In [1]:
```
import pandas as pd
```

In [2]:
```
# Load the dataset
file_path = '/Users/macbookpro/Desktop/MFT Energy Case Study/messy_trading_da
df = pd.read_csv(file_path)
```

In [3]:
```
# Output basic info about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Date         367 non-null    object 
 1   Price        367 non-null    float64
 2   Volume       367 non-null    float64
 3   Temperature  362 non-null    float64
 4   Day_Type     367 non-null    object 
dtypes: float64(3), object(2)
memory usage: 14.5+ KB
None
```

In [4]:
```
# Display number of rows and columns
print(f"\nNumber of rows: {df.shape[0]}")
print(f"Number of columns: {df.shape[1]}")
```

```

Number of rows: 367
Number of columns: 5
```

In [5]:
```
# Check for NaN values
print(df.isna().sum())
```

```
Date           0
Price          0
Volume         0
Temperature    5
Day_Type       0
dtype: int64
```

In [6]:
```
# Check for empty cells
print((df == '').sum())
```

```
Date           0
Price          0
Volume         0
Temperature    0
Day_Type       0
dtype: int64
```

In [7]:
```
# Display the first few rows to inspect the data
print(df.head())
```

```
         Date      Price      Volume  Temperature Day_Type
0  2022-01-01  52.483571  196.942732    22.980721  Weekend
1  2022-01-02  49.756010  189.759896    21.256343  Weekend
2  2022-01-03  54.133055  199.614946    22.324721  Weekday
3  2022-01-04  58.956939  225.802920    20.594690  Weekday
4  2022-01-05  50.618046  192.816402    19.400500  Weekday
```

In [8]:
```
# Remove rows where the Temperature column has NaN values
df_cleaned = df.dropna(subset=['Temperature'])
```

In [9]:
```
# Display the first few rows to verify the data
print(df_cleaned.head())
```

```
         Date      Price      Volume  Temperature Day_Type
0  2022-01-01  52.483571  196.942732    22.980721  Weekend
1  2022-01-02  49.756010  189.759896    21.256343  Weekend
2  2022-01-03  54.133055  199.614946    22.324721  Weekday
3  2022-01-04  58.956939  225.802920    20.594690  Weekday
4  2022-01-05  50.618046  192.816402    19.400500  Weekday
```

In [10]:
```
# Use .loc to modify the column without raising a warning
df_cleaned.loc[:, 'Date'] = pd.to_datetime(df_cleaned['Date'])
```

In [11]:
```
# Sort the DataFrame by the date column in ascending order
df_cleaned = df_cleaned.sort_values(by='Date', ascending=True)
```

In [12]:
```
# Reset the index
df_cleaned.reset_index(drop=True, inplace=True)
```

In [13]:
```
# Count the number of negative values and text values in the PRICE column
negative_price_count = (df_cleaned['Price'] < 0).sum()
text_price_count = df_cleaned['Price'].apply(lambda x: isinstance(x, str)).s

print(f"Negative values in Price column: {negative_price_count}")
print(f"Text values in Price column: {text_price_count}")
```

```
Negative values in Price column: 6
Text values in Price column: 0
```

In [14]:
```
# Count the number of negative values and text values in the VOLUME column
negative_volume_count = (df_cleaned['Volume'] < 0).sum()
text_volume_count = df_cleaned['Volume'].apply(lambda x: isinstance(x, str))

print(f"Negative values in Volume column: {negative_volume_count}")
print(f"Text values in Volume column: {text_volume_count}")
```

```
Negative values in Volume column: 5
Text values in Volume column: 0
```

In [15]:
```
# Remove rows where Price or Volume has negative values
df_cleaned = df_cleaned[(df_cleaned['Price'] >= 0) & (df_cleaned['Volume'] >
```

