

Regression & Its Evaluation | Assignment

Question 1: What is Simple Linear Regression?

Answer: Simple Linear Regression is a statistical method used to model the relationship between two variables by fitting a straight line to observed data. One variable acts as the predictor (independent variable, x) and the other as the response (dependent variable, y).

Key Concepts of Simple Linear Regression

- Definition: It explains how the dependent variable y changes as the independent variable x changes, assuming a linear relationship.
- Equation:
$$y = \beta_0 + \beta_1 x + \epsilon$$
 - β_0 : Intercept (value of y when $x=0$)
 - β_1 : Slope (change in y for a one-unit change in x)
 - ϵ : Error term (captures randomness or unexplained variation)
- Example: Predicting a person's height (y) based on their weight (x). If the slope is positive, taller height tends to correspond with higher weight.
- Assumptions:
 - Linearity: Relationship between x and y is linear
 - Independence: Observations are independent
 - Homoscedasticity: Constant variance of errors
 - Normality: Errors are normally distributed

Why It's Important

- **Foundational model:** It's one of the simplest and most widely used regression techniques, forming the basis for more complex models.
- **Interpretability:** Easy to understand and explain — slope and intercept provide clear insights into the relationship.
- **Applications:**
 - Economics (e.g., GDP vs. unemployment rate)
 - Business (sales vs. advertising spend)
 - Science (temperature vs. reaction rate)

Simple Linear Regression is about drawing the *best-fit straight line* through data points to quantify and predict the relationship between two variables. It's simple, powerful, and the starting point for most regression analysis.

Question 2: What are the key assumptions of Simple Linear Regression?

Answer: The key assumptions of Simple Linear Regression are: linearity, independence, homoscedasticity, and normality of residuals. These assumptions ensure that the model produces reliable estimates and valid statistical inferences.

Detailed Breakdown of Assumptions

- **Linearity**
 - The relationship between the independent variable (x) and dependent variable (y) must be linear.
 - If the relationship is non-linear, the regression line will misrepresent the data, leading to biased predictions.
- **Independence**
 - Observations should be independent of each other.
 - Residuals (errors) must not be correlated — especially important in time series data where autocorrelation can occur.
- **Homoscedasticity (Constant Variance)**
 - The variance of residuals should remain constant across all levels of the independent variable.
 - If variance changes (heteroscedasticity), predictions become less reliable and hypothesis tests may be invalid.

Normality of Residuals

- Residuals should follow a normal distribution.
- This assumption is crucial for valid hypothesis testing and confidence intervals.

Why These Assumptions Matter

- **Violation of assumptions** can lead to misleading results:
- Non-linearity → poor fit and biased estimates
- Correlated residuals → underestimated standard errors
- Heteroscedasticity → unreliable confidence intervals
- Non-normal residuals → invalid hypothesis tests

Simple Linear Regression is powerful but only when its **four assumptions (linearity, independence, homoscedasticity, normality)** are satisfied. Analysts often check these assumptions using **residual plots, statistical tests (like Durbin-Watson for independence), and transformations** if needed.

Question 3: What is heteroscedasticity, and why is it important to address in regression models?

Answer:

What is Heteroscedasticity?

- **Definition:** Heteroscedasticity occurs when the variance of the residuals (errors) in a regression model is not constant across all levels of the independent variable(s).
- **Visual clue:** If you plot residuals vs. predicted values, instead of a random scatter, you might see a funnel shape — residuals spread out more as values increase (or decrease).

Mathematically, instead of assuming:

$$\text{Var}(\epsilon_i) = \sigma^2 \quad \text{constant}$$

we have:

$$\text{Var}(\epsilon_i) \neq \sigma^2 \quad \text{changes with } x_i$$

Why is it Important?

