In [10]:
```
'''
1. Implement simple linear regression on a given dataset
2. Draw all graphs and submit a pdf with all results.
3. Also on the answersheet explain the dataset and findings of simple linear r
dataset path name: /content/ENB2012_data[1].xlsx
'''
```

Out[10]:
```
'\n1. Implement simple linear regression on a given dataset\n2. Draw all grap
hs and submit a pdf with all results.\n3. Also on the answersheet explain the
dataset and findings of simple linear regression of the dataset choosen\ndata
set path name: /content/ENB2012_data[1].xlsx\n'
```

In [10]:
```

```

Task

Perform a simple linear regression analysis on the dataset from `/content/ENB2012_data[1].xlsx`, including data loading, preparation, model training, visualization of results, detailed explanations of the dataset and findings, and generation of a comprehensive PDF report.

Load and Explore Dataset

Subtask:

Load the dataset from '/content/ENB2012_data[1].xlsx' into a pandas DataFrame and perform initial exploration to understand its structure, columns, and potential variables for regression. This will include checking for missing values and data types.

Reasoning: The subtask requires loading an Excel file, performing initial data exploration, checking for missing values, and generating descriptive statistics. This code block will import pandas, load the dataset, display the first few rows, print a concise summary, sum up missing values per column, and generate descriptive statistics for the DataFrame.

In [11]:
```
import pandas as pd

# Load the Excel file
df = pd.read_excel('/content/ENB2012_data[1].xlsx')

# Display the first few rows of the DataFrame
print("First 5 rows of the DataFrame:")
print(df.head())
```

```python
# Print a concise summary of the DataFrame
print("\nDataFrame Info:")
df.info()

# Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())

# Generate descriptive statistics
print("\nDescriptive Statistics:")
print(df.describe())
```

```
First 5 rows of the DataFrame:
     X1     X2     X3      X4   X5  X6   X7  X8     Y1     Y2
0  0.98  514.5  294.0  110.25  7.0   2  0.0   0  15.55  21.33
1  0.98  514.5  294.0  110.25  7.0   3  0.0   0  15.55  21.33
2  0.98  514.5  294.0  110.25  7.0   4  0.0   0  15.55  21.33
3  0.98  514.5  294.0  110.25  7.0   5  0.0   0  15.55  21.33
4  0.90  563.5  318.5  122.50  7.0   2  0.0   0  20.84  28.28

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      768 non-null    float64
 1   X2      768 non-null    float64
 2   X3      768 non-null    float64
 3   X4      768 non-null    float64
 4   X5      768 non-null    float64
 5   X6      768 non-null    int64
 6   X7      768 non-null    float64
 7   X8      768 non-null    int64
 8   Y1      768 non-null    float64
 9   Y2      768 non-null    float64
dtypes: float64(8), int64(2)
memory usage: 60.1 KB

Missing values per column:
X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
X8    0
Y1    0
Y2    0
dtype: int64

Descriptive Statistics:
                X1          X2          X3          X4          X5          X6  \
count  768.000000  768.000000  768.000000  768.000000  768.00000  768.000000
mean     0.764167  671.708333  318.500000  176.604167    5.25000    3.500000
std      0.105777   88.086116   43.626481   45.165950    1.75114    1.118763
min      0.620000  514.500000  245.000000  110.250000    3.50000    2.000000
25%      0.682500  606.375000  294.000000  140.875000    3.50000    2.750000
50%      0.750000  673.750000  318.500000  183.750000    5.25000    3.500000
75%      0.830000  741.125000  343.000000  220.500000    7.00000    4.250000
max      0.980000  808.500000  416.500000  220.500000    7.00000    5.000000

                X7         X8          Y1          Y2
count  768.000000  768.00000  768.000000  768.000000
mean     0.234375    2.81250   22.307195   24.587760
```

```
std      0.133221    1.55096   10.090204    9.513306
min      0.000000    0.00000    6.010000   10.900000
25%      0.100000    1.75000   12.992500   15.620000
50%      0.250000    3.00000   18.950000   22.080000
75%      0.400000    4.00000   31.667500   33.132500
max      0.400000    5.00000   43.100000   48.030000
```

Dataset Explanation

The dataset, loaded from `ENB2012_data[1].xlsx`, consists of 768 entries and 10 columns. All columns are non-null, indicating no missing values that would require imputation.