In [16]:
```
# Check for values containing "," in Price, Volume, and Temperature columns
comma_price_count = df_cleaned['Price'].astype(str).str.contains(',').sum()
comma_volume_count = df_cleaned['Volume'].astype(str).str.contains(',').sum(
comma_temperature_count = df_cleaned['Temperature'].astype(str).str.contains

print(f"Number of values containing ',' in Price: {comma_price_count}")
print(f"Number of values containing ',' in Volume: {comma_volume_count}")
print(f"Number of values containing ',' in Temperature: {comma_temperature_c
```

```
Number of values containing ',' in Price: 0
Number of values containing ',' in Volume: 0
Number of values containing ',' in Temperature: 0
```

In [17]:
```
# Ensure Price, Volume, and Temperature columns are in numeric format
# If there are any non-numeric values, they will be converted to NaN
df_cleaned['Price'] = pd.to_numeric(df_cleaned['Price'], errors='coerce')
df_cleaned['Volume'] = pd.to_numeric(df_cleaned['Volume'], errors='coerce')
df_cleaned['Temperature'] = pd.to_numeric(df_cleaned['Temperature'], errors=
```

In [18]:
```
# Display any rows with NaNs (to inspect if type conversion introduced NaNs)
print("\nRows with NaNs after conversion (if any):")
print(df_cleaned[df_cleaned.isna().any(axis=1)])
```

```

Rows with NaNs after conversion (if any):
Empty DataFrame
Columns: [Date, Price, Volume, Temperature, Day_Type]
Index: []
```

In [19]:
```
# Count the number of duplicate values in the Date column
duplicate_count = df_cleaned.duplicated(subset=['Date']).sum()

print(f"Number of duplicate values in the Date column: {duplicate_count}")
```

```
Number of duplicate values in the Date column: 2
```

In [20]:
```
# Drop any rows where Date is duplicated, keeping only the first occurrence
df_cleaned = df_cleaned.drop_duplicates(subset=['Date'], keep='first')
```

In [21]:
```
# Display info to verify changes
print("Data types after conversion:")
print(df_cleaned.dtypes)

print("\nNumber of rows after removing duplicates based on Date:")
print(df_cleaned.shape[0])
```

```
Data types after conversion:
Date            object
Price          float64
Volume         float64
Temperature    float64
Day_Type        object
dtype: object

Number of rows after removing duplicates based on Date:
349
```

In [22]:
```
# Convert Date column into datetime64 format
df_cleaned['Date'] = pd.to_datetime(df_cleaned['Date'])
```

In [23]:
```
#Convert Day_Type column into category format for memory efficiency
df_cleaned['Day_Type'] = df_cleaned['Day_Type'].astype('category')
```

In [24]:
```
import numpy as np

# 1) Check if Date values are not in yyyy-mm-dd format
invalid_date_format = df_cleaned['Date'].isna()

# 2) Check for non-numeric values in Price, Volume, and Temperature
non_numeric_price = ~df_cleaned['Price'].apply(np.isreal)
non_numeric_volume = ~df_cleaned['Volume'].apply(np.isreal)
non_numeric_temperature = ~df_cleaned['Temperature'].apply(np.isreal)

# Combine conditions to find rows where any condition is true
invalid_rows = df_cleaned[invalid_date_format | non_numeric_price | non_nume

# Display rows that meet the conditions
print("Rows with invalid date formats or non-numeric values in Price, Volume
print(invalid_rows)
```

```
Rows with invalid date formats or non-numeric values in Price, Volume, or T
emperature:
Empty DataFrame
Columns: [Date, Price, Volume, Temperature, Day_Type]
Index: []
```

In [25]:
```
# Display unique values in the Day_Type column
day_type_values = df_cleaned['Day_Type'].cat.categories
print("Possible values in Day_Type:")
print(day_type_values)
```

```
Possible values in Day_Type:
Index(['Holiday', 'Weekday', 'Weekend'], dtype='object')
```

The records with "WhoAmI?" and "WhyIsDataAlwaysMessy?" are removed in the duplication removal of the Date column stage. Thus, there is no additional step required to remove them.

