e11-code

November 13, 2023

[18]:
```python
import pandas as pd

# Load the CSV file
file_path = 'e11.csv'  # Replace with the actual path to your CSV file
df = pd.read_csv(file_path)

# Replace all non-numeric values with zero
df = df.apply(pd.to_numeric, errors='coerce').fillna(0)

# Save the cleaned data to a new CSV file or overwrite the existing one
cleaned_file_path = 'cleaned_e11.csv'  # Replace with the desired path
df.to_csv(cleaned_file_path, index=False)
import pandas as pd

# Assuming df is your DataFrame with the cleaned data

# Set the threshold for dropping columns
threshold = 0.5  # You can adjust this threshold as needed

# Drop columns with more than 50% of values as zero
df_filtered = df.loc[:, (df == 0).mean() <= threshold]

# Now df_filtered contains only the columns that meet the specified condition
import pandas as pd

# Assuming df is your DataFrame with the cleaned data

# Set the threshold for dropping columns based on mean deviation
deviation_threshold = 0.1  # You can adjust this threshold as needed

# Normalize the data for each column
normalized_df = (df - df.mean()) / (df.std())

# Drop columns with mean deviation below the threshold
df_filtered = normalized_df.loc[:, normalized_df.abs().mean() >=
 ↪deviation_threshold]
```

```
# Now df_filtered contains only the columns that meet the specified condition
import pandas as pd

# Assuming df is your DataFrame with the cleaned data

# Interpolate zero values in each column
df_interpolated = df_filtered.apply(lambda col: col.
 ↪interpolate(method='linear', limit_direction='both', inplace=False))

# Now df_interpolated contains the DataFrame with zero values interpolated in␣
 ↪each column
import pandas as pd
import numpy as np

# Assuming df_interpolated is your DataFrame with the interpolated data

# Set the threshold for identifying outliers
outlier_threshold = 1  # You can adjust this threshold as needed

# Identify regions with abrupt changes using the Z-score
z_scores = np.abs((df_interpolated - df_interpolated.mean()) / df_interpolated.
 ↪std())
outlier_mask = z_scores > outlier_threshold

# Replace identified outlier regions with NaN
df_interpolated_outliers = df_interpolated.mask(outlier_mask)

# Interpolate NaN values without considering outliers
df_interpolated_outliers = df_interpolated_outliers.interpolate(method='linear')

# Now df_interpolated_outliers contains the DataFrame with identified outlier␣
 ↪regions interpolated
# Replace NaN values in df_interpolated_outliers with values from␣
 ↪df_interpolated
df_interpolated_outliers = df_interpolated_outliers.fillna(df_interpolated)
import pandas as pd

# Assuming df is your DataFrame with the cleaned data
df1=df_interpolated_outliers
# Set the threshold for dropping columns based on mean deviation
deviation_threshold = 0.1  # You can adjust this threshold as needed

# Normalize the data for each column
normalized_df1 = (df1 - df1.mean()) / (df1.std())

# Drop columns with mean deviation below the threshold
```

```python
df_filtered1 = normalized_df1.loc[:, normalized_df1.abs().mean() >=␣
 ↪deviation_threshold]

# Now df_filtered contains only the columns that meet the specified condition
import pandas as pd
import numpy as np
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt

# Assuming df_interpolated_outliers is your DataFrame
correlation_matrix = df_filtered1.corr()

# Set the threshold for high correlation
high_correlation_threshold = 0.8  # You can adjust this threshold as needed

# Replace NaN values with a specific value (e.g., 0)
correlation_matrix = correlation_matrix.fillna(0)

# Create hierarchical clusters based on the absolute correlation matrix
linkage_matrix = hierarchy.linkage(correlation_matrix.abs().values,␣
 ↪method='complete')
clusters = hierarchy.fcluster(linkage_matrix, high_correlation_threshold,␣
 ↪criterion='distance')

# Create a dictionary to store the columns in each cluster
cluster_dict = {}
for col, cluster in zip(correlation_matrix.columns, clusters):
    if cluster not in cluster_dict:
        cluster_dict[cluster] = [col]
    else:
        cluster_dict[cluster].append(col)

# Choose a representative column from each cluster (e.g., the first column)
selected_columns = [cluster[0] for cluster in cluster_dict.values()]

# Create a new DataFrame with selected columns
df_selected = df_filtered1[selected_columns]

# Now df_selected contains columns without high correlation
import pandas as pd

# Assuming df_selected is your DataFrame

# Columns to remove and place in a separate DataFrame
vibration_columns = ['c51', 'c52', 'c53', 'c54']
specific_energy_column = 'c241'
```

```python
# Create a DataFrame for vibration columns
df_vibration = df_selected[vibration_columns].copy()

# Create a DataFrame for the specific energy column
df_specific_energy = df_selected[specific_energy_column].to_frame()

# Remove the vibration and specific energy columns from df_selected
df_selected = df_selected.drop(columns=vibration_columns)
df_selected= df_selected.drop(columns=[specific_energy_column])

# Now you have three separate DataFrames:
# - df_selected_without_vibration (df_selected without vibration columns)
# - df_vibration (DataFrame containing c51, c52, c53, c54)
# - df_specific_energy (DataFrame containing c241)
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assuming df_selected is your DataFrame with the relevant predictors

# Create a DataFrame to store VIF values
vif_data = pd.DataFrame()
vif_data["Variable"] = df_selected.columns
vif_data["VIF"] = [variance_inflation_factor(df_selected.values, i) for i in␣
 ↪range(df_selected.shape[1])]

# Set the threshold for dropping columns
vif_threshold = 10

# Identify columns with VIF above the threshold
high_vif_columns = vif_data[vif_data["VIF"] > vif_threshold]["Variable"]

# Drop columns with high VIF from the original DataFrame
df_selected_filtered = df_selected.drop(columns=high_vif_columns)

# Display the DataFrame after dropping columns with high VIF
print(df_selected_filtered)
```

```
            c5        c6        c7        c8       c12       c13       c14  \
0     1.077686 -0.620112 -0.583908 -2.345130 -5.994284 -2.894029  0.434436
1    -1.038875 -0.710192 -0.623838 -2.036731 -6.466142 -4.360946  0.434358
2     0.466331 -0.800272 -0.631667 -1.728332 -5.351553 -4.800539  0.434419
3     0.566375 -0.890352 -0.639495 -1.419932 -2.367856 -1.695886 -1.498128
4     0.878250 -0.980432 -0.685252 -1.111533 -0.908675 -0.401377 -1.588547
…          …         …         …         …         …         …         …
1020  0.264346 -0.003848 -1.502663 -1.523425  0.645351  0.606723  0.435099
1021 -0.149455 -0.003848 -1.549860 -1.520990  0.859607  0.846734  0.435091
1022 -0.563255 -0.003848 -1.515050 -1.588091  0.791280  0.759059  0.435083
```

```
1023 -0.977055 -0.003848 -1.551450 -1.562049  0.872698  0.715155  0.435108
1024 -0.977055 -0.003848 -1.554019 -1.506483  0.836988  0.851783  0.435187

           c16       c20       c21  …      c146      c147      c156  \
0    -3.984699 -0.058447 -0.586570  … -3.958093 -5.005045 -0.999512
1    -5.274191  0.509824 -0.634807  … -5.316902 -5.140248 -0.999512
2    -5.516213  0.083688 -0.167798  … -6.795694 -5.303635 -0.999512
3    -1.674810  0.648976 -0.367610  … -2.559943 -2.144536 -0.999512
4    -0.726209  1.324640  0.266923  … -0.130495 -8.114437 -0.999512
…          …         …         …  …         …         …         …
1020  0.676463  0.352133  0.711210  … -0.398185  0.292528 -0.999512
1021  1.228540  0.763008  0.710078  …  0.043806  0.296045 -0.999512
1022  1.379658  0.647405  0.856444  …  0.250073  0.313906 -0.999512
1023  1.498950  0.672338  0.998389  …  0.338943  0.308518 -0.999512
1024  1.196592  0.808282  1.069930  …  0.118599  0.311145 -0.999512

          c160      c162      c163      c177      c179      c238      c239
0    -5.888668 -1.073725 -0.818691 -8.476040 -2.927616 -3.722118  1.054677
1    -5.888668 -1.073725 -0.818691 -3.098376  0.519717 -3.417373  1.121804
2    -5.888668 -1.073725 -0.818691 -2.981887  0.091237 -1.798721  1.164766
3    -5.888668 -1.073725 -0.818691 -3.827758  0.552376 -0.749763  1.285186
4    -5.888668 -1.073725 -0.818691 -2.029978  0.478127  0.369444  1.464617
…          …         …         …         …         …         …         …
1020  0.291235 -0.659019  1.764924 -0.580644  0.168941 -0.058725 -1.697298
1021  0.291235  0.999806 -0.388088 -0.420545 -0.006024 -0.968301 -0.138142
1022  0.291235  0.999806 -0.388088 -0.329986 -0.058370 -1.248025  0.064266
1023  0.291235  0.999806 -0.818691 -0.295098  0.012685 -1.245924  0.135513
1024  0.291235  0.999806 -0.818691 -0.286966  0.027418 -1.021563  0.395885

[1025 rows x 36 columns]
```

[19]:

```
#for column in df_interpolated.columns:
#    plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
#    plt.plot(df_interpolated.index, df_interpolated[column])
#    plt.title(f'Plot of {column}')
#    plt.xlabel('Index')  # You can customize the x-axis label
#    plt.ylabel(column)  # You can customize the y-axis label
#    plt.grid(True)
#    plt.show()
```

[20]:

```
#for column in df_interpolated_outliers.columns:
  #  plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
  #  plt.plot(df_interpolated_outliers.index, df_interpolated_outliers[column])
 #   plt.title(f'Plot of {column}')
 #   plt.xlabel('Index')  # You can customize the x-axis label
 #   plt.ylabel(column)  # You can customize the y-axis label
 #   plt.grid(True)
```

```
#   plt.show()
```

[21]:
```
# Assuming df_selected_filtered is your DataFrame

# Get the names of all columns left in df_selected_filtered
selected_columns_names = df_selected_filtered.columns

# Display the column names
print(selected_columns_names)
```

```
Index(['c5', 'c6', 'c7', 'c8', 'c12', 'c13', 'c14', 'c16', 'c20', 'c21', 'c22',
       'c23', 'c26', 'c27', 'c29', 'c30', 'c34', 'c35', 'c36', 'c37', 'c42',
       'c63', 'c68', 'c72', 'c73', 'c133', 'c146', 'c147', 'c156', 'c160',
       'c162', 'c163', 'c177', 'c179', 'c238', 'c239'],
      dtype='object')
```

