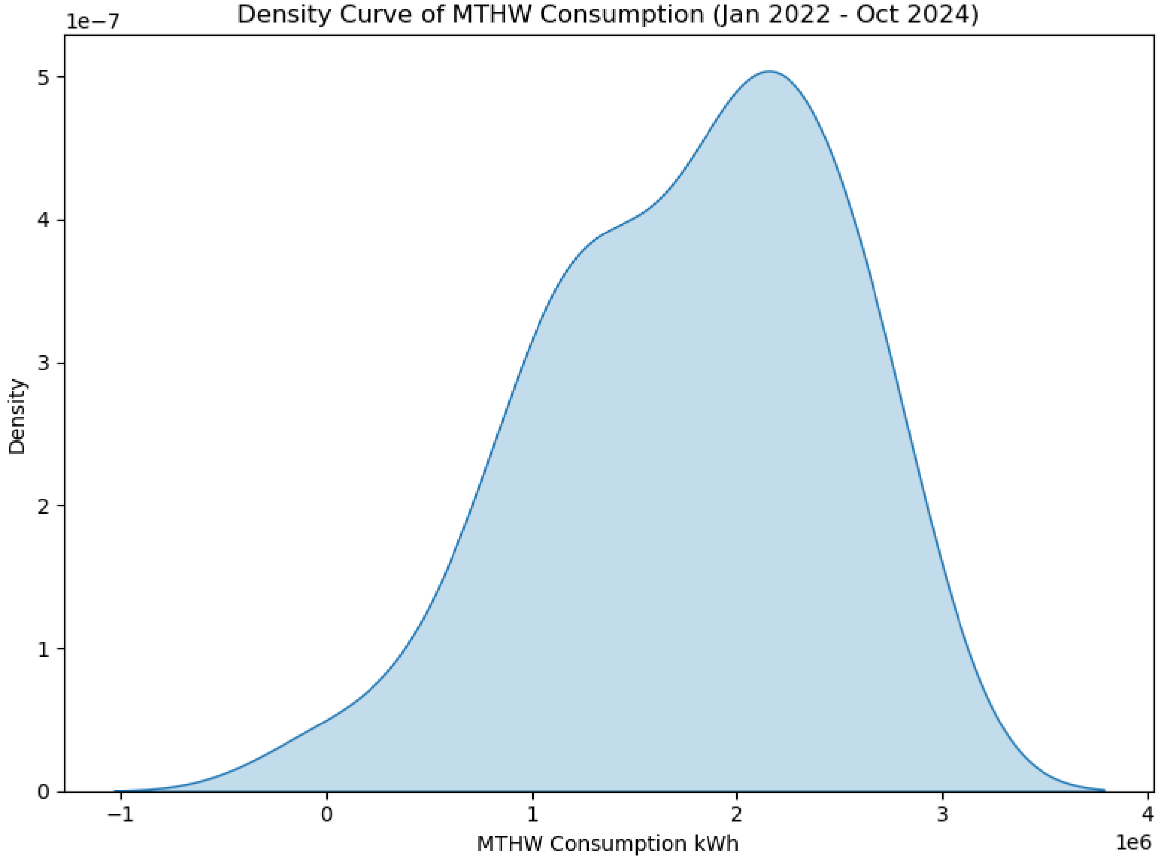



In [33]:
```
# Violin Plot
plt.figure(figsize=(8, 6))
sns.violinplot(data=filtered_data, y='MTHW Consumption kWh')
plt.title('Violin Plot of MTHW Consumption (Jan 2022 - Oct 2024)')
plt.ylabel('MTHW Consumption kWh')
plt.tight_layout()
plt.show()
```




In [35]:
```
# Density Curve
plt.figure(figsize=(8, 6))
sns.kdeplot(filtered_data['MTHW Consumption kWh'], shade=True)
plt.title('Density Curve of MTHW Consumption (Jan 2022 - Oct 2024)')
plt.xlabel('MTHW Consumption kWh')
plt.tight_layout()
plt.show()
```