DATA ANALYSIS

In [26]:
```
# Compute variation (standard deviation) for Price, Volume, and Temperature
price_variation = df_cleaned['Price'].std()
volume_variation = df_cleaned['Volume'].std()
temperature_variation = df_cleaned['Temperature'].std()

print(f"Variation in Price: {price_variation}")
print(f"Variation in Volume: {volume_variation}")
print(f"Variation in Temperature: {temperature_variation}")
```

```
Variation in Price: 51.865364205562635
Variation in Volume: 1155.4308637074241
Variation in Temperature: 2.234094384020828
```

In [27]:
```
# Compute ratios of each Day_Type category to the sum of all records
day_type_counts = df_cleaned['Day_Type'].value_counts(normalize=True)
print("\nRatios of Day_Type values to the total number of records:")
print(day_type_counts)
```

```

Ratios of Day_Type values to the total number of records:
Day_Type
Weekday    0.702006
Weekend    0.286533
Holiday    0.011461
Name: proportion, dtype: float64
```

In [28]:
```
# Group by Day_Type and calculate the standard deviation for Price and Volum
price_volume_variation_by_day_type = df_cleaned.groupby('Day_Type', observed

print("Variation of Price and Volume within each Day_Type category:")
print(price_volume_variation_by_day_type)
```

```
Variation of Price and Volume within each Day_Type category:
              Price       Volume
Day_Type
Holiday   28.580780    87.618122
Weekday   59.600678  1232.812724
Weekend   25.486220   971.336296
```

In [29]:
```
import matplotlib.pyplot as plt
import os
```

1st Plots

In [30]:
```
# Define the directory to save the plots
save_dir = '/Users/macbookpro/Desktop/MFT Energy Case Study/1st Plots'
os.makedirs(save_dir, exist_ok=True)  # Create directory if it does not exis
```

In [31]:
```
# Scatter plot of Price vs. Date
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned['Date'], df_cleaned['Price'], alpha=0.5)
plt.title("Price vs. Date")
plt.xlabel("Date")
plt.ylabel("Price")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(f"{save_dir}/Price_vs_Date.png")  # Save the plot
plt.close()

# Scatter plot of Price vs. Volume
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned['Volume'], df_cleaned['Price'], alpha=0.5, color='ora
plt.title("Price vs. Volume")
plt.xlabel("Volume")
plt.ylabel("Price")
plt.savefig(f"{save_dir}/Price_vs_Volume.png")  # Save the plot
plt.close()

# Scatter plot of Price vs. Temperature
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned['Temperature'], df_cleaned['Price'], alpha=0.5, color
plt.title("Price vs. Temperature")
plt.xlabel("Temperature")
plt.ylabel("Price")
plt.savefig(f"{save_dir}/Price_vs_Temperature.png")  # Save the plot
plt.close()

# Box plot of Price vs. Day_Type
plt.figure(figsize=(8, 6))
df_cleaned.boxplot(column='Price', by='Day_Type')
plt.title("Price vs. Day_Type")
plt.suptitle("")  # Remove default boxplot title
plt.xlabel("Day_Type")
plt.ylabel("Price")
plt.savefig(f"{save_dir}/Price_vs_Day_Type.png")  # Save the plot
plt.close()
```

```
<Figure size 576x432 with 0 Axes>
```

Outlier Removal

In [32]:
```
# Remove records with Price greater than 400
df_cleaned_no_outliers = df_cleaned[df_cleaned['Price'] <= 400]

# Further remove records with Volume greater than 1000
df_cleaned_no_outliers = df_cleaned_no_outliers[df_cleaned_no_outliers['Volu

# Display the number of records after removing outliers
print(f"Number of records after removing outliers: {df_cleaned_no_outliers.s

# Display the first few rows to confirm changes
print("\nFirst few rows of df_cleaned_no_outliers:")
print(df_cleaned_no_outliers.head())
```

```
Number of records after removing outliers: 340

First few rows of df_cleaned_no_outliers:
        Date      Price      Volume  Temperature Day_Type
0 2022-01-01  52.483571  196.942732    22.980721  Weekend
1 2022-01-02  49.756010  189.759896    21.256343  Weekend
2 2022-01-03  54.133055  199.614946    22.324721  Weekday
3 2022-01-04  58.956939  225.802920    20.594690  Weekday
4 2022-01-05  50.618046  192.816402    19.400500  Weekday
```

