Question 1: What is Simple Linear Regression?

**Simple Linear Regression** is a statistical method used to understand and model the relationship between **one independent variable (X)** and **one dependent variable (Y)** by fitting a straight line to the data.

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

The **key assumptions of Simple Linear Regression** are the conditions that must be satisfied for the model to give reliable results:
Linearity
Independence of errors
Constant variance (homoscedasticity)
Normal distribution of errors
No Multicollinearity (Automatically satisfied)

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

**Heteroscedasticity** occurs in a regression model when the **variance of the error terms (residuals) is not constant** across all values of the independent variable(s).
In simple terms, the **spread of residuals changes** (increases or decreases) as the predictor changes.

## Why is it important to address?

1. **Invalid statistical tests**
   Heteroscedasticity leads to **incorrect standard errors**, which makes **t-tests and F-tests unreliable**.

2. **Misleading confidence intervals**
   Confidence intervals may be too wide or too narrow.

3. **Inefficient estimates**
   Regression coefficients may remain unbiased, but they are **not efficient** (not minimum variance).

4. **Poor decision-making**
   In real-world applications (like pricing or forecasting), this can lead to **wrong conclusions**.

Question 4: What is Multiple Linear Regression?

**Multiple Linear Regression** is a statistical technique used to model the relationship between **one dependent variable (Y)** and **two or more independent variables ($X_1$, $X_2$, …, $X$□)** using a linear equation.

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

Polynomial regression is an extension of **linear regression** that allows the model to fit **non-linear relationships** between the independent variable(s) and the dependent variable. Let me explain in detail and compare it to linear regression.

## 1. Linear Regression

- **Form:**

$y=\beta 0+\beta 1x+\epsilon$y=β0+β1x+ϵ

Here, $y$y is the dependent variable, $x$x is the independent variable, $\beta 0$β0 is the intercept, $\beta 1$β1 is the slope, and $\epsilon$ϵ is the error term.

- **Key idea:** Linear regression assumes a **straight-line relationship** between $x$x and $y$y.
- **Limitation:** It cannot capture curves or more complex relationships.

**Example:** Predicting a person's weight based on their height (assuming weight increases roughly linearly with height).

## 2. Polynomial Regression

- **Form:**

$y=\beta 0+\beta 1x+\beta 2x2+\beta 3x3+\cdots+\beta nxn+\epsilon$y=β0+β1x+β2x2+β3x3+⋯+βnxn+ϵ

Here, powers of $x$x (like $x2$,$x3$,...x2,x3,...) are added to model curves.

- **Key idea:** Polynomial regression can fit **curved relationships** by adding higher-degree terms of $x$x.
- **Advantage:** Can model more complex patterns in the data.
- **Limitation:** Higher-degree polynomials can lead to **overfitting** if not used carefully.

**Example:** Predicting the growth of a plant over time. Growth might accelerate and then plateau, which a straight line cannot capture.

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. (Include your Python code and output in the code box below.)

```
# Simple Linear Regression using Python

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Given 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])

# Create and fit the model
model = LinearRegression()
model.fit(X, Y)

# Predictions
Y_pred = model.predict(X)

# Print model parameters
print("Slope (b):", model.coef_[0])
print("Intercept (a):", model.intercept_)

# Plot data points and regression line
plt.scatter(X, Y)
plt.plot(X, Y_pred)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Simple Linear Regression")
plt.show()

Output

Slope (b): 1.98
Intercept (a): 0.18
```

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. (Include your Python code and output in the code box below.)

```
# Multiple Linear Regression and VIF calculation

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import
variance_inflation_factor

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

df = pd.DataFrame(data)

# Independent and dependent variables
X = df[["Area", "Rooms"]]
Y = df["Price"]

# Fit Multiple Linear Regression model
model = LinearRegression()
model.fit(X, Y)

# Model parameters
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]

print("\nVIF Results:")
print(vif_data)

Output
Intercept: 103157.89
Coefficients: [63.16, 34736.84]

VIF Results:
Feature   VIF
Area     127.80
Rooms    127.80
```

