

Regression & Its Evaluation | Assignment

Question 1: What is Simple Linear Regression?

Simple Linear Regression is a method to **predict the value of one variable (the dependent variable, or response)** based on the value of **one other variable (the independent variable, or predictor)** by fitting a straight line through the data points.

The Equation:

The relationship is modeled by the **equation of a straight line**:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Where:

- Y = Dependent variable (what you're trying to predict)
- X = Independent variable (the input/predictor)
- β_0 = Intercept (the value of Y when $X = 0$)
- β_1 = Slope (the change in Y for a one-unit change in X)
- ϵ = Error term (captures the difference between the actual and predicted values)

Key Idea

The goal is to **find the line of best fit**, which minimizes the total squared differences between the actual values and the predicted values. This is called the **least squares method**.

Example

-

Predicting someone's weight (Y) based on their height (X).

-

Predicting house price (Y) based on its size (X).

Assumptions

To use simple linear regression properly, you assume:

- 1.

The relationship between X and Y is linear.

- 2.

The residuals (errors) are normally distributed.

- 3.

Homoscedasticity — constant variance of residuals.

- 4.

Independence of observations.

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

1. Linearity

-

What it means: The relationship between the independent variable X and the dependent variable Y is **linear** — that is, the change in Y is proportional to the change in X .

-

How to check: Look at a scatter plot of X vs. Y . The points should form roughly a straight-line pattern.

2. Independence of Errors

-

What it means: The residuals (errors) are **independent** — there should be no correlation between consecutive residuals.

-

Why it matters: If errors are correlated (common in time series data), the standard errors may be underestimated.

-

How to check: Use the **Durbin-Watson test** or plot residuals vs. time/order.

3. Homoscedasticity (Constant Variance of Errors)

-

What it means: The variance of residuals should be **constant** across all levels of X .

-

Why it matters: If the spread of residuals grows or shrinks with X , predictions become less reliable.

-

How to check: Plot residuals vs. fitted values. The spread should look random and even — not a funnel shape.

4. Normality of Errors

-

What it means: The residuals (errors) should be **normally distributed** around the regression line.

-

Why it matters: This mainly affects confidence intervals and hypothesis tests.

-

How to check: Use a histogram or a Q-Q plot of residuals.

5. No Significant Outliers or High Leverage Points

-

What it means: Outliers can overly influence the slope and intercept.

-

How to check: Look at scatter plots, Cook's distance, or leverage plots.

Summary of Assumptions:

Assumption Checks for

Linearity

Is the relationship straight-line?

Independence

Are residuals independent?

Homoscedasticity

Is variance constant?

Normality of Errors

Are residuals normal?

No major outliers

Are there extreme values distorting

the line? Question 3: What is heteroscedasticity, and why

is it important to address in regression models?

Heteroscedasticity means that the **variance of the residuals (errors)** is **not constant** across all levels of the independent variable(s).

In simple words:

-

With **homoscedasticity**, the spread of the residuals is even — the scatter around the regression line stays roughly the same for all predicted values.

-

With **heteroscedasticity**, the spread changes — often getting wider or narrower as the value of X changes.

Example

Imagine you're predicting people's income based on their years of education.

-

For people with low years of education, the variance in income is small (most earn similar amounts).

-

For people with higher education, incomes can vary widely (some earn a lot, some not so much). This creates a **fan shape** in a residual plot — classic heteroscedasticity.

How to Detect Heteroscedasticity

-

Residuals vs. Fitted plot: If you see a funnel or cone shape (residuals spread out more as fitted values increase), it indicates heteroscedasticity.

-

Formal tests:

—

Breusch–Pagan test

—

White's test

Why is Heteroscedasticity a Problem?

Heteroscedasticity **doesn't bias the regression coefficients**, but it **violates an important OLS assumption**, leading to:

1. **Incorrect standard errors** → t-tests & p-values become unreliable.
2. **Confidence intervals & hypothesis tests may be invalid.**
3. Model predictions may be less efficient (less precise).

How to Fix It

Possible solutions:

-

Transform the dependent variable (e.g., $\log(Y)$, square root).

-

Use **Weighted Least Squares (WLS)** instead of OLS.

Use **robust standard errors** to adjust for the unequal variance.

- Sometimes add missing variables —

heteroscedasticity can appear if a relevant variable is omitted.

Question 4: What is Multiple Linear Regression?

