Lesson 06: Selection and Filtering

Data Science with Python – BSc Course

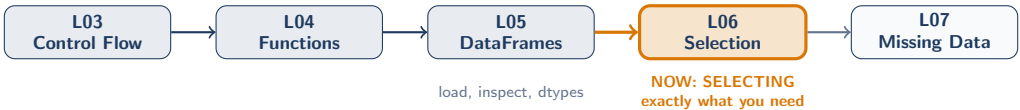
Data Science Program

BSc Course

45 Minutes

Where We Are

Last time: DataFrames gave you tabular data power



You can load a DataFrame with 10,000 rows. But you need 50 specific rows that match your criteria. How do you extract them?

Selection and filtering are the query language of pandas

Learning Objectives

After this lesson, you will be able to:

- **Select** columns using bracket and dot notation
- **Access** rows with `iloc` (position) and `loc` (label)
- **Filter** data using boolean conditions
- **Combine** multiple conditions with `&`, `|`, and `~`
- **Build** a stock screener using selection and filtering

Finance Application: Screen stocks by price, volume, sector, and returns.

Module 1 capstone – you will use all skills from L01–L06 together

The Selection Problem

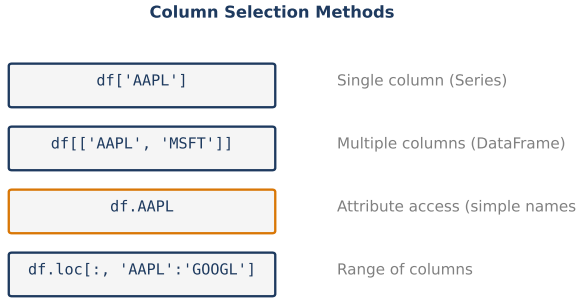
You have 10,000 rows. You need 50. How?

Boolean conditions select matching rows

Three tools: **column selection** (which columns), **row indexing** (`loc`/`iloc`), and **boolean filtering** (which rows match).

Selection = choosing columns. Filtering = choosing rows by condition.

Column Selection



Single brackets return Series; double brackets return DataFrame

loc: Label-Based Selection

Select by row and column labels

Single row:

```
df.loc["2024-01-02"]
# Returns that row as Series
```

Row range (inclusive!):

```
df.loc["2024-01":"2024-03"]
# All rows Jan through Mar
```

Row + column:

```
df.loc["2024-01-02", "AAPL"]
# Single value:  185.2
```

Multiple columns:

```
df.loc[:, ["AAPL", "MSFT"]]
# All rows, 2 columns
```

Key rule: `loc` uses **labels** and the end is **inclusive**.

---

loc = label-based. Both start and end are INCLUDED in the range.

iloc: Integer-Based Selection

Select by row and column positions (0-based)

Single row:

```
df.iloc[0]  # First row
df.iloc[-1] # Last row
```

Row range (exclusive end):

```
df.iloc[0:5]
# Rows 0, 1, 2, 3, 4
```

Row + column:

```
df.iloc[0, 1]
# Row 0, column 1
```

Slicing both:

```
df.iloc[0:5, 1:3]
# Rows 0--4, columns 1--2
```

Key rule: `iloc` uses **integers** and the end is **exclusive** (like Python slicing).

---

iloc = integer location. End index is EXCLUDED, just like list slicing.

Boolean Indexing: The Filter Pipeline


```
df[df["AAPL"] > 188]  # returns only matching rows
```

How it works:

1. Write a condition → produces a True/False Series (the "mask")
2. Pass the mask into `df[mask]` → keeps only True rows

Boolean indexing: condition creates mask, mask filters rows

Boolean Masking in Action

AAPL

185
190
188
195
182

Condition:

`df["AAPL"] > 188`

Mask:

`False`
`True`
`False`
`True`
`False`

Result:

Only rows where True

190
195

2 of 5 rows selected

The mask is a Series of booleans, same length as the DataFrame.