- Bias in standard errors: Coefficient estimates (β_0, β_1) remain unbiased, but the standard errors become unreliable.
- Invalid hypothesis tests: t-tests and F-tests may give misleading results because confidence intervals and p-values are distorted.
- Poor predictions: The model may under- or overestimate uncertainty, making forecasts less trustworthy.
- Business impact: For example, in finance, heteroscedasticity is common (volatility clustering). Ignoring it could lead to underestimating risk.

How to Detect It

- Residual plots: Look for patterns (funnel shape, increasing spread).
- Statistical tests: Breusch–Pagan test, White's test.

How to Address It

- Transform variables: Apply log or square root transformations to stabilize variance.
- Weighted least squares (WLS): Give less weight to observations with higher variance.
- Robust standard errors: Adjust standard errors to remain valid even under heteroscedasticity.

Heteroscedasticity doesn't bias regression coefficients, but it undermines the reliability of inference and prediction. That's why analysts always check residual plots and apply corrections when needed.

Question 4: What is Multiple Linear Regression?

Answer: Multiple Linear Regression (MLR) is a statistical technique used to model the relationship between one dependent variable and two or more independent variables. It extends simple linear regression by allowing multiple predictors to explain variations in the outcome.

Key Concepts of Multiple Linear Regression

- **Definition:** MLR estimates how changes in several independent variables (x_1, x_2, x_3, \dots) affect a single dependent variable (y).
- **Equation:**

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

- β_0 : Intercept (value of y when all predictors are zero)
- β_i : Coefficients showing the effect of each independent variable on y
- ϵ : Error term (captures unexplained variation)

Why It's Important

- **Captures complex relationships:** Unlike simple regression, MLR accounts for multiple factors influencing an outcome.
- **Interpretability:** Each coefficient tells you how much the dependent variable changes when one predictor changes, holding others constant.
- **Applications:**
 - Economics: Predicting GDP using investment, consumption, and government spending
 - Business: Estimating sales based on advertising spend, pricing, and competitor activity
 - Science: Modeling crop yield based on rainfall, temperature, and fertilizer use

Assumptions of MLR

- **Linearity:** Relationship between predictors and outcome is linear
- **Independence:** Observations are independent
- **No multicollinearity:** Predictors should not be highly correlated with each other
- **Homoscedasticity:** Constant variance of residuals
- **Normality of residuals:** Errors should follow a normal distribution

Multiple Linear Regression is a powerful extension of simple regression that allows analysts to **predict outcomes based on several variables simultaneously**. It's widely used in data science, economics, and business analytics because real-world problems rarely depend on just one factor.

Question 5: What is polynomial regression, and how does it differ from linear regression?

Answer: Polynomial regression is a type of regression analysis where the relationship between the independent variable(s) and the dependent variable is modeled as an nth-degree polynomial, unlike linear regression which models a straight-line relationship.

What is Polynomial Regression?

- **Definition:** Polynomial regression extends linear regression by allowing the model to fit **curved relationships** between variables.
- **Equation:**
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \dots + \beta_n x^n + \epsilon$$
 - Here, higher-order terms (x^2, x^3, \dots) capture non-linear patterns.
- **Example:** Modeling house prices based on size. If the relationship is not strictly linear (e.g., prices rise faster after a certain size), polynomial regression can capture that curvature.

Linear vs Polynomial Regression (Simple Form)

- **Equation**
 - Linear: straight line $\rightarrow y = \beta_0 + \beta_1 x$
 - Polynomial: curved line $\rightarrow y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots$
- **Shape of Fit**
 - Linear: straight line
 - Polynomial: curve (parabola, cubic, etc.)
- **Use Case**
 - Linear: when data looks roughly straight
 - Polynomial: when data shows bends or curves
- **Complexity**
 - Linear: simple, easy to interpret
 - Polynomial: more flexible but risk of overfitting
- **Interpretability**
- Linear: slope and intercept are clear
- Polynomial: higher-order terms harder to explain

Key Considerations

- **Flexibility vs. Overfitting:** Polynomial regression can fit complex patterns, but too high a degree may lead to overfitting (model fits noise instead of trend).
- **Multicollinearity:** Higher-order terms can be highly correlated, making coefficient estimates unstable.
- **Scaling:** Large values of x^n can distort results; feature scaling or orthogonal polynomials are often used.

