Python and SQL: intro / SQL platforms

Ewa Weychert

Class 2: If statements and pandas

Overview

- **Pandas** is a Python library for data manipulation and analysis.
- It provides two main data structures:
  - **Series** – 1-dimensional labeled array.
  - **DataFrame** – 2-dimensional labeled data table.
- Built on top of `NumPy`.
- Commonly used for:
  - Data cleaning
  - Data exploration
  - Data transformation

Why the Name "Pandas"?

- The name **Pandas** does **not** come from the animal .
- It comes from the term **"Panel Data"**, a concept in *econometrics*.
- **Panel Data** means:

  Data sets that include observations over multiple time periods for the same individuals.

- Therefore:

$$\text{Pandas} = \text{PANel} + \text{DAta}$$

- Created by **Wes McKinney** in 2008 to simplify data analysis in Python.

"Pandas makes panel and tabular data easy to handle!"

Pandas 1/3


I ALWAYS IMPORT PANDAS AS PD
You know, I'm something of a data scientist myself
imgflip.com


Google
Sheets
X
Excel

Pandas 3/3

Installation

To install Pandas, use:

```
pip install pandas
```

Importing Pandas

```
import pandas as pd
```

- `pd` is the standard alias.
- You can now use all Pandas functions with `pd`.

Series

```
import pandas as pd
s = pd.Series([10, 20, 30, 40])
print(s)
```

- A one-dimensional labeled array.
- Each element has an index.

DataFrame

```
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

- A two-dimensional labeled structure.
- Similar to a spreadsheet or SQL table.

Reading Data

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

- Load data from CSV, Excel, JSON, SQL, etc.
- Example: `pd.read`$_{e}xcel("file.xlsx")$

Inspecting Data

```
df.head()        # First 5 rows
df.tail()        # Last 5 rows
df.info()        # Summary
df.describe()    # Statistics
```

Selecting Data

```
df["Age"]            # Select column
df.loc[0]            # Select row by label
df.iloc[1]           # Select row by index
df[df["Age"] > 30]   # Filter rows
```

Adding / Removing Columns

```
df["Country"] = ["USA", "UK", "Canada"]
df.drop("Age", axis=1, inplace=True)
```

Handling Missing Data

```
df.dropna()       # Remove missing values
df.fillna(0)      # Replace missing values
```

Exporting Data

```
df.to_csv("output.csv", index=False)
df.to_excel("output.xlsx", index=False)
```

Summary

- Pandas makes data analysis easy and powerful.
- Key structures: **Series** and **DataFrame**.
- Supports reading/writing many file types.
- Perfect for cleaning, transforming, and analyzing data.

Why Pandas Is Popular in Data Science

- **Easy to learn:** Excel-like operations with concise Python syntax.
- **Fast and efficient:** Built on NumPy; handles larger datasets than spreadsheets.
- **Powerful cleaning tools:** `dropna()`, `fillna()`, `replace()`, `astype()`.
- **Ecosystem integration:** Works with NumPy, Matplotlib/Seaborn, scikit-learn, SQL.
- **Automation & reproducibility:** Code = repeatable steps & version control.

Real-World Uses: Finance

- Analyzing stock prices, returns, and portfolio performance.
- Cleaning and aggregating large sets of transaction or trading data.
- Automating reporting and visualizing trends over time.

Real-World Uses: Biology / Healthcare

- Working with gene expression or clinical datasets.
- Cleaning and normalizing data from lab instruments or sensors.
- Tracking patient records, outcomes, or drug effectiveness.

Real-World Uses: Marketing / Business

- Analyzing customer behavior and sales performance.
- Segmenting audiences using filters and conditions.
- Measuring campaign impact and visualizing conversion rates.

Pandas in the Data Analysis Workflow

Stage	Purpose	Example Pandas Tasks
Import/Load	Bring data into Python	`read_csv()`, `read_excel()`, `read_sql()`
Clean	Handle missing/inconsistent data	`dropna()`, `fillna()`, `rename()`, `astype()`
Explore/Analyze	Inspect, filter, compute stats	`df.describe()`, `df[df["age"]>30]`, `df.corr()`
Visualize	Show patterns/trends	`df.plot()`, `df.hist()`
Report/Export	Save results	`to_csv()`, `to_excel()`, `to_json()`

Excel vs Pandas (Quick Comparison)

- **Excel:** Great for small datasets, manual exploration, quick charts.
- **Pandas:** Scales to larger datasets, automation via code, reproducible pipelines.
- **Together:** Prototype in Excel, productionize and automate with Pandas.

Key Takeaways

- Pandas is the Swiss Army knife for tabular data in Python.
- Real-world impact across finance, healthcare, and marketing.
- Clean $\rightarrow$ Analyze $\rightarrow$ Visualize $\rightarrow$ Report: pandas supports every step.
- Start small, write readable code, and automate repetitive work.

Why Pandas Struggles with Big Data

- **Pandas** is great for small to medium datasets, but has limits:
  - Runs on a single CPU core (no true parallelism)
  - Data must fit into memory (RAM)
  - Large datasets cause slowdowns or crashes
- As data grows, we need **distributed** or **parallel** solutions.

"When your data doesn't fit in memory, it's time to look beyond Pandas."

Alternatives to Pandas

- Several tools extend or replace Pandas for large-scale processing:
  - **Dask** — parallel, chunked DataFrame with Pandas-like syntax.
  - **Terality** — serverless engine that scales Pandas code automatically.
  - **PySpark** — distributed processing via Apache Spark.
- All aim to process big data efficiently while keeping code familiar.

How These Tools Work

Dask

- Breaks data into smaller chunks.
- Executes tasks in parallel.
- Uses task scheduling (DAG).

Terality

- Serverless, no setup needed.
- Auto-scales in the cloud.
- Fully compatible with Pandas API.

PySpark

- Distributed across many machines.
- Uses Spark DataFrames or RDDs.
- Great for very large datasets.

Performance and Tradeoffs

- **Pandas:** Fast for small datasets, simple and intuitive.
- **Dask:** Parallelizes operations on local or cluster hardware.
- **Terality:** Scales automatically, minimal setup.
- **PySpark:** Extremely scalable but requires cluster management.

Each tool balances speed, scalability, and simplicity differently.

When to Choose Each Tool

- **Pandas:** Small or medium data on one machine.
- **Dask:** Larger data on multi-core or small clusters.
- **Terality:** Need to scale automatically with minimal setup.
- **PySpark:** Huge, distributed datasets (enterprise or cloud).

Choose based on scale, complexity, and available infrastructure.

Key Takeaways

- Pandas is powerful, but not infinite — it's best for local analysis.
- Dask, Terality, and PySpark extend Python's reach to Big Data.
- Think about:
    - **Data size**
    - **Hardware availability**
    - **Ease of scaling**
- Select the right tool for your data scale and workflow.

Adiós Pandas? Not Quite — Just Know When to Move On!

Comparison: Pandas vs Dask vs Terality vs PySpark

Feature	Pandas	Dask	Terality	PySpark
Scale	Single machine, fits in RAM	Parallel on one or few machines	Auto-scales in the cloud	Distributed across large clusters
Speed	Fast for small data	Faster on multi-core tasks	Very fast (serverless scaling)	Optimized for very large datasets
Ease of Use	Easiest to learn	Similar API, minor setup	Same API, no setup	Steeper learning curve
Setup / Infra	Local only	Needs local cluster or scheduler	Cloud managed, no setup	Requires Spark cluster setup
Best For	Small / Medium data analysis	Larger datasets on local or small cluster	Cloud-based scaling with minimal effort	Enterprise-level big data processing
Limitations	Not scalable, memory bound	Some overhead, setup needed	Closed-source, cloud dependent	Complex to manage, higher overhead

Each tool balances ease, scalability, and control differently.

The following are some interesting links from the pandas documentation:

- **Styling DataFrames:** https://pandas.pydata.org/pandas-docs/stable/style.html
- **The pandas ecosystem:** https://pandas.pydata.org/pandas-docs/stable/ecosystem.html
- Those with an R and/or SQL background may find it helpful to see how the pandas syntax compares:
  - **Comparison with R / R Libraries:** https://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html
  - **Comparison with SQL:** https://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html
  - **SQL Queries:** https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#sql-queries
  - https://github.com/stefmolin/Hands-On-Data-Analysis-with-Pandas-2nd-edition
  - https://medium.com/sfu-cspmp/adios-pandas-process-big-data-in-a-flash-using-terality-dask-or-pyspark-74e65adeb922

Thank you!

See you next week

e.weychert@uw.edu.pl

UNIWERSYTET WARSZAWSKI | WYDZIAŁ NAUK EKONOMICZNYCH