Learning to Talk to Data: My Journey in the WiDS 5.0 Program at IIT Bombay

Student Name: Keshav Sureka

Roll Number: 25b2204

Department: Mechanical

Organizers: Analytics Club & Web and Coding Club (WnCC), IIT Bombay

Duration: Winter Break 2025–2026

---

1. Introduction: What is WiDS?

The Winter in Data Science (WiDS) program at IIT Bombay is a special learning path designed to take students from knowing nothing about data to being able to build smart computer models. It follows a simple rule: **"Learn, Build, Connect."**

First, we **Learn** the basics through a three-week bootcamp. Then, we **Build** real projects using actual data from the real world. Finally, we **Connect** with mentors and other students to share what we found. This report explains everything I learned, from how to write basic Python code to how to make a computer predict the future using math.

2. Setting Up the Workshop: Python and Its Friends

Before we can analyze data, we need the right tools. We used the **Python** programming language because it is easy to read and has many "libraries" (pre-written code) that do the heavy lifting for us.

2.1 The Tools of the Trade

- **Pandas:** Think of this as "Excel for Python." It allows us to load data into tables called DataFrames and clean them up easily.
- **NumPy:** This is the math engine. It handles large lists of numbers very quickly using a trick called "vectorization" (doing math on a whole list at once instead of one by one).
- **Matplotlib & Seaborn:** These are the "artists." They take boring numbers and turn them into colorful charts and graphs so we can actually see what the data is telling us.

3. Data Wrangling: Getting the Data Ready

In the real world, data is messy. It's like finding a box of old, disorganized photos. "Data Wrangling" is the process of sorting those photos and fixing the blurry ones.

3.1 Loading the Data

We start by bringing our data into Python using a command like pd.read_csv(). We then split the data into two parts:

1. **Independent Features (X):** These are the "hints" or "clues." For example, if we are predicting house prices, the clues would be the number of rooms, the location, and the age of the house.
2. **Dependent Target (y):** This is the "answer" we want the computer to guess, like the actual price of the house.

4. The "Practice vs. Final Exam" Strategy: Train-Test Split

If you give a student the answers to a test before they take it, they haven't really learned; they've just memorized. Machine learning is the same.

To make sure our model actually "learns" patterns rather than just memorizing our data, we use a technique called **Train-Test Split**:

- **Training Set (70%):** We give this data to the computer to let it practice and find patterns.
- **Test Set (30%):** We hide this data. Once the computer thinks it's smart, we give it this "final exam" to see if it can guess the answers for data it has never seen before.

4.2 Why Use random_state=42?

When we split data, the computer picks rows randomly. If we run the code again, it might pick different rows, giving us different results. By setting random_state=42, we tell the computer to always pick the **same** random rows. This is like a "seed" that makes sure our experiment is reproducible—anyone else running the code will get the exact same split.

5. Fairness in Math: Feature Scaling

Imagine you are comparing two people. One is 1.8 meters tall, and the other weighs 80 kilograms. A computer is "dumb"—it might think 80 is much more important than 1.8 just because the number is bigger.

To fix this, we use the **StandardScaler**. It makes every column "fair" by putting them on the same scale.

5.1 The Magic Formula

The scaler uses this math for every single number:

$z = (x-u)/z$

- **x:** The original number.
- **u (Mean):** The average of all numbers in that column.

- **z (Standard Deviation):** How much the numbers "spread out" from the average.

After this, the average of every column becomes 0, and the spread becomes 1. Now the computer can compare height and weight fairly.

6. The "Line of Best Fit": Linear Regression

Linear Regression is the "mother of all algorithms." Its goal is simple: find a straight line that goes through our data points as perfectly as possible.

6.1 The Analogy: The Lemonade Stand

Imagine you have a lemonade stand. You notice that when it's hotter outside, you sell more lemonade. You plot the temperature (X) and sales (y) on a graph. Linear Regression draws a line through those points so you can say, "If it's 35 degrees tomorrow, I'll probably sell 100 cups."

6.2 The Equation of the Line

The computer is trying to solve this puzzle:

$y = a x + b$

- **y:** The prediction (Sales).
- **x:** The input (Temperature).
- **a(Slope):** How much y changes when x goes up by one.
- **b(Intercept):** The starting point on the graph when x is zero.

6.3 Ordinary Least Squares (OLS)

How does the computer find the "best" line? It uses **Ordinary Least Squares**. It measures the distance between each real data point and the line (this distance is called the **Residual** or error), squares those distances (to make them all positive), and adds them up. The "best" line is the one where this total sum is the smallest possible.

7. Checking Our Work: Cross-Validation

One "final exam" might not be enough. What if the test set was too easy? To be extra sure, we use **Cross-Validation** (specifically K-Fold).

We divide our data into 10 groups (folds). The computer trains on 9 groups and tests on the 10th one. It repeats this 10 times, using a different group as the test set each time. Then, we take the average of all 10 scores. This gives us a much more honest view of how good our model really is.

7.1 Why is it called neg_mean_squared_error?

In Scikit-Learn, the computer always wants to "maximize" a score (higher is better). But in data science, "Error" is bad—we want it to be as low as possible.

To fix this, the software flips the sign. If our error is 5, it calls the score -5. If the error is 10, it calls the score -10. Since -5 is a "higher" number than -10, the computer correctly picks the smaller error.

8. Did We Win? Measuring Success

After the computer makes its guesses, we need to know how well it did.

8.1 The R-Squared (R^2) Score

This is a number between 0 and 1 that tells us how much of the "story" our model explained.

- **R^2 = 1:** Perfect! The model explained everything.

- **R^2 = 0:** Useless. The model is just guessing the average.
- **R^2 = 0.80:** Great! The model explained 80% of why the data looks the way it does.

8.2 Residual Analysis (The KDE Plot)

We also look at the "leftover" errors (residuals). We use a **KDE Plot**, which looks like a smooth hill. If the hill is centered at 0, it means our errors are small and balanced—our model is doing a great job.

9. Advanced Concepts: Beyond the Basics

While I spent most of my time on the basics, the WiDS program also introduced me to more complex ideas:

- **Decision Trees:** A model that asks "Yes/No" questions to find an answer. It's like the game "20 Questions."
- **Random Forests:** Instead of one tree, we use a whole forest (a committee of trees) and let them vote on the best answer. This is usually much more accurate.
- **Deep Learning:** Models inspired by the human brain (Neural Networks) that are great for recognizing images or text.
- **Generative AI:** The newest part of the program, where we learned how models like ChatGPT predict the next word in a sentence and how we can "prompt" them to give us better answers.

10. Conclusion and Personal Learnings

The WiDS program was more than just a coding class. It taught me how to think logically about problems. I learned that data isn't just a list of numbers; it's a story waiting to be told.

By following the rigorous steps of cleaning data, scaling features, and validating models with math, I now have the foundation to tackle real-world problems in finance, engineering, or even medicine. Being a student at IITB, this experience has opened doors for me to join the global community of researchers and innovators in Artificial Intelligence.
