Lesson 05: DataFrames Introduction

Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

Where We Are

Last time: Functions gave your code reusability

You can write functions and loops. But real data has thousands of rows and dozens of columns. You need a structure designed for **tabular data**.

---

pandas DataFrames are THE workhorse of data science in Python

Learning Objectives

After this lesson, you will be able to:

- **Import** pandas and create DataFrames from dictionaries and CSV files
- **Inspect** data with `head()`, `tail()`, `shape`, `info()`, `describe()`
- **Distinguish** Series (1D) from DataFrame (2D)
- **Explain** the role of index, columns, and dtypes
- **Apply** `set_index()` and basic column operations to stock data

Finance Application: Load and explore real stock price data.

---

pandas is THE library for data manipulation in Python

Why Not Just Use Excel?

Excel has rows and columns. pandas has that PLUS the power of Python.

Excel limitations:

- ~1 million row limit
- Manual point-and-click
- Hard to reproduce analysis
- One file at a time

pandas advantages:

- Millions of rows, no limit
- Scriptable and automated
- Fully reproducible
- Process 100 files in a loop

```
import pandas as pd
df = pd.read_csv("stock_prices.csv")  # One line to load data!
```

pandas was created by Wes McKinney at a hedge fund – built for finance.

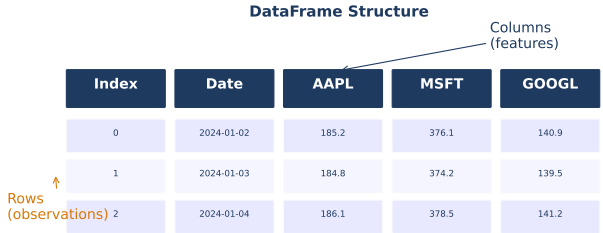
---

pandas was literally built for financial data analysis

A DataFrame is a table with labeled rows and columns

DataFrame = labeled 2D table. Series = labeled 1D column.

DataFrame Structure



2D labeled data structure with rows and columns

DataFrames are 2D labeled data structures with index, columns, and values

Creating DataFrames

From a dictionary:

```
import pandas as pd
data = {
    "Ticker":  ["AAPL", "MSFT"],
    "Price":  [185.50, 340.20],
    "Shares":  [100, 50]
}
df = pd.DataFrame(data)
```

Keys become column names.
Lists become column values.

From a list of lists:

```
data = [
    ["AAPL", 185.50, 100],
    ["MSFT", 340.20, 50]
]
df = pd.DataFrame(data,
    columns=["Ticker",
             "Price", "Shares"])
```

From CSV (most common):

```
df = pd.read_csv("prices.csv")
```

Dictionary method is most common for small data; CSV for real datasets

Series vs DataFrame

Series (1D):
Single column with index

Index	Price
0	185.2
1	184.8
2	186.1

```
s = df["AAPL"]  # Series
type(s)  # pd.Series
```

DataFrame (2D):
Multiple columns sharing an index

	AAPL	MSFT	VOL
0	185.2	376.1	1.2M
1	184.8	374.2	1.1M
2	186.1	378.5	1.3M

```
sub = df[["AAPL","MSFT"]]  # DF
type(sub)  # pd.DataFrame
```

Single brackets `["col"]` → Series. *Double brackets* `[["col"]]` → DataFrame.

Series = 1 column; DataFrame = table of Series sharing an index

First Look: head(), tail(), shape

df.head(3) – First 3 rows

Date	Price
2024-01-02	185.2
2024-01-03	184.8
2024-01-04	186.1

df.shape → `(252, 4)`
252 rows, 4 columns

df.tail(3) – Last 3 rows

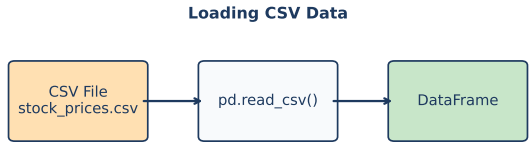
Date	Price
2024-12-27	195.8
2024-12-30	196.2
2024-12-31	197.1

df.columns
→ `Index(['Date', 'AAPL', ...])`

Default: 5 rows — **Customize:** `head(10)`, `tail(20)`

Always inspect data after loading – head() is your first reflex

Loading CSV Data



Common Parameters:

```
filepath: "data/prices.csv"
index_col: "Date"
parse_dates: True
usecols: ["AAPL", "MSFT"]
```