2nd Plots

In [33]:
```
# Define the directory to save the plots
save_dir = '/Users/macbookpro/Desktop/MFT Energy Case Study/2nd Plots'
os.makedirs(save_dir, exist_ok=True)  # Create directory if it doesn't exist

# Scatter plot of Price vs. Date
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned_no_outliers['Date'], df_cleaned_no_outliers['Price'],
plt.title("Price vs. Date")
plt.xlabel("Date")
plt.ylabel("Price")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig(f"{save_dir}/Price_vs_Date.png")  # Save the plot
plt.close()

# Scatter plot of Price vs. Volume
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned_no_outliers['Volume'], df_cleaned_no_outliers['Price'
plt.title("Price vs. Volume")
plt.xlabel("Volume")
plt.ylabel("Price")
plt.savefig(f"{save_dir}/Price_vs_Volume.png")  # Save the plot
plt.close()

# Scatter plot of Price vs. Temperature
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned_no_outliers['Temperature'], df_cleaned_no_outliers['P
plt.title("Price vs. Temperature")
plt.xlabel("Temperature")
plt.ylabel("Price")
plt.savefig(f"{save_dir}/Price_vs_Temperature.png")  # Save the plot
plt.close()

# Box plot of Price vs. Day_Type
plt.figure(figsize=(8, 6))
df_cleaned_no_outliers.boxplot(column='Price', by='Day_Type')
plt.title("Price vs. Day_Type")
plt.suptitle("")  # Remove default boxplot title
plt.xlabel("Day_Type")
plt.ylabel("Price")
plt.savefig(f"{save_dir}/Price_vs_Day_Type.png")  # Save the plot
plt.close()
```

```
<Figure size 576x432 with 0 Axes>
```

Correlations

In [34]:
```
# Compute Pearson correlation for linear relationships
pearson_corr = df_cleaned_no_outliers[['Price', 'Volume', 'Temperature']].co
print("Pearson Correlation (linear relationships):")
print(pearson_corr)

# Compute Spearman correlation for monotonic relationships
spearman_corr = df_cleaned_no_outliers[['Price', 'Volume', 'Temperature']].c
print("\nSpearman Correlation (nonlinear monotonic relationships):")
print(spearman_corr)
```

```
Pearson Correlation (linear relationships):
                Price    Volume  Temperature
Price        1.000000  0.936573    -0.514155
Volume       0.936573  1.000000    -0.498390
Temperature -0.514155 -0.498390     1.000000

Spearman Correlation (nonlinear monotonic relationships):
                Price    Volume  Temperature
Price        1.000000  0.929015    -0.513516
Volume       0.929015  1.000000    -0.493402
Temperature -0.513516 -0.493402     1.000000
```

Feature Engineering

In [35]:
```
# Extract Season from Date
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    elif month in [9, 10, 11]:
        return 'Autumn'
```

In [36]:
```
# Add the Season column
df_cleaned_no_outliers['Season'] = df_cleaned_no_outliers['Date'].dt.month.a
```

In [37]:
```
# Replace "Weekday" values in Day_Type with specific weekday names based on
df_cleaned_no_outliers['Day_Type'] = df_cleaned_no_outliers.apply(
    lambda row: row['Date'].day_name() if row['Day_Type'] == 'Weekday' else
    axis=1
)

# Display the first few rows to confirm changes
print("\nFirst few rows after feature engineering:")
print(df_cleaned_no_outliers.head())
```

```
First few rows after feature engineering:
        Date      Price      Volume  Temperature   Day_Type  Season
0 2022-01-01  52.483571  196.942732    22.980721    Weekend  Winter
1 2022-01-02  49.756010  189.759896    21.256343    Weekend  Winter
2 2022-01-03  54.133055  199.614946    22.324721     Monday  Winter
3 2022-01-04  58.956939  225.802920    20.594690    Tuesday  Winter
4 2022-01-05  50.618046  192.816402    19.400500  Wednesday  Winter
```