[22]:
```
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt

# Assuming df_filtered1 is your DataFrame with cleaned and processed data

# Controllable variables
controllable_columns = ['c26', 'c27', 'c28', 'c29', 'c30', 'c31', 'c32', 'c33',␣
 ↪'c39', 'c139', 'c142', 'c143', 'c155', 'c156', 'c157', 'c158', 'c160',␣
 ↪'c161', 'c162', 'c163']

# Create a DataFrame with only controllable variables
df_controllable = df_filtered1[controllable_columns].copy()

# Correlation analysis on controllable variables
correlation_matrix_controllable = df_controllable.corr()

# Set the threshold for high correlation
high_correlation_threshold = 0.8  # You can adjust this threshold as needed

# Replace NaN values with a specific value (e.g., 0)
correlation_matrix_controllable = correlation_matrix_controllable.fillna(0)

# Create hierarchical clusters based on the absolute correlation matrix
linkage_matrix_controllable = hierarchy.linkage(correlation_matrix_controllable.
 ↪abs().values, method='complete')
clusters_controllable = hierarchy.fcluster(linkage_matrix_controllable,␣
 ↪high_correlation_threshold, criterion='distance')

# Create a dictionary to store the columns in each cluster
```

```python
cluster_dict_controllable = {}
for col, cluster in zip(correlation_matrix_controllable.columns,␣
 ↪clusters_controllable):
    if cluster not in cluster_dict_controllable:
        cluster_dict_controllable[cluster] = [col]
    else:
        cluster_dict_controllable[cluster].append(col)

# Choose a representative column from each cluster (e.g., the first column)
selected_columns_controllable = [cluster[0] for cluster in␣
 ↪cluster_dict_controllable.values()]

# Create a new DataFrame with selected columns from controllable variables
df_controllable_selected = df_controllable[selected_columns_controllable]

# VIF calculation on controllable variables
vif_data_controllable = pd.DataFrame()
vif_data_controllable["Variable"] = df_controllable_selected.columns
vif_data_controllable["VIF"] =␣
 ↪[variance_inflation_factor(df_controllable_selected.values, i) for i in␣
 ↪range(df_controllable_selected.shape[1])]

# Set the threshold for dropping columns
vif_threshold_controllable = 10

# Identify columns with VIF above the threshold
high_vif_columns_controllable =␣
 ↪vif_data_controllable[vif_data_controllable["VIF"] >␣
 ↪vif_threshold_controllable]["Variable"]

# Drop columns with high VIF from the original DataFrame of controllable␣
 ↪variables
df_controllable_filtered = df_controllable_selected.
 ↪drop(columns=high_vif_columns_controllable)

# Display the results
print("DataFrame with Controllable Variables:")
print(df_controllable_selected)

print("\nCorrelation Matrix for Controllable Variables:")
print(correlation_matrix_controllable)

print("\nDataFrame after dropping columns with high VIF for Controllable␣
 ↪Variables:")
print(df_controllable_filtered)
```

```
DataFrame with Controllable Variables:
```

```
           c26       c27       c28       c30       c31       c32       c39  \
0     1.261065  0.466452 -1.283271 -2.013954 -3.796822 -2.374609 -2.018021
1     1.168615 -0.123706 -1.149661 -2.741322 -4.165876 -2.964296 -2.032982
2     1.405832 -0.279690 -1.433831  0.099538 -6.003108 -3.663312 -2.019702
3     1.643049 -0.999321 -1.588933  0.541207 -3.143254 -3.037304 -1.999225
4     0.064689 -0.481034 -1.256392 -0.095862 -1.909756 -2.812619 -1.995785
…          …         …         …         …         …         …         …
1020  0.798630  0.085605 -1.658007 -1.206199 -1.445568  0.187849  1.076150
1021  0.798630  0.085605 -1.658007 -1.192785 -1.134421  0.403056  1.076150
1022  0.798630  0.085605 -1.658007 -1.352933 -0.858724  0.804668  1.076150
1023  0.798630  0.085605 -1.658007 -1.221850 -0.753362  1.009922  1.076150
1024  0.798630  0.085605 -1.658007 -1.176781 -0.717170  1.138604  1.076150

          c139      c142      c155      c156      c157      c158      c163
0    -0.211039 -3.798572 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
1    -0.631378 -3.971087 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
2    -1.172680 -5.382301 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
3     0.989308 -3.450346 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
4     1.109907 -2.388482 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
…          …         …         …         …         …         …         …
1020 -1.437224 -1.177156  0.751245 -0.999512  0.756536 -0.335384  1.764924
1021 -0.787145 -1.177156  0.955511 -0.999512  0.893634 -0.222931 -0.388088
1022 -0.437316 -1.177156  0.955511 -0.999512  1.167830 -0.222931 -0.388088
1023 -0.221686 -1.177156  0.989555 -0.999512  0.756536 -0.194817 -0.818691
1024  0.149813 -1.177156  0.938489 -0.999512  0.893634 -0.307271 -0.818691

[1025 rows x 14 columns]

Correlation Matrix for Controllable Variables:
           c26       c27       c28       c29       c30       c31       c32  \
c26   1.000000  0.343828 -0.196719  0.713787 -0.330267 -0.132484 -0.061507
c27   0.343828  1.000000 -0.087154  0.086185 -0.286325  0.029645  0.158741
c28  -0.196719 -0.087154  1.000000 -0.154847  0.211838 -0.075117 -0.274693
c29   0.713787  0.086185 -0.154847  1.000000  0.012065 -0.315855 -0.102917
c30  -0.330267 -0.286325  0.211838  0.012065  1.000000 -0.134027  0.025822
c31  -0.132484  0.029645 -0.075117 -0.315855 -0.134027  1.000000  0.573122
c32  -0.061507  0.158741 -0.274693 -0.102917  0.025822  0.573122  1.000000
c33  -0.024384  0.185624 -0.269728 -0.045132  0.044852  0.517003  0.961759
c39   0.161277  0.271898 -0.287675  0.111891 -0.226386  0.044817  0.243285
c139 -0.348402 -0.142280  0.196754 -0.353297  0.011562  0.447620  0.217858
c142 -0.161228  0.024769  0.445317 -0.390324 -0.023636  0.720036  0.365443
c143 -0.242280  0.063663  0.693751 -0.367932  0.077654  0.435791  0.235615
c155  0.198351  0.294034  0.009746 -0.022745 -0.471700  0.390688  0.163661
c156  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
c157 -0.135476  0.095964  0.041299 -0.148481 -0.100360  0.483010  0.507445
c158  0.261215  0.294696 -0.048619  0.070061 -0.210049  0.294852  0.250676
c160 -0.026902  0.119000 -0.013273 -0.100913 -0.066857  0.479353  0.433353
c161 -0.088847  0.080947  0.131829 -0.143132 -0.043372  0.386932  0.247823
```

```
c162 -0.030866  0.254972 -0.042205 -0.162373 -0.282663  0.358612  0.282248
c163  0.095586  0.328563 -0.004330 -0.075022 -0.142791  0.188230  0.337504

           c33       c39      c139      c142      c143      c155  c156  \
c26  -0.024384  0.161277 -0.348402 -0.161228 -0.242280  0.198351   0.0
c27   0.185624  0.271898 -0.142280  0.024769  0.063663  0.294034   0.0
c28  -0.269728 -0.287675  0.196754  0.445317  0.693751  0.009746   0.0
c29  -0.045132  0.111891 -0.353297 -0.390324 -0.367932 -0.022745   0.0
c30   0.044852 -0.226386  0.011562 -0.023636  0.077654 -0.471700   0.0
c31   0.517003  0.044817  0.447620  0.720036  0.435791  0.390688   0.0
c32   0.961759  0.243285  0.217858  0.365443  0.235615  0.163661   0.0
c33   1.000000  0.245722  0.225149  0.299432  0.196413  0.160385   0.0
c39   0.245722  1.000000 -0.141478 -0.088547 -0.059662  0.523475   0.0
c139  0.225149 -0.141478  1.000000  0.410623  0.407533  0.209588   0.0
c142  0.299432 -0.088547  0.410623  1.000000  0.747831  0.318446   0.0
c143  0.196413 -0.059662  0.407533  0.747831  1.000000  0.338437   0.0
c155  0.160385  0.523475  0.209588  0.318446  0.338437  1.000000   0.0
c156  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000   0.0
c157  0.471607  0.469537  0.170238  0.399520  0.477216  0.492755   0.0
c158  0.216201  0.367155 -0.151315  0.280323  0.157874  0.421750   0.0
c160  0.407681  0.396987  0.042788  0.366506  0.333653  0.414385   0.0
c161  0.198360  0.450028 -0.006599  0.349994  0.394075  0.477143   0.0
c162  0.248689  0.440008  0.247743  0.223315  0.365123  0.628358   0.0
c163  0.338318  0.270242  0.158556  0.182304  0.304210  0.306878   0.0

          c157      c158      c160      c161      c162      c163
c26  -0.135476  0.261215 -0.026902 -0.088847 -0.030866  0.095586
c27   0.095964  0.294696  0.119000  0.080947  0.254972  0.328563
c28   0.041299 -0.048619 -0.013273  0.131829 -0.042205 -0.004330
c29  -0.148481  0.070061 -0.100913 -0.143132 -0.162373 -0.075022
c30  -0.100360 -0.210049 -0.066857 -0.043372 -0.282663 -0.142791
c31   0.483010  0.294852  0.479353  0.386932  0.358612  0.188230
c32   0.507445  0.250676  0.433353  0.247823  0.282248  0.337504
c33   0.471607  0.216201  0.407681  0.198360  0.248689  0.338318
c39   0.469537  0.367155  0.396987  0.450028  0.440008  0.270242
c139  0.170238 -0.151315  0.042788 -0.006599  0.247743  0.158556
c142  0.399520  0.280323  0.366506  0.349994  0.223315  0.182304
c143  0.477216  0.157874  0.333653  0.394075  0.365123  0.304210
c155  0.492755  0.421750  0.414385  0.477143  0.628358  0.306878
c156  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
c157  1.000000  0.561382  0.740737  0.697276  0.523917  0.262506
c158  0.561382  1.000000  0.554942  0.551230  0.213909  0.220912
c160  0.740737  0.554942  1.000000  0.664166  0.400402  0.183958
c161  0.697276  0.551230  0.664166  1.000000  0.477547  0.109980
c162  0.523917  0.213909  0.400402  0.477547  1.000000  0.420283
c163  0.262506  0.220912  0.183958  0.109980  0.420283  1.000000

DataFrame after dropping columns with high VIF for Controllable Variables:
```

```
           c26       c27       c28       c30       c31       c32       c39  \
0     1.261065  0.466452 -1.283271 -2.013954 -3.796822 -2.374609 -2.018021
1     1.168615 -0.123706 -1.149661 -2.741322 -4.165876 -2.964296 -2.032982
2     1.405832 -0.279690 -1.433831  0.099538 -6.003108 -3.663312 -2.019702
3     1.643049 -0.999321 -1.588933  0.541207 -3.143254 -3.037304 -1.999225
4     0.064689 -0.481034 -1.256392 -0.095862 -1.909756 -2.812619 -1.995785
…          …         …         …         …         …         …         …
1020  0.798630  0.085605 -1.658007 -1.206199 -1.445568  0.187849  1.076150
1021  0.798630  0.085605 -1.658007 -1.192785 -1.134421  0.403056  1.076150
1022  0.798630  0.085605 -1.658007 -1.352933 -0.858724  0.804668  1.076150
1023  0.798630  0.085605 -1.658007 -1.221850 -0.753362  1.009922  1.076150
1024  0.798630  0.085605 -1.658007 -1.176781 -0.717170  1.138604  1.076150

          c139      c142      c155      c156      c157      c158      c163
0    -0.211039 -3.798572 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
1    -0.631378 -3.971087 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
2    -1.172680 -5.382301 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
3     0.989308 -3.450346 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
4     1.109907 -2.388482 -1.768029 -0.999512 -5.001587 -4.046346 -0.818691
…          …         …         …         …         …         …         …
1020 -1.437224 -1.177156  0.751245 -0.999512  0.756536 -0.335384  1.764924
1021 -0.787145 -1.177156  0.955511 -0.999512  0.893634 -0.222931 -0.388088
1022 -0.437316 -1.177156  0.955511 -0.999512  1.167830 -0.222931 -0.388088
1023 -0.221686 -1.177156  0.989555 -0.999512  0.756536 -0.194817 -0.818691
1024  0.149813 -1.177156  0.938489 -0.999512  0.893634 -0.307271 -0.818691

[1025 rows x 14 columns]
```