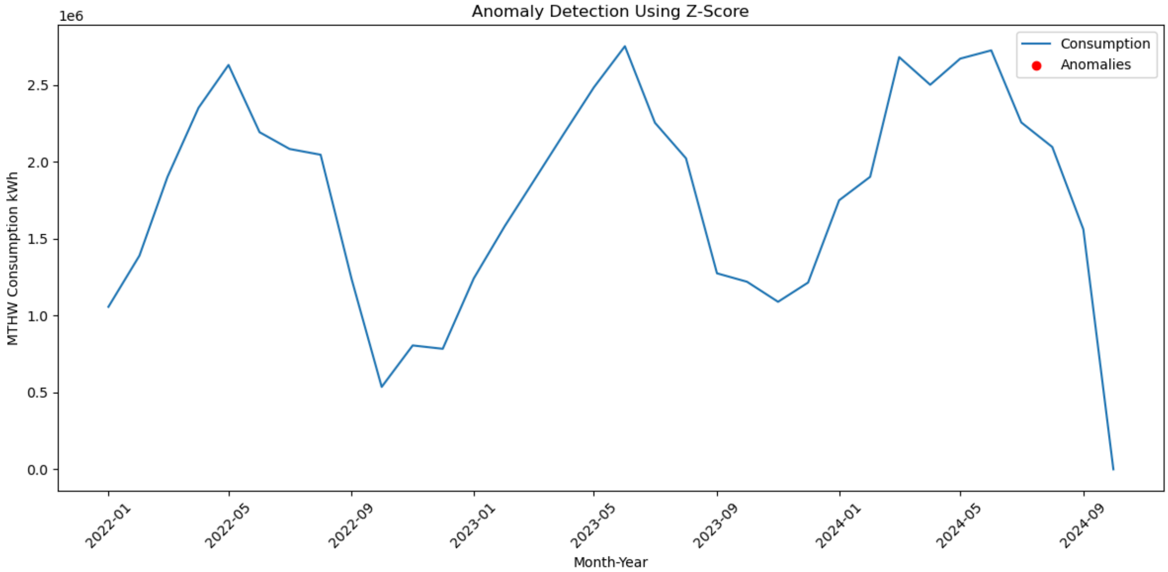
```
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\2157925163.py:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(filtered_data['MTHW Consumption kWh'], shade=True)
```




In [37]:

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

# Load the dataset (ensure it's filtered for the required time range)
data = pd.read_csv("processed_steam_mthw_data.csv")
data['MonthYear'] = pd.to_datetime(data['MonthYear'], format='%b_%Y')
filtered_data = data[(data['MonthYear'] >= '2022-01-01') & (data['MonthYear'] <=

# Calculate Z-Scores
mean = filtered_data['MTHW Consumption kWh'].mean()
std_dev = filtered_data['MTHW Consumption kWh'].std()
filtered_data['Z-Score'] = (filtered_data['MTHW Consumption kWh'] - mean) / std_

# Define threshold for anomalies
threshold = 3
filtered_data['Anomaly_Z'] = filtered_data['Z-Score'].apply(lambda x: 'Anomaly'