In [38]:
```
# Day of the Month
df_cleaned_no_outliers['Day_of_Month'] = df_cleaned_no_outliers['Date'].dt.d
```

In [39]:
```
# 1-Day Lagged Price
df_cleaned_no_outliers['Price_Lag_1d'] = df_cleaned_no_outliers['Price'].shi
```

In [40]:
```
# 7-Day Rolling Average of Price
df_cleaned_no_outliers['Price_7d_MA'] = df_cleaned_no_outliers['Price'].roll
```

In [41]:
```
# Display the first few rows to verify the new features
print("\nFirst few rows after adding new features:")
print(df_cleaned_no_outliers.head(10))
```

```
First few rows after adding new features:
        Date      Price      Volume  Temperature   Day_Type  Season  \
0 2022-01-01  52.483571  196.942732    22.980721    Weekend  Winter
1 2022-01-02  49.756010  189.759896    21.256343    Weekend  Winter
2 2022-01-03  54.133055  199.614946    22.324721     Monday  Winter
3 2022-01-04  58.956939  225.802920    20.594690    Tuesday  Winter
4 2022-01-05  50.618046  192.816402    19.400500  Wednesday  Winter
5 2022-01-06  51.064945  207.925388    24.271623   Thursday  Winter
6 2022-01-07  60.578255  262.664526    22.970141     Friday  Winter
7 2022-01-08  56.965617  231.353729    26.729083    Weekend  Winter
8 2022-01-09  51.226965  195.933460    20.350706    Weekend  Winter
9 2022-01-10  56.732621  237.489521    21.745448     Monday  Winter

   Day_of_Month  Price_Lag_1d  Price_7d_MA
0             1           NaN    52.483571
1             2     52.483571    51.119790
2             3     49.756010    52.124212
3             4     54.133055    53.832394
4             5     58.956939    53.189524
5             6     50.618046    52.835428
6             7     51.064945    53.941546
7             8     60.578255    54.581838
8             9     56.965617    54.791975
9            10     51.226965    55.163341
```

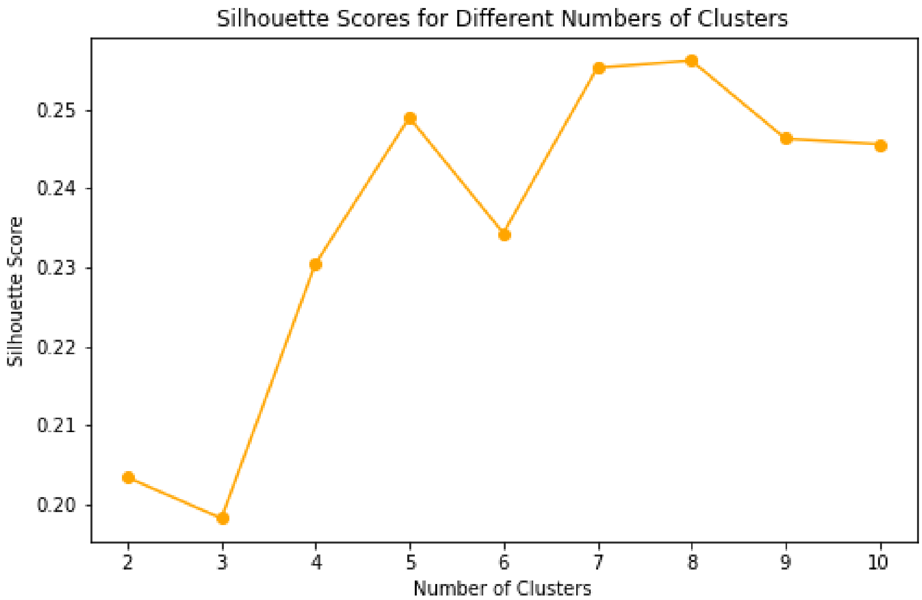
In [42]:
```
# Drop the first row (with NaN in Price_Lag_1d) and the Date column
data_matrix = df_cleaned_no_outliers.iloc[1:].drop(columns=['Date'])

# One-hot encode Day_Type and Season, while keeping Day_of_Month as-is
data_matrix = pd.get_dummies(data_matrix, columns=['Day_Type', 'Season'], dr

# Display the first few rows to confirm
print("First few rows of data_matrix:")
print(data_matrix.head())
```

```
First few rows of data_matrix:
       Price      Volume  Temperature  Day_of_Month  Price_Lag_1d  \
1  49.756010  189.759896    21.256343             2     52.483571
2  54.133055  199.614946    22.324721             3     49.756010
3  58.956939  225.802920    20.594690             4     54.133055
4  50.618046  192.816402    19.400500             5     58.956939
5  51.064945  207.925388    24.271623             6     50.618046

   Price_7d_MA  Day_Type_Friday  Day_Type_Holiday  Day_Type_Monday  \
1    51.119790            False             False            False
2    52.124212            False             False             True
3    53.832394            False             False            False
4    53.189524            False             False            False
5    52.835428            False             False            False

   Day_Type_Thursday  Day_Type_Tuesday  Day_Type_Wednesday  Day_Type_Weeken
d  \
1              False             False               False              Tru
e
```

