Homoscedasticity & Heteroscedasticity

✅ Thumb Rule for Homo- and Heteroscedasticity:

- **Homoscedasticity**: Constant variance of errors/residuals across all levels of the independent variable(s).
  Thumb Rule: Residuals should appear randomly scattered around zero in a residual plot.
  → **Purpose**: A key assumption in **linear regression**. It ensures:
  - Unbiased estimation of coefficients.
  - Valid confidence intervals and hypothesis tests.
- **Heteroscedasticity**: Non-constant variance (errors spread increases or decreases with the predictor).
  Thumb Rule: If residuals form a **fan shape** or **pattern**, heteroscedasticity may be present.
  → **Purpose**: Indicates model issues (e.g., missing variables, incorrect functional form).
  Used in:
  - **Econometrics** (common in income, price data)
  - **Diagnostics** (to improve models using transformation or robust regression)

📊 Which Is Best?

Type	Suitability	When It's "Best"
Homoscedasticity	Ideal for OLS (Ordinary Least Squares) regression	✅ Best when using standard linear regression
Heteroscedasticity	Not ideal but can be modeled using alternatives (e.g., Weighted Least Squares, transformation, robust SE)	Acceptable if handled properly, especially in real-world data

🧠 In general, **homoscedasticity is better**, especially for valid regression assumptions.
But if heteroscedasticity is present and common in your data (e.g., economics), handle it properly.

💡 Summary:

- **Thumb Rule**: Residuals randomly scattered = Homoscedastic; Fan shape/pattern = Heteroscedastic
- **Purpose**: Ensures reliability of regression results.
- **Best**: Homoscedasticity is best **if assumptions of linear regression are to be trusted**.

Python Code Example:

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

```
np.random.seed(42)
X = np.linspace(1, 100, 100)
Y_homo = 2 * X + np.random.normal(0, 10, 100)
Y_hetero = 2 * X + np.random.normal(0, X * 0.5, 100)
```

- This creates an array X of **100 numbers** evenly spaced from 1 to 100.
- X is the predictor variable.

This creates **two Y variables**:

1. **Y_homo** → Homoscedastic data
    - Follows the line Y = 2X + noise
    - The noise is **random with constant spread** (mean = 0, std = 10), same for all X.
2. **Y_hetero** → Heteroscedastic data
    - Also Y = 2X + noise
    - But here, the **noise increases as X increases**: std = X * 0.5. So bigger X → bigger errors.

```
X_const = sm.add_constant(X)
model_homo = sm.OLS(Y_homo, X_const).fit()
model_hetero = sm.OLS(Y_hetero, X_const).fit()
```

- Adds a **constant column** (intercept) to X for regression.
- OLS() creates the regression model.
- .fit() fits the model to the data.
- Now we have:
    - model_homo: model for constant-error data
    - model_hetero: model for increasing-error data

```
resid_homo = model_homo.resid
resid_hetero = model_hetero.resid
```

- Residuals = Actual Y – Predicted Y
- Shows the difference between what the model predicted and what really happened.

```
plt.figure(figsize=(12, 5))
```

- Starts a figure with size 12x5 (inches)

```
plt.subplot(1, 2, 1)
plt.scatter(X, resid_homo, color='green')
plt.axhline(0, color='black', linestyle='--')
plt.title("Homoscedasticity (Residuals vs X)")
plt.xlabel("X")
plt.ylabel("Residuals")
```

- Makes a scatter plot: X on x-axis, residuals on y-axis.
- axhline(0) draws a horizontal line at 0.
- If residuals are randomly spread, it confirms **homoscedasticity**.

```
plt.subplot(1, 2, 2)
plt.scatter(X, resid_hetero, color='red')
plt.axhline(0, color='black', linestyle='--')
plt.title("Heteroscedasticity (Residuals vs X)")
plt.xlabel("X")
```

- Same as above but for Y_hetero.
- Residuals **spread out more** as X increases → shows **heteroscedasticity**.