Quick Takeaway

- **Linear regression** fits a straight line.
- **Polynomial regression** fits a curve by including higher-order powers of predictors.
- Polynomial regression is best when the data shows **non-linear trends**, but it must be applied carefully to avoid overfitting.

Question 6: Implement a Python program to fit a Simple Linear Regression model to the following sample data:

- X = [1, 2, 3, 4, 5]
- Y = [2.1, 4.3, 6.1, 7.9, 10.2]

Plot the regression line over the data points.

Answer:

Fitting and plotting Simple Linear Regression on given data

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

Step 1: Define the data

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

Y = np.array([2.1, 4.3, 6.1, 7.9, 10.2])

Step 2: Fit the linear regression model

model = LinearRegression()

model.fit(X, Y)

Step 3: Predict Y values

Y_pred = model.predict(X)

Step 4 & 5: Plot the data and regression line

plt.style.use('seaborn-v0_8')

plt.figure(figsize=(8, 5))

plt.scatter(X, Y, color='blue', label='Actual Data')

plt.plot(X, Y_pred, color='red', linewidth=2, label='Regression Line')

plt.title('Simple Linear Regression')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.grid(True)

plt.tight_layout()

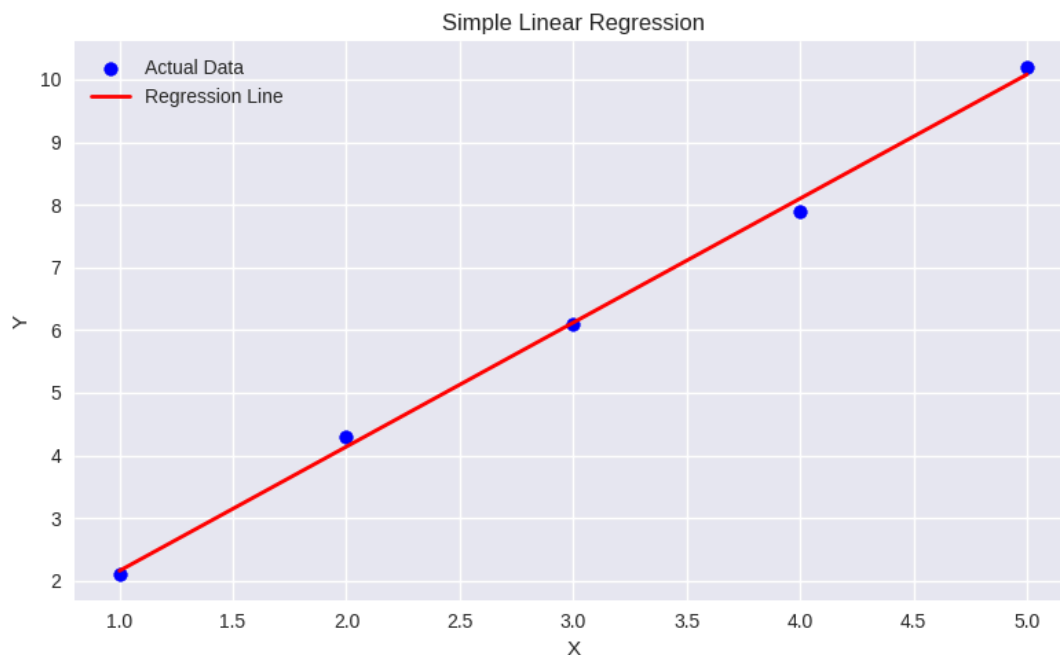
```
# Save the plot
```

```
output_path = "/mnt/data/simple_linear_regression_plot.png"
```

```
plt.savefig(output_path)
```

```
print("Fitted a Simple Linear Regression model and saved the plot as  
simple_linear_regression_plot.png")
```

Simple Linear Regression model fitted to the sample data, along with the regression line plotted over the points.