pd.read_csv() handles most CSV formats automatically

Inspecting Data: info() and describe()

df.info() reveals structure:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex:  252 entries
Data columns (4 total):
  Date    object    252 non-null
  AAPL    float64   252 non-null
  MSFT    float64   250 non-null
  Volume  int64     252 non-null
```

MSFT has 2 missing values!

df.describe() shows statistics:

Stat	AAPL	MSFT
count	252	250
mean	189.5	385.2
std	8.2	12.5
min	175.1	355.8
50%	188.9	384.1
max	210.3	420.5

info() = data types + missing values. *describe()* = statistical summary.

---

info() finds data quality issues; describe() finds statistical patterns

DataFrame Info Breakdown

DataFrame Info: df.info()

```
<class pandas.DataFrame>
RangeIndex: 252 entries, 0 to 251
Data columns (5 columns):
  Date    252 non-null datetime64
  AAPL    252 non-null float64
  MSFT    252 non-null float64
  GOOGL   250 non-null float64  (2 missing)
memory usage: 10.0 KB
```

info() reveals data types, missing values, and memory usage

Checkpoint: Test Your Understanding

Q1: What is the difference between a Series and a DataFrame?

Q2: You run `df.shape` and get `(1000, 5)`. How many rows? How many columns?

Q3: Which method tells you about missing values – `info()` or `describe()`?

Think for 30 seconds. Answers: Q1: Series=1 column, DF=table of columns. Q2: 1000 rows, 5 columns. Q3: info().

If you got all three, you understand DataFrame basics

Data Types (dtypes) and Memory

Common pandas dtypes:

dtype	Contains
`int64`	Whole numbers
`float64`	Decimals
`object`	Strings/mixed
`datetime64`	Dates/times
`bool`	True/False
`category`	Categorical

```
df.dtypes  # Check all columns
```

Why dtypes matter:

- `object` columns cannot be summed
- Math only works on numeric dtypes
- Wrong dtype = wrong results silently

Converting types:

```
df["Price"] = df["Price"].astype(float)
```

```
df["Date"] = pd.to_datetime(df["Date"])
```

Memory:

```
df.memory_usage(deep=True)
```

Always check dtypes after loading – string numbers need conversion

Reading CSV Files in Detail

pd.read_csv() – the most-used pandas function

```
# Basic load
df = pd.read_csv("stock_prices.csv")
# With options
df = pd.read_csv("stock_prices.csv",
    index_col="Date",       # Use Date as index
    parse_dates=True,       # Convert dates automatically
    usecols=["Date", "AAPL", "MSFT"])  # Only these columns
```

Useful parameters:

`sep=","`	Column separator (default comma)
`header=0`	Row number for column names
`na_values=["N/A"]`	Treat these as missing
`nrows=100`	Only read first 100 rows

---

read_csv handles 90% of data loading in practice

Finance: Loading Stock Price Data

A typical workflow for stock data

```
import pandas as pd
# Step 1: Load the CSV
df = pd.read_csv("AAPL_daily.csv",
    parse_dates=["Date"])
# Step 2: Quick inspection
print(df.shape)      # (252, 6)
print(df.head())     # First 5 rows
print(df.dtypes)     # Check types
# Step 3: Set date as index
df = df.set_index("Date")
# Step 4: Check for problems
print(df.isnull().sum())  # Missing values?
```

Always follow this 4-step pattern when loading any new dataset.

Load → inspect → set index → check quality

Setting the Index

Use dates as row labels for time series data

Before (default integer index):

	Date	Price
0	2024-01-02	185.2
1	2024-01-03	184.8
2	2024-01-04	186.1

```
df.set_index("Date", inplace=True)
```

After (date index):

Date	Price
2024-01-02	185.2
2024-01-03	184.8
2024-01-04	186.1

Now you can slice by date:

```
df.loc["2024-01":"2024-03"]
```

Reset: `df.reset_index()` moves the index back to a regular column.

Date indices enable powerful time-based slicing and resampling

Basic Column Operations

Create new columns from existing ones