Clustering

In [43]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [44]:
```
# Step 1: Scale the features in data_matrix
scaler = StandardScaler()
scaled_data_matrix = scaler.fit_transform(data_matrix)

# Step 2: Determine the optimal number of clusters using Silhouette Score
silhouette_scores = []
range_clusters = range(2, 11)

for k in range_clusters:
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(scaled_data_matrix)
    silhouette_scores.append(silhouette_score(scaled_data_matrix, kmeans.lab

# Plot Silhouette Scores
plt.figure(figsize=(8, 5))
plt.plot(range_clusters, silhouette_scores, marker='o', color='orange')
plt.title("Silhouette Scores for Different Numbers of Clusters")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.show()
```



In [45]:
```
# Step 3: Choose the optimal number of clusters based on the maximum Silhoue
optimal_clusters = range_clusters[silhouette_scores.index(max(silhouette_sco
print(f"Optimal number of clusters: {optimal_clusters}")
```

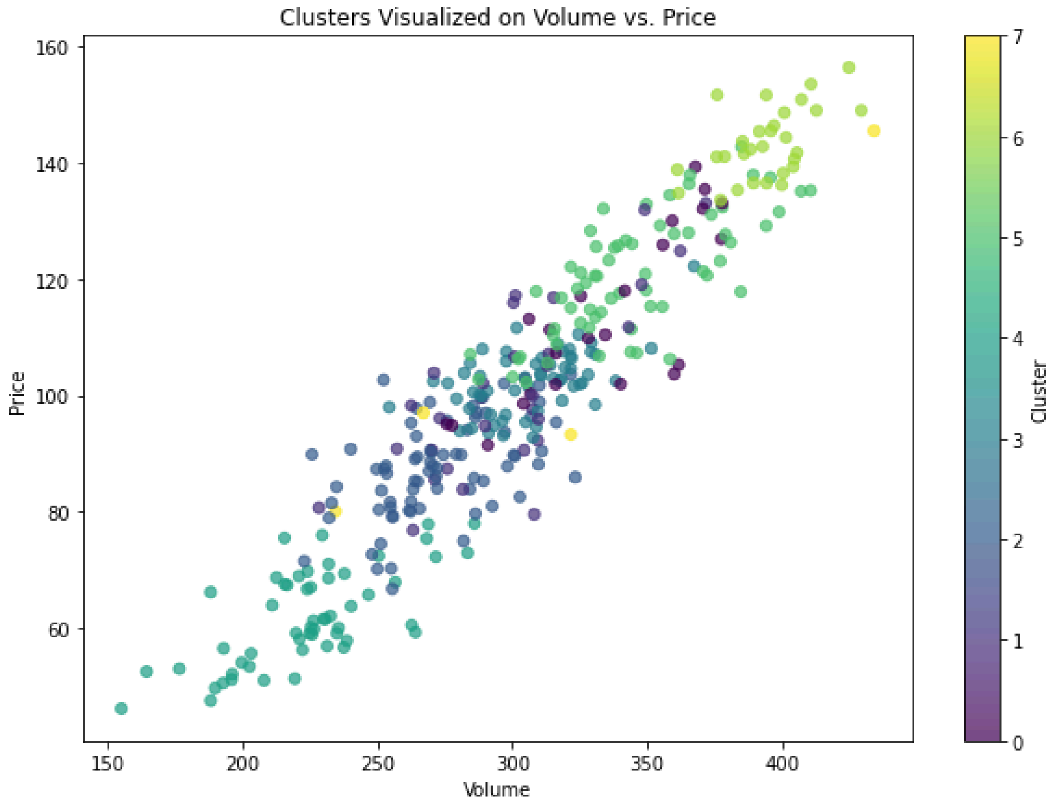
```
Optimal number of clusters: 8
```

In [46]:
```
# Step 4: Run K-Means with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_clusters, random_state=0)
data_matrix['Cluster'] = kmeans.fit_predict(scaled_data_matrix)
```

In [47]:
```
# Display the first few rows with the new Cluster feature
print("\nFirst few rows of data_matrix with Cluster feature:")
print(data_matrix.head())
```

```
First few rows of data_matrix with Cluster feature:
       Price      Volume  Temperature  Day_of_Month  Price_Lag_1d  \
1  49.756010  189.759896    21.256343             2     52.483571
2  54.133055  199.614946    22.324721             3     49.756010
3  58.956939  225.802920    20.594690             4     54.133055
4  50.618046  192.816402    19.400500             5     58.956939