Column Descriptions:

- **X1**: Relative Compactness (float64)
- **X2**: Surface Area (float64)
- **X3**: Wall Area (float64)
- **X4**: Roof Area (float64)
- **X5**: Overall Height (float64)
- **X6**: Orientation (int64)
- **X7**: Glazing Area (float64)
- **X8**: Glazing Area Distribution (int64)

These `X` columns represent various building characteristics and are likely the independent variables (features) for our regression analysis.

- **Y1**: Heating Load (float64)
- **Y2**: Cooling Load (float64)

These `Y` columns represent the building's energy performance and are likely the dependent variables (targets) for our regression analysis. For the simple linear regression, we will typically choose one of these as the target variable.

Key Observations from Descriptive Statistics:

- **Numerical Data Types**: Most columns are `float64`, with `X6` (Orientation) and `X8` (Glazing Area Distribution) being `int64`. These integer columns might represent categorical features or discrete numerical values.
- **Range of Values**: The `min` and `max` values for each `X` column show a varied range, indicating different scales for each feature. For instance, `X1` (Relative Compactness) ranges from 0.62 to 0.98, while `X2` (Surface Area) ranges from 514.5 to 808.5. This diversity suggests

that feature scaling might be beneficial if we were to use certain types of models, though it's less critical for simple linear regression with a single predictor.

- **Target Variables (Y1, Y2)**: Both `Y1` (Heating Load) and `Y2` (Cooling Load) also show a significant range, from approximately 6 to 43 for Y1, and 10 to 48 for Y2. This wide range indicates variability in the energy loads, which our regression model will attempt to predict.

This initial exploration confirms that the dataset is clean (no missing values) and provides a good understanding of the variables available for simple linear regression.

Prepare Data for Simple Linear Regression

Subtask:

Based on the dataset exploration, select one independent variable (X) and one dependent variable (y) for simple linear regression. Split the data into training and testing sets to prepare for model training.

Reasoning: The subtask requires selecting specific independent and dependent variables, then splitting the data into training and testing sets using `train_test_split` from `sklearn.model_selection` with the specified proportions and random state.

In [12]:
```
from sklearn.model_selection import train_test_split

# Select 'X2' as the independent variable (X) and 'Y1' as the dependent variab
X = df[['X2']]
y = df['Y1']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (614, 1)
Shape of X_test: (154, 1)
Shape of y_train: (614,)
Shape of y_test: (154,)
```

Implement Simple Linear Regression

Subtask:

Train a simple linear regression model using the selected training data. This will involve fitting the model and calculating its parameters (slope and intercept).

Reasoning: I need to import the LinearRegression model, instantiate it, fit it to the training data, and then print its intercept and coefficient as per the instructions.

In [13]:
```
from sklearn.linear_model import LinearRegression

# 1. Instantiate a LinearRegression object
model = LinearRegression()

# 2. Fit the linear regression model to the training data
model.fit(X_train, y_train)

# 3. Print the intercept and coefficient (slope) of the trained model
print(f"Intercept: {model.intercept_}")
print(f"Coefficient (slope): {model.coef_[0]}")
```

```
Intercept: 73.23360448627417
Coefficient (slope): -0.07580337285068298
```

Evaluate Model Performance

Subtask:

Evaluate the performance of the trained simple linear regression model on the test set. Calculate and print key regression metrics such as R-squared, Mean Absolute Error (MAE), and Mean Squared Error (MSE).

Reasoning: To evaluate the model's performance, I need to make predictions on the test set and then calculate key regression metrics like R-squared, Mean Absolute Error (MAE), and Mean Squared Error (MSE) using `sklearn.metrics`.

