Pandas: The Ultimate Syntax Guide

Pandas is the tool for data manipulation in Python. To master it, you need to understand its **Mental Models** and **Syntax Patterns**.

1. The Mental Model

Don't just memorize methods. Understand the structure.

The "Dictionary" Intuition

A DataFrame is essentially a **Dictionary of Series (Columns)**.

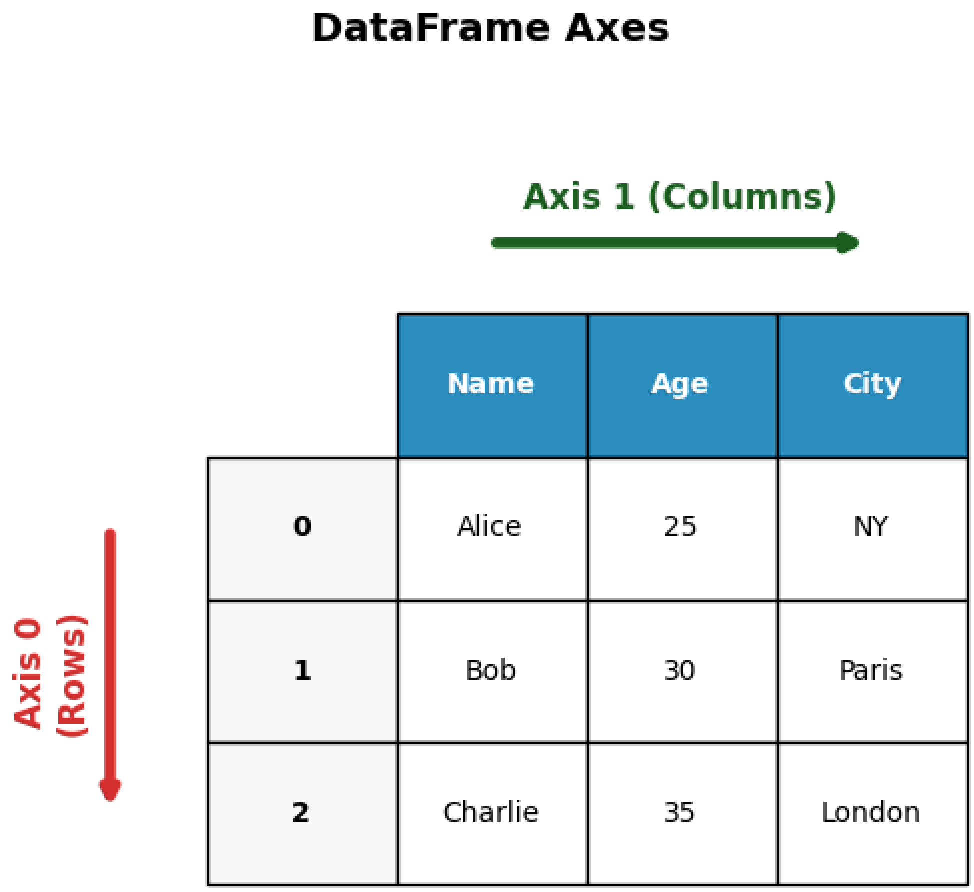
- **Keys** = Column Names
- **Values** = Columns (Series)

```
# Just like a dict!
df['Name']  # Get column 'Name'
df['Age'] = 30  # Set column 'Age'
```

The "Axis" Intuition

This is where most beginners get stuck.

- **Axis 0 (Rows)**: The direction of the index. "Move down".
  - `mean(axis=0)`: Collapse rows (calculate mean for each column).
- **Axis 1 (Columns)**: The direction of the columns. "Move across".
  - `mean(axis=1)`: Collapse columns (calculate mean for each row).



---

2. Universal Syntax Patterns

Memorize these three patterns to handle 90% of tasks.

Pattern 1: Selection (`loc`)

Syntax: `df.loc[row_labels, col_labels]`

Think: "I want **these rows** and **these columns**."

```
# Row 0, Column 'Name'
df.loc[0, 'Name']

# Rows 0 to 5, Columns 'Name' and 'Age'
df.loc[0:5, ['Name', 'Age']]
```

df.loc[0:1, ['Name', 'Age']]

	Name	Age	City
0	Alice	25	NY
1	Bob	30	Paris
2	Charlie	35	London

Pattern 2: Filtering (Boolean Indexing)

Syntax: `df[ condition ]`

Think: "Keep rows where **condition is True**."

```
# Single Condition
df[df['Age'] > 25]

# Multiple Conditions (Parentheses are mandatory!)
df[(df['Age'] > 25) & (df['City'] == 'Paris')]
```

df[df['Age'] > 25]

	Name	Age	City
0	Alice	25	NY
1	Bob	30	Paris
2	Charlie	35	London

Pattern 3: Assignment

Syntax: `df['new_col'] = values`

Think: "Create (or overwrite) this key with these values."

```
df['Is_Adult'] = df['Age'] >= 18
```