5  51.064945  207.925388    24.271623             6     50.618046

   Price_7d_MA  Day_Type_Friday  Day_Type_Holiday  Day_Type_Monday  \
1    51.119790            False             False            False
2    52.124212            False             False             True
3    53.832394            False             False            False
4    53.189524            False             False            False
5    52.835428            False             False            False

   Day_Type_Thursday  Day_Type_Tuesday  Day_Type_Wednesday  Day_Type_Weeken
d  \
1              False             False               False              Tru
e
2              False             False               False              Fals
e
3              False              True               False              Fals
e
4              False             False                True              Fals
e
5               True             False               False              Fals
e

   Season_Autumn  Season_Spring  Season_Summer  Season_Winter  Cluster
1          False          False          False           True        4
2          False          False          False           True        4
3          False          False          False           True        4
4          False          False          False           True        4
5          False          False          False           True        4
```

In [48]:
```
# Define x and y for the plot
x_feature = 'Volume'  # Change this to another feature if you have a differe
y_feature = 'Price'

# Plotting
plt.figure(figsize=(10, 7))
scatter = plt.scatter(data_matrix[x_feature], data_matrix[y_feature], c=data_
plt.colorbar(scatter, label='Cluster')
plt.title(f"Clusters Visualized on {x_feature} vs. {y_feature}")
plt.xlabel(x_feature)
plt.ylabel(y_feature)
plt.savefig('/Users/macbookpro/Desktop/MFT Energy Case Study/Cluster_Plot.png
plt.show()
```