# Plot anomalies
plt.figure(figsize=(12, 6))
plt.plot(filtered_data['MonthYear'], filtered_data['MTHW Consumption kWh'], labe
anomalies = filtered_data[filtered_data['Anomaly_Z'] == 'Anomaly']
plt.scatter(anomalies['MonthYear'], anomalies['MTHW Consumption kWh'], color='re
plt.title('Anomaly Detection Using Z-Score')
plt.xlabel('Month-Year')
plt.ylabel('MTHW Consumption kWh')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\3831728223.py:13: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_data['Z-Score'] = (filtered_data['MTHW Consumption kWh'] - mean) / std
_dev
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\3831728223.py:17: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_data['Anomaly_Z'] = filtered_data['Z-Score'].apply(lambda x: 'Anomaly'
if abs(x) > threshold else 'Normal')
```

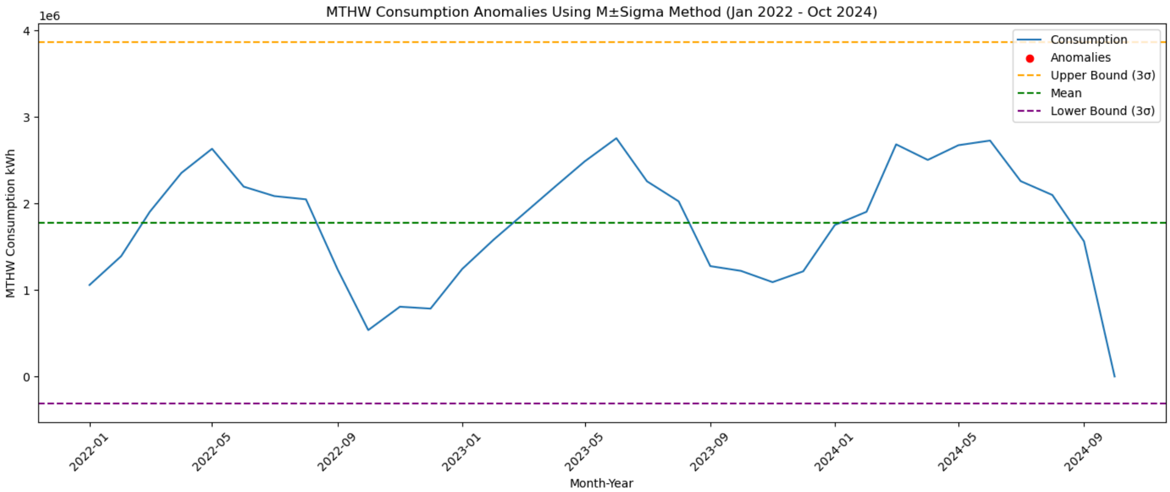



In [39]:

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

# Load and filter data
data = pd.read_csv("processed_steam_mthw_data.csv")
data['MonthYear'] = pd.to_datetime(data['MonthYear'], format='%b_%Y')
filtered_data = data[(data['MonthYear'] >= '2022-01-01') & (data['MonthYear'] <=

# Calculate mean and standard deviation
mean = filtered_data['MTHW Consumption kWh'].mean()
std_dev = filtered_data['MTHW Consumption kWh'].std()

# Define thresholds (using k=3 for 3σ)
k = 3
upper_bound = mean + k * std_dev
lower_bound = mean - k * std_dev

# Flag anomalies
filtered_data['Anomaly'] = filtered_data['MTHW Consumption kWh'].apply(
    lambda x: 'Anomaly' if (x > upper_bound) or (x < lower_bound) else 'Normal'
)

# Plot results
```

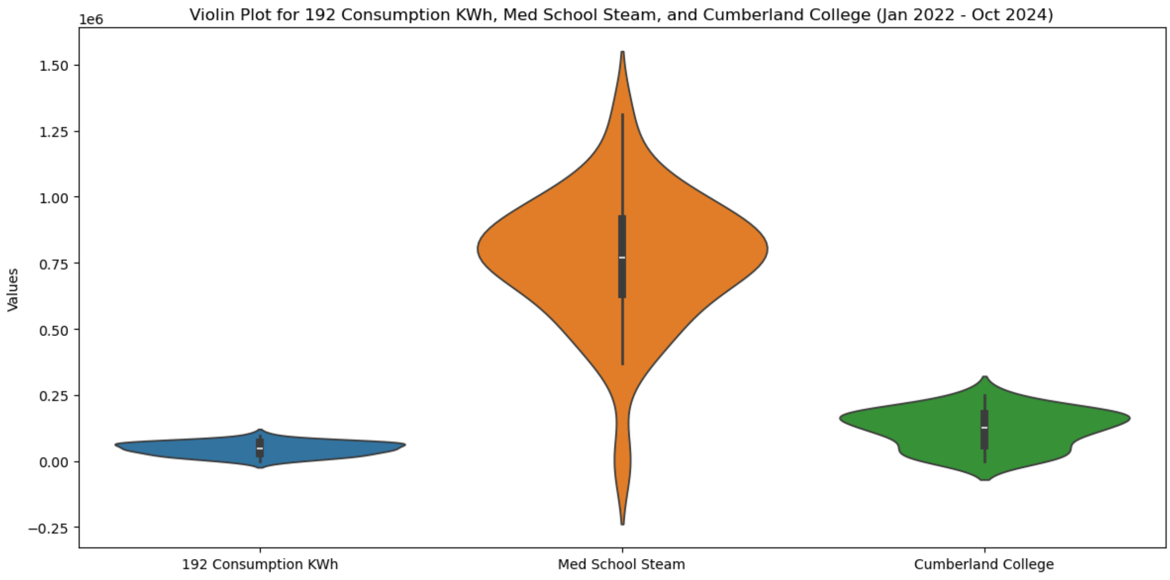
```
plt.figure(figsize=(14, 6))
plt.plot(filtered_data['MonthYear'], filtered_data['MTHW Consumption kWh'], labe
anomalies = filtered_data[filtered_data['Anomaly'] == 'Anomaly']
plt.scatter(anomalies['MonthYear'], anomalies['MTHW Consumption kWh'], color='re
plt.axhline(upper_bound, color='orange', linestyle='--', label=f'Upper Bound ({k
plt.axhline(mean, color='green', linestyle='--', label='Mean')
plt.axhline(lower_bound, color='purple', linestyle='--', label=f'Lower Bound ({k
plt.title('MTHW Consumption Anomalies Using M±Sigma Method (Jan 2022 - Oct 2024)
plt.xlabel('Month-Year')
plt.ylabel('MTHW Consumption kWh')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Display anomalies
print("Identified Anomalies:")
print(anomalies[['MonthYear', 'MTHW Consumption kWh']])
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\882568294.py:19: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered_data['Anomaly'] = filtered_data['MTHW Consumption kWh'].apply(
```