[23]:
```python
import pandas as pd
import statsmodels.api as sm

# Assuming df_selected_filtered is your DataFrame with the cleaned and
 ↪processed data
# Assuming 'target_column' is the column you want to predict (e.g., 'c51')

# Select predictor variables (features) and the target variable
X = df_selected_filtered
y = df_vibration['c51']
print(X)
print(y)
# Add a constant term to the predictor variables (OLS in statsmodels does not
 ↪add it automatically)
X = sm.add_constant(X)

# Fit the OLS model
model = sm.OLS(y, X).fit()
```

```python
# Display the model summary
print(model.summary())
```

```
            c5        c6        c7        c8       c12       c13       c14  \
0     1.077686 -0.620112 -0.583908 -2.345130 -5.994284 -2.894029  0.434436
1    -1.038875 -0.710192 -0.623838 -2.036731 -6.466142 -4.360946  0.434358
2     0.466331 -0.800272 -0.631667 -1.728332 -5.351553 -4.800539  0.434419
3     0.566375 -0.890352 -0.639495 -1.419932 -2.367856 -1.695886 -1.498128
4     0.878250 -0.980432 -0.685252 -1.111533 -0.908675 -0.401377 -1.588547
…          …         …         …         …         …         …         …
1020  0.264346 -0.003848 -1.502663 -1.523425  0.645351  0.606723  0.435099
1021 -0.149455 -0.003848 -1.549860 -1.520990  0.859607  0.846734  0.435091
1022 -0.563255 -0.003848 -1.515050 -1.588091  0.791280  0.759059  0.435083
1023 -0.977055 -0.003848 -1.551450 -1.562049  0.872698  0.715155  0.435108
1024 -0.977055 -0.003848 -1.554019 -1.506483  0.836988  0.851783  0.435187

           c16       c20       c21  …      c146      c147      c156  \
0    -3.984699 -0.058447 -0.586570  … -3.958093 -5.005045 -0.999512
1    -5.274191  0.509824 -0.634807  … -5.316902 -5.140248 -0.999512
2    -5.516213  0.083688 -0.167798  … -6.795694 -5.303635 -0.999512
3    -1.674810  0.648976 -0.367610  … -2.559943 -2.144536 -0.999512
4    -0.726209  1.324640  0.266923  … -0.130495 -8.114437 -0.999512
…          …         …         …  …         …         …         …
1020  0.676463  0.352133  0.711210  … -0.398185  0.292528 -0.999512
1021  1.228540  0.763008  0.710078  …  0.043806  0.296045 -0.999512
1022  1.379658  0.647405  0.856444  …  0.250073  0.313906 -0.999512
1023  1.498950  0.672338  0.998389  …  0.338943  0.308518 -0.999512
1024  1.196592  0.808282  1.069930  …  0.118599  0.311145 -0.999512

          c160      c162      c163      c177      c179      c238      c239
0    -5.888668 -1.073725 -0.818691 -8.476040 -2.927616 -3.722118  1.054677
1    -5.888668 -1.073725 -0.818691 -3.098376  0.519717 -3.417373  1.121804
2    -5.888668 -1.073725 -0.818691 -2.981887  0.091237 -1.798721  1.164766
3    -5.888668 -1.073725 -0.818691 -3.827758  0.552376 -0.749763  1.285186
4    -5.888668 -1.073725 -0.818691 -2.029978  0.478127  0.369444  1.464617
…          …         …         …         …         …         …         …
1020  0.291235 -0.659019  1.764924 -0.580644  0.168941 -0.058725 -1.697298
1021  0.291235  0.999806 -0.388088 -0.420545 -0.006024 -0.968301 -0.138142
1022  0.291235  0.999806 -0.388088 -0.329986 -0.058370 -1.248025  0.064266
1023  0.291235  0.999806 -0.818691 -0.295098  0.012685 -1.245924  0.135513
1024  0.291235  0.999806 -0.818691 -0.286966  0.027418 -1.021563  0.395885

[1025 rows x 36 columns]
0      -1.057308
1      -1.041112
2      -1.160680
3      -0.824690
```

```
4      -0.844904
          …
1020   -1.194113
1021   -1.194113
1022   -1.194113
1023   -1.194113
1024   -1.194113
Name: c51, Length: 1025, dtype: float64
                            OLS Regression Results
==============================================================================
Dep. Variable:                    c51   R-squared:                       0.408
Model:                            OLS   Adj. R-squared:                  0.387
Method:                 Least Squares   F-statistic:                     19.47
Date:                Mon, 13 Nov 2023   Prob (F-statistic):           4.94e-89
Time:                        17:32:57   Log-Likelihood:                -1185.3
No. Observations:                1025   AIC:                             2443.
Df Residuals:                     989   BIC:                             2620.
Df Model:                          35
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c5             0.0569      0.028      2.057      0.040       0.003       0.111
c6            -0.0308      0.030     -1.043      0.297      -0.089       0.027
c7            -0.1192      0.035     -3.368      0.001      -0.189      -0.050
c8             0.3024      0.042      7.262      0.000       0.221       0.384
c12           -0.0053      0.046     -0.117      0.907      -0.095       0.084
c13           -0.0335      0.047     -0.719      0.472      -0.125       0.058
c14            0.0197      0.033      0.595      0.552      -0.045       0.085
c16           -0.1366      0.045     -3.049      0.002      -0.225      -0.049
c20            0.1500      0.043      3.470      0.001       0.065       0.235
c21           -0.1749      0.039     -4.476      0.000      -0.252      -0.098
c22            0.0005      0.037      0.014      0.989      -0.072       0.073
c23            0.0761      0.043      1.785      0.075      -0.008       0.160
c26            0.1097      0.050      2.207      0.028       0.012       0.207
c27            0.1727      0.032      5.399      0.000       0.110       0.235
c29           -0.0497      0.045     -1.094      0.274      -0.139       0.039
c30            0.0777      0.036      2.181      0.029       0.008       0.148
c34            0.1573      0.035      4.470      0.000       0.088       0.226
c35           -0.1320      0.035     -3.815      0.000      -0.200      -0.064
c36           -0.0039      0.028     -0.140      0.889      -0.058       0.050
c37            0.0023      0.027      0.084      0.933      -0.052       0.056
c42            0.0107      0.032      0.333      0.739      -0.052       0.074
c63            0.0125      0.036      0.350      0.726      -0.058       0.083
c68           -0.0507      0.036     -1.403      0.161      -0.122       0.020
c72            0.0487      0.037      1.303      0.193      -0.025       0.122
c73           -0.3148      0.040     -7.837      0.000      -0.394      -0.236
c133          -0.1568      0.036     -4.325      0.000      -0.228      -0.086
```

```
c146          0.0297      0.046      0.643      0.520     -0.061      0.120
c147          0.0007      0.033      0.021      0.984     -0.063      0.064
c156       5.741e-17      0.024   2.35e-15      1.000     -0.048      0.048
c160          0.0522      0.038      1.367      0.172     -0.023      0.127
c162          0.3024      0.039      7.853      0.000      0.227      0.378
c163         -0.0073      0.034     -0.218      0.828     -0.073      0.059
c177          0.1161      0.031      3.689      0.000      0.054      0.178
c179         -0.1964      0.036     -5.458      0.000     -0.267     -0.126
c238          0.1804      0.033      5.493      0.000      0.116      0.245
c239          0.0131      0.035      0.372      0.710     -0.056      0.082
==============================================================================
Omnibus:                       34.363   Durbin-Watson:                   0.215
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               19.142
Skew:                           0.160   Prob(JB):                     6.97e-05
Kurtosis:                       2.412   Cond. No.                         6.40
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

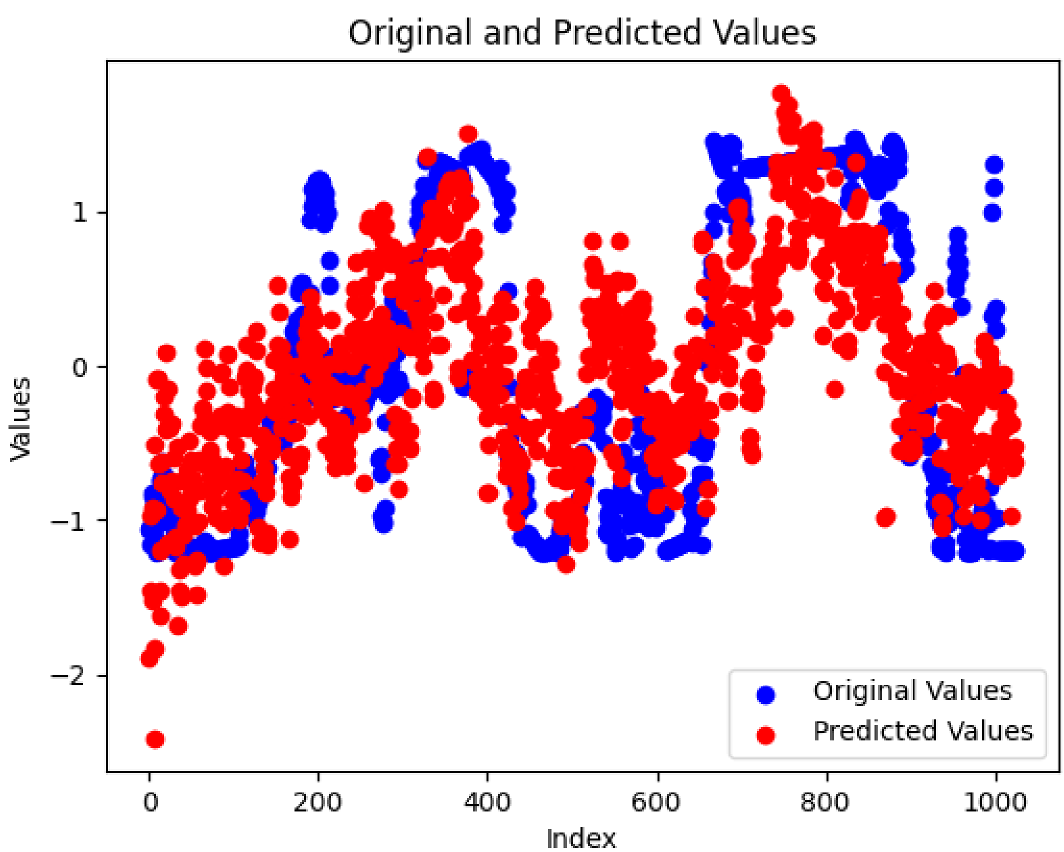
[24]:
```python
import pandas as pd
import statsmodels.api as sm

# Assuming df_selected_filtered is your DataFrame with the cleaned and␣
↪processed data
# Assuming 'target_column' is the column you want to predict (e.g., 'c51')

# Select predictor variables (features) and the target variable
X = df_selected_filtered
y = df_vibration['c51']

# Iterate until all p-values are below 0.05
while True:
    # Add a constant term to the predictor variables
    X = sm.add_constant(X)

    # Fit the OLS model
    model = sm.OLS(y, X).fit()

    # Check p-values
    p_values = model.pvalues[1:]  # Exclude the constant term
    max_p_value = p_values.max()

    if max_p_value > 0.05:
        # Drop the variable with the highest p-value
        variable_to_drop = p_values.idxmax()
```

```python
        X = X.drop(columns=[variable_to_drop])
    else:
        # Break the loop if all p-values are below 0.05
        break

