# Unlock Your Data Superpowers: A Beginner's Guide to Pandas

Minor in AI, IIT Ropar
25th Feb, 2025

Welcome, future data wizards! This module is your launchpad into the wonderful world of Pandas, a powerful Python library that will become your best friend when working with data. We'll start with an intuitive explanation of the need for Pandas, then dive into its core components, and finally, explore essential operations to manipulate data like a pro. Let's begin!

## The OTT Platform Problem - Why Pandas?

Imagine you're building your own Online TV (OTT) Platform, a competitor to Netflix or Disney+. What cool new feature would you add? Think about it. Maybe AI-generated storylines based on user preferences? Or perhaps multi-angle viewing with live commentary?

Now, consider this: to make these features *actually* work, you need a LOT of data. You need information about:

- **Each Movie/Show:** Title, Genre, Release Year, Director, Actors, Ratings, Reviews, Specifications, etc.
- **Each User:** Viewing History, Ratings Given, Search Queries, Subscription Type, Devices Used, location, demographic.
- **Platform Performance:** Streaming Quality, Server Load, Ad Revenue, Subscription Numbers.

All this data needs to be organized, cleaned, and analyzed to provide recommendations, fix bugs, and make smart business decisions. Imagine trying to do this with just basic Python lists and dictionaries... It would be a nightmare!

That's where Pandas comes in. Pandas is designed to handle large amounts of *structured data* efficiently. It's like a super-powered spreadsheet for Python, making data manipulation a breeze.

## Understanding Different Types of Data

Before we dive into Pandas, let's take a step back and classify the different types of data we encounter in the real world. Understanding these data types will give you a better appreciation for what Pandas is trying to solve.

1. **Cross-Sectional Data:** This is a snapshot of data at a *single point in time*. Think of it like taking a survey of students in a class to understand how many of them understand the day's lecture. You're collecting data from many individuals (students) at one specific time.

Figure 1: A snapshot of a classroom with students raising their hands to indicate understanding.

2. **Time Series Data:** This is data collected *over a period of time.* Imagine tracking the daily stock prices of a company for the last 20 years. You're collecting data from one entity (the company) over a long period of time.

3. **Panel Data:** This is a *combination of both cross-sectional and time series data.* Think of tracking the performance of 10 different OTT platforms over a year, for a specified time. You have multiple entities (OTT platforms) and data collected over a period of time (a year).

   Pandas was born out of the need to efficiently work with panel data. Operations that were previously slow became significantly faster thanks to the data structures and algorithms implemented in Pandas.

## Introducing Pandas - From Panel Data to DataFrames

You might have wondered where the name "Pandas" comes from. Well, it's derived from "Panel Data"! Initially designed to handle panel data effectively, Pandas has evolved to handle a wide variety of structured data. This library emerged when a company wanted to analyze huge sets of panel data and wanted the processing to be fast. Now, Pandas is used for virtually every type of data processing!

The core of Pandas is the **DataFrame**.

Figure 2: Pandas - The data manipulation tool!

A DataFrame is like a table (rows and columns), where each column can have a different data type (numbers, text, dates, etc.). It's a powerful way to organize and represent data, and it's the primary data structure you'll be working with in Pandas.

But what makes a DataFrame so special? It's the fact that it can hold heterogeneous Data. The different types of data are combined to form one collective organization. Each attribute, whether strings or integers, can exist with the same data frame.

Think back to our OTT platform example. You can create a DataFrame to hold movie data, where each row represents a movie, and the columns represent the movie's title, genre, release year, rating, and so on. The power of the data frame is also that you can frame data together!