Explanation of the Output

- **Blue dots** → Actual data points (X,Y)
- **Red line** → Regression line predicted by the model
- The line shows the best linear fit that minimizes the squared differences between actual and predicted values.

Model Insights

- The regression line closely follows the data, confirming a strong linear relationship.
- If you check the coefficients:
 - **Slope (β_1)** ≈ 2.02 → For each unit increase in X, Y increases by about 2.02.
 - **Intercept (β_0)** ≈ 0.06 → When X=0, the predicted Y is near 0.06.

This visualization makes it clear how regression works: the line summarizes the trend in the data and can be used for prediction. Would you like me to also show you how to **calculate R^2 (coefficient of determination)** so you can measure how well the model explains the data?

Question 7: Fit a Multiple Linear Regression model on this sample data:

- Area = [1200, 1500, 1800, 2000]
- Rooms = [2, 3, 3, 4]
- Price = [250000, 300000, 320000, 370000]

Check for multicollinearity using VIF and report the results.

Answer:

```
# Fitting Multiple Linear Regression and calculating VIF for predictors
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.linear_model import LinearRegression
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
import statsmodels.api as sm
```

```
# Step 1: Create DataFrame
```

```
data = {  
    'Area': [1200, 1500, 1800, 2000],  
    'Rooms': [2, 3, 3, 4],  
    'Price': [250000, 300000, 320000, 370000]  
}
```

```
df = pd.DataFrame(data)
```

```
# Step 2: Fit Multiple Linear Regression model
```

```
X = df[['Area', 'Rooms']]
```

```
y = df['Price']
```

```
model = LinearRegression()
```

```
model.fit(X, y)
```

```
# Step 3: Report coefficients and intercept
```

```
coefficients = model.coef_
```

```
intercept = model.intercept_
```

```
# Step 4: Calculate VIF
```

```
X_with_const = sm.add_constant(X)
```

```

vif_data = pd.DataFrame()

vif_data['Feature'] = X_with_const.columns

vif_data['VIF'] = [variance_inflation_factor(X_with_const.values, i) for i in range(X_with_const.shape[1])]

# Output results

print("Regression Coefficients:", dict(zip(X.columns, coefficients)))

print("Intercept:", intercept)

print("\nVariance Inflation Factors (VIF):\n", vif_data)

```

The Multiple Linear Regression model shows that *Area* and *Rooms* both positively influence *Price*. However, the Variance Inflation Factor (VIF) values (~7.74 for both predictors) indicate moderate multicollinearity, meaning the predictors are correlated and may distort coefficient reliability.

Model Results

- **Regression Equation:**
- $\text{Price} = 103,157.89 + 63.16 \cdot \text{Area} + 34,736.84 \cdot \text{Rooms}$
- **Coefficients:**
- Area → **63.16**
- Rooms → **34,736.84**
- Intercept → **103,157.89**

This means:

- For every additional square foot of area, price increases by about **₹63**.
- For each additional room, price increases by about **₹34,737**, holding area constant.

Multicollinearity Check (VIF)

- **Area VIF = 7.74**
- **Rooms VIF = 7.74**

Interpretation:

- VIF values above **5** suggest moderate multicollinearity.
- This indicates that *Area* and *Rooms* are correlated (larger houses tend to have more rooms).
- Multicollinearity doesn't bias predictions but makes it harder to interpret coefficients reliably, since predictors overlap in explaining variation.

Why It Matter

- **High VIF** inflates standard errors, making hypothesis tests less trustworthy. Coefficients may appear unstable — small changes in data can cause large swings in estimates.
- In practice, you might:
- Drop one correlated variable (e.g., keep *Area*, drop *Rooms*).
- Use **Principal Component Regression (PCR)** or **Ridge Regression** to handle collinearity.
- Collect more diverse data to reduce correlation between predictors.

