Excellent! 🔍 Verifying assumptions is **super important** to ensure your **linear regression model is valid** and **interpretable**.

Let’s walk through the **5 main assumptions** of linear regression — and how to **verify each one** with simple Python code.

**✅ Assumptions of Linear Regression & How to Check Them**

**1. Linearity**

➡️ The relationship between X (features) and y (target) is **linear**.

**🔍 How to check:**

* Plot **actual vs predicted values**.
* Plot **residuals vs fitted values** (should show no clear pattern).

import matplotlib.pyplot as plt

plt.scatter(model.predict(X), y)

plt.xlabel("Predicted values")

plt.ylabel("Actual values")

plt.title("Actual vs Predicted")

plt.show()

**2. No Multicollinearity**

➡️ Features (X variables) should not be **highly correlated** with each other.

**🔍 How to check:**

* Use a **correlation matrix**
* Use **Variance Inflation Factor (VIF)**

import pandas as pd

import seaborn as sns

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

# Correlation heatmap

sns.heatmap(X.corr(), annot=True, cmap='coolwarm')

# VIF

vif\_data = pd.DataFrame()

vif\_data["feature"] = X.columns

vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(len(X.columns))]

print(vif\_data)

⚠️ VIF > 10 = high multicollinearity (bad)

**3. Homoscedasticity (Equal Error Variance)**

➡️ The spread (variance) of residuals should be constant.

**🔍 How to check:**

* Plot **residuals vs predicted values**
* Look for a **funnel shape** (bad) vs random spread (good)

residuals = y - model.predict(X)

plt.scatter(model.predict(X), residuals)

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

plt.xlabel("Predicted values")

plt.ylabel("Residuals")

plt.title("Residuals vs Predicted")

plt.show()

**4. Normality of Residuals**

➡️ Residuals (errors) should be **normally distributed**.

**🔍 How to check:**

* Plot a **histogram** or **Q-Q plot**
* Use **Shapiro-Wilk test** for normality

import scipy.stats as stats

import seaborn as sns

sns.histplot(residuals, kde=True)

plt.title("Histogram of Residuals")

plt.show()

# Q-Q plot

stats.probplot(residuals, dist="norm", plot=plt)

plt.show()

**5. No Autocorrelation of Errors**

➡️ Residuals should not be correlated with each other (especially in time-series).

**🔍 How to check:**

* Use **Durbin-Watson statistic**

from statsmodels.stats.stattools import durbin\_watson

dw\_stat = durbin\_watson(residuals)

print("Durbin-Watson:", dw\_stat)

✅ Ideal value ≈ 2  
< 1 or > 3 → Strong autocorrelation (bad)

**🛠 What If Assumptions Are Violated?**

| **Problem** | **Fix** |
| --- | --- |
| Non-linearity | Try polynomial regression |
| Heteroscedasticity | Log-transform y, or use Weighted Least Squares |
| Multicollinearity | Drop correlated features or use PCA |
| Non-normal residuals | Try robust regression |
| Autocorrelated errors | Use time-series models like ARIMA |

Want me to help you do this with a real dataset (like diabetes, housing, etc.)?

Awesome follow-up! 👇

**❓ Do We Need to Check Assumptions *Before* Running Regression?**

**✅ Short Answer: No, but...**

You **don’t have to check them *before*** you *fit* the model —  
But you **must check them *after*** fitting the model to **validate** the results.

**💡 Why?**

When you call:

model.fit(X, y)

The model will still **run successfully** — even if the assumptions are violated.

But...

📉 If assumptions are **not met**, your model might give:

* **Wrong coefficients**
* **Misleading predictions**
* **Invalid p-values and confidence intervals**
* **Poor generalization**

**🔁 So What’s the Proper Flow?**

1. ✅ **Fit the model**: model.fit(X\_train, y\_train)
2. ✅ **Get predictions**: y\_pred = model.predict(X\_test)
3. ✅ **Check assumptions** (residuals, VIF, etc.)
4. ✅ **Interpret results** only if assumptions are okay
5. 🔁 **Fix or transform data** if assumptions are violated
6. ✅ **Retrain if needed**

**🧠 Real-World Practice:**

* In **exploratory data analysis (EDA)**, you might spot issues early (e.g. multicollinearity).
* But formal checks like **residual plots** and **Durbin-Watson test** come **after model training**.

Want to try a mini example now where we train a model and check assumptions together step-by-step?

Great question! The **Dummy Variable Trap** is a common issue in regression models — and it’s super easy to avoid once you understand it. Let’s break it down simply 👇

**🧩 What is a Dummy Variable?**

A **dummy variable** is used to represent **categorical data** as **numerical values** (0 or 1) so that machine learning models (like linear regression) can use them.