---

3. Essential Methods Cheat Sheet

Inspection (Know your data)

Method	Description
`df.head()`	First 5 rows
`df.info()`	Data types & non-null counts
`df.describe()`	Summary stats (mean, min, max)
`df.shape`	(Rows, Columns)
`df.columns`	List of column names

I/O (Read & Write)

Method	Description

Method	Description
`pd.read_csv('file.csv')`	Load CSV
`pd.read_excel('file.xlsx')`	Load Excel
`df.to_csv('file.csv', index=False)`	Save to CSV

Cleaning (Fix your data)

Method	Description
`df.dropna()`	Drop rows with missing values
`df.fillna(value)`	Fill missing values
`df.drop_duplicates()`	Remove duplicate rows

Visualizing Missing Data Handling:

<table>
<tr><th>Original with NaN</th><th>→</th><th>fillna(0)</th></tr>
<tr><td><table><tr><td>1.0</td><td>2.0</td></tr><tr><td>NaN</td><td>4.0</td></tr></table></td><td></td><td><table><tr><td>1.0</td><td>2.0</td></tr><tr><td>0.0</td><td>4.0</td></tr></table></td></tr>
</table>

---

4. The "Split-Apply-Combine" (GroupBy)

Intuition:

1. **Split** the data into groups based on a key (e.g., 'City').
2. **Apply** a function to each group (e.g., 'mean').
3. **Combine** the results into a new table.

Syntax: `df.groupby('group_col')['target_col'].function()`

```
# For each City, what is the average Age?
df.groupby('City')['Age'].mean()
```

Visualizing GroupBy:

<table>
<tr><th>Original Data</th><th>→</th><th>Group By 'City' -> Mean 'Age'</th></tr>
<tr><td><table><tr><th>City</th><th>Age</th></tr><tr><td>NY</td><td>25</td></tr><tr><td>NY</td><td>35</td></tr><tr><td>Paris</td><td>30</td></tr></table></td><td></td><td><table><tr><th>City</th><th>Age</th></tr><tr><td>NY</td><td>30.0</td></tr><tr><td>Paris</td><td>30.0</td></tr></table></td></tr>
</table>

---

5. Merging & Concatenation

Concatenation (Stacking)

Intuition: "Glues" DataFrames together. Usually vertical (adding rows).

Syntax: `pd.concat([df1, df2])`

Visualizing Concatenation:

<table>
<tr><th>df1</th><th>df2</th><th>→</th><th>pd.concat([df1, df2])</th></tr>
<tr>
<td><table><tr><th>A</th><th>B</th></tr><tr><td>1</td><td>5</td></tr><tr><td>2</td><td>6</td></tr></table></td>
<td><table><tr><th>A</th><th>B</th></tr><tr><td>3</td><td>7</td></tr><tr><td>4</td><td>8</td></tr></table></td>
<td></td>
<td><table><tr><th>A</th><th>B</th></tr><tr><td>1</td><td>5</td></tr><tr><td>2</td><td>6</td></tr><tr><td>3</td><td>7</td></tr><tr><td>4</td><td>8</td></tr></table></td>
</tr>
</table>

Merging (Joining)

Intuition: Combines DataFrames based on a **common key** (like SQL).

Syntax: `pd.merge(left, right, on='key', how='inner')`

- **inner**: Keep only matches (Default).
- **left**: Keep all left rows, match right where possible.
- **outer**: Keep everything.

Visualizing Merge Types:

<table>
<tr><th>Left DF</th><th>Right DF</th><th>Inner Join (Match Only)</th><th>Left Join (Keep Left)</th></tr>
<tr>
<td><table><tr><th>key</th><th>val1</th></tr><tr><td>A</td><td>1</td></tr><tr><td>B</td><td>2</td></tr></table></td>
<td><table><tr><th>key</th><th>val2</th></tr><tr><td>B</td><td>3</td></tr><tr><td>C</td><td>4</td></tr></table></td>
<td><table><tr><th>key</th><th>val1</th><th>val2</th></tr><tr><td>B</td><td>2</td><td>3</td></tr></table></td>
<td><table><tr><th>key</th><th>val1</th><th>val2</th></tr><tr><td>A</td><td>1</td><td>NaN</td></tr><tr><td>B</td><td>2</td><td>3</td></tr></table></td>
</tr>
</table>

---

6. Map & Apply

Transform data using custom functions.

Map (Series only)

Intuition: "Substitute values". Used for simple 1-to-1 mapping.

```
# Map 'Male' to 0, 'Female' to 1
df['Gender_Code'] = df['Gender'].map({'Male': 0, 'Female': 1})
```

```python
df['Name_Length'] = df['Name'].map(len)
```

Apply (DataFrame)

Intuition: "Run this function across **Rows** (axis=1) or **Columns** (axis=0)". Power move: Use multiple columns at once.

```python
# Create a summary using Name AND Age (Row-wise)
def summarize(row):
    return f"{row['Name']} is {row['Age']} years old"

df['Summary'] = df.apply(summarize, axis=1)
```