Boolean mask selects rows where condition is True

Combining Multiple Conditions

AND: &

```
(df["AAPL"] > 185) &
(df["MSFT"] > 380)
```

Both conditions must be True

NOT: ~

```
~(df["AAPL"] > 190)
```

Inverts the condition

OR: |

```
(df["AAPL"] > 200) |
(df["MSFT"] > 400)
```

Either condition can be True

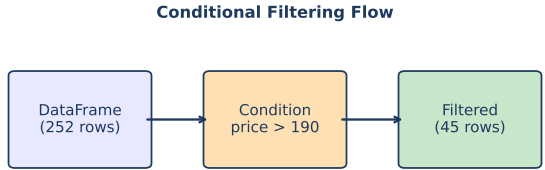
Full example:

```
mask = (df["AAPL"] > 185) &
       (df["Volume"] > 1e6)
result = df[mask]
```

Critical: Always use **parentheses** around each condition!

Without parentheses, operator precedence causes bugs

Conditional Filtering




```
df_filtered = df[df["AAPL"] > 190]
```

Filtering creates a new DataFrame – original remains unchanged

The query() Method

Traditional (verbose):

```
df[(df["AAPL"] > 185) &
   (df["Volume"] > 1e6)]
```

Repeated `df` everywhere.

With variables:

```
threshold = 185
df.query("AAPL > @threshold")
```

`@` prefix accesses Python variables.

query() (cleaner):

```
df.query("AAPL > 185 and
         Volume > 1e6")
```

More readable, no parentheses needed.

Supports:

- `and`, `or`, `not`
- Comparisons: `>`, `<`, `==`
- `in` for membership
- Backticks for spaces in names

---

query() is more readable for complex filters

Checkpoint: Test Your Understanding

Q1: `loc` vs `iloc` – which uses labels and which uses integer positions?

Q2: What does `df.iloc[0:3]` return? How many rows?

Q3: Why do we need parentheses in `(df["A"] > 5) & (df["B"] < 10)`?

Think for 30 seconds. Answers: Q1: loc=labels, iloc=integers. Q2: 3 rows (0,1,2). Q3: & has higher precedence than comparison operators.

---

loc = inclusive end; iloc = exclusive end. This trips up everyone.

Membership Testing: isin()

Select rows where a column matches any value in a list

```
# Select only tech stocks
tech = ["AAPL", "MSFT", "GOOGL", "AMZN"]
df[df["Ticker"].isin(tech)]
```

Without isin (verbose):

```
(df["Ticker"] == "AAPL") |
(df["Ticker"] == "MSFT") |
(df["Ticker"] == "GOOGL") |
(df["Ticker"] == "AMZN")
```

With isin (clean):

```
df["Ticker"].isin(tech)
```

Exclude with ~:

```
df[~df["Ticker"].isin(tech)]
# Everything EXCEPT tech
```

---

isin() replaces long chains of OR conditions

String Methods: .str Accessor

Filter and transform text columns

```
# Contains a substring
df[df["Name"].str.contains("Tech")]
# Starts with
df[df["Ticker"].str.startswith("A")]
# Case-insensitive search
df[df["Sector"].str.lower() == "finance"]
```

Useful .str methods:

`.str.contains("x")`	Substring match
`.str.startswith("A")`	Starts with
`.str.len()`	String length
`.str.upper()`	Uppercase
`.str.strip()`	Remove whitespace

.str accessor unlocks all Python string methods for entire columns

Finance: Filtering by Sector

Stock screeners start with sector and industry filters

```
# Load S&P 500 constituents
sp500 = pd.read_csv("sp500.csv")
# Filter to financial sector
financials = sp500[sp500["Sector"] == "Financials"]
print(f"Financial stocks:  {len(financials)}")
# Further filter:  large cap only
large_fin = financials[financials["MarketCap"] > 50e9]
# Or in one step with query():
large_fin = sp500.query(
    'Sector == "Financials" and MarketCap > 50e9')
```

Chaining filters narrows 500 stocks down to exactly what you need.

Stock screening = sector filter → size filter → metric filter

Finance: Multi-Criteria Stock Screener

Combine all tools to build a real screener

```
# Criteria:  cheap, liquid, positive momentum
screened = df[
    (df["PE_Ratio"] < 20) &         # Value
    (df["AvgVolume"] > 1e6) &       # Liquidity
    (df["Return_3M"] > 0) &         # Momentum
    (df["Sector"].isin(["Tech", "Health"]))  # Sector
]
# Sort by return, best first
screened = screened.sort_values(
    "Return_3M", ascending=False)
print(f"Passed screening:  {len(screened)} stocks")
```

4 conditions, 3 tools (&, isin, sort) – this is professional-grade screening.