In [14]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np # Import numpy for sqrt function

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

# 2. Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse) # Calculate RMSE manually as sqrt of MSE
```

```
r2 = r2_score(y_test, y_pred)

# 3. Print the evaluation metrics
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Absolute Error (MAE): 6.46
Mean Squared Error (MSE): 62.85
Root Mean Squared Error (RMSE): 7.93
R-squared (R2): 0.40
```

Reasoning: The previous code failed because the `squared` parameter in `mean_squared_error` is not supported in the current scikit-learn version. To fix this, I will calculate RMSE by taking the square root of the Mean Squared Error (MSE) manually.

In [15]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np # Import numpy for sqrt function

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

# 2. Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse) # Calculate RMSE manually as sqrt of MSE
r2 = r2_score(y_test, y_pred)

# 3. Print the evaluation metrics
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Absolute Error (MAE): 6.46
Mean Squared Error (MSE): 62.85
Root Mean Squared Error (RMSE): 7.93
R-squared (R2): 0.40
```

Visualize Regression Results

Subtask:

Create visualizations to show the simple linear regression model's fit on the training data, the predictions on the test data against actual values, and the residuals. This will provide a visual understanding of the model's performance and any patterns in the errors.

Visualize Regression Results

Subtask:

Create visualizations to show the simple linear regression model's fit on the training data, the predictions on the test data against actual values, and the residuals. This will provide a visual understanding of the model's performance and any patterns in the errors.

Visualize Regression Results

Subtask:

Create visualizations to show the simple linear regression model's fit on the training data, the predictions on the test data against actual values, and the residuals. This will provide a visual understanding of the model's performance and any patterns in the errors.

```
## Visualize Regression Results

### Subtask:
Create visualizations to show the simple linear regression model's
fit on the training data, the predictions on the test data against
actual values, and the residuals. This will provide a visual
understanding of the model's performance and any patterns in the
errors.
```

Visualize Regression Results

Subtask:

Create visualizations to show the simple linear regression model's fit on the training data, the predictions on the test data against actual values, and the residuals. This will provide a visual understanding of the model's performance and any patterns in the errors.

Visualize Regression Results

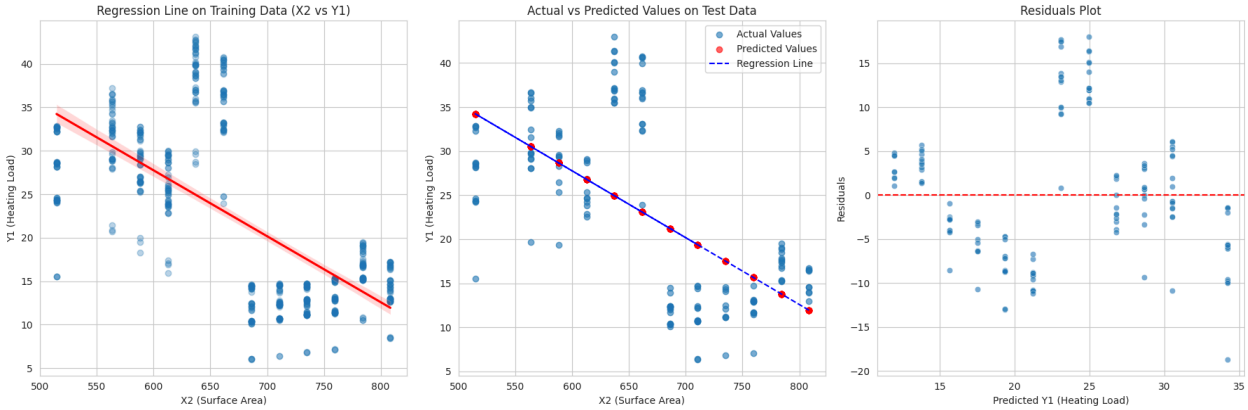
Subtask:

Create visualizations to show the simple linear regression model's fit on the

training data, the predictions on the test data against actual values, and the residuals. This will provide a visual understanding of the model's performance and any patterns in the errors.