# Display the final model summary
print(model.summary())
```

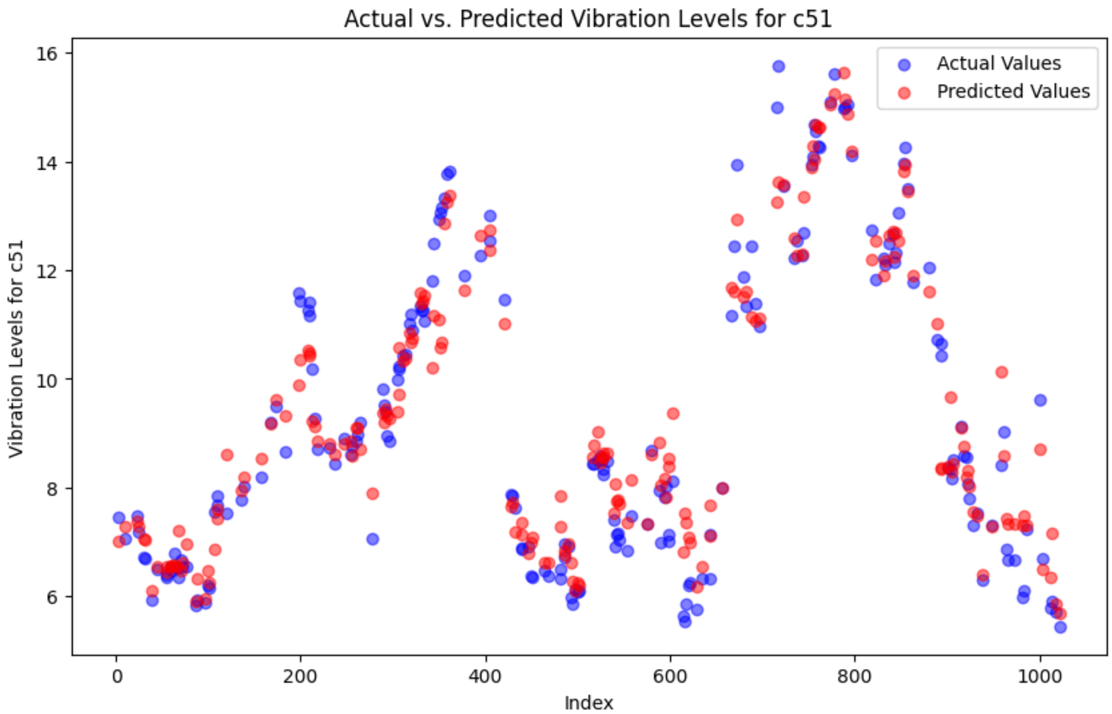
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    c51   R-squared:                       0.398
Model:                            OLS   Adj. R-squared:                  0.389
Method:                 Least Squares   F-statistic:                     41.68
Date:                Mon, 13 Nov 2023   Prob (F-statistic):           2.02e-99
Time:                        17:32:58   Log-Likelihood:                -1193.7
No. Observations:                1025   AIC:                             2421.
Df Residuals:                    1008   BIC:                             2505.
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.888e-17      0.024   1.18e-15      1.000      -0.048       0.048
c7            -0.1305      0.030     -4.304      0.000      -0.190      -0.071
c8             0.3162      0.034      9.184      0.000       0.249       0.384
c16           -0.1745      0.032     -5.426      0.000      -0.238      -0.111
c20            0.1327      0.036      3.656      0.000       0.061       0.204
c21           -0.1760      0.037     -4.746      0.000      -0.249      -0.103
c26            0.0845      0.030      2.854      0.004       0.026       0.143
c27            0.1761      0.029      6.064      0.000       0.119       0.233
c30            0.0653      0.032      2.016      0.044       0.002       0.129
c34            0.1567      0.032      4.919      0.000       0.094       0.219
c35           -0.1197      0.030     -4.026      0.000      -0.178      -0.061
c73           -0.2997      0.036     -8.401      0.000      -0.370      -0.230
c133          -0.1345      0.031     -4.367      0.000      -0.195      -0.074
c162           0.3055      0.035      8.690      0.000       0.237       0.374
c177           0.1043      0.030      3.441      0.001       0.045       0.164
c179          -0.1925      0.030     -6.454      0.000      -0.251      -0.134
c238           0.1654      0.028      5.955      0.000       0.111       0.220
==============================================================================
Omnibus:                       42.713   Durbin-Watson:                   0.209
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.860
Skew:                           0.136   Prob(JB):                     2.95e-05
Kurtosis:                       2.356   Cond. No.                         3.63
==============================================================================

Notes:
```

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[25]:
```
import matplotlib.pyplot as plt

# Assuming X_test is your test set and y_test is the corresponding true values

# Make predictions on the test set
X_test = X
y_test=y
X_test = sm.add_constant(X_test)  # Add a constant term
predictions = model.predict(X_test)
plt.scatter(y_test.index, y_test, label='Original Values', color='blue')
# Plot predicted values
plt.scatter(y_test.index, predictions, label='Predicted Values', color='red')

plt.xlabel('Index')  # Assuming y_test has an index
plt.ylabel('Values')
plt.title('Original and Predicted Values')
plt.legend()
plt.show()
```



[26]:
```python
# Assuming the data has been preprocessed and split into features (X) and␣
↪target (y)

# Selecting 'c51' as the target variable
target_column = 'c51'
X = df_controllable_filtered  # Features
y = df[target_column]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Model fitting - Random Forest Regression
from sklearn.ensemble import RandomForestRegressor

# Train the model
model_c51 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c51.fit(X_train, y_train)

# Prediction on test set
predictions_c51 = model_c51.predict(X_test)

# Visualize the results
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values
plt.scatter(y_test.index, y_test, label='Actual Values', color='blue', alpha=0.
↪5)
plt.scatter(y_test.index, predictions_c51, label='Predicted Values',␣
↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c51')
plt.title('Actual vs. Predicted Vibration Levels for c51')
plt.legend()
plt.show()
```



[27]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, predictions_c51)
print(f"Mean Squared Error for c51: {mse}")

# Calculate R-squared
r2 = r2_score(y_test, predictions_c51)
print(f"R-squared for c51: {r2}")
```

```
Mean Squared Error for c51: 0.4888078613912953
R-squared for c51: 0.9359045196858679
```

[28]:
```
df[controllable_columns]
```

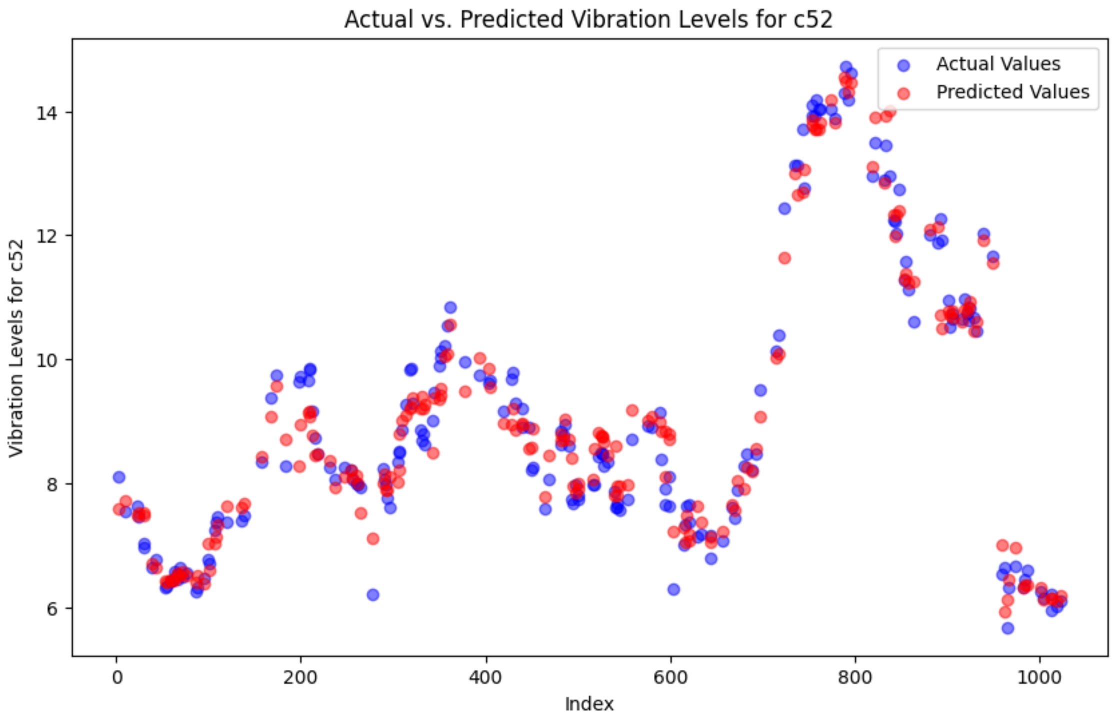
[28]:
```
             c26         c27        c28         c29        c30        c31  \
0     493.796764  104.553871  41.187601  290.965340  14.379552  70.678458
1     493.661889  104.513206  41.580752  290.621190  14.315323  69.516257
2     495.644947  104.502457  40.744572  295.416417  14.566180  63.730582
3     494.354041  104.452871  40.288181  292.676229  14.605181  72.736626
4     492.051373  104.488584  41.266692  289.017462  14.548926  76.621067
…          …           …          …          …          …          …
1020  497.999661  104.985408  35.389852  295.193044  14.450879  78.082852
1021  497.139686  104.968285  35.658364  293.975309  14.452064  79.062697
```

```
1022  497.557435  104.958298  35.666902  294.001376  14.437922  79.930899
1023  497.669483  104.916068  35.685112  294.049924  14.449497  80.262698
1024  498.180745  104.924628  35.738588  294.500885  14.453477  80.376672

            c32        c33      c39       c139       c142       c143  c155  \
0     48.679005 -69.203403  0.410096  13.599070  48.457544  37.000000   0.0
1     48.057417 -69.414081  0.409465  13.167193  48.123814  37.000000   0.0
2     47.320586 -69.645378  0.410025  12.611031  45.393828  37.000000   0.0
3     47.980460 -69.452794  0.410889  14.832367  49.131185  37.000000   0.0
4     48.217299 -69.344057  0.411034  15.943873  51.185354  37.000000   0.0
…           …          …        …          …          …          …    …
1020  51.380085 -68.004586  0.550094  12.339225  50.383117  32.000168  14.8
1021  51.606934 -67.893767  0.550332  13.007149  51.008752  32.000168  16.0
1022  52.030272 -67.727372  0.550160  13.366582  51.452608  32.000168  16.0
1023  52.246631 -67.620510  0.550423  13.588131  51.645568  32.000168  16.2
1024  52.382273 -67.546656  0.550027  13.969828  51.737968  32.000168  15.9

      c156  c157  c158  c160  c161  c162  c163
0      0.0   9.0   9.0   450   150   150    50
1      0.0   9.0   9.0   450   150   150    50
2      0.0   9.0   9.0   450   150   150    50
3      0.0   9.0   9.0   450   150   150    50
4      0.0   9.0   9.0   450   150   150    50
…      …     …     …     …     …     …     …
1020   0.0  30.0  22.2   800   370   160    80
1021   0.0  30.5  22.6   800   400   200    55
1022   0.0  31.5  22.6   800   370   200    55
1023   0.0  30.0  22.7   800   350   100    50
1024   0.0  30.5  22.3   800   200   400   100

[1025 rows x 20 columns]
```

[29]:
```python
# Assuming the data has been preprocessed and split into features (X) and
↪target (y)

# Selecting 'c52' as the target variable
target_column_c52 = 'c52'
X = df_controllable_filtered  # Features
y = df[target_column_c52]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c52, X_test_c52, y_train_c52, y_test_c52 = train_test_split(X, y,
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c52
from sklearn.ensemble import RandomForestRegressor
```

```python
# Train the model for c52
model_c52 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c52.fit(X_train_c52, y_train_c52)

# Prediction on test set for c52
predictions_c52 = model_c52.predict(X_test_c52)

# Evaluation metrics for c52
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c52
mse_c52 = mean_squared_error(y_test_c52, predictions_c52)
print(f"Mean Squared Error for c52: {mse_c52}")

# Calculate R-squared for c52
r2_c52 = r2_score(y_test_c52, predictions_c52)
print(f"R-squared for c52: {r2_c52}")

# Visualization for c52
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c52
plt.scatter(y_test_c52.index, y_test_c52, label='Actual Values', color='blue',␣
 ↪alpha=0.5)
plt.scatter(y_test_c52.index, predictions_c52, label='Predicted Values',␣
 ↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c52')
plt.title('Actual vs. Predicted Vibration Levels for c52')
plt.legend()
plt.show()
```

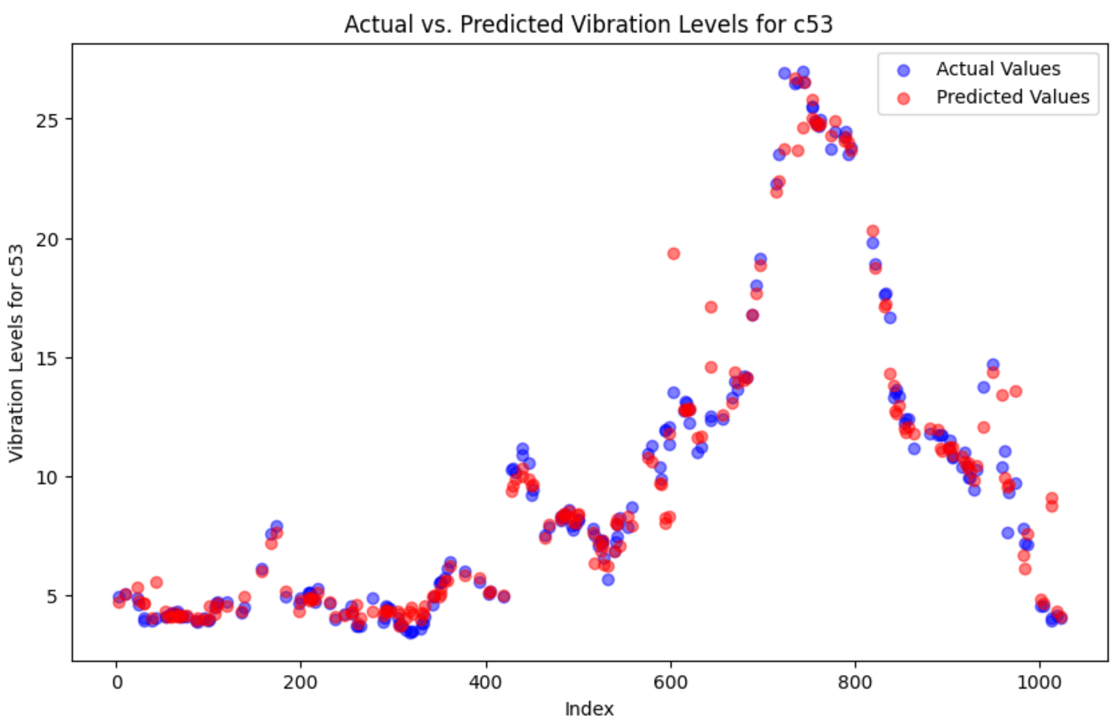
```
Mean Squared Error for c52: 0.15923022309906587
R-squared for c52: 0.9674925248170679
```



[30]:
```
# Assuming the data has been preprocessed and split into features (X) and
↪target (y)