Real hedge funds use exactly this logic, just with more criteria

Sorting Data

Order rows by one or more columns

Single column:

```
# Ascending (default)
df.sort_values("AAPL")
# Descending
df.sort_values("AAPL",
    ascending=False)
```

By index:

```
df.sort_index()  # By date
```

Multiple columns:

```
df.sort_values(
    ["Sector", "Return"],
    ascending=[True, False])
```

Sort by sector A-Z, then within each sector sort by return descending.

Top N:

```
df.nlargest(10, "Return")
df.nsmallest(5, "Price")
```

---

sort_values returns a new DataFrame; use inplace=True to modify original

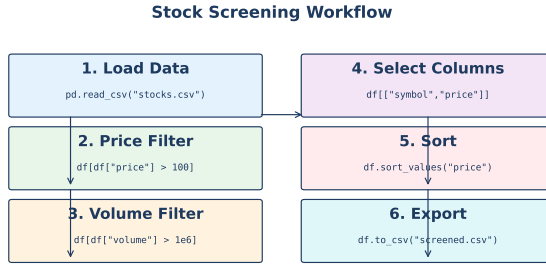
Selection Methods Comparison

Selection Methods Comparison

Method	Use Case	Returns
`df["col"]`	Single column	Series
`df[["col1","col2"]]`	Multiple columns	DataFrame
`df.iloc[0]`	Row by position	Series
`df.loc["date"]`	Row by label	Series
`df[df.col > x]`	Filter rows	DataFrame

Choose method based on what you need to select

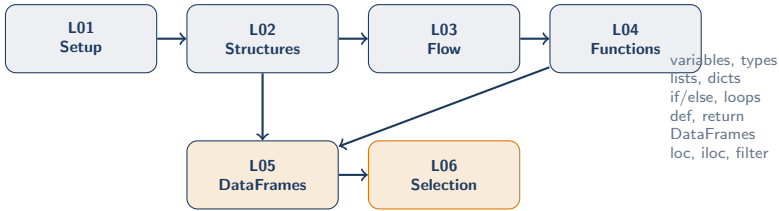
Stock Screening Workflow



Combine filters to build powerful stock screeners

Stock screening = loading + filtering + selecting + sorting

Module 1 Complete!



You now speak Python + pandas!

Six lessons: from zero Python to filtering stock data

Hands-on Exercise (25 min)

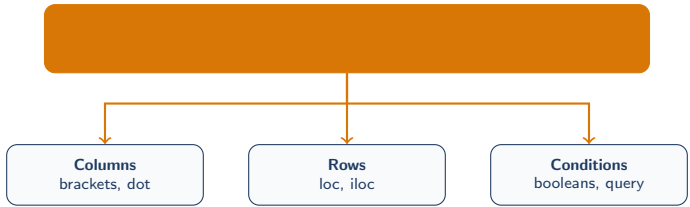
Build a stock screener:

1. Load stock data from CSV
2. Select only AAPL and MSFT columns
3. Filter rows where AAPL ¿ 185
4. Filter rows where AAPL ¿ 185 AND MSFT ¿ 375
5. Use query() for the same filter
6. Select first 10 trading days using iloc
7. Sort by AAPL price descending

Bonus: Use `isin()` to filter for a list of tickers. Add `.str.contains()` to search company names.

Stock screeners are fundamental tools in quantitative finance

The Big Idea



With these three tools, you can extract any subset from any dataset.

Columns + rows + conditions: the complete selection toolkit

Lesson Summary

Key Takeaways:

- `df["col"]` selects a column as Series; `df[["col"]]` as DataFrame
- `iloc` uses integer positions (exclusive end); `loc` uses labels (inclusive end)
- Boolean conditions create True/False masks for filtering
- Combine conditions with `&` (and), `|` (or), `~` (not)
- `query()` and `isin()` make complex filters readable

Up Next: Module 2 – Data Manipulation

L07: Missing Data. Real datasets are messy. Next you learn to detect, handle, and clean missing values.

Module 1 complete – you can load, explore, select, and filter data!