## Getting Started with Pandas - Basic DataFrames

Let's create a simple DataFrame to see how it works. Open your Python editor (or a Colab notebook if you're using Google Colab) and type in the following code:

```python
import pandas as pd

# Create a dictionary of data
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 28],
    'City': ['New York', 'London', 'Paris']
}

# Create a DataFrame from the dictionary
df = pd.DataFrame(data)

# Print the DataFrame
print(df)
```

Listing 1: Creating a Basic DataFrame

This code does the following:

(a) `import pandas as pd`: This line imports the Pandas library and gives it the alias `pd`. This is a standard convention, and it means you can now access Pandas functions using `pd.` (e.g., `pd.DataFrame`).

(b) `data = { ... }`: This creates a Python dictionary called `data`. The keys of the dictionary are the column names ('Name', 'Age', 'City'), and the values are lists containing the data for each column.

(c) `df = pd.DataFrame(data)`: This is the magic line! It creates a Pandas DataFrame named `df` from the `data` dictionary. The dictionary's keys become the column names, and the lists become the column data.

(d) `print(df)`: This line prints the DataFrame to your console.

Run the code, and you should see something like this:

```
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
2  Charlie   28     Paris
```

Congratulations! You've created your first Pandas DataFrame. Notice how Pandas automatically assigns row labels (0, 1, 2) to each row.

## Essential DataFrame Operations

Now that you have a DataFrame, let's explore some essential operations you can perform to manipulate and analyze your data. We'll expand upon what was covered before.

(a) **Data Types:**

You can check the data types of each column in your DataFrame using the `.dtypes` attribute. For example:

```python
print(df.dtypes)
```

Listing 2: Checking Data Types

This will output something like:

```
Name      object
Age        int64
City      object
dtype: object
```

Notice that Pandas infers the data type of the 'Name' and 'City' columns as `object` (which is generally used for strings) and the 'Age' column as `int64` (a 64-bit integer). Pandas can also infer floating point numbers and Booleans.

(b) **Accessing Data:**

- **Selecting Columns:** You can select a single column using square brackets:

```python
print(df['Name'])
```

Listing 3: Selecting Single Column

This will print the 'Name' column. You can select multiple columns by passing a list of column names within the square brackets:

```python
print(df[['Name', 'Age']])
```

Listing 4: Selecting Multiple Columns

This will print the 'Name' and 'Age' columns. Notice the double square brackets! The outer brackets are for accessing the DataFrame, and the inner brackets create a list of column names.

- **Selecting Rows Using '.loc[]' and '.iloc[]':** Pandas provides two primary ways to select rows: '.loc[]' (label-based) and '.iloc[]' (integer-based).
  - `.loc[]`: Accesses rows by label (index name) and columns by name.
  - `.iloc[]`: Accesses rows and columns by integer position.

```python
import pandas as pd

data = {
    'Name': ['Aryan', 'Aman', 'Akhil', 'Akash', 'Ayush'],
    'Age': [25, 30, 22, 28, 24],
    'City': ['Pune', 'Blore', 'Hyd', 'Pune', 'Delhi'],
    'Salary': [60000, 60000, 45000, 70000, 55000]
}

df = pd.DataFrame(data)

# Get Akhil's Name and Age using .loc[]
akhil_info = df.loc[2, ['Name', 'Age']]
print(akhil_info)

# Get Aryan's Name and Age using .iloc[]
aryan_info = df.iloc[0, [0, 1]]
print(aryan_info)
```

Listing 5: Selecting Rows and Columns Using loc and iloc

- **Selecting Rows with `.head()` and `.tail()`:** You can select the first few rows using the `.head()` method and last few rows using the `.tail()` method.

```python
print(df.head(2)) # Display the first 2 rows
print(df.tail(1)) # Display the last row
```

Listing 6: Selecting Rows Using head() and tail()

This gives a great high level view about the data.

(c) **Filtering Data:**

One of the most powerful features of Pandas is the ability to filter data based on conditions. For example, let's say you want to find all the people in your DataFrame who are older than 27. You can do this using boolean indexing:

```python
import pandas as pd

data = {
    'Name': ['Aryan', 'Aman', 'Akhil', 'Akash', 'Ayush'],
    'Age': [25, 30, 22, 28, 24],
    'City': ['Pune', 'Blore', 'Hyd', 'Pune', 'Delhi'],
    'Salary': [60000, 60000, 45000, 70000, 55000]
}

df = pd.DataFrame(data)

filtered_df = df[df['Age'] < 25]
print(filtered_df)
```

Listing 7: Filtering Data Using Boolean Indexing

This code does the following:

- `df['Age'] < 25`: This creates a *boolean series* where each element is `True` if the corresponding age is less than 25, and `False` otherwise.
- `df[ ... ]`: This uses the boolean series to select only the rows where the condition is `True`.

You can combine multiple conditions using logical operators (`&` for "and", `|` for "or"). For example, to find all people who are in Pune with salary more than 50000:

```python
import pandas as pd

data = {
    'Name': ['Aryan', 'Aman', 'Akhil', 'Akash', 'Ayush'],
    'Age': [25, 30, 22, 28, 24],
    'City': ['Pune', 'Blore', 'Hyd', 'Pune', 'Delhi'],
    'Salary': [60000, 60000, 45000, 70000, 55000]
}

df = pd.DataFrame(data)

salary =  df[(df['Salary'] > 50000)  & (df['City'] == 'Pune')]
print(salary)
```

Listing 8: Filtering with Multiple Conditions

## Working with Missing Data

Real-world data is messy. It often contains missing values, which can cause problems if you try to perform calculations or analyses. Pandas provides tools to handle these missing values.

(a) **Representing Missing Data:** Pandas uses `NaN` (Not a Number) to represent missing numerical values. The string counterpart is `None`.

(b) **Detecting Missing Values:** You can use the `.isnull()` method to detect missing values in your DataFrame. This method returns a boolean DataFrame where `True` indicates a missing value and `False` indicates a non-missing value.

```python
import pandas as pd

data = {
    'Team':  ['Ind', 'Aus', 'Eng'],
    'Score': [250, None, None],
    'Result': ['W', 'W', 'L']
}

df = pd.DataFrame(data)
print(df)

print("\n", df.isnull())
```

Listing 9: Detecting Missing Values using isnull()

Output:

```
  Team  Score Result
0  Ind  250.0      W
1  Aus    NaN      W
2  Eng    NaN      L

      Team  Score  Result
0  False  False   False
1  False   True   False
2  False   True   False
```

(c) **Handling Missing Values:**

- `dropna()`: This method removes rows or columns containing missing values.

```python
import pandas as pd

data = {
    'Team':  ['Ind', 'Aus', 'Eng'],
    'Score': [250, None, None],
    'Result': ['W', 'W', 'L']
}

df = pd.DataFrame(data)
```

```python

df_dropped = df.dropna()
print("\n", df_dropped, "\n")
```

Listing 10: Removing Missing Values using dropna()

This will drop any row with at least one missing value. Output:

```
   Team  Score Result
0  Ind  250.0      W
```

- `fillna()`: This method fills missing values with a specified value.

```python
import pandas as pd

data = {
    'Team':  ['Ind', 'Aus', 'Eng'],
    'Score': [250, None, None],
    'Result': ['W', 'W', 'L']
}

df = pd.DataFrame(data)

df_filled = df.fillna(0)
print(df_filled)
```

Listing 11: Filling Missing Values using fillna()

Output:

```
   Team  Score Result
0  Ind  250.0      W
1  Aus    0.0      W
2  Eng    0.0      L
```

This gives a new data frame, and all `NaN`s are now `0`. You can also fill missing values with the mean, median, or mode of a column.

This is only a brief introduction, but a strong beginning to understanding Pandas, DataFrames, and essential operations for data manipulation. Think of Pandas as a language that can interpret the requirements of a Data Scientist, and run through large numbers of data to filter, process, analyze, and give data insight to help your company grow.