**Example:**

Category → "Color" with 3 values: Red, Blue, Green

We convert this into dummy variables:

Color\_Red Color\_Blue Color\_Green

1 0 0 ← Red

0 1 0 ← Blue

0 0 1 ← Green

**⚠️ What is the Dummy Variable Trap?**

The **Dummy Variable Trap** is when you **include all dummy variables** for a categorical feature, causing **perfect multicollinearity**.

📌 One variable can be **perfectly predicted** by the others.

This breaks one of the core assumptions of linear regression: **no multicollinearity**.

**👎 Problem:**

Using **all 3** dummy variables (Red, Blue, Green) introduces **redundancy**.

Why?

Because if you know:

* Color\_Red = 0
* Color\_Blue = 0  
  → Then it **must be Green (1)**

So the third dummy is **completely predictable** → this causes multicollinearity.

**✅ Solution: Drop One Dummy**

We just drop **one dummy variable** (any one — usually the first one).

Using pandas.get\_dummies() in Python:

pd.get\_dummies(data["Color"], drop\_first=True)

Now it looks like:

Color\_Blue Color\_Green

0 0 ← Red (baseline)

1 0 ← Blue

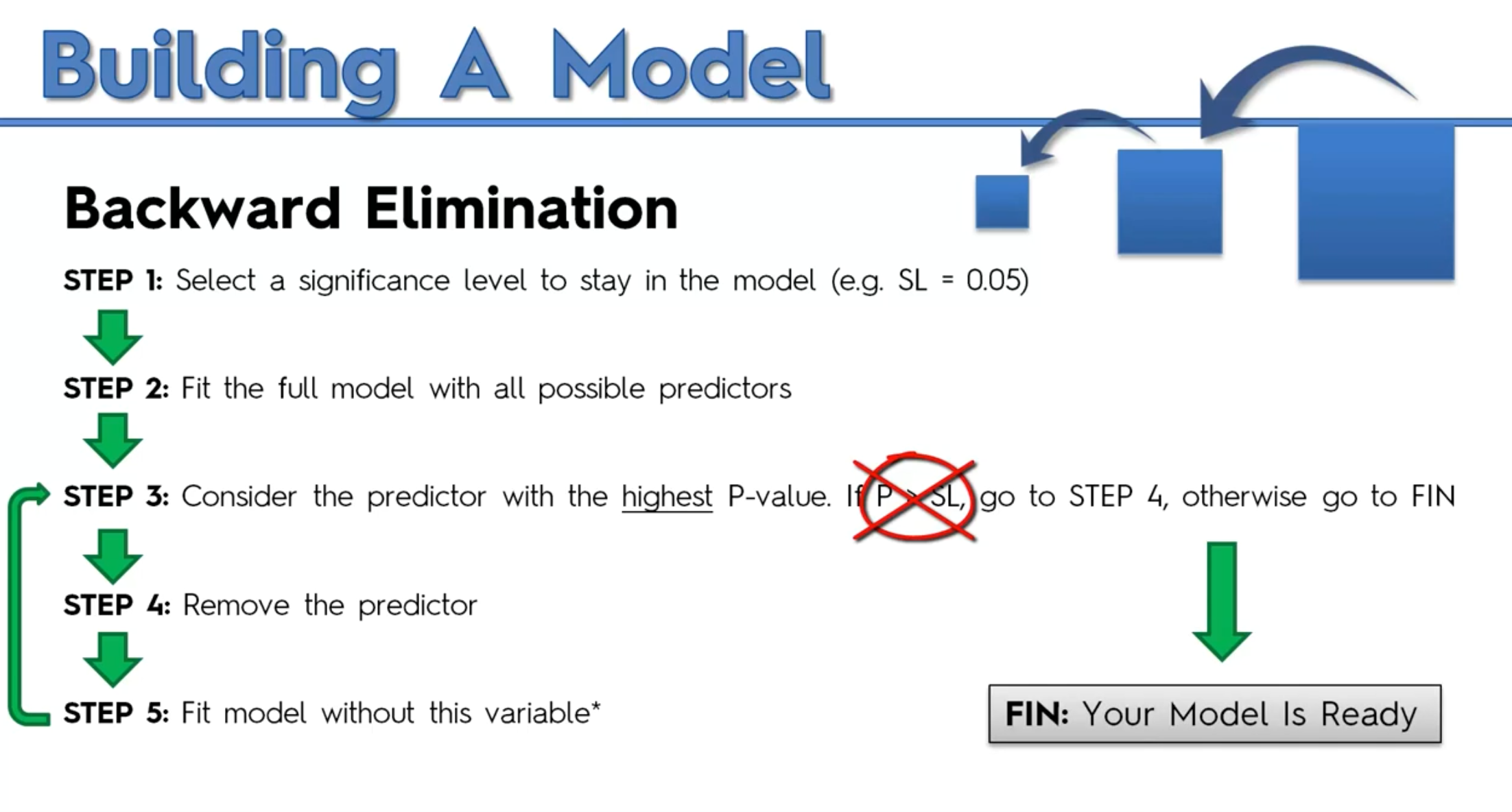
0 1 ← Green

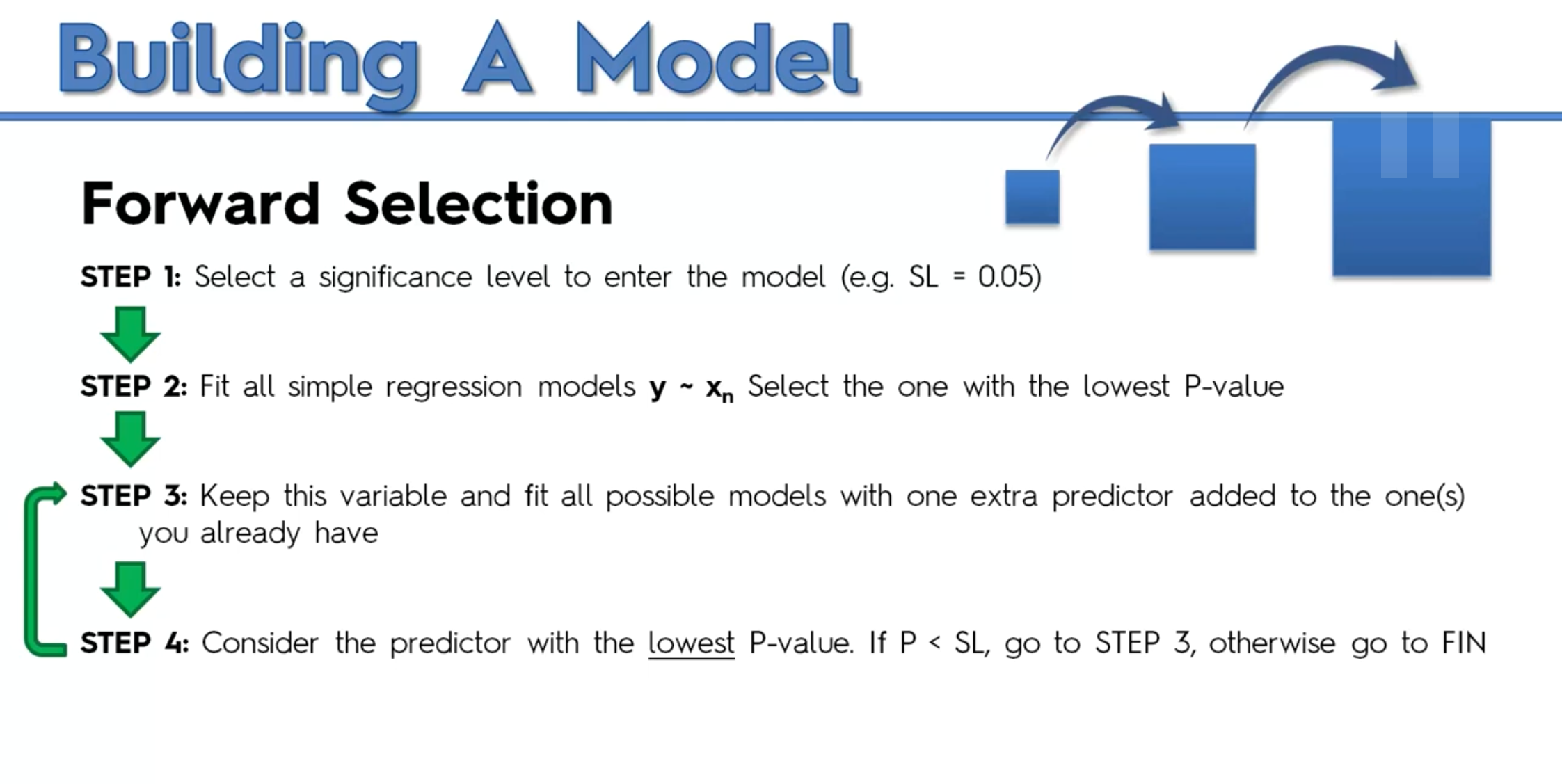
This avoids the trap. The dropped category (Red) becomes the **baseline/reference category**.

**🧠 Summary**

| **Concept** | **Meaning** |
| --- | --- |
| **Dummy Variable** | A way to convert categories into numbers |
| **Dummy Variable Trap** | Occurs when you use all dummy variables, causing multicollinearity |
| **Fix** | Drop one dummy (using drop\_first=True) |

Let me know if you want to try this with a real dataset (like Titanic or Iris) and see the trap in action!





Build a reg model with with 1 var one by one and get the least p value one

New put that one in a bucket and add second var with 1st try all var and get another and keep doing only stop when p> sl

