Assignment Code: DA-AG-010

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 continuous variables**:

- One **independent variable** (also called the **predictor** or **explanatory variable**, usually denoted as **X**)
- One **dependent variable** (also called the **response** or **output variable**, usually denoted as **Y**)

Mathematical Equation

The relationship is modeled using a straight line:

Y=a+bX+εY = a + bX + \varepsilonY=a+bX+ε

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

Answer:

The **key assumptions of Simple Linear Regression** are crucial for the model to provide reliable and interpretable results. Here are the main ones:

1. Linearity
2. Independence of Errors
3. Homoscedasticity (Constant Variance)
4. Normality of Errors
5. **No Perfect Multicollinearity** (only relevant in **multiple** linear regression)

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

Answer:

Heteroscedasticity refers to the situation in regression models where the **variance of the residuals (errors)** is **not constant** across all levels of the independent variable(s). In simple terms, the **spread of the errors increases or decreases** as the value of the predictor changes.

Why Is It Important to Address?

1. **Violates Regression Assumptions**
   - Linear regression assumes **constant variance of errors** (homoscedasticity)
   - Heteroscedasticity breaks this assumption
2. **Incorrect Standard Errors**
   - Standard errors of the regression coefficients become **biased**
   - This leads to **invalid confidence intervals and hypothesis tests**
3. **Unreliable Predictions**
   - The model may give **inaccurate forecasts** for some ranges of the data
4. **Inefficient Estimates**
   - The regression coefficients remain **unbiased**, but are **not efficient**
   - There's a **better way to estimate them** if heteroscedasticity is present

---

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 (Y)** and **two or more independent variables (X_1, X_2, ..., X_n)**.

Mathematical Form

$$Y = a + b_1 X_1 + b_2 X_2 + \cdots + b_n X_n + \varepsilon$$

Where:

- **Y** = Dependent variable (what you're predicting)
- **X_1, X_2, ..., X_n** = Independent variables (predictors/features)
- **a** = Intercept
- **b_1, b_2, ..., b_n** = Coefficients (effect of each X on Y)
- **ε** = Error term (unexplained variability)

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

Answer:

Polynomial Regression is a type of regression that models the relationship between the independent variable XXX and the dependent variable YYY as an nth-degree polynomial.

How It Differs from Linear Regression

Feature	Linear Regression	Polynomial Regression
Relationship type	Linear (straight line)	Non-linear (curve)
Equation	Y=a+bX+εY = a + bX + \varepsilon	Y=a+b1X+b2X2+⋯+bnXn+εY = a + b_1X + b_2X^2 + \dots + b_nX^n + \varepsilon
Shape of model	Line	Curve (e.g., parabola, cubic, etc.)
Still linear in params?	Yes	Yes (despite non-linear in XX)
Used when	Data has a straight-line trend	Data shows curved or complex patterns

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.*)
Answer:

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

# Sample data
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # Reshape to 2D for sklearn
Y = np.array([2.1, 4.3, 6.1, 7.9, 10.2])

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

# Predict values
Y_pred = model.predict(X)

# Print coefficients
print(f"Intercept (a): {model.intercept_:.2f}")
print(f"Slope (b): {model.coef_[0]:.2f}")

# Plotting
plt.scatter(X, Y, color='blue', label='Data Points')
plt.plot(X, Y_pred, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Simple Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

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.*)

Answer:

Here's a complete Python program to:

1. Fit a **Multiple Linear Regression** model
2. Compute the **Variance Inflation Factor (VIF)** to check for multicollinearity

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

# Sample 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']

# Add constant term for statsmodels
X_sm = sm.add_constant(X)

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

# Display regression results
print(model.summary())

# Calculate VIF for each feature
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("\nVariance Inflation Factor (VIF):")
print(vif_data)
```

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.

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

Answer:

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

# Input 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])

# Transform to 2nd-degree polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

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

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

# Plot the original data and polynomial regression curve
plt.scatter(X, Y, color='blue', label='Original Data')
plt.plot(X, Y_pred, color='red', label='2nd-Degree Polynomial Fit')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polynomial Regression (Degree 2)')
plt.legend()
```

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

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.*)
Answer:

Here's a complete Python program to:

1. Fit a **Simple Linear Regression** model
2. Compute and plot the **residuals**
3. Visually **assess heteroscedasticity** based on residual spread

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

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

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

# Predict Y values
Y_pred = model.predict(X)

# Calculate residuals
residuals = Y - Y_pred

# Print residuals
print("Residuals:", residuals)

# Plot residuals
plt.scatter(X, residuals, color='purple', marker='o', label='Residuals')
```

```
plt.axhline(y=0, color='black', linestyle='--', linewidth=1)
plt.xlabel('X (Independent Variable)')
plt.ylabel('Residuals')
plt.title('Residual Plot for Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

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:

Scenario: House Price Prediction with Issues in the Regression Model

As a **data scientist at a real estate company**, you're building a **regression model** to predict **house prices** based on features like:

- **Area (in sqft)**
- **Number of Rooms**
- **Location**

You detect two common issues:

1. **Heteroscedasticity** – variance of errors is not constant
2. **Multicollinearity** – predictor variables are highly correlated

---

Steps to Address These Issues & Build a Robust Model

---

1. Detect and Fix Heteroscedasticity

Detection

- **Residual plots**: Plot residuals vs. predicted values. A "funnel" shape indicates heteroscedasticity.
- **Statistical tests**: Use the **Breusch-Pagan** or **White test** for confirmation.

Solutions

- **Transform the target variable (Y)**:

Apply a **logarithmic or Box-Cox transformation** to stabilize variance:

```
y_transformed = np.log(y)
```

- **Use robust standard errors**:

Use **HC3-robust covariance matrix** with statsmodels:

```
model = sm.OLS(y, X).fit(cov_type='HC3')
```

- **Switch to Weighted Least Squares (WLS)**:
    - Gives less weight to points with high variance in errors

---

2. Detect and Fix Multicollinearity

Detection

- **Variance Inflation Factor (VIF)**:

Check VIF for each predictor:

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

- VIF > 5 (or >10) indicates multicollinearity

Solutions

- **Remove or combine highly correlated features**:
  - For example, if Area and Rooms are highly correlated, consider using only one or creating a ratio feature like `AreaPerRoom`
- **Use Regularization (Ridge or Lasso)**:

Penalizes large coefficients and reduces overfitting/multicollinearity:

```python
from sklearn.linear_model import Ridge
model = Ridge(alpha=1.0)
```

- **Use Principal Component Analysis (PCA)** if many numeric features are correlated
  - Reduces dimensionality while preserving variance

---

3. Final Model Refinement and Validation

- **Scale features** for regularization models
- **Split data into train/test sets** or use **cross-validation**
- **Evaluate model performance** with metrics like:
  - **RMSE**, **MAE**, **R^2**
- **Check residual plots again** after transformations

- **Interpret coefficients** carefully, especially after regularization