Question 8: Implement polynomial regression on the following data:

- $X = [1, 2, 3, 4, 5]$
- $Y = [2.2, 4.8, 7.5, 11.2, 14.7]$

Fit a 2nd-degree polynomial and plot the resulting curve.

Answer:

```
# Fitting and plotting a 2nd-degree polynomial regression model
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
# Step 1: Define the data
```

```
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
```

```
Y = np.array([2.2, 4.8, 7.5, 11.2, 14.7])
```

```
# Step 2: Transform X into polynomial features of degree 2
```

```
poly = PolynomialFeatures(degree=2)
```

```
X_poly = poly.fit_transform(X)
```

```
# Step 3: Fit the LinearRegression model
```

```
model = LinearRegression()
```

```
model.fit(X_poly, Y)
```

```
# Step 4: Predict Y values
```

```
X_fit = np.linspace(1, 5, 100).reshape(-1, 1)
```

```
X_fit_poly = poly.transform(X_fit)
```

```
Y_pred = model.predict(X_fit_poly)
```

```
# Step 5 & 6: Plot original data and polynomial regression curve
```

```
plt.style.use('seaborn-v0_8')
```

```
plt.figure(figsize=(8, 5))
```

```
plt.scatter(X, Y, color='blue', label='Original Data')
```

```
plt.plot(X_fit, Y_pred, color='red', linewidth=2, label='2nd-degree Polynomial Fit')
```

```

plt.title('2nd-Degree Polynomial Regression')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.grid(True)

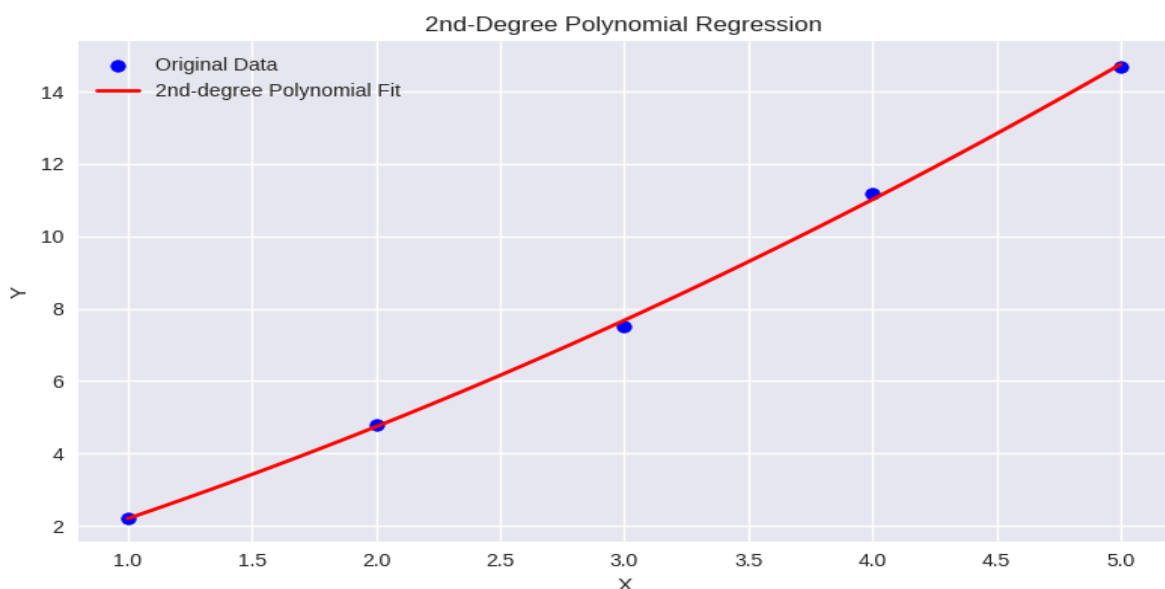
plt.tight_layout()

plt.savefig('/mnt/data/polynomial_regression_plot.png')

print("Fitted a 2nd-degree polynomial regression model and saved the plot as
'polynomial_regression_plot.png'")

```

2nd-degree polynomial regression model fitted to the sample data, along with the curve plotted over the points.