```
# Simple arithmetic
df["Return"] = df["Close"].pct_change()

# Difference
df["Spread"] = df["High"] - df["Low"]

# From a function (L04!)
def calc_return(series):
    return series.pct_change()

df["Return"] = calc_return(df["Close"])
```

Key operations:

`df["A"] + df["B"]`	Element-wise addition
`df["A"] * 100`	Scalar multiplication
`df["A"].pct_change()`	Percentage change

Column operations are vectorized – no loops needed!

Finance: Calculating Daily Returns

Returns are the language of finance

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \texttt{pct\_change()}$$

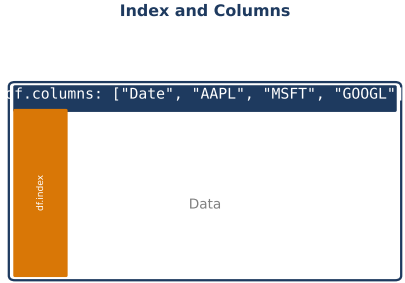
```
# One line to compute all daily returns
df["Return"] = df["Close"].pct_change()
# Log returns (for statistical analysis)
import numpy as np
df["LogReturn"] = np.log(df["Close"] / df["Close"].shift(1))
# Quick statistics
print(f"Mean daily return:  {df['Return'].mean():.4f}")
print(f"Daily volatility:  {df['Return'].std():.4f}")
```

No loops needed. pandas computes returns for all 252 days at once.

---

pct_change() computes returns in one vectorized operation

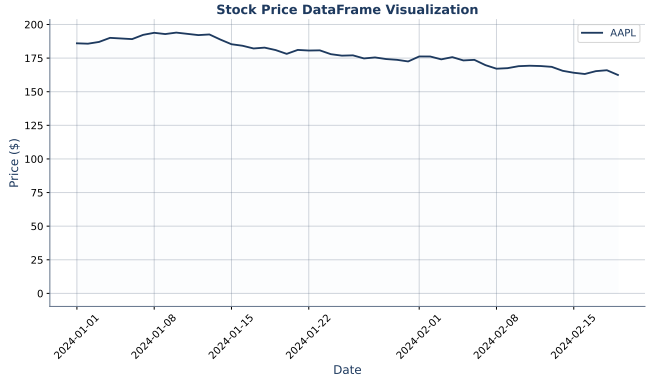
Index and Columns



df.shape: (252, 4) | df.dtypes: column data types

Index labels rows; columns label data fields

Stock Data Example



Financial time series are natural DataFrames – dates as index, prices as columns

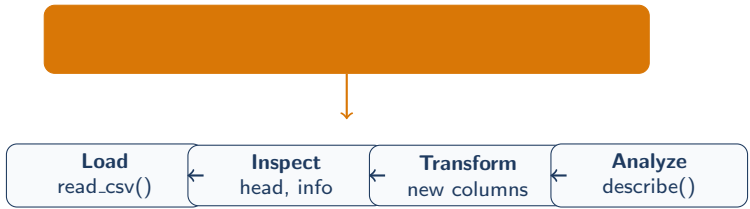
Hands-on Exercise (25 min)

Explore stock price data:

1. Load stock data:
   `df = pd.read_csv("stock_prices.csv", parse_dates=["Date"])`
2. Set the date index: `df.set_index("Date", inplace=True)`
3. View first 10 rows: `df.head(10)`
4. Check data types: `df.info()`
5. Get statistics: `df.describe()`
6. Create a return column: `df["Return"] = df["AAPL"].pct_change()`
7. Find which stock has the highest mean price

Exploration before analysis prevents costly mistakes

The Big Idea



This four-step workflow applies to every dataset you will ever touch.

Load → Inspect → Transform → Analyze: the universal data workflow

Lesson Summary

Key Takeaways:

- pandas DataFrame is the core data structure for tabular data
- `pd.read_csv()` loads CSV files in one line
- `head()`/`tail()` for quick inspection; `info()` for structure; `describe()` for stats
- Single brackets return Series; double brackets return DataFrame
- Date indices enable time-based slicing of financial data

Next Lesson: L06 – Selection and Filtering
You have 10,000 rows. You need 50. Next, you learn how to extract exactly the data you need.

Loading data is step 1 – now we learn to slice and filter it