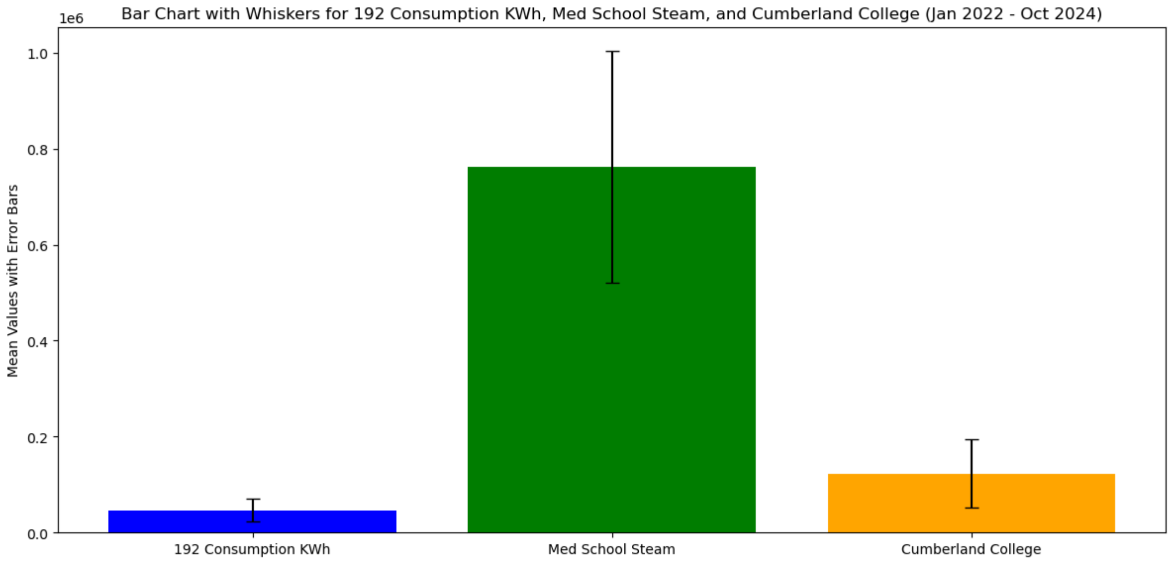



```
Identified Anomalies:
Empty DataFrame
Columns: [MonthYear, MTHW Consumption kWh]
Index: []
```

