Python Pandas Cheatsheet

By Ahnaf Chowdhury

Python Pandas Cheatsheet (Beginner-Friendly)

Pandas is a powerful Python library that makes it easier to handle and explore data in rows and columns — much like a spreadsheet or a SQL table.

📦 Importing Pandas

```
import pandas as pd
```

You'll often see pd used as the short name for pandas.

📁 Reading Data

1. Reading a CSV file

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

Loads data from a CSV file into a table (called a DataFrame).

2. Reading Excel files

```
df = pd.read_excel("file.xlsx")
```

Loads data from an Excel file. You may need to install openpyxl.

📄 Exploring the Data

3. View the top rows

```
df.head()
```

Shows the first 5 rows. Helpful to quickly peek at your data.

```
df.head(10)
```

Shows the first 10 rows.

4. View the bottom rows

```
df.tail()
```

Shows the last 5 rows.

5. Get basic info

```
df.info()
```

Displays column names, non-empty counts, and data types.

6. Summary statistics

```
df.describe()
```

Gives quick stats like mean, count, min, and max for number columns.

7. Show the shape

```
df.shape
```

Tells you how many rows and columns the table has.

🔍 Selecting Data

8. Select one column

```
df["column_name"]
```

Returns a single column as a Series (like a list with labels).

9. Select multiple columns

```
df[["name", "age"]]
```

Returns multiple columns in a new DataFrame.

10. Select rows using numbers

```
df.iloc[0]
```

Gives you the first row (by position).

11. Select rows by labels

```
df.loc[0]
```

Gives you the row with label/index 0. Often the same as iloc[0].

🧹 Cleaning and Updating

12. Rename columns

```
df.rename(columns={"old_name": "new_name"}, inplace=True)
```

Changes the column name.

13. Drop columns

```
df.drop("column_name", axis=1, inplace=True)
```

Removes the column. axis=1 means columns, axis=0 means rows.

14. Drop rows with missing data

```
df.dropna(inplace=True)
```

Removes rows that have any empty (NaN) cells.

15. Fill missing values

```
df.fillna(0)
```

Replaces missing (NaN) cells with 0.

🔳 Filtering Data

16. Simple condition

```
df[df["age"] > 30]
```

Gives all rows where age is more than 30.

17. Multiple conditions

```
df[(df["age"] > 30) & (df["city"] == "New York")]
```

Gives rows where both conditions are true. Use & for and, | for or.

🏗️ Sorting and Rearranging

18. Sort by values

```
df.sort_values("age")
```

Sorts the rows by the age column.

```
df.sort_values("age", ascending=False)
```

Sorts by age in reverse order.

19. Reset index

```
df.reset_index(drop=True, inplace=True)
```

Resets row numbers back to 0, 1, 2... after filtering.

➕ Creating and Modifying Columns

20. Create a new column

```
df["new_column"] = df["price"] * df["quantity"]
```

Adds a new column by doing math on other columns.

21. Change values in a column

```
df["age"] = df["age"] + 1
```

Adds 1 to every value in the age column.

📊 Grouping and Aggregating

22. Group and summarize

```
df.groupby("department")["salary"].mean()
```

Groups the data by department and shows the average salary.

23. Count values in a column

```
df["city"].value_counts()
```

Tells you how many times each city appears.

💾 Save or Export Data

24. Save to CSV

```
df.to_csv("new_file.csv", index=False)
```

Saves your data to a new CSV file. index=False removes row numbers.

🔄 Other Handy Functions

25. Rename a specific value

```
df["gender"].replace("M", "Male", inplace=True)
```

Changes 'M' to 'Male' in the gender column.

26. Remove duplicates

```
df.drop_duplicates(inplace=True)
```

Keeps only the first time a row appears.