Mastering Pandas: From Basics to Expert Level

Powerful Data Analysis & Manipulation with Python

pandas

Dated: 24 July 2025

Github: bahadurCorvit

Instructor : Fawad Bahadur

Training Objectives & Outcomes

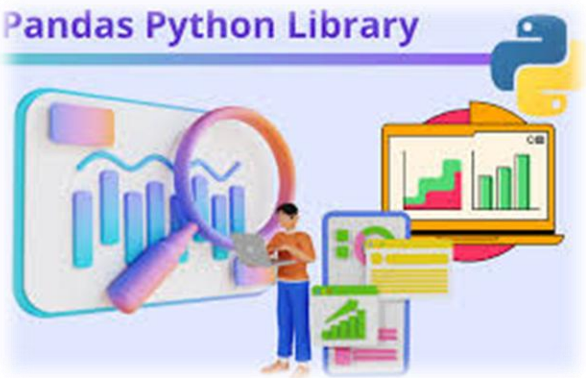
1. Understand the fundamentals of Pandas Series and DataFrames
2. Perform efficient data cleaning, filtering, and transformation
3. Master advanced techniques like grouping, merging, reshaping, and time series handling
4. Apply Pandas to real-world data analysis, reporting, and machine learning workflows

Instructor : Fawad Bahadur

Introduction to Pandas

What is Pandas?

- Open-source Python library for **data manipulation and analysis.**
- Provides powerful data structures: Series and DataFrame.
- Built on top of NumPy, designed for working with structured data.



Why Use Pandas?

- Easy handling of missing data.
- Flexible reshaping and pivoting.
- Powerful group-by functionality.
- Supports time series data.

Example:

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

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Instructor : Fawad Bahadur

Setting Up Pandas in Your Environment

What is Pandas?

- Open-source Python library for data analysis and manipulation
- Built on top of NumPy

Prerequisites-

- Python installed (preferably 3.6 or above)
- pip (Python package installer)

Installation Using pip:

```
pip install pandas
```

Verify Installation:

```
import pandas as pd
print(pd._version_)
```

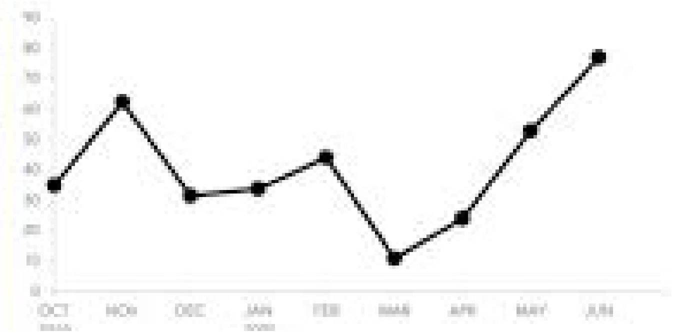
Instructor : Fawad Bahadur

pandas

Pandas Series

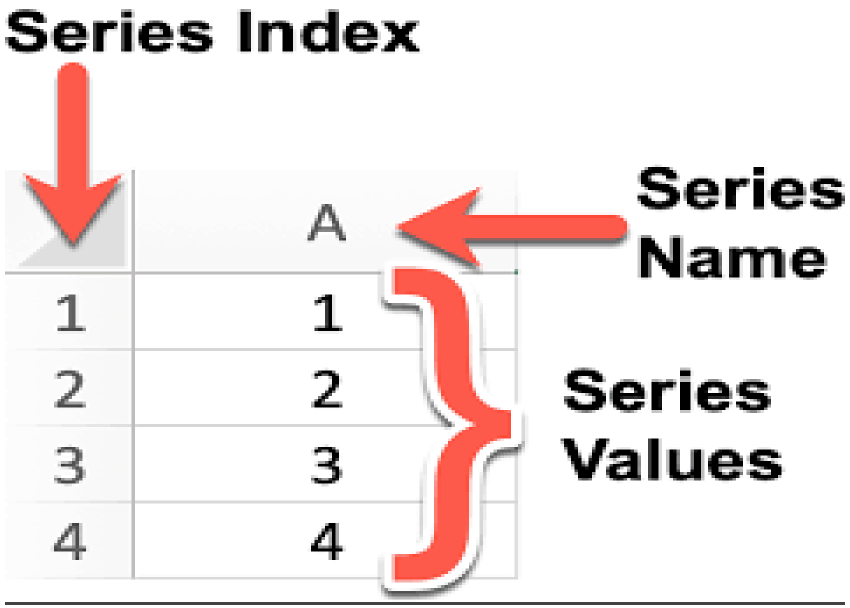
What is a Series?

- One-dimensional labeled array.
- Can hold any data type (integers, strings, floats, etc.).
- Labels (index) identify each element.



