Reflective Journal — Lab 02

When I first started this lab, I thought working with data in Python would feel similar to Excel: rows, columns, and basic formulas. Very quickly I realized that tools like pandas and NumPy aren't just replacements for spreadsheets—they change the way I think about data. Using the Iris dataset gave me the chance to test that shift, and I ended up learning more from my mistakes and questions than from the "successful" steps.

What was difficult at first

Indexing in pandas was one of the first things that confused me. I kept mixing up loc and iloc, and I often got errors that didn't make sense to me. At some point it clicked: loc is about labels (like "species"), while iloc is about positions (row numbers). Once I saw it that way, pandas started to feel more logical, almost like Excel with superpowers. That was a small breakthrough because it stopped me from treating errors as random problems and helped me see them as signs that I was thinking about the data incorrectly.

My "aha" moment with NumPy

I also noticed a big change in my mindset when I used NumPy. At first, I wanted to write for-loops to calculate things like averages, but then I saw how vectorized operations could do the same task in one line. The interesting part wasn't just that it was faster—it actually matched the way math is written, where you apply operations to whole arrays instead of single numbers. That moment made me realize that coding in data science is less about telling the computer every small step, and more about thinking in sets and letting NumPy do the heavy lifting.

Visualizations as more than pictures

Making scatter plots with matplotlib felt simple at first, but then I noticed something important: the plots were telling me about class separability. For example, setosa clustered clearly on its own, while versicolor and virginica overlapped. This taught me that graphs are not just for decoration—they answer questions. I also saw how species counts were balanced (50 each), and that reminded me to always check distributions before trusting a model. I used to skip those details, but now I understand they matter for accuracy later on.

Learning from errors

Some of my best learning came from error messages. I got a KeyError when I mistyped a column name, and a ValueError when shapes didn't match between arrays. At first I was frustrated, but then I learned to slow down, read the traceback carefully, and even add small print() or .head() checks to verify assumptions. Debugging started to feel like a teacher rather than a punishment. I realized that handling errors well is just as much a part of programming as writing correct code.

A new perspective on data cleaning

I always thought data cleaning was boring, but this lab showed me why it is critical. Even a small inconsistency can affect statistics and plots, and that directly changes model performance. When I saw how petal features separated species better than sepal features, it made me think about feature selection as part of “data quality” rather than just preparation.

How my approach changed

At the beginning, I treated the lab as a checklist: import libraries, load the dataset, make some graphs. By the end, I was using each step to test my own understanding. Instead of rushing through, I asked: *Does this operation reflect my intention? Does this plot actually answer a question? Is my data reliable?* The most valuable lesson for me was realizing that errors and small checks are not detours—they are part of the workflow. That mindset shift made the tools (Colab, pandas, NumPy, matplotlib) feel like a unified way of reasoning about data, not just separate programs.