# Selecting 'c53' as the target variable
target_column_c53 = 'c53'
X = df_controllable_filtered  # Features
y = df[target_column_c53]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c53, X_test_c53, y_train_c53, y_test_c53 = train_test_split(X, y,
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c52
from sklearn.ensemble import RandomForestRegressor

# Train the model for c52
model_c53 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c53.fit(X_train_c53, y_train_c53)

# Prediction on test set for c52
predictions_c53 = model_c53.predict(X_test_c53)

# Evaluation metrics for c52
```

```python
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c52
mse_c53 = mean_squared_error(y_test_c53, predictions_c53)
print(f"Mean Squared Error for c53: {mse_c53}")

# Calculate R-squared for c53
r2_c53 = r2_score(y_test_c53, predictions_c53)
print(f"R-squared for c53: {r2_c53}")

# Visualization for c53
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c52
plt.scatter(y_test_c53.index, y_test_c53, label='Actual Values', color='blue',␣
 ↪alpha=0.5)
plt.scatter(y_test_c53.index, predictions_c53, label='Predicted Values',␣
 ↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c53')
plt.title('Actual vs. Predicted Vibration Levels for c53')
plt.legend()
plt.show()
```

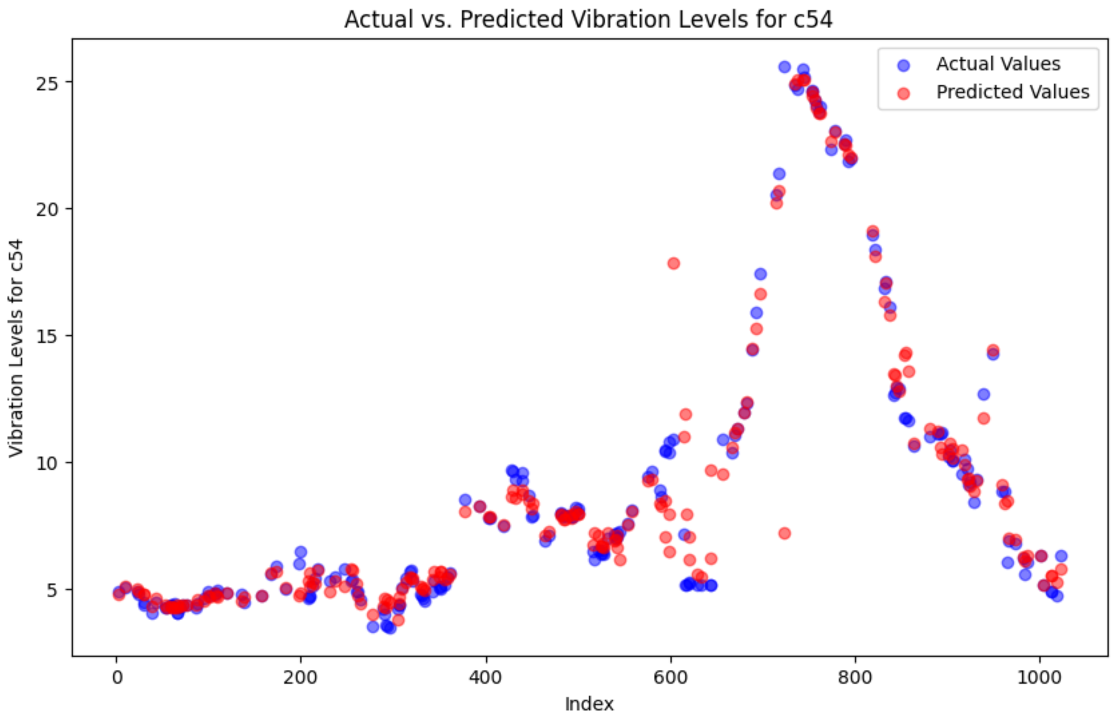
```
Mean Squared Error for c53: 1.212257674269031
R-squared for c53: 0.9683511115174663
```



[31]:
```python
# Assuming the data has been preprocessed and split into features (X) and
 target (y)

# Selecting 'c54' as the target variable
target_column_c54 = 'c54'
X = df_controllable_filtered  # Features
y = df[target_column_c54]   # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c54, X_test_c54, y_train_c54, y_test_c54 = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c54
from sklearn.ensemble import RandomForestRegressor

# Train the model for c54
model_c54 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c54.fit(X_train_c54, y_train_c54)

# Prediction on test set for c52
predictions_c54 = model_c54.predict(X_test_c54)

# Evaluation metrics for c54
```

```python
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c54
mse_c54 = mean_squared_error(y_test_c54, predictions_c54)
print(f"Mean Squared Error for c54: {mse_c54}")

# Calculate R-squared for c53
r2_c54 = r2_score(y_test_c54, predictions_c54)
print(f"R-squared for c54: {r2_c54}")

# Visualization for c53
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c52
plt.scatter(y_test_c54.index, y_test_c54, label='Actual Values', color='blue',
 ↪alpha=0.5)
plt.scatter(y_test_c54.index, predictions_c54, label='Predicted Values',
 ↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c54')
plt.title('Actual vs. Predicted Vibration Levels for c54')
plt.legend()
plt.show()
```

```
Mean Squared Error for c54: 2.815548632675955
R-squared for c54: 0.9110286000147784
```



[32]:
```
# Selecting 'c51' as the target variable
target_column_c51 = 'c51'
X = df_controllable_filtered  # Features for c51
y = df[target_column_c51]  # Target for c51

# Train-test split for c51
from sklearn.model_selection import train_test_split
X_train_c51, X_test_c51, y_train_c51, y_test_c51 = train_test_split(X, y,␣
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c51
from sklearn.ensemble import RandomForestRegressor

# Train the model for c51
model_c51 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c51.fit(X_train_c51, y_train_c51)

# Prediction on test set for c51
predictions_c51 = model_c51.predict(X_test_c51)

# Linear regression model for p-values of c51
import statsmodels.api as sm

# Perform linear regression to get p-values for c51
```

```
X_train_with_const_c51 = sm.add_constant(X_train_c51)
model_c51_pvalues = sm.OLS(y_train_c51, X_train_with_const_c51).fit()

# Show first five p-values for c51 in ascending order
pvalues_c51 = model_c51_pvalues.pvalues.sort_values()
print("First five p-values for c51:")
print(pvalues_c51.head())

# Check predictions and trigger alarm for c51
c51_alarms = []
for val in predictions_c51:
    if val > 20:
        c51_alarms.append("CRITICAL")
    elif val >= 10:
        c51_alarms.append("HIGH")
    else:
        c51_alarms.append("SAFE")

print("Alarms for c51:", c51_alarms)

# Selecting 'c52' as the target variable
target_column_c52 = 'c52'
X = df_controllable_filtered  # Features for c52
y = df[target_column_c52]  # Target for c52

# Train-test split for c52
X_train_c52, X_test_c52, y_train_c52, y_test_c52 = train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c52
model_c52 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c52.fit(X_train_c52, y_train_c52)

# Prediction on test set for c52
predictions_c52 = model_c52.predict(X_test_c52)

# Linear regression model for p-values of c52
X_train_with_const_c52 = sm.add_constant(X_train_c52)
model_c52_pvalues = sm.OLS(y_train_c52, X_train_with_const_c52).fit()

# Show first five p-values for c52 in ascending order
pvalues_c52 = model_c52_pvalues.pvalues.sort_values()
print("First five p-values for c52:")
print(pvalues_c52.head())

# Check predictions and trigger alarm for c52
c52_alarms = []
```

```python
for val in predictions_c52:
    if val > 20:
        c52_alarms.append("CRITICAL")
    elif val >= 10:
        c52_alarms.append("HIGH")
    else:
        c52_alarms.append("SAFE")

print("Alarms for c52:", c52_alarms)
```

```
First five p-values for c51:
c156    0.000000e+00
c157    1.006006e-17
c39     2.975249e-16
c158    1.122867e-14
c32     2.475449e-13
dtype: float64
Alarms for c51: ['SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'HIGH',
'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'HIGH', 'SAFE',
'HIGH', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'HIGH',
'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'HIGH', 'HIGH',
'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'HIGH', 'HIGH',
'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'HIGH', 'HIGH', 'SAFE', 'SAFE',
'HIGH', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE',
'SAFE', 'HIGH', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE',
'SAFE', 'HIGH', 'HIGH', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE',
'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH',
'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'HIGH', 'HIGH']
First five p-values for c52:
c156    0.000000e+00
c27     7.473334e-14
c39     1.012629e-13
c163    1.914934e-11
c157    1.636827e-09
dtype: float64
Alarms for c52: ['SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH',
'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE',
```

```
'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE',
'HIGH', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE',
'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE',
'HIGH', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE',
'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE',
'HIGH', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE',
'SAFE', 'SAFE', 'HIGH', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE',
'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE',
'HIGH', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE', 'HIGH', 'HIGH',
'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'SAFE', 'HIGH', 'SAFE', 'SAFE',
'SAFE', 'SAFE', 'HIGH', 'HIGH', 'SAFE', 'SAFE', 'HIGH']
```

[33]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
import statsmodels.api as sm

# Selecting 'c51' as the target variable
target_column_c51 = 'c51'
X = df_controllable_filtered  # Features for c51
y = df[target_column_c51]  # Target for c51

# Train-test split for c51
X_train_c51, X_test_c51, y_train_c51, y_test_c51 = train_test_split(X, y,␣
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c51
model_c51 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c51.fit(X_train_c51, y_train_c51)

# Linear regression model for p-values of c51
X_train_with_const_c51 = sm.add_constant(X_train_c51)
model_c51_pvalues = sm.OLS(y_train_c51, X_train_with_const_c51).fit()

# Get the coefficients for c51
coefficients_c51 = model_c51_pvalues.params.abs().drop('const', errors='ignore')
top_five_coeffs_c51 = coefficients_c51.nlargest(5)
print("Top five highest magnitude coefficients for c51:")
print(top_five_coeffs_c51)
```

```python
# Calculate R-squared for c51
r2_c51 = r2_score(y_test_c51, model_c51.predict(X_test_c51))
print("R-squared for c51:", r2_c51)

# Selecting 'c52' as the target variable
target_column_c52 = 'c52'
X = df[controllable_columns]  # Features for c52
y = df[target_column_c52]  # Target for c52

# Train-test split for c52
X_train_c52, X_test_c52, y_train_c52, y_test_c52 = train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c52
model_c52 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c52.fit(X_train_c52, y_train_c52)

# Linear regression model for p-values of c52
X_train_with_const_c52 = sm.add_constant(X_train_c52)
model_c52_pvalues = sm.OLS(y_train_c52, X_train_with_const_c52).fit()

# Get the coefficients for c52
coefficients_c52 = model_c52_pvalues.params.abs().drop('const', errors='ignore')
top_five_coeffs_c52 = coefficients_c52.nlargest(5)
print("Top five highest magnitude coefficients for c52:")
print(top_five_coeffs_c52)

# Calculate R-squared for c52
r2_c52 = r2_score(y_test_c52, model_c52.predict(X_test_c52))
print("R-squared for c52:", r2_c52)
```

```
Top five highest magnitude coefficients for c51:
c156    9.417776
c157    1.123372
c39     0.925989
c32     0.873966
c158    0.827802
dtype: float64
R-squared for c51: 0.9359045196858679
Top five highest magnitude coefficients for c52:
c39     7.471601
c142    2.001005
c33     1.275193
c31     0.994919
c28     0.724889
dtype: float64
```

