Power of Indexing with “ If ” and “ elif “ in Python

When working with data, efficiency and clarity are crucial. In my recent projects, I've discovered the power of using indexing with `if` and `elif` statements in Python. This technique helped me identify patterns, merge data, and improve overall workflow. Let’s dive into how it worked for me and why it’s so much better than doing things the hard way.

1. Spotting Patterns in Data

Imagine manually sifting through each column to find specific ones for merging. Boring, right? Indexing can make this task a breeze.

Example:

```
s = []
for col in tr1.columns:
    if col[-1] in ("d", "1"):
        s.append(col)
print(s)
# Output: ['case_id', 'num_group1']
```

Here, I’m looking at the last character of each column name. If it’s a 'd' or '1', I add that column to my list. This way, I quickly found the columns I needed without the tedious manual search.

2. Making Data *Handling Efficient*

Structuring data manually is another monotonous task. Setting the right data types for each column can be time-consuming and error-prone. Indexing simplifies this process.

Example:

```
for col in df.columns:
    if col in ["case_id", "WEEK_NUM", "num_group1", "num_group2"]:
        df = df.with_columns(pl.col(col).cast(pl.Int64))
    elif col[-1] in ("P", "A"):
        df = df.with_columns(pl.col(col).cast(pl.Float64))
```

```
    elif col[-1] in ("M",):
        df = df.with_columns(pl.col(col).cast(pl.String))
    elif col[-1] in ("D",):
        df = df.with_columns(pl.col(col).cast(pl.Date))
```

This code checks each column and sets the appropriate data type based on the column name or its last character. It ensured my data was correctly structured without manual intervention, saving me from a dull and repetitive task.

Conclusion

Using indexing with `if` and `elif` statements in Python can make your data work more efficient and understandable. Whether identifying patterns or structuring data, these techniques save time and reduce errors, making your life much easier.