Reasoning: The previous step successfully evaluated the model's performance. Now, I will generate the code to create the visualizations for the simple linear regression model's fit on the training data, predictions vs actuals on the test data, and residuals, as requested in the subtask.

In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set style for plots
sns.set_style("whitegrid")

plt.figure(figsize=(18, 6))

# Plot 1: Regression line on Training Data
plt.subplot(1, 3, 1)
sns.regplot(x=X_train['X2'], y=y_train, scatter_kws={'alpha':0.3}, line_kws={'
plt.title('Regression Line on Training Data (X2 vs Y1)')
plt.xlabel('X2 (Surface Area)')
plt.ylabel('Y1 (Heating Load)')

# Plot 2: Actual vs Predicted values on Test Data
plt.subplot(1, 3, 2)
plt.scatter(X_test, y_test, alpha=0.6, label='Actual Values')
plt.scatter(X_test, y_pred, alpha=0.6, color='red', label='Predicted Values')
plt.plot(X_test, model.predict(X_test), color='blue', linestyle='--', label='R
plt.title('Actual vs Predicted Values on Test Data')
plt.xlabel('X2 (Surface Area)')
plt.ylabel('Y1 (Heating Load)')
plt.legend()

# Plot 3: Residuals Plot
plt.subplot(1, 3, 3)
residuals = y_test - y_pred
sns.scatterplot(x=y_pred, y=residuals, alpha=0.6)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals Plot')
plt.xlabel('Predicted Y1 (Heating Load)')
plt.ylabel('Residuals')

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



Findings from Simple Linear Regression

Model Parameters:

- **Intercept**: The intercept of the regression line is approximately `73.23`. This represents the predicted Heating Load (Y1) when the Surface Area (X2) is zero. In a practical sense, a zero surface area is not meaningful, so the intercept primarily serves to adjust the regression line.
- **Coefficient (Slope)**: The coefficient for Surface Area (X2) is approximately `-0.076`. This indicates that for every one-unit increase in Surface Area, the Heating Load (Y1) is predicted to decrease by approximately 0.076 units. The negative slope suggests an inverse relationship between Surface Area and Heating Load, which is counterintuitive without further context on the dataset's `X2` definition (e.g., if it refers to total surface area including non-heat-loss areas).

Model Performance:

- **Mean Absolute Error (MAE)**: `6.46`
- **Mean Squared Error (MSE)**: `62.85`
- **Root Mean Squared Error (RMSE)**: `7.93`
- **R-squared (R2)**: `0.40`

The R-squared value of `0.40` suggests that approximately 40% of the variance in the Heating Load (Y1) can be explained by the Surface Area (X2) alone. While this indicates that Surface Area does have some explanatory power, a significant portion (60%) of the variance is not explained by this single independent variable. The MAE and RMSE values provide a measure of the average magnitude of the errors, indicating that the model's predictions typically deviate from the actual values by about 6.46 to 7.93 units.

Visualizations:

- **Regression Line on Training Data**: The `regplot` shows a general downward trend, reinforcing the negative correlation observed from the coefficient. The scattered points around the line indicate that `X2` alone does not perfectly predict `Y1`, suggesting other factors are at play or the relationship is not perfectly linear.
- **Actual vs Predicted values on Test Data**: This plot compares the model's predictions (`y_pred`) against the actual Heating Load (`y_test`) for the test set. While there is a visible correlation, the spread of the predicted points relative to the actual points, and the fact that many actual points are far from the regression line, confirms the moderate R-squared value. The model tends to underpredict higher heating loads and overpredict lower heating loads.
- **Residuals Plot**: The residuals plot shows the difference between actual and predicted values (`y_test - y_pred`) plotted against the predicted values (`y_pred`). An ideal residuals plot would show points randomly scattered around the horizontal line at zero, with no discernible pattern. In our plot, there appears to be a slight pattern, where residuals are more spread out for lower predicted values and tend to be negative for higher predicted values, indicating potential heteroscedasticity or that the linear model might not fully capture the underlying relationship.