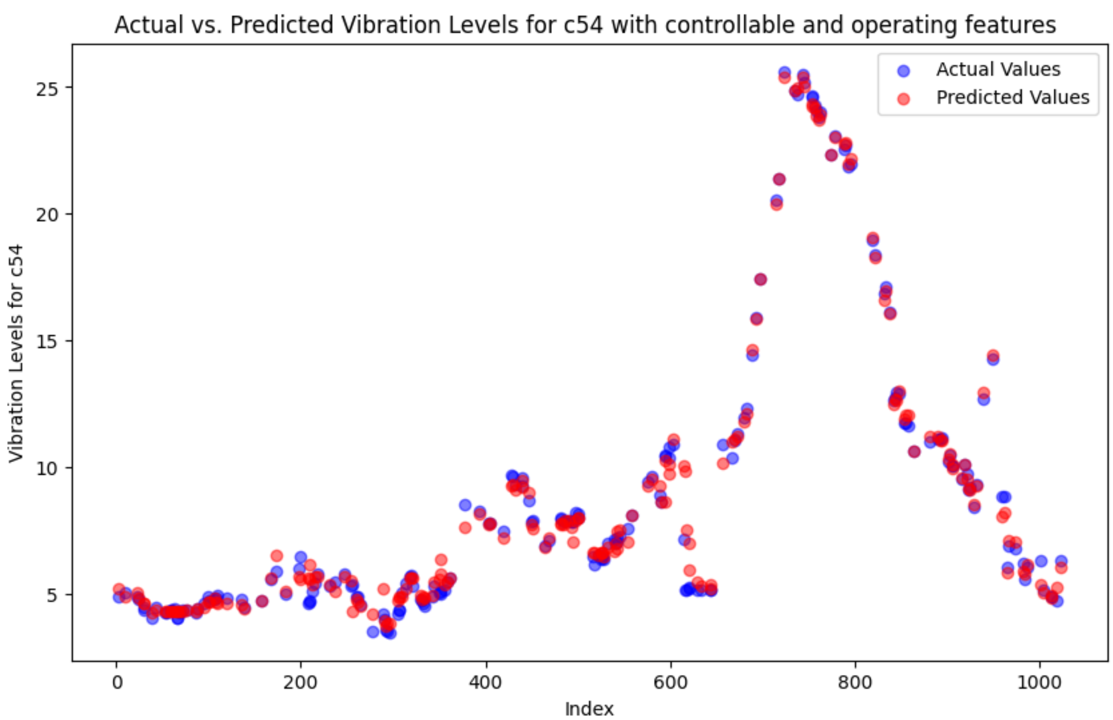
```
R-squared for c52: 0.9679653145771651
```

[35]:
```python
# Assuming df_controllable_filtered is a DataFrame containing controllable␣
↪features

# Selecting 'c51' and 'c52' from the original DataFrame
c51_c52 = df[['c51', 'c52']]

# Concatenating 'c51', 'c52', and df_controllable_filtered into a new DataFrame
features_c51_c52_controllable = pd.concat([c51_c52, df_controllable_filtered],␣
↪axis=1)
```

[36]:
```python
# Assuming the data has been preprocessed and split into features (X) and␣
↪target (y)
c51_c52_c53 = df[['c51', 'c52', 'c53']]

# Concatenating 'c51', 'c52', and df_controllable_filtered into a new DataFrame
features_c51_c52_c53_controllable = pd.concat([c51_c52_c53,␣
↪df_controllable_filtered], axis=1)

# Selecting 'c54' as the target variable
target_column_c54 = 'c54'
X = features_c51_c52_c53_controllable  # Features
y = df[target_column_c54]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c54, X_test_c54, y_train_c54, y_test_c54 = train_test_split(X, y,␣
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c54
from sklearn.ensemble import RandomForestRegressor

# Train the model for c54
model_c54 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c54.fit(X_train_c54, y_train_c54)

# Prediction on test set for c54
predictions_c54 = model_c54.predict(X_test_c54)

# Evaluation metrics for c54
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c54
mse_c54 = mean_squared_error(y_test_c54, predictions_c54)
print(f"Mean Squared Error for c54: {mse_c54}")
```

```python
# Calculate R-squared for c54
r2_c54 = r2_score(y_test_c54, predictions_c54)
print(f"R-squared for c54: {r2_c54}")

# Visualization for c54
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c54
plt.scatter(y_test_c54.index, y_test_c54, label='Actual Values', color='blue',
    alpha=0.5)
plt.scatter(y_test_c54.index, predictions_c54, label='Predicted Values',
    color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c54')
plt.title('Actual vs. Predicted Vibration Levels for c54 with controllable and
    operating features ')
plt.legend()
plt.show()
```

```
Mean Squared Error for c54: 0.31871110651656576
R-squared for c54: 0.9899287218808692
```

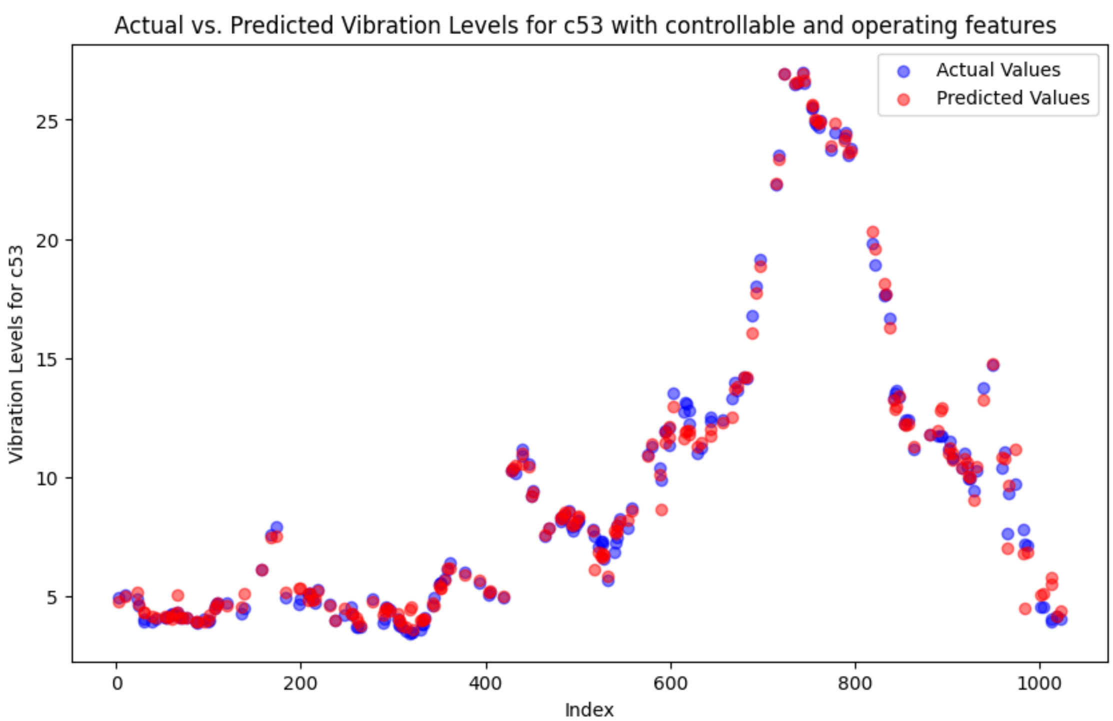


[37]:

```python
# Assuming the data has been preprocessed and split into features (X) and␣
↪target (y)
c51_c52_c54 = df[['c51', 'c52', 'c54']]

# Concatenating 'c51', 'c52', and df_controllable_filtered into a new DataFrame
features_c51_c52_c54_controllable = pd.concat([c51_c52_c54,␣
↪df_controllable_filtered], axis=1)

# Selecting 'c53' as the target variable
target_column_c53 = 'c53'
X = features_c51_c52_c54_controllable  # Features
y = df[target_column_c53]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c53, X_test_c53, y_train_c53, y_test_c53 = train_test_split(X, y,␣
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c53
from sklearn.ensemble import RandomForestRegressor

# Train the model for c53
model_c53 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c53.fit(X_train_c53, y_train_c53)

# Prediction on test set for c53
predictions_c53 = model_c53.predict(X_test_c53)

# Evaluation metrics for c53
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c53
mse_c54 = mean_squared_error(y_test_c53, predictions_c53)
print(f"Mean Squared Error for c53: {mse_c53}")

# Calculate R-squared for c53
r2_c53 = r2_score(y_test_c53, predictions_c53)
print(f"R-squared for c53: {r2_c53}")

# Visualization for c53
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c53
plt.scatter(y_test_c53.index, y_test_c53, label='Actual Values', color='blue',␣
↪alpha=0.5)
```

```
plt.scatter(y_test_c53.index, predictions_c53, label='Predicted Values',␣
 ↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c53')
plt.title('Actual vs. Predicted Vibration Levels for c53 with controllable and␣
 ↪operating features ')
plt.legend()
plt.show()
```

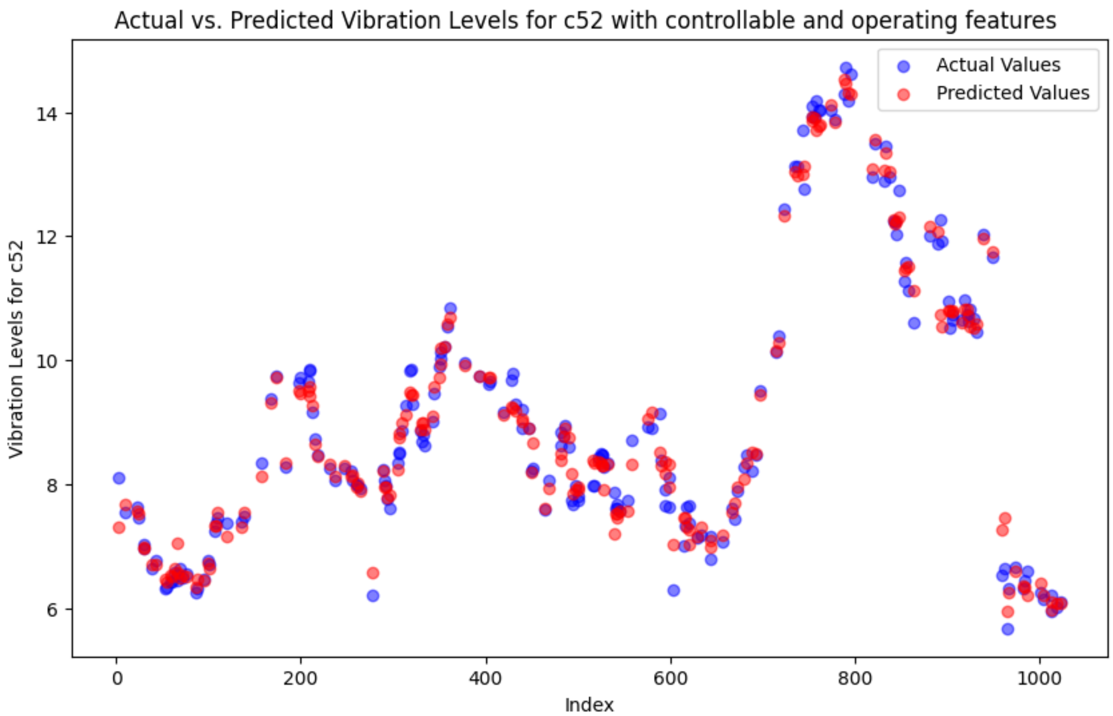
```
Mean Squared Error for c53: 1.212257674269031
R-squared for c53: 0.994472023961037
```