Multiple Linear Regression (MLR) is an extension of Simple Linear Regression — **but instead of using *one* independent variable (predictor), it uses *two or more*.**

It models the relationship between:

-

One dependent variable (Y)

-

And **two or more independent variables** ($X_1, X_2, X_3, \dots, X_n$)

The Equation

The general form is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \dots + \beta_n X_n + \epsilon$$

Where:

-

Y = Dependent variable (what you're predicting)

-

X_1, X_2, \dots, X_n = Independent variables (predictors)

-

β_0 = Intercept

-

$\beta_1, \beta_2, \dots, \beta_n$ = Regression coefficients (how much Y changes for a unit change in X , *holding other X 's constant*)

-

ϵ = Error term (residual)

What Does It Do?

Multiple Linear Regression lets you:

-

Predict Y using *several factors at once*.

-

Understand the effect of each predictor while **controlling for others**.

-

Estimate how much each X uniquely contributes to Y .

Example

Predicting house prices:

$$\text{House Price} = \beta_0 + \beta_1 (\text{Size}) + \beta_2 (\text{Number of Rooms}) + \beta_3 (\text{Distance to City}) + \dots + \epsilon$$

-

Size \rightarrow bigger houses cost more

-

More rooms \rightarrow higher price

-

Closer to city \rightarrow higher price

Each coefficient shows the **effect of one factor while holding the others constant**.

Key Assumptions

Same as Simple Linear Regression **plus**:

- 1.

Linearity: Relationship between each X and Y is linear.

- 2.

Independence of errors: Residuals are independent.

- 3.

Homoscedasticity: Constant variance of residuals.

4.

Normality of errors: Residuals are normally distributed.

5.

No multicollinearity: Independent variables shouldn't be highly correlated with each other. (High multicollinearity makes it hard to estimate individual effects.)

Why Use Multiple Linear Regression?

-

Real-world problems are rarely explained by one factor.

-

MLR allows for **better predictions**.

-

It helps understand the **relative importance of each predictor**.

- But: It's more complex → more room for assumption violations.

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

Polynomial Regression is an **extension of Linear Regression** that models the **relationship between the independent variable(s) and the dependent variable as an nth-degree polynomial**.

In other words, instead of fitting a straight line, you fit a **curved line** to capture **non-linear relationships**.

The General Equation

A **polynomial regression** with one predictor looks like:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \dots + \beta_n X^n + \epsilon$$

Where:

-

Y = Dependent variable

-

X, X^2, X^3, \dots, X^n = Predictor variable raised to different powers

-

$\beta_0, \beta_1, \beta_2, \dots, \beta_n$ = Coefficients

ϵ = Error term

How is it Different from Linear Regression?

Aspect

Simple/Multiple Linear Regression

Polynomial Regression

Relation

ship

Models **linear** relationships (straight line or hyperplane)

Models **non-linear** relationships (curve)

Predicto

rs

Uses original X variables only

Uses original X plus **powers of X**

Equatio

n form

Straight-line equation

Polynomial equation

Exempl

e

$$Y = \beta_0 + \beta_1 X$$

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2$$

0

1

2

Important Point

Even though the relationship is **non-linear in X**, **polynomial regression is still linear in the coefficients!** That's why it can be solved using ordinary least squares (OLS) just like linear regression.

When to Use Polynomial Regression

When your data shows a **curved trend** that a straight line can't capture. Example:

-

Growth curves (population growth)

-

Price vs. advertising spend

-

Physics or engineering problems where relationships bend

Watch Out For:

-

Overfitting: Higher-degree polynomials can fit the training data perfectly but fail to generalize.

-


Extrapolation: Predictions outside the observed range can become wildly inaccurate.

-


Collinearity: Powers of X are often correlated → can inflate variance.

Visual Example

-

Linear: Fits a straight line: 

-

Polynomial: Fits a curve: 

(Imagine a U-shaped scatter plot: a line cuts through awkwardly, but a quadratic curve fits nicely.)

Key Difference:

Linear Regression → Best for straight-line trends. **Polynomial Regression** → Best when the

trend is **curved**, but you still want to use linear regression methods by transforming X. 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.)

```
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)
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)
# Plot
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 Example')
plt.legend()
plt.grid(True)
plt.show()
# Coefficients
model.intercept_, model.coef_[0](np.float64(0.179999999999999794), np.float64(1.98000000000000004))
```

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.)

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

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.)

feature

0

const

1

Area

VIF

34.210526

7.736842

Sample data

Area = np.array([1200, 1500, 1800, 2000])

Rooms = np.array([2, 3, 3, 4])

Price = np.array([250000, 300000, 320000, 370000])

Prepare the data

X = pd.DataFrame({

'Area': Area,

'Rooms': Rooms

})

Y = Price

Fit Multiple Linear Regression

model = LinearRegression()

model.fit(X, Y)

Add constant for statsmodels VIF

X_with_const = sm.add_constant(X)

Calculate VIF

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])]

Results

model.intercept_, model.coef_, vif_data

(np.float64(103157.89473684214),

array([

63.15789474, 34736.84210526]),

2

Rooms

7.736842)import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression

Sample 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)
# Predictions for smooth curve
X_range = np.linspace(1, 5, 100).reshape(-1, 1)
X_range_poly = poly.transform(X_range)
Y_pred = model.predict(X_range_poly)
# Plot
plt.scatter(X, Y, color='blue', label='Data Points')
plt.plot(X_range, 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()
# Coefficients
model.intercept_, model.coef_(np.float64(0.060000000000000938), array([0. , 1.94, 0.2 ]))

```

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.)

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
# Sample data
X = np.array([10, 20, 30, 40, 50]).reshape(-1, 1)
Y = np.array([15, 35, 40, 50, 65]) # Fit Linear Regression
model = LinearRegression()
model.fit(X, Y)
# Predictions and residuals
Y_pred = model.predict(X)
residuals = Y - Y_pred
# Plot residuals
plt.scatter(Y_pred, residuals, color='blue')
plt.axhline(y=0, color='red', linestyle='--')

```

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

```
plt.ylabel('Residuals')
```

```
plt.title('Residuals Plot')
```

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

```
plt.show()
```

```
# Residuals
```

```
residuals
```

```
array([-3. , 5.5, -1. , -2.5, 1. ])
```

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.

-

Goal: Predict house prices using features like **area**, **number of rooms**, and **location**.

-

Challenges: Heteroscedasticity: The spread of residuals is not constant — larger houses

may have wider price variability. **Multicollinearity:** Predictors like **area** and **number of**

rooms are likely correlated.

1. How to Address Heteroscedasticity

Why it's a problem:

-

Heteroscedasticity doesn't bias coefficients, but it makes standard errors unreliable →

bad p-values → misleading significance tests.

Common solutions:

1. Transform the dependent variable (Y):

-

Take the log of house prices. This often stabilizes variance.

-

New model:

$\log(\text{Price}) = \beta_0 + \beta_1 (\text{Area}) + \beta_2 (\text{Rooms}) + \beta_3 (\text{Location}) + \epsilon$

2. Use robust standard errors:

-

Instead of changing the model, adjust the calculation of standard errors to be robust to heteroscedasticity.

-

E.g., in Python: statsmodels → HC0, HC3 robust covariance estimators.

3. Weighted Least Squares (WLS):

-

Give less weight to observations with higher variance.

-

More complex, but effective for severe heteroscedasticity.

4. Check for omitted variables:•

Sometimes heteroscedasticity appears because important predictors are missing (e.g., neighborhood income level).

2. How to Address Multicollinearity

Why it's a problem:

-

High correlation among predictors inflates standard errors → unstable estimates → hard to interpret individual effects.

Common solutions:

1. Check VIF:

-

Drop or combine predictors with high VIF (typically > 10 is a red flag).

-

Example: If **area** and **rooms** are highly correlated, maybe replace them with a composite feature like **area per room**.

2. Use dimensionality reduction:

-

Principal Component Analysis (PCA): Combines correlated variables into uncorrelated components.

-

This sacrifices interpretability but can stabilize the model.

3. Regularization:

-

Use **Ridge Regression (L2 penalty)** to shrink coefficients of correlated predictors.

-

Or **Lasso Regression (L1 penalty)** to shrink some coefficients to zero → automatic feature selection.

-

Example: `sklearn.linear_model.Ridge` or `Lasso`.

4. Domain knowledge:

-

Maybe you realize **area** is more meaningful than **rooms**, so you drop **rooms** altogether.

3. Model Diagnostics and Validation

After fixing issues:

-

Re-check residual plots to confirm variance is more stable.

-

Re-calculate VIF to verify multicollinearity is under control.

-

Use cross-validation to check generalization.

-

Compare model performance (R^2 , RMSE) with and without fixes.

4. Final Deliverable

Provide a **clear, interpretable model**:

-

Communicate the impact of each predictor.

-

Explain any transformations used (e.g., log price).

-

Justify any features dropped or combined.