Data Loading: Load the data from the CSV file. Data Preprocessing: Drop non-numeric columns if they are not needed for modeling. Data Scaling: Scale the features due to the difference in ranges. Model Training: Split the data into training and test sets. Train a linear regression model on the training set. Model Evaluation: Evaluate the model on the test set using appropriate metrics. Visualization: Plot the actual vs. predicted values for evaluation. Coefficient Analysis: Analyze the model coefficients to understand the influence of each chemical component.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
```

Data Loading: The dataset is loaded from a CSV file into a Pandas DataFrame. A raw string path is used to prevent errors from escape characters in the file path.

In [ ]:
```
# Step 1: Data Loading
data = pd.read_csv(r'C:\Users\Elena\Documents\GitHub\steel_strength\metals_data.csv
```

If there's a 'formula' column with non-numeric data, it should be dropped since linear regression requires numerical input.

In [ ]:
```
# Step 2: Data Preprocessing (Assuming 'formula' column to be dropped)
# data = data.drop('formula', axis=1)  # if 'formula' column is non-numeric and not
```

The features (independent variables) and outputs (dependent variables) are separated. Here, it's assumed that all columns except the last three are features (chemical components of the alloys), and the last three columns are outputs (mechanical properties like yield strength, tensile strength, and elongation).

In [ ]:
```
# Step 3: Feature Selection
# Assuming all columns except the last three are features (chemical components)
features = data.iloc[:, :-3]  # Exclude last three columns which are mechanical pro
outputs = data.iloc[:, -3:]   # Last three columns as outputs
```

StandardScaler standardizes the features by removing the mean and scaling to unit variance. This is crucial when features have different scales and ranges.

In [ ]:
```
# Step 4: Data Scaling
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

The dataset is split into training and test sets with an 80/20 ratio, and a linear regression model is initialized. For each output variable, a separate model is trained using the training

data.

In [ ]:
```
# Step 5: Model Training
# Split data
X_train, X_test, y_train, y_test = train_test_split(features_scaled, outputs, test_

# Initialize model
model = LinearRegression()

# Train model for each mechanical property
models = {}
for col in outputs:
    model.fit(X_train, y_train[col])
    models[col] = model.coef_
```

For each output, predictions are made using the test set. Mean squared error (MSE) is calculated for each prediction compared to the actual values, which quantifies the average squared difference between estimated values and actual value.

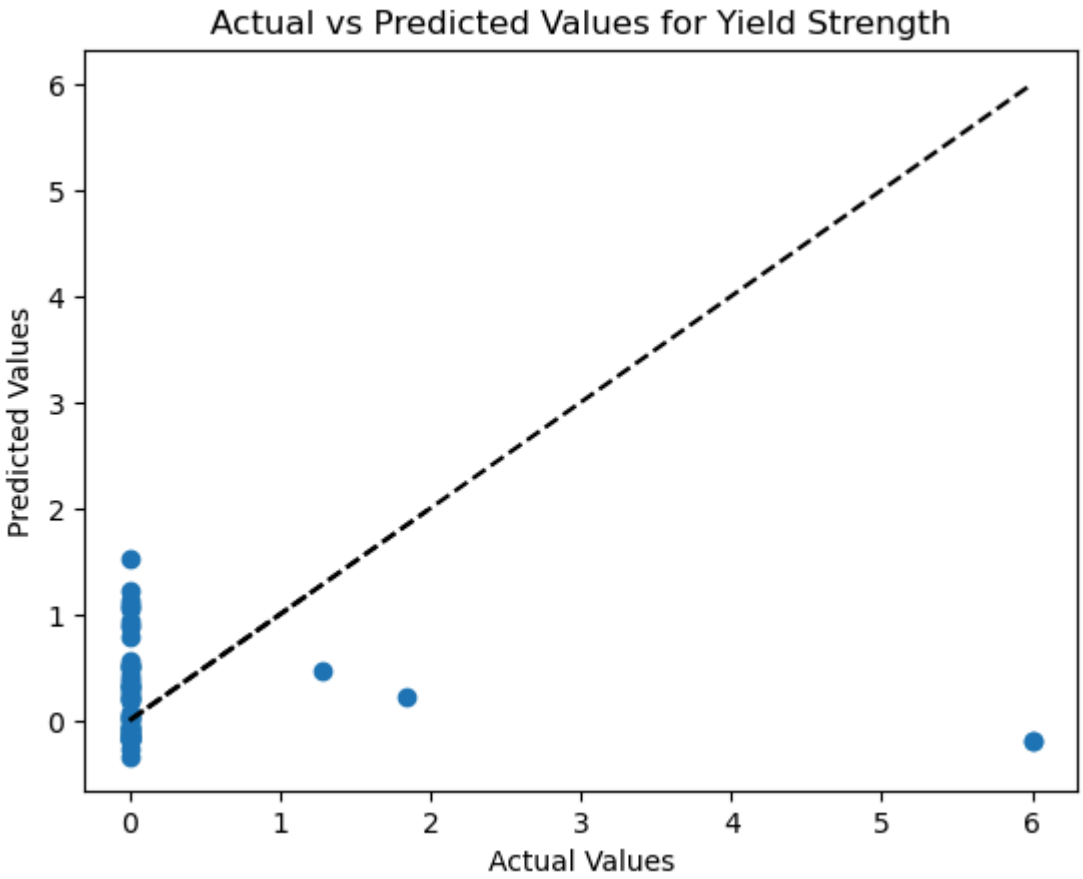
In [ ]:
```
# Step 6: Model Evaluation
predictions = {}
for col in outputs:
    predictions[col] = model.predict(X_test)
    mse = mean_squared_error(y_test[col], predictions[col])
    print(f'MSE for {col}:', mse)
```

```
MSE for w: 1.510297792372733
MSE for al: 0.26714664308621694
MSE for ti: 0.12480798374420149
```

Visualization: A scatter plot is generated comparing the actual vs. predicted values for the first mechanical property in the outputs. A diagonal line (y=x) is also plotted to represent the perfect predictions for context.

The coefficients (weights) of each feature for predicting the output are printed out. These coefficients indicate the relative influence of each chemical component on the predicted mechanical property.

In [ ]:
```
# Step 7: Visualization
# Plotting actual vs predicted values for the first mechanical property
plt.scatter(y_test.iloc[:, 0], predictions[outputs.columns[0]])
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values for Yield Strength')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--')  # Diag
plt.show()
```



In [ ]:
```
# Step 8: Coefficient Analysis
# Print out coefficients for each output
for col in outputs:
    print(f'Coefficients for {col}:', models[col])
```

```
Coefficients for w: [-0.18451812 -0.01381481 -0.10818113 -0.55391007 -0.75022121 -0.
34377962
  0.04524908  0.07902058 -0.1470551  -0.10459605]
Coefficients for al: [-0.01835512 -0.04346689 -0.04995344  0.23702275  0.20984778
0.06747065
 -0.02383538 -0.05781436  0.03395742 -0.07753892]
Coefficients for ti: [-0.03979393 -0.00481166 -0.08767851 -0.06316696  0.36662686
0.01787687
  0.06423954  0.03082288  0.00716187  0.07413564]
```