Findings from Simple Linear Regression

Model Parameters:

- **Intercept**: The intercept of the regression line is approximately `73.23`. This represents the predicted Heating Load (Y1) when the Surface Area (X2) is zero. In a practical sense, a zero surface area is not meaningful, so the intercept primarily serves to adjust the regression line.
- **Coefficient (Slope)**: The coefficient for Surface Area (X2) is approximately `-0.076`. This indicates that for every one-unit increase in Surface Area, the Heating Load (Y1) is predicted to decrease by approximately 0.076 units. The negative slope suggests an inverse relationship between Surface Area and Heating Load, which is counterintuitive without further context on the dataset's `X2` definition (e.g., if it refers to total surface area including non-heat-loss areas).

Model Performance:

- **Mean Absolute Error (MAE)**: `6.46`
- **Mean Squared Error (MSE)**: `62.85`
- **Root Mean Squared Error (RMSE)**: `7.93`
- **R-squared (R2)**: `0.40`

The R-squared value of `0.40` suggests that approximately 40% of the variance in the Heating Load (Y1) can be explained by the Surface Area (X2) alone. While this indicates that Surface Area does have some explanatory power, a significant portion (60%) of the variance is not explained by this single independent variable. The MAE and RMSE values provide a measure of the average magnitude of the errors, indicating that the model's predictions typically deviate from the actual values by about 6.46 to 7.93 units.

Visualizations:

- **Regression Line on Training Data**: The `regplot` shows a general downward trend, reinforcing the negative correlation observed from the coefficient. The scattered points around the line indicate that `X2` alone does not perfectly predict `Y1`, suggesting other factors are at play or the relationship is not perfectly linear.
- **Actual vs Predicted values on Test Data**: This plot compares the model's predictions (`y_pred`) against the actual Heating Load (`y_test`) for the test set. While there is a visible correlation, the spread of the predicted points relative to the actual points, and the fact that many actual points are far from the regression line, confirms the moderate R-squared value. The model tends to underpredict higher heating loads and overpredict lower heating loads.
- **Residuals Plot**: The residuals plot shows the difference between actual and predicted values (`y_test - y_pred`) plotted against the predicted values (`y_pred`). An ideal residuals plot would show points randomly scattered around the horizontal line at zero, with no discernible pattern. In our plot, there appears to be a slight pattern, where residuals are more spread out for lower predicted values and tend to be negative for higher predicted values, indicating potential heteroscedasticity or that the linear model might not fully capture the underlying relationship.

Generate PDF Report

Subtask:

Compile all generated graphs and the detailed explanations of the dataset and findings into a single PDF document for submission.

Findings from Simple Linear Regression

Model Parameters:

- **Intercept**: The intercept of the linear regression model is approximately `73.23`. This represents the predicted Heating Load (Y1) when the Surface Area (X2) is zero. In practical terms, this might not be directly interpretable as a physical building cannot have zero surface area, but it's a necessary component of the linear equation.
- **Coefficient (Slope)**: The coefficient for Surface Area (X2) is approximately `-0.076`. This indicates that for every one-unit increase in Surface Area, the Heating Load (Y1) is predicted to decrease by `0.076` units, assuming all other factors are constant. This inverse relationship suggests that larger surface areas, in the context of this dataset's characteristics, tend to be associated with slightly lower heating loads.