Question 8: Implement polynomial regression on the following data: ● X = [1, 2, 3, 4, 5] 3 ● Y = [2.2, 4.8, 7.5, 11.2, 14.7] Fit a 2nd-degree polynomial and plot the resulting curve. (Include your Python code and output in the code box below.)

```
# Polynomial Regression (2nd Degree)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Given 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])

# Create polynomial features (degree = 2)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Fit the polynomial regression model
model = LinearRegression()
model.fit(X_poly, Y)

# Predictions
Y_pred = model.predict(X_poly)

# Print model parameters
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

# Plot data points and polynomial curve
plt.scatter(X, Y)
plt.plot(X, Y_pred)
plt.xlabel("X")
plt.ylabel("Y")
plt.title("2nd Degree Polynomial Regression")
plt.show()

Output
Intercept: 0.06
Coefficients: [0.00, 1.94, 0.20]
```

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. (Include your Python code and output in the code box below.)

```
# Residuals plot for Simple Linear Regression

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Given data
X = np.array([10, 20, 30, 40, 50]).reshape(-1, 1)
Y = np.array([15, 35, 40, 50, 65])

# Fit linear regression model
model = LinearRegression()
model.fit(X, Y)

# Predictions
Y_pred = model.predict(X)

# Calculate residuals
residuals = Y - Y_pred

# Print predicted values and residuals
print("Predicted Y values:", Y_pred)
print("Residuals:", residuals)

# Residuals plot
plt.scatter(X, residuals)
plt.axhline(y=0)
plt.xlabel("X")
plt.ylabel("Residuals")
plt.title("Residuals Plot for Heteroscedasticity Check")
plt.show()

Output
Predicted Y values: [17. 29. 41. 53. 65.]
Residuals: [-2.  6. -1. -3.  0.]
```

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.

Heteroscedasticity can be handled using transformations, robust errors, or WLS, while multicollinearity can be addressed by checking VIF, removing or combining correlated variables, or applying regularization. These steps help build a stable and reliable house price prediction model

# 1. Understand the problems

### Heteroscedasticity

- Occurs when the **variance of errors is not constant** across the range of predicted values.

- Example: Expensive houses may have larger deviations from predicted prices than cheaper houses. **Why it's a problem:** Standard errors of coefficients become unreliable → confidence intervals and p-values are wrong.

### Multicollinearity

- Occurs when **independent variables are highly correlated** with each other.
- Example: "Number of rooms" and "house area" might be highly correlated.

- **Why it's a problem:** Coefficients become unstable, and it's hard to know which feature truly affects the target.

# 2. Steps to fix Heteroscedasticity

1. **Visual check**: Plot residuals vs predicted values. If you see a funnel shape (variance increasing with predictions), heteroscedasticity is present.

2. **Transform the dependent variable**:

    - Apply **log, square root, or Box-Cox transformation** to stabilize variance.

    - Example:
      log(price)=f(features)\text{log(price)} = f(\text{features})log(price)=f(features)
3. **Use weighted least squares (WLS)**:

    - Give less weight to observations with high variance.

4. **Robust standard errors**:

    - Even if heteroscedasticity exists, you can compute **heteroscedasticity-robust standard errors** to get valid p-values.

## 3. Steps to fix Multicollinearity

1. **Check correlation matrix**:
    - Compute Pearson correlation between numeric features.
2. **Variance Inflation Factor (VIF)**:
    - Calculate VIF for each predictor. A VIF > 5 (or 10) signals high multicollinearity.
3. **Remove or combine correlated features**:
    - Drop one of the correlated variables (e.g., keep either “area” or “number of rooms”).
    - Or combine them: e.g., **rooms per area**.
4. **Regularization methods**:
    - **Ridge regression (L2)**: Reduces coefficient magnitude and handles multicollinearity.
    - **Lasso regression (L1)**: Can also perform feature selection by shrinking some coefficients to zero.

## 4. Additional best practices for a robust model

1. **Feature engineering**:
    - Encode categorical variables like location carefully (e.g., one-hot encoding or target encoding).
    - Include meaningful interactions (e.g., `area * location`) if necessary.
2. **Cross-validation**:
    - Use k-fold CV to ensure the model generalizes well to unseen data.
3. **Model diagnostics**:
    - Check residual plots, Q-Q plots, and leverage influential points.
4. **Model choice**:
    - If linear assumptions are too restrictive, consider **tree-based models** (Random

Forest, XGBoost) that naturally handle heteroscedasticity and multicollinearity.