In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a violin plot
plt.figure(figsize=(12, 6))
sns.violinplot(data=filtered_data[['192 Consumption KWh', 'Med School Steam', 'C
plt.title('Violin Plot for 192 Consumption KWh, Med School Steam, and Cumberland
plt.ylabel('Values')
plt.xticks([0, 1, 2], ['192 Consumption KWh', 'Med School Steam', 'Cumberland Co
plt.tight_layout()
plt.show()
```



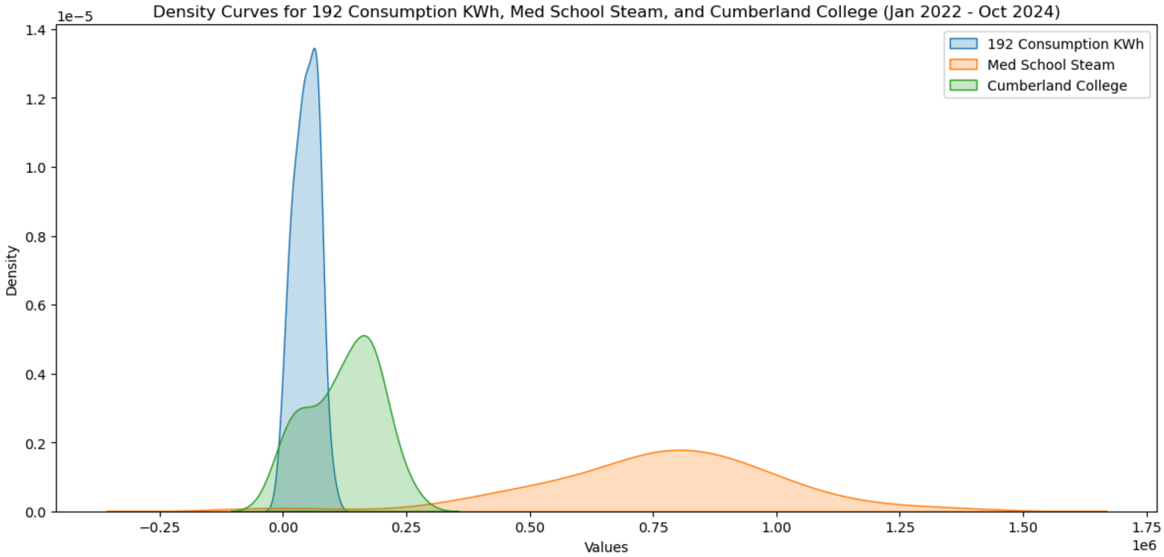

In [43]:
```
# Calculate means and standard deviations
means = filtered_data[['192 Consumption KWh', 'Med School Steam', 'Cumberland Co
stds = filtered_data[['192 Consumption KWh', 'Med School Steam', 'Cumberland Col

# Create the bar chart
plt.figure(figsize=(12, 6))
plt.bar(means.index, means, yerr=stds, capsize=5, color=['blue', 'green', 'orang
plt.title('Bar Chart with Whiskers for 192 Consumption KWh, Med School Steam, an
plt.ylabel('Mean Values with Error Bars')
plt.xticks([0, 1, 2], ['192 Consumption KWh', 'Med School Steam', 'Cumberland Co
plt.tight_layout()
plt.show()
```




In [45]:
```
# Plot density curves
plt.figure(figsize=(12, 6))
sns.kdeplot(filtered_data['192 Consumption KWh'], label='192 Consumption KWh', s
sns.kdeplot(filtered_data['Med School Steam'], label='Med School Steam', shade=T
sns.kdeplot(filtered_data['Cumberland College'], label='Cumberland College', sha

plt.title('Density Curves for 192 Consumption KWh, Med School Steam, and Cumberl
plt.xlabel('Values')
plt.ylabel('Density')
plt.legend()
```

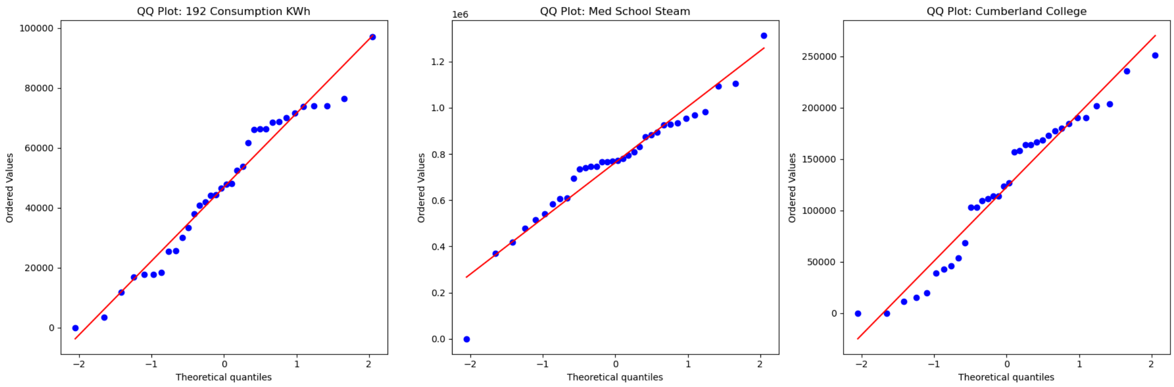
```
plt.tight_layout()
plt.show()
```

