Lesson Reflection

Summary

This lesson covered different techniques for accessing and filtering data stored in Pandas DataFrames. It introduced logical and comparison operators like ==, !=, >, < for elementwise comparisons, and &, | for combining multiple Boolean filter conditions. Using these operators, we can select subsets of data that meet specific criteria.

We also saw functions like isin() and DataFrame indexer loc[] to apply filter arrays and select data. Overall, these methods allow slicing data in flexible ways during analysis.

Key Points

- Comparison operators compare DataFrame elements and produce Boolean arrays
- Boolean operators combine multiple comparison expressions
- Filters subset DataFrame rows where expression is True
- loc[] selects data by label or Boolean filter
- isin() filters rows if value in list of values

Reflection Questions

- What are some common use cases for filtering DataFrames?
- How can chaining multiple Boolean operators enable more precise data selection?
- When might you use isin() versus repeated equality checks with == ?
- What are benefits of using loc[] over column-wise data access?
- Where might complex Boolean filters become difficult to manage?

Challenges

- Filter a DataFrame to only show values greater than the mean
- Combine filters to find rows matching multiple Boolean conditions
- Use isin() to filter categorical data like product types
- Slice DataFrame with .loc[] using a Boolean filter
- Chain complex filters with multiple Boolean operators

```python
import pandas as pd
import numpy as np

# Sample fruit price DataFrame
data = {'Fruit': ['Apple', 'Banana', 'Orange'],
        'Price': [2.5, 1.2, 3.3]}
df = pd.DataFrame(data)

# Calculate average price
avg_price = df['Price'].mean()
print(avg_price)

# Filter prices > average
filter = df['Price'] > avg_price
df.loc[filter]

# isin() filter on fruits
fruit_filter = df['Fruit'].isin(['Apple','Orange'])
df.loc[fruit_filter]
```

```
2.3333333333333335
    Fruit  Price
0   Apple    2.5
2  Orange    3.3
```