Add the clusters as a feature to data_matrix

In [49]:
```
data_matrix['Cluster'] = kmeans.labels_

# Display the first few rows to confirm the Cluster column
print("First few rows of data_matrix with Cluster column:")
print(data_matrix.head())
```

```
First few rows of data_matrix with Cluster column:
       Price      Volume  Temperature  Day_of_Month  Price_Lag_1d  \
1  49.756010  189.759896    21.256343             2     52.483571
2  54.133055  199.614946    22.324721             3     49.756010
3  58.956939  225.802920    20.594690             4     54.133055
4  50.618046  192.816402    19.400500             5     58.956939
5  51.064945  207.925388    24.271623             6     50.618046

   Price_7d_MA  Day_Type_Friday  Day_Type_Holiday  Day_Type_Monday  \
1    51.119790            False             False            False
2    52.124212            False             False             True
3    53.832394            False             False            False
4    53.189524            False             False            False
5    52.835428            False             False            False

   Day_Type_Thursday  Day_Type_Tuesday  Day_Type_Wednesday  Day_Type_Weeken
d  \
1              False             False               False               Tru
e
2              False             False               False              Fals
e
3              False              True               False              Fals
e
4              False             False                True              Fals
e
5               True             False               False              Fals
e

   Season_Autumn  Season_Spring  Season_Summer  Season_Winter  Cluster
1          False          False          False           True        4
2          False          False          False           True        4
3          False          False          False           True        4
4          False          False          False           True        4
5          False          False          False           True        4
```

Support Vector Regression (SVR) with a linear kernel

In [50]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
```

In [51]:
```
# Step 1: Define the target and features
X = data_matrix.drop(columns=['Price'])  # Features (all columns except targ
y = data_matrix['Price']                 # Target

# Step 2: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Step 3: Standardize the features (important for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Train the SVR model with a linear kernel
svr_model = SVR(kernel='linear', C=1.0, epsilon=0.1)
svr_model.fit(X_train_scaled, y_train)

# Step 5: Make predictions and evaluate the model
y_pred_train = svr_model.predict(X_train_scaled)
y_pred_test = svr_model.predict(X_test_scaled)
```

In [52]:
```
# Evaluation metrics
train_mae = mean_absolute_error(y_train, y_pred_train)
test_mae = mean_absolute_error(y_test, y_pred_test)
train_mse = mean_squared_error(y_train, y_pred_train)
test_mse = mean_squared_error(y_test, y_pred_test)
train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)
```

In [53]:
```
print("SVR Model Evaluation (Linear Kernel):")
print(f"Training MAE: {train_mae}")
print(f"Test MAE: {test_mae}")
print(f"Training MSE: {train_mse}")
print(f"Test MSE: {test_mse}")
print(f"Training R²: {train_r2}")
print(f"Test R²: {test_r2}")
```

```
SVR Model Evaluation (Linear Kernel):
Training MAE: 3.067268785896988
Test MAE: 3.4330816557585218
Training MSE: 17.233677772988155
Test MSE: 19.700921422594885
Training R²: 0.9706199311947556
Test R²: 0.9730692901200902
```

Hyperparameter Tuning for the SVR with a linear kernel

In [54]:
```
from sklearn.model_selection import GridSearchCV
```

In [55]:
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the SVR model with linear kernel
svr_model = SVR(kernel='linear')

# Set up the parameter grid for C and epsilon
param_grid = {
    'C': [0.1, 1, 10, 100],          # Regularization strength
    'epsilon': [0.01, 0.1, 0.5, 1]   # Margin of tolerance
}

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=svr_model, param_grid=param_grid, cv=5,
grid_search.fit(X_train_scaled, y_train)

# Best parameters and the best score from GridSearch
print("Best Parameters:", grid_search.best_params_)
print("Best CV Score (negative MAE):", grid_search.best_score_)
```

```
Best Parameters: {'C': 100, 'epsilon': 0.5}
Best CV Score (negative MAE): -3.2204163298202877
```

In [56]:
```
# Train the SVR model with the best parameters
best_svr = grid_search.best_estimator_
best_svr.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_test = best_svr.predict(X_test_scaled)

# Evaluate the model
test_mae = mean_absolute_error(y_test, y_pred_test)
test_mse = mean_squared_error(y_test, y_pred_test)
test_r2 = r2_score(y_test, y_pred_test)

print("\nTuned SVR Model Evaluation:")
print(f"Test MAE: {test_mae}")
print(f"Test MSE: {test_mse}")
print(f"Test R²: {test_r2}")
```

```

Tuned SVR Model Evaluation:
Test MAE: 3.317636156799765
Test MSE: 18.03202313748865
Test R²: 0.9753506359805802
```