```
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\4089502887.py:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(filtered_data['192 Consumption KWh'], label='192 Consumption KWh',
shade=True)
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\4089502887.py:4: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(filtered_data['Med School Steam'], label='Med School Steam', shade=
True)
C:\Users\sugan\AppData\Local\Temp\ipykernel_26340\4089502887.py:5: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(filtered_data['Cumberland College'], label='Cumberland College', sh
ade=True)
```




In [47]:

```
import scipy.stats as stats

# Generate QQ plots
plt.figure(figsize=(18, 6))

# QQ plot for each column
columns = ['192 Consumption KWh', 'Med School Steam', 'Cumberland College']
for i, col in enumerate(columns):
    plt.subplot(1, 3, i + 1)
    stats.probplot(filtered_data[col].dropna(), dist="norm", plot=plt)
    plt.title(f'QQ Plot: {col}')

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



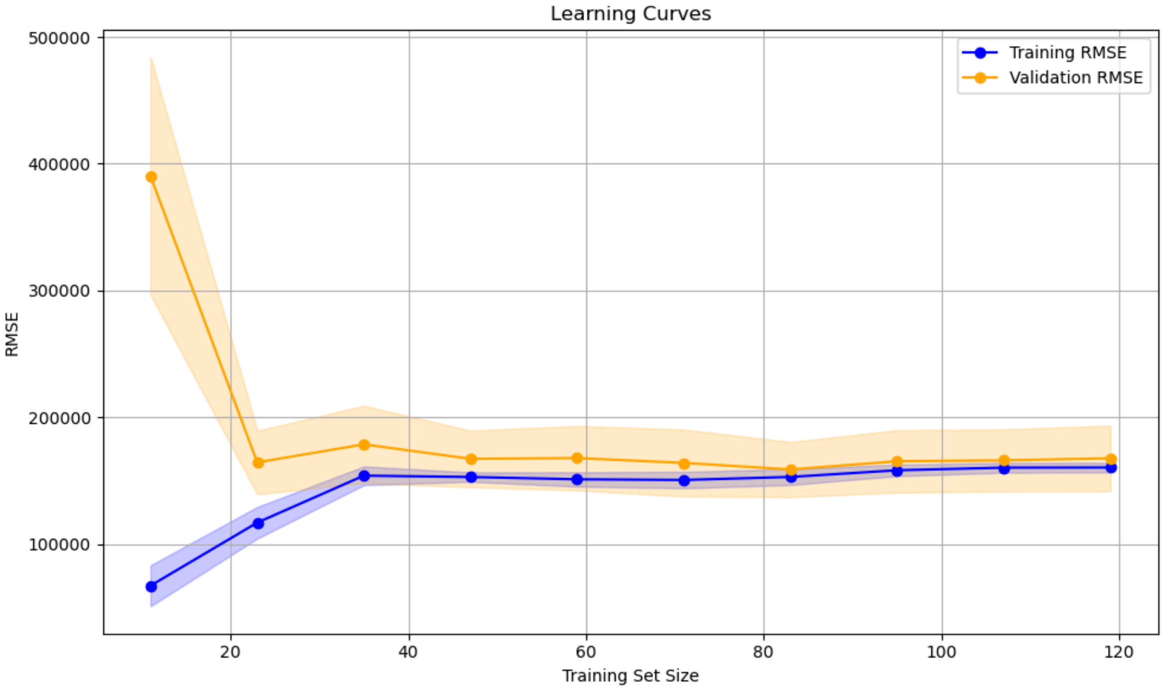

In [100…

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import KFold


# Feature preparation function
def prepare_features(data):
    # First melt the DataFrame to get proper time series format
    data_melted = data.melt(
        id_vars=['MonthYear'],
        var_name='meter_description',
        value_name='consumption'
    )

    # Convert MonthYear column to datetime
    data_melted['MonthYear'] = pd.to_datetime(data_melted['MonthYear'], format='

    # Create features
    features = pd.DataFrame({
        #'month_sin': np.sin(2 * np.pi * data_melted['MonthYear'].dt.month / 12)
        #'month_cos': np.cos(2 * np.pi * data_melted['MonthYear'].dt.month / 12)
        #'consumption_lag1': data_melted.groupby('meter_description')['consumpti
        'rolling_mean_3': data_melted.groupby('meter_description')['consumption'
        'rolling_std_3': data_melted.groupby('meter_description')['consumption']
    })