[38]:
```
# Assuming the data has been preprocessed and split into features (X) and␣
 ↪target (y)
c51_c53_c54 = df[['c51', 'c53', 'c54']]

# Concatenating 'c51', 'c52', and df_controllable_filtered into a new DataFrame
features_c51_c53_c54_controllable = pd.concat([c51_c53_c54,␣
 ↪df_controllable_filtered], axis=1)

# Selecting 'c52' as the target variable
target_column_c52 = 'c52'
```

```python
X = features_c51_c53_c54_controllable  # Features
y = df[target_column_c52]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c52, X_test_c52, y_train_c52, y_test_c52 = train_test_split(X, y,␣
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c52
from sklearn.ensemble import RandomForestRegressor

# Train the model for c52
model_c52 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c52.fit(X_train_c52, y_train_c52)

# Prediction on test set for c52
predictions_c52 = model_c52.predict(X_test_c52)

# Evaluation metrics for c52
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c52
mse_c52 = mean_squared_error(y_test_c52, predictions_c52)
print(f"Mean Squared Error for c52: {mse_c52}")

# Calculate R-squared for c52
r2_c52 = r2_score(y_test_c52, predictions_c52)
print(f"R-squared for c52: {r2_c52}")

# Visualization for c52
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c52
plt.scatter(y_test_c52.index, y_test_c52, label='Actual Values', color='blue',␣
↪alpha=0.5)
plt.scatter(y_test_c52.index, predictions_c52, label='Predicted Values',␣
↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c52')
plt.title('Actual vs. Predicted Vibration Levels for c52 with controllable and␣
↪operating features ')
plt.legend()
plt.show()
```

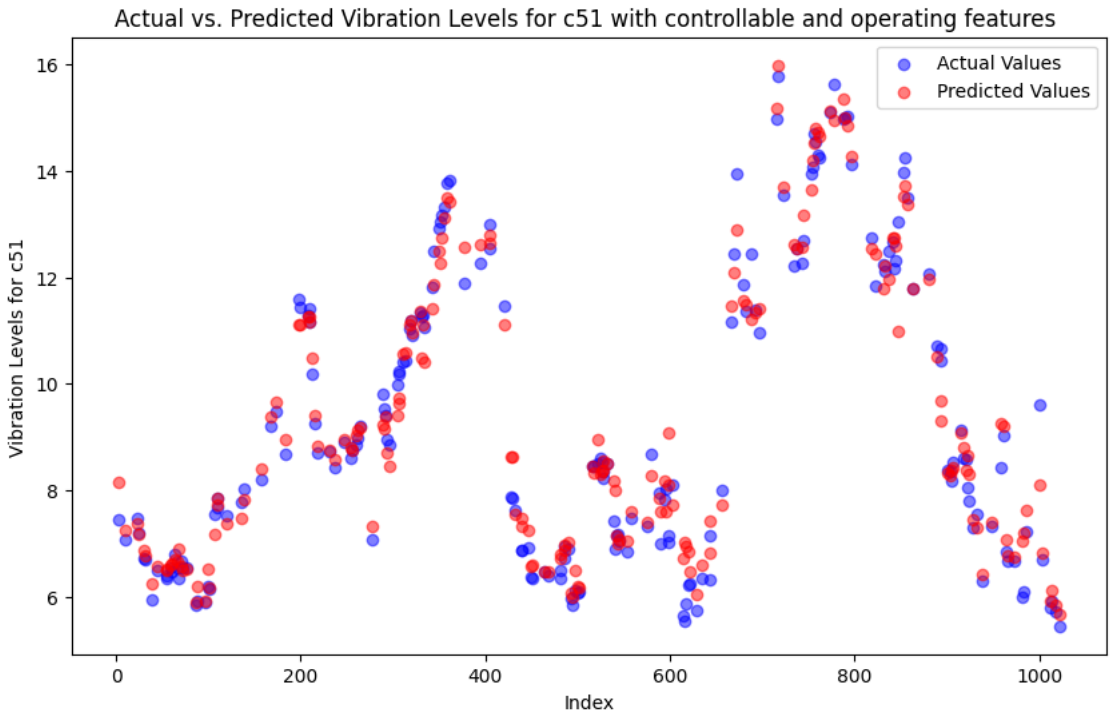
```
Mean Squared Error for c52: 0.07913196566925913
R-squared for c52: 0.9838448985368208
```



[39]:
```python
# Assuming the data has been preprocessed and split into features (X) and
↪target (y)
c52_c53_c54 = df[['c52', 'c53', 'c54']]

# Concatenating 'c51', 'c52', and df_controllable_filtered into a new DataFrame
features_c52_c53_c54_controllable = pd.concat([c52_c53_c54,
↪df_controllable_filtered], axis=1)

# Selecting 'c51' as the target variable
target_column_c52 = 'c51'
X = features_c52_c53_c54_controllable  # Features
y = df[target_column_c51]  # Target

# Train-test split
from sklearn.model_selection import train_test_split
X_train_c51, X_test_c51, y_train_c51, y_test_c51 = train_test_split(X, y,
↪test_size=0.2, random_state=42)

# Model fitting - Random Forest Regression for c51
from sklearn.ensemble import RandomForestRegressor
```

```python
# Train the model for c51
model_c51 = RandomForestRegressor(n_estimators=100, random_state=42)
model_c51.fit(X_train_c51, y_train_c51)

# Prediction on test set for c51
predictions_c51 = model_c51.predict(X_test_c51)

# Evaluation metrics for c51
from sklearn.metrics import mean_squared_error, r2_score

# Calculate Mean Squared Error for c51
mse_c51 = mean_squared_error(y_test_c51, predictions_c51)
print(f"Mean Squared Error for c51: {mse_c51}")

# Calculate R-squared for c51
r2_c51 = r2_score(y_test_c51, predictions_c51)
print(f"R-squared for c51: {r2_c51}")

# Visualization for c51
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Plotting actual vs. predicted values for c51
plt.scatter(y_test_c51.index, y_test_c51, label='Actual Values', color='blue',␣
↪alpha=0.5)
plt.scatter(y_test_c51.index, predictions_c51, label='Predicted Values',␣
↪color='red', alpha=0.5)

plt.xlabel('Index')
plt.ylabel('Vibration Levels for c51')
plt.title('Actual vs. Predicted Vibration Levels for c51 with controllable and␣
↪operating features ')
plt.legend()
plt.show()
```

```
Mean Squared Error for c51: 0.21260748349134898
R-squared for c51: 0.9721216047263032
```

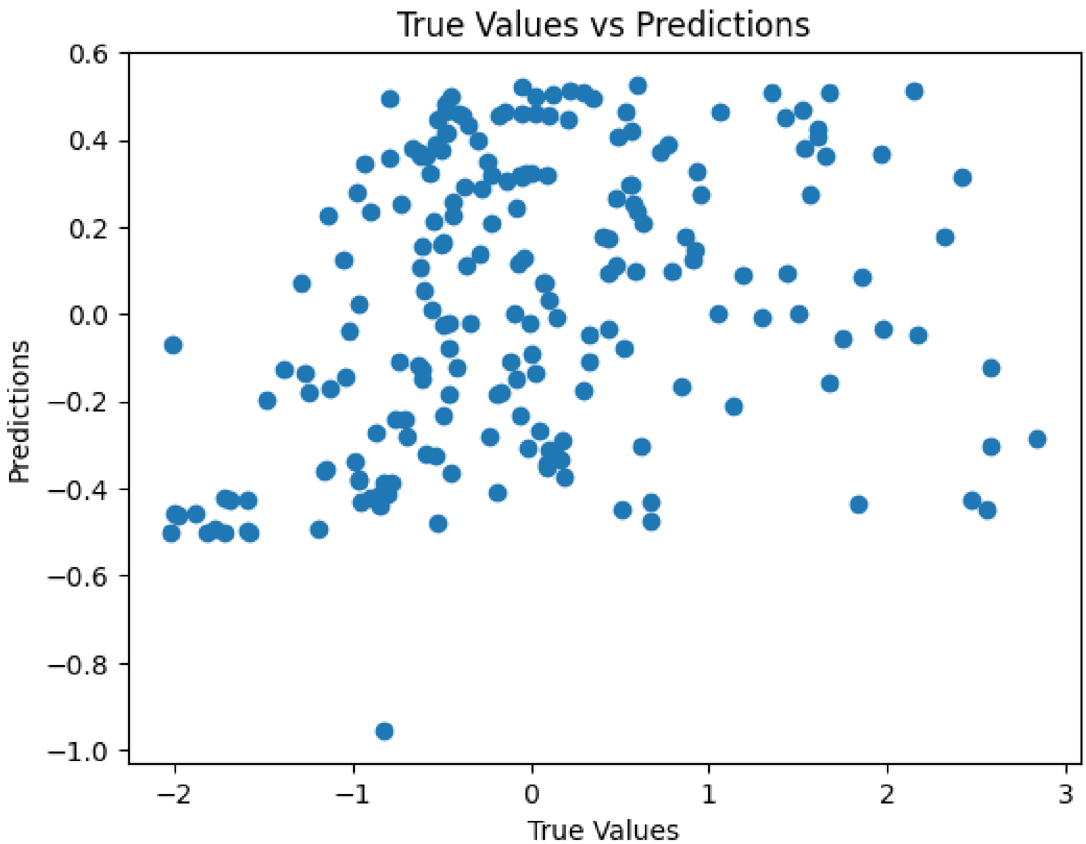


[40]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_selected_filtered,␣
 ↪df_specific_energy, test_size=0.2, random_state=42)
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

model = LinearRegression()
rfe = RFE(model, n_features_to_select=1)
fit = rfe.fit(X_train, y_train)

selected_features = X_train.columns[fit.support_]
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()
model.fit(X_train[selected_features], y_train)

# Predict on the test set
y_pred = model.predict(X_test[selected_features])

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

Mean Squared Error: 0.946863161561553

[41]:
```python
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

# Continue from the existing code
model.fit(X_train[selected_features], y_train)

# Predict on the test set
y_pred = model.predict(X_test[selected_features])

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Scatter plot
plt.scatter(y_test, y_pred)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('True Values vs Predictions')
plt.show()
```

Mean Squared Error: 0.946863161561553
R-squared: 0.07911339942218265



[42]:
```
coef_df = pd.DataFrame({'Feature': selected_features, 'Coefficient': model.
↪coef_[0]})
coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
print(coef_df)
```

```
  Feature  Coefficient
0     c29    -0.319927
```

[43]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

# Assuming df_selected_filtered is your DataFrame with independent variables
# and df_specific_energy is your DataFrame with the target variable 'c241'

# Combine independent variables and target variable
df_combined = pd.concat([df_selected_filtered, df_specific_energy], axis=1)

# Split the data into training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    df_selected_filtered, df_specific_energy, test_size=0.2, random_state=42
)

# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Initialize R^2 value
r2 = 0

# Set the desired R^2 threshold
r2_threshold = 0.85  # Adjust as needed

# Set maximum number of iterations to avoid infinite loop
max_iterations = 10  # You can adjust this limit as needed

# Iterate until the desired R^2 is achieved or the maximum number of iterations␣
 ↪is reached
for iteration in range(1, max_iterations + 1):
    # Train the model
    rf_model.fit(X_train, y_train.values.ravel())

    # Make predictions on the test set
    y_pred = rf_model.predict(X_test)

    # Calculate the R^2 value
    r2 = r2_score(y_test, y_pred)

    # Display the current iteration, R^2 value, and selected features
    print(f"Iteration {iteration}: Current R^2 Score: {r2}")

    # Feature importances
    feature_importances = rf_model.feature_importances_

    # Display feature importances
    feature_importance_df = pd.DataFrame(
        {"Feature": X_train.columns, "Importance": feature_importances}
    ).sort_values(by="Importance", ascending=False)

    print("Feature Importances:")
    print(feature_importance_df)

    # Check if the R^2 threshold is reached
    if r2 >= r2_threshold:
        print(f"Target R^2 ({r2_threshold}) reached in {iteration} iterations.")
        break
```

```python
    # Remove the least important feature
    least_important_feature = feature_importance_df.iloc[-1]["Feature"]
    X_train = X_train.drop(columns=least_important_feature)
    X_test = X_test.drop(columns=least_important_feature)

# If the loop completes without reaching the R^2 threshold, print a message
if r2 < r2_threshold:
    print(f"Maximum number of iterations ({max_iterations}) reached. Current␣
 ↪R^2: {r2}")
```

```
Iteration 1: Current R^2 Score: 0.8162877260610757
Feature Importances:
   Feature  Importance
25    c133    0.237627
20     c42    0.146881
33    c179    0.126713
16     c34    0.074405
14     c29    0.045247
3       c8    0.027065
9      c21    0.022082
34    c238    0.020996
23     c72    0.020543
27    c147    0.017957
32    c177    0.017503
2       c7    0.017117
8      c20    0.016009
0       c5    0.015874
15     c30    0.015549
24     c73    0.015375
7      c16    0.014003
22     c68    0.012293
1       c6    0.011926
21     c63    0.010503
17     c35    0.010475
5      c13    0.010437
4      c12    0.009783
12     c26    0.009641
11     c23    0.009547
26    c146    0.008409
18     c36    0.008347
6      c14    0.007890
35    c239    0.007738
30    c162    0.006910
31    c163    0.006626
10     c22    0.005291
19     c37    0.004911
13     c27    0.004228
```

```
29    c160    0.004097
28    c156    0.000000
Iteration 2: Current R^2 Score: 0.8133221680623501
Feature Importances:
   Feature  Importance
25    c133    0.237631
20     c42    0.146541
32    c179    0.126209
16     c34    0.074191
14     c29    0.045044
3       c8    0.026783
9      c21    0.022999
23     c72    0.021275
33    c238    0.021095
27    c147    0.018699
2       c7    0.018468
31    c177    0.017083
8      c20    0.016307
15     c30    0.015815
24     c73    0.015201
0       c5    0.015129
7      c16    0.013783
22     c68    0.012490
21     c63    0.011516
17     c35    0.011085
1       c6    0.010636
5      c13    0.010504
12     c26    0.009696
18     c36    0.009053
11     c23    0.008607
4      c12    0.008162
26    c146    0.007943
34    c239    0.007816
6      c14    0.007492
29    c162    0.006673
30    c163    0.006534
10     c22    0.005641
19     c37    0.004699
13     c27    0.004691
28    c160    0.004511
Iteration 3: Current R^2 Score: 0.8206186451871905
Feature Importances:
   Feature  Importance
25    c133    0.237724
20     c42    0.147169
31    c179    0.126619
16     c34    0.074375
14     c29    0.044955
```

```
3        c8    0.026457
9       c21    0.022765
23      c72    0.021062
32     c238    0.020854
2        c7    0.019023
30     c177    0.018212
27     c147    0.017403
24      c73    0.016759
8       c20    0.016342
0        c5    0.015469
15      c30    0.015297
7       c16    0.013837
22      c68    0.013276
1        c6    0.013229
17      c35    0.010896
5       c13    0.010693
21      c63    0.010662
12      c26    0.009499
11      c23    0.008816
18      c36    0.008677
4       c12    0.008662
33     c239    0.007732
26     c146    0.007709
6       c14    0.007667
28     c162    0.006881
29     c163    0.005939
10      c22    0.005580
13      c27    0.004981
19      c37    0.004782
Iteration 4: Current R^2 Score: 0.8208731524954036
Feature Importances:
   Feature  Importance
24    c133    0.237977
19     c42    0.147173
30    c179    0.128006
16     c34    0.074237
14     c29    0.045068
3       c8    0.026437
9      c21    0.022958
22     c72    0.021734
31    c238    0.020766
26    c147    0.019062
29    c177    0.018646
2       c7    0.017850
23     c73    0.017561
8      c20    0.016687
15     c30    0.015561
0       c5    0.014983
```

```
21      c68    0.013713
7       c16    0.013134
1        c6    0.012504
5       c13    0.010779
17      c35    0.010752
20      c63    0.010287
25     c146    0.009883
12      c26    0.009493
4       c12    0.009047
18      c36    0.008955
11      c23    0.008510
32     c239    0.008125
6       c14    0.007212
27     c162    0.006889
10      c22    0.005766
28     c163    0.005380
13      c27    0.004868
Iteration 5: Current R^2 Score: 0.8140733977836991
Feature Importances:
   Feature  Importance
23    c133    0.237895
18     c42    0.146801
29    c179    0.127960
15     c34    0.074638
13     c29    0.045367
3       c8    0.026693
9      c21    0.023093
21     c72    0.021839
30    c238    0.020647
25    c147    0.019471
2       c7    0.018559
28    c177    0.018477
22     c73    0.017328
14     c30    0.017123
8      c20    0.016540
0       c5    0.014290
7      c16    0.013507
19     c63    0.012859
1       c6    0.012577
20     c68    0.012095
16     c35    0.011342
5      c13    0.010577
4      c12    0.009900
12     c26    0.009833
11     c23    0.009085
17     c36    0.008929
24    c146    0.008128
6      c14    0.007696
```

```
31    c239    0.007501
26    c162    0.007002
27    c163    0.006682
10     c22    0.005566
Iteration 6: Current R^2 Score: 0.8215059416860512
Feature Importances:
   Feature  Importance
22    c133    0.238523
17     c42    0.147667
28    c179    0.128463
14     c34    0.074558
12     c29    0.045460
3       c8    0.027108
9      c21    0.023939
29    c238    0.021449
20     c72    0.021220
24    c147    0.018764
2       c7    0.018237
27    c177    0.018189
21     c73    0.017455
13     c30    0.016414
8      c20    0.016238
0       c5    0.015003
19     c68    0.013804
1       c6    0.012927
7      c16    0.012247
15     c35    0.011337
18     c63    0.011094
5      c13    0.011060
16     c36    0.010986
4      c12    0.009756
10     c23    0.009724
11     c26    0.009542
23    c146    0.008982
30    c239    0.008384
6      c14    0.007842
25    c162    0.007031
26    c163    0.006599
Iteration 7: Current R^2 Score: 0.8178531372122186
Feature Importances:
   Feature  Importance
22    c133    0.238217
17     c42    0.147654
27    c179    0.127440
14     c34    0.074380
12     c29    0.046058
3       c8    0.027616
9      c21    0.023355
```

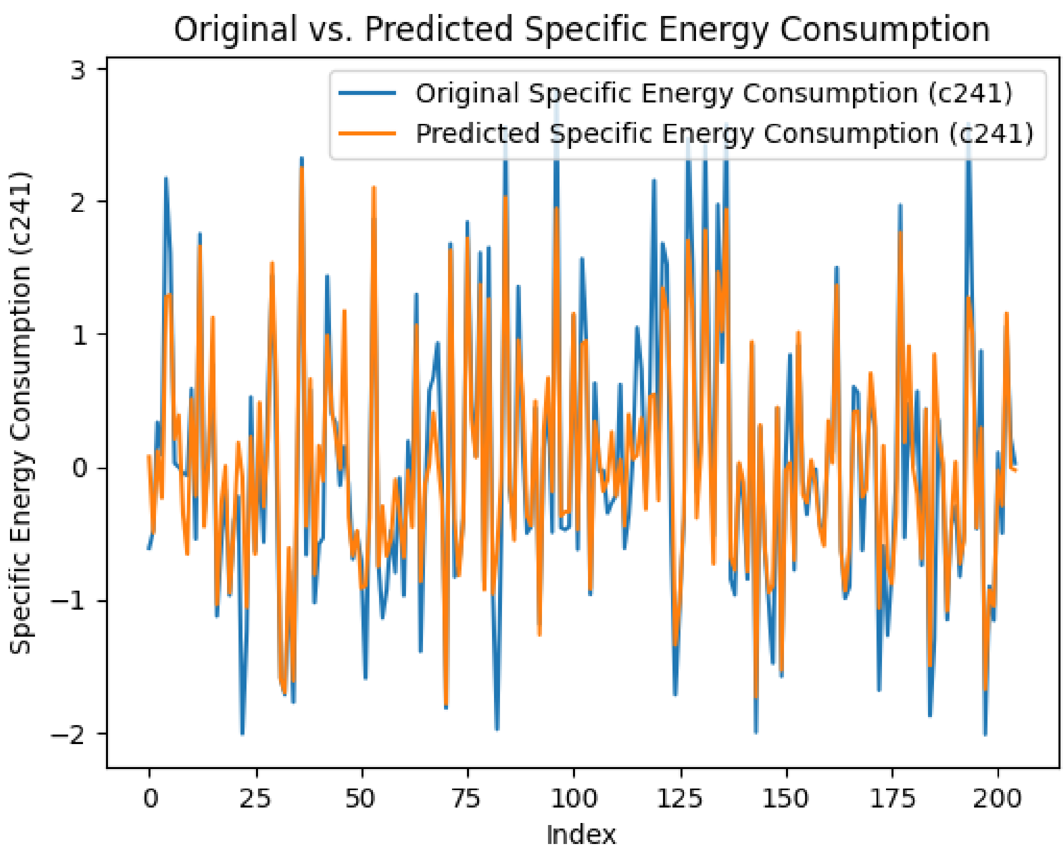
```
20      c72    0.021859
28     c238    0.021668
2        c7    0.019485
24     c147    0.019015
26     c177    0.018548
0        c5    0.017735
21      c73    0.017611
8       c20    0.016652
13      c30    0.015821
19      c68    0.014464
1        c6    0.013529
7       c16    0.013083
15      c35    0.011643
18      c63    0.011219
5       c13    0.010705
11      c26    0.010576
4       c12    0.009620
23     c146    0.009542
6       c14    0.009200
10      c23    0.009090
16      c36    0.009021
29     c239    0.007899
25     c162    0.007296
Iteration 8: Current R^2 Score: 0.8192629191607321
Feature Importances:
   Feature  Importance
22    c133    0.238646
17     c42    0.149204
26    c179    0.129848
14     c34    0.075293
12     c29    0.045529
3       c8    0.027456
9      c21    0.023230
20     c72    0.021874
27    c238    0.021255
2       c7    0.020765
25    c177    0.019528
24    c147    0.019149
21     c73    0.017224
8      c20    0.016812
13     c30    0.016634
0       c5    0.015821
19     c68    0.014947
7      c16    0.012956
1       c6    0.012732
15     c35    0.012186
5      c13    0.011742
18     c63    0.011216
```

```
6      c14    0.010340
11     c26    0.009974
16     c36    0.009965
4      c12    0.009177
10     c23    0.009134
23    c146    0.009092
28    c239    0.008272
Iteration 9: Current R^2 Score: 0.8195018816827709
Feature Importances:
   Feature  Importance
22    c133    0.238886
17     c42    0.149708
26    c179    0.129289
14     c34    0.074851
12     c29    0.045779
3       c8    0.027768
9      c21    0.023930
27    c238    0.022078
20     c72    0.021698
25    c177    0.019887
2       c7    0.019790
24    c147    0.019490
21     c73    0.019232
13     c30    0.017697
0       c5    0.017525
8      c20    0.017423
19     c68    0.013830
1       c6    0.013408
7      c16    0.013020
15     c35    0.012564
18     c63    0.011865
5      c13    0.011720
16     c36    0.010567
4      c12    0.010251
11     c26    0.009975
6      c14    0.009610
10     c23    0.009306
23    c146    0.008852
Iteration 10: Current R^2 Score: 0.8223594644774774
Feature Importances:
   Feature  Importance
22    c133    0.239202
17     c42    0.150177
25    c179    0.128727
14     c34    0.076359
12     c29    0.046462
3       c8    0.028335
9      c21    0.023792
```

```
26    c238    0.022444
20     c72    0.022443
24    c177    0.020203
2       c7    0.020119
23    c147    0.019781
21     c73    0.018681
13     c30    0.017876
8      c20    0.017322
0       c5    0.016586
7      c16    0.015105
1       c6    0.014035
19     c68    0.013758
5      c13    0.013290
18     c63    0.012718
15     c35    0.011862
4      c12    0.011446
11     c26    0.010271
10     c23    0.009714
6      c14    0.009709
16     c36    0.009582
Maximum number of iterations (10) reached. Current R^2: 0.8223594644774774
```

[44]:

```python
import matplotlib.pyplot as plt

# Plotting the original and predicted values
plt.plot(y_test.values, label='Original Specific Energy Consumption (c241)')
plt.plot(y_pred, label='Predicted Specific Energy Consumption (c241)')
plt.title('Original vs. Predicted Specific Energy Consumption')
plt.xlabel('Index')
plt.ylabel('Specific Energy Consumption (c241)')
plt.legend()
plt.show()
```


Original vs. Predicted Specific Energy Consumption
Original Specific Energy Consumption (c241)
Predicted Specific Energy Consumption (c241)
Specific Energy Consumption (c241)
Index
3
2
1
0
−1
−2
0
25
50
75
100
125
150
175
200