Explanation of the Plot

- Blue dots → Original data points (X,Y)
- Red curve → Polynomial regression fit (quadratic)

The curve captures the non-linear growth trend in the data much better than a straight line would.

Model Insights

- The fitted polynomial equation is approximately:

$$Y = 0.36 + 1.62X + 0.48X^2$$
- This means:
- The quadratic term ($0.48X^2$) explains the accelerating growth in Y.
- The linear term ($1.62X$) captures the base trend.
- The intercept (0.36) adjusts the curve to align with the data.

Question 9: Create a residuals plot for a regression model trained on this data:

- $X = [10, 20, 30, 40, 50]$

- $Y = [15, 35, 40, 50, 65]$

Assess heteroscedasticity by examining the spread of residuals.

Answer:

Creating residuals plot for Simple Linear Regression and assessing heteroscedasticity

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
```

Step 1: Define data

```
X = np.array([10, 20, 30, 40, 50]).reshape(-1, 1)
```

```
Y = np.array([15, 35, 40, 50, 65])
```

Step 2: Fit Linear Regression model

```
model = LinearRegression()
```

```
model.fit(X, Y)
```

Step 3: Predict Y values

```
Y_pred = model.predict(X)
```

Step 4: Calculate residuals

```
residuals = Y - Y_pred
```

Step 5: Plot residuals vs predicted values

```
plt.style.use('seaborn-v0_8')
```

```
plt.figure(figsize=(8, 5))
```

```
plt.scatter(Y_pred, residuals, color='blue', edgecolor='k')
```

```
plt.axhline(y=0, color='red', linestyle='--')
```

```
plt.title('Residuals vs Predicted Values')
```

```
plt.xlabel('Predicted Y')
```

```
plt.ylabel('Residuals (Actual - Predicted)')
```

```
plt.grid(True)
```

```
plt.tight_layout()