    # Add seasonal indicators
    season_dummies = pd.get_dummies(
        data_melted['MonthYear'].dt.month.map(
            lambda m: 'Summer' if m in [12, 1, 2] else
            'Autumn' if m in [3, 4, 5] else
            'Winter' if m in [6, 7, 8] else 'Spring'
        ),
        prefix='season'
    )

    # Add meter description indicators (building-specific)
    building_dummies = pd.get_dummies(data_melted['meter_description'], prefix='

    return pd.concat([features, season_dummies, building_dummies], axis=1), data
```

```
# Random Forest optimization function
def optimize_rf_model(X_train, X_test, y_train, y_test):
    param_grid = {
     'n_estimators': [50],
    'max_depth': [2],
    'max_features': ['sqrt', 'log2']
    }

    rf = RandomForestRegressor(random_state=40)
    grid_search = GridSearchCV(
        estimator=rf,
        param_grid=param_grid,
        cv=5,
        scoring='neg_mean_squared_error',
        n_jobs=-1,
        verbose=1
    )

    grid_search.fit(X_train, y_train)
    predictions = grid_search.predict(X_test)

    return grid_search.best_estimator_, grid_search.best_params_, predictions


def plot_learning_curve(estimator, X, y, cv, scoring='neg_mean_squared_error', t
    # Generate learning curve data
    train_sizes, train_scores, validation_scores = learning_curve(
        estimator,
        X,
        y,
        cv=cv,
        scoring=scoring,
        train_sizes=train_sizes,
        n_jobs=-1
    )

    # Calculate RMSE for training and validation
    train_rmse = np.sqrt(-train_scores)
    validation_rmse = np.sqrt(-validation_scores)

    # Plot learning curves
    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_rmse.mean(axis=1), 'o-', color='blue', label='Tr
    plt.plot(train_sizes, validation_rmse.mean(axis=1), 'o-', color='orange', la

    # Add shaded areas for standard deviation
    plt.fill_between(
        train_sizes,
        train_rmse.mean(axis=1) - train_rmse.std(axis=1),
        train_rmse.mean(axis=1) + train_rmse.std(axis=1),
        alpha=0.2,
        color='blue'
    )
    plt.fill_between(
        train_sizes,
        validation_rmse.mean(axis=1) - validation_rmse.std(axis=1),
        validation_rmse.mean(axis=1) + validation_rmse.std(axis=1),
        alpha=0.2,
        color='orange'
    )
```

```
    # Add labels and title
    plt.title('Learning Curves')
    plt.xlabel('Training Set Size')
    plt.ylabel('RMSE')
    plt.legend(loc='best')
    plt.grid()
    plt.tight_layout()
    plt.show()


# Main function to orchestrate modeling
def main():
    # Load dataset
    FILE_PATH = "processed_steam_mthw_data.csv"
    data = pd.read_csv(FILE_PATH)

    # Filter relevant columns and rows (Jan 2022 - Oct 2024)
    data = data[['MonthYear', '192 Consumption KWh', 'Med School Steam', 'Cumber
    data = data[(data['MonthYear'] >= "Jan_2022") & (data['MonthYear'] <= "Oct_2

    # Prepare features and target
    X, y = prepare_features(data)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=40
    )

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

    # Optimize model
    best_model, best_params, predictions = optimize_rf_model(
        X_train_scaled, X_test_scaled, y_train, y_test
    )

    rf_model = RandomForestRegressor(n_estimators=50, max_depth=2, max_features=
    cv_strategy = KFold(n_splits=5, shuffle=True, random_state=40)

    # Plot the learning curve
    plot_learning_curve(rf_model, X_train_scaled, y_train, cv=cv_strategy)

    # Calculate metrics
    metrics = {
        'RMSE': np.sqrt(mean_squared_error(y_test, predictions)),
        'MAE': mean_absolute_error(y_test, predictions),
        'R2': r2_score(y_test, predictions)
    }

    print("Best Model Parameters:", best_params)
    print("Model Metrics:", metrics)
if __name__ == "__main__":
    main()
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```




```
Best Model Parameters: {'max_depth': 2, 'max_features': 'sqrt', 'n_estimators': 5
0}
Model Metrics: {'RMSE': 204624.1263564878, 'MAE': 143129.07466783188, 'R2': 0.791
9624056851104}
```

In [ ]:

In [ ]: