

Pandas-1: Introduction to Pandas

Motivation

- Must-have for **AI/ML coding rounds** in startups as well as Product based Companies.
 - **Easy to Use toolkit** for both beginners and pro developers.
 - pandas lets you quickly load, clean, analyze, and visualize data like a pro.
 - Whenever data is **Tabular or 2-Dimensional**, always think of Pandas.
 - Knowing pandas is a sought-after skill for roles in **data science, business analysis, research, and beyond**. It makes your resume stand out.
 - Essential for working on **machine learning, data science**, and **deep learning** projects
-

What is Pandas?

pandas is an **open-source Python library** for **data manipulation and analysis**.

It is built on top of **NumPy** and provides powerful tools to work with structured data like tables (rows and columns).

Think of pandas as **Excel for Python** — but faster, more flexible, and much more powerful.

Why Pandas?

Without pandas, you'd need to manually manage Tabular data for data operations — which gets messy and slow.

Pandas:

- Handles large datasets efficiently.
 - Provides easy syntax for **filtering, grouping, joining, reshaping** data.
 - Integrates well with **NumPy, Matplotlib**, and other data-science libraries.
-

Core Data Structures

Pandas provides two main data structures:

1. **Series**

- 1D labeled array (like one column in a spreadsheet)

- Labels are called **index**

```
import pandas as pd
s = pd.Series([10, 20, 30], index=["a", "b", "c"])
print(s)
```

2. DataFrame

- 2D table with rows and columns
- Each column is a **Series**

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

Key Features

- **Data cleaning** → Handle missing values (`fillna` , `dropna`)
- **Data selection & filtering** → Use labels (`loc`) or positions (`iloc`)
- **Aggregation & grouping** → `groupby` , `sum` , `mean`
- **Merging & joining** → Combine multiple datasets (`merge` , `concat`)
- **Reshaping** → Pivot tables, stack/unstack
- **I/O operations** → Read/write CSV, Excel, SQL, JSON, etc.

Basic Example

```
In [3]: import pandas as pd

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# The Iris dataset has no header, so specify column names
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']

# Read CSV without index column
df = pd.read_csv(url, header=None, names=column_names, index_col=False)

print(df.head())
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [5]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   sepal_length    150 non-null    float64
1   sepal_width     150 non-null    float64
2   petal_length    150 non-null    float64
3   petal_width     150 non-null    float64
4   class           150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Pandas vs NumPy - Which one to use and When?

- Dimension > 2: Use NumPy
 - Use NumPy (ndarray) when your 2-D data is essentially a numeric matrix and you need fast, memory-efficient numerical work (linear algebra, broadcasting, heavy elementwise operations).
 - Use pandas (DataFrame) when your 2-D data is a table — i.e., columns with names, mixed dtypes, missing values, need for grouping/joins/time-series/CSV/Excel IO or easy selection by column name.
-

Practical rule of Thumb

Start with pandas for data cleaning and exploration. When doing heavy numeric computations (matrix factorization, linear algebra), convert the numeric columns to a NumPy array.

Will Pandas work when data is larger than memory ?

Since Pandas loads the entire dataset into memory. if data is larger than the main memory Here are practical approaches:

1. **Dask** : A library that extends pandas with parallel computing and chunked processing.
 2. **Polars** : A fast DataFrame library written in Rust, with lazy execution and streaming.
 3. "PySpark" : A distributed big-data engine. PySpark splits the dataset into chunks (partitions) and processes them sequentially or in parallel — only a small chunk is in memory at a time.
-

Happy Learning ! Team DecodeAiML !!