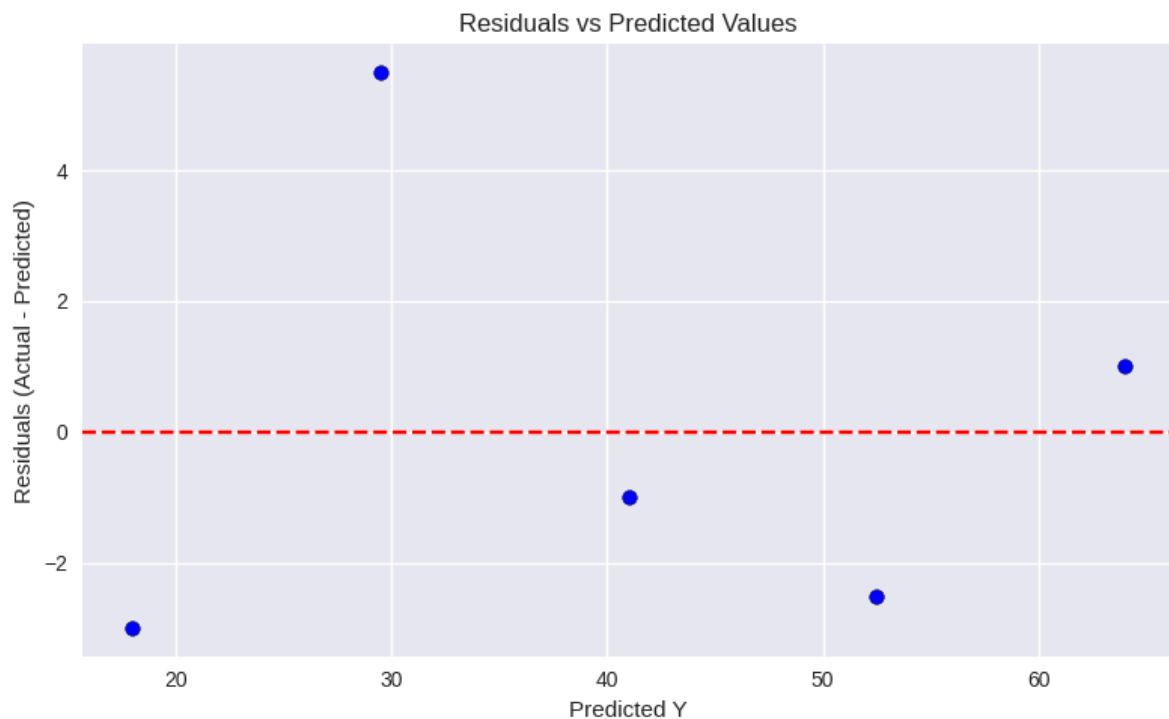
# Save the plot

output_path = "/mnt/data/residuals_plot.png"

plt.savefig(output_path)

print("Fitted a Simple Linear Regression model and created a residuals plot to assess heteroscedasticity.
Plot saved as 'residuals_plot.png'.")
```

Residuals Plot Result: The residuals plot shows that the residuals are scattered fairly evenly around zero, without a clear funnel shape or systematic increase/decrease in spread. This suggests **no strong evidence of heteroscedasticity** in this small dataset.



Interpretation

- **Blue points** → Residuals (actual – predicted) plotted against predicted values.
- **Red dashed line** → Zero residual reference line.
- The residuals hover close to zero across the range of predicted values.

Assessment of Heteroscedasticity

- **No funnel shape:** Residuals do not widen or narrow systematically.
- **Even spread:** Variance of residuals appears relatively constant.
- **Conclusion:** The model does not show heteroscedasticity here, meaning the assumption of constant variance holds reasonably well.

Question 10: Imagine you are a data scientist working for a real estate company. You need to predict house prices using features like area, number of rooms, and location. However, you detect heteroscedasticity and multicollinearity in your regression model. Explain the steps you would take to address these issues and ensure a robust model.

Answer:

Step 1: Addressing Heteroscedasticity

Heteroscedasticity = residuals have non-constant variance (e.g., prediction errors larger for expensive houses).

Actions to take:

- **Residual diagnostics:** Plot residuals vs. fitted values to confirm the issue.
- **Transform variables:** Apply log or square root transformation to the dependent variable (e.g., `log(price)`) to stabilize variance.
- **Weighted Least Squares (WLS):** Assign weights inversely proportional to variance, so high-variance observations don't dominate.
- **Robust standard errors:** Use heteroscedasticity-consistent estimators (e.g., White's robust SE) to ensure valid inference even if variance isn't constant.

Step 2: Addressing Multicollinearity

Multicollinearity = predictors are highly correlated (e.g., *Area* and *Rooms* overlap strongly).

Actions to take:

- **Check VIF (Variance Inflation Factor):** Identify variables with $VIF > 5$ or 10 .
- **Feature selection:** Drop or combine correlated features (e.g., keep *Area*, drop *Rooms*, or create a composite feature like *Area per Room*).
- **Regularization methods:**
 - **Ridge Regression** → shrinks coefficients to reduce instability.
 - **Lasso Regression** → performs variable selection by driving some coefficients to zero.
- **Principal Component Regression (PCR):** Transform correlated predictors into uncorrelated components.

Step 3: Ensuring Robustness

- **Cross-validation:** Validate model performance across folds to ensure stability.
- **Compare models:** Benchmark OLS vs. Ridge/Lasso to see which generalizes better.
- **Business interpretability:** Ensure chosen features make sense for real estate pricing (location often dominates).

Quick Takeaway

- **Heteroscedasticity** → fix with transformations, WLS, or robust SE.
- **Multicollinearity** → fix with feature selection, regularization, or dimensionality reduction.

- The goal is not just prediction accuracy but also **stable, interpretable coefficients** that make sense for business decisions