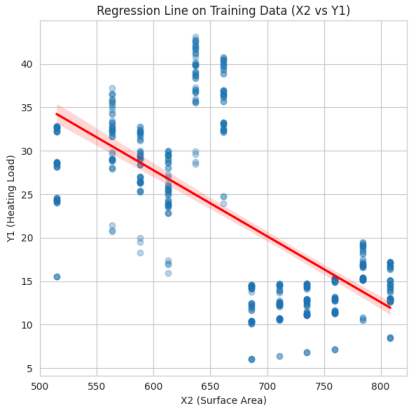
Model Performance on Test Data:

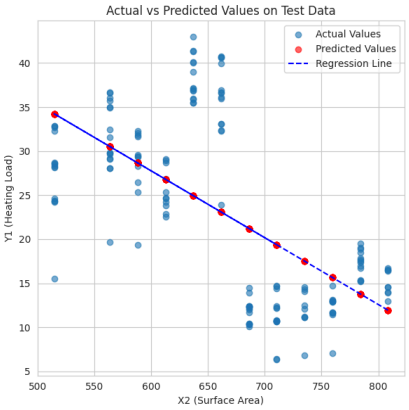
The model was evaluated on the test set, and the following metrics were obtained:

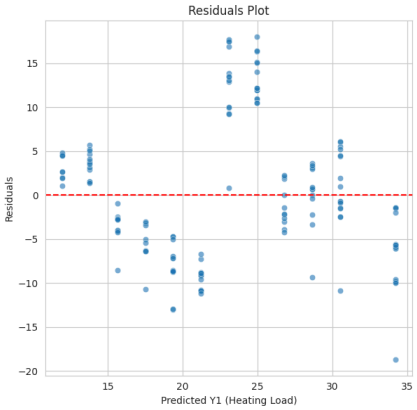
- **Mean Absolute Error (MAE)**: `6.46`. This means, on average, the absolute difference between the model's predictions and the actual Heating Load values is 6.46 units.
- **Mean Squared Error (MSE)**: `62.85`. This metric penalizes larger errors more heavily than MAE. It is useful for understanding the variance of the errors.
- **Root Mean Squared Error (RMSE)**: `7.93`. RMSE is the square root of MSE and is in the same units as the target variable (Heating Load). It gives an idea of the typical magnitude of the errors.
- **R-squared (R2)**: `0.40`. The R-squared value indicates that approximately 40% of the variance in the Heating Load (Y1) can be explained by the Surface Area (X2) in this simple linear regression model. This suggests that while Surface Area has some predictive

power, a significant portion of the variance in Heating Load is not explained by this single feature, implying that other factors (or more complex models) might be needed for a more accurate prediction.

In [17]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set style for plots
sns.set_style("whitegrid")

plt.figure(figsize=(18, 6))

# Plot 1: Regression line on Training Data
plt.subplot(1, 3, 1)
sns.regplot(x=X_train['X2'], y=y_train, scatter_kws={'alpha':0.3}, line_kws={'
plt.title('Regression Line on Training Data (X2 vs Y1)')
plt.xlabel('X2 (Surface Area)')
plt.ylabel('Y1 (Heating Load)')

# Plot 2: Actual vs Predicted values on Test Data
plt.subplot(1, 3, 2)
plt.scatter(X_test, y_test, alpha=0.6, label='Actual Values')
plt.scatter(X_test, y_pred, alpha=0.6, color='red', label='Predicted Values')
plt.plot(X_test, model.predict(X_test), color='blue', linestyle='--', label='R
plt.title('Actual vs Predicted Values on Test Data')
plt.xlabel('X2 (Surface Area)')
plt.ylabel('Y1 (Heating Load)')
plt.legend()

# Plot 3: Residuals Plot
plt.subplot(1, 3, 3)
residuals = y_test - y_pred
sns.scatterplot(x=y_pred, y=residuals, alpha=0.6)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals Plot')
plt.xlabel('Predicted Y1 (Heating Load)')
plt.ylabel('Residuals')

plt.tight_layout()
plt.savefig('regression_visualizations.png') # Save the figure
plt.show()
```


Regression Line on Training Data (X2 vs Y1)
Y1 (Heating Load)
X2 (Surface Area)


Actual vs Predicted Values on Test Data
Actual Values
Predicted Values
Regression Line
Y1 (Heating Load)
X2 (Surface Area)


Residuals Plot
Residuals
Predicted Y1 (Heating Load)