Creating a Series

```
import pandas as pd
 s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
print(s)
```



Accessing Data

```
print(s['b'])    # 20
print(s[1])      # 20 (by position)
```

Series Attributes

Values: returns the data as NumPy array
Index: returns the index labels

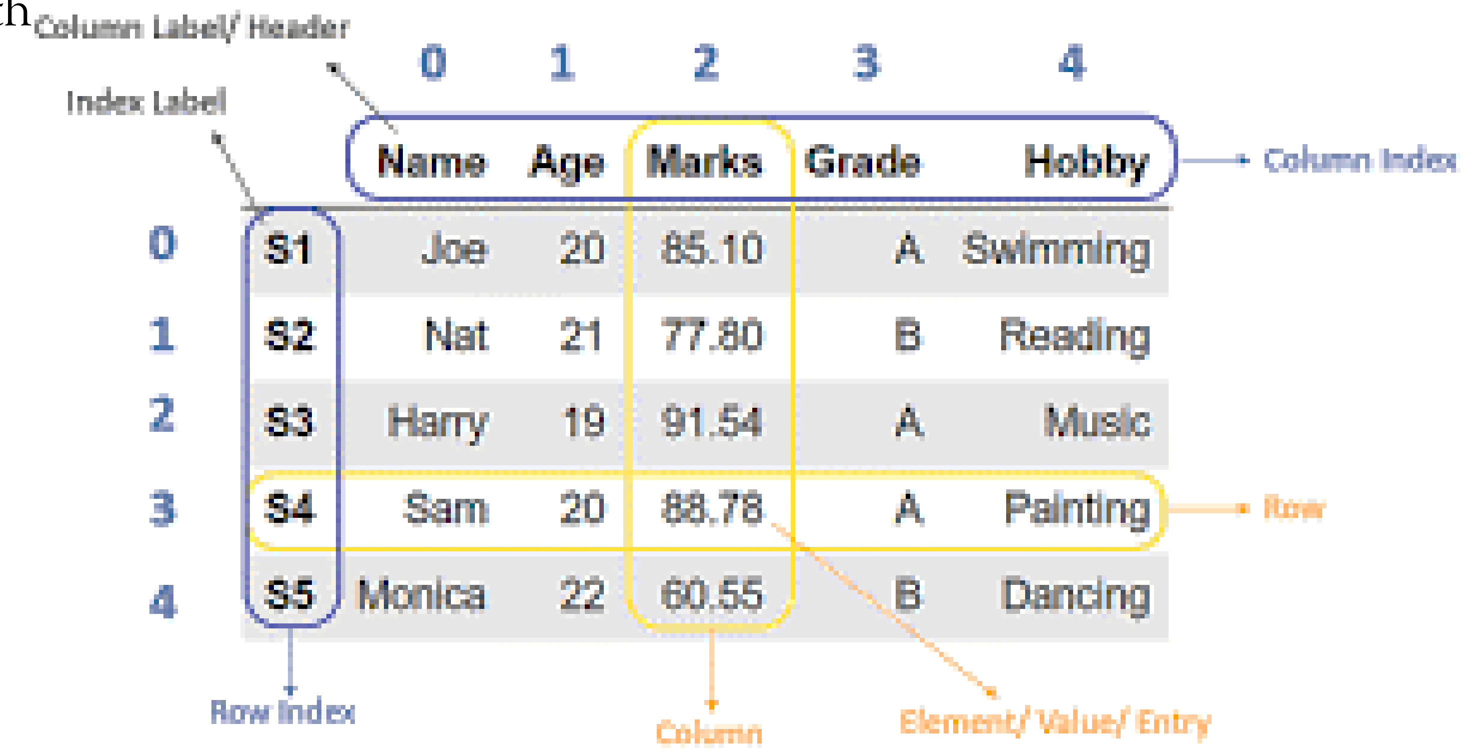
Tip
Series is the building block for DataFrame columns.

Instructor : Fawad Bahadur

Pandas DataFrame Basics

What is a DataFrame?

- Two-dimensional labeled data structure with columns of potentially different types.
- Like a spreadsheet or SQL table.



Creating a DataFrame

```
import pandas as pd
 data = {
'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35],
    'Salary': [50000, 60000, 70000]
}
df = pd.DataFrame(data)
print(df)
```

Accessing Columns

```
print(df['Name'])  # Returns Series of names
```

Accessing Rows By

index label: `df.loc[0]`

By integer position: `df.iloc[0]`

Tip

- DataFrame is the core structure for data analysis in Pandas.

Instructor : Fawad Bahadur

Data Selection & Indexing?

Selecting Columns

```
df['Age']          # Single column (Series)
df[['Name', 'Age']] # Multiple columns (DataFrame)
```

Aditional Selection

- df[df['Age'] > 25] # Rows where Age > 25

Selecting Rows

Using .loc (label-based)

```
df.loc[0]          # First row by index label
df.loc[0:2]        # Rows 0 to 2 inclusive
```

Using .iloc (integer position-based)

```
df.iloc[0]         # First row by position
df.iloc[0:2]       # Rows 0 and 1
```

Setting Values

```
df.loc[1, 'Salary'] = 65000
```

Tip

Use .loc for label indexing and .iloc for positional indexing.

Handling Missing Data?

Detecting Missing Data

```
df.isnull()          # Returns DataFrame of True/False for missing
df.isnull().sum()    # Count missing values per column
```

Changing Data Types

```
df['Age'] = df['Age'].astype(float)
```

Removing Duplicates

```
df.drop_duplicates(inplace=True)
df.drop_duplicates(subset=['ID', 'Email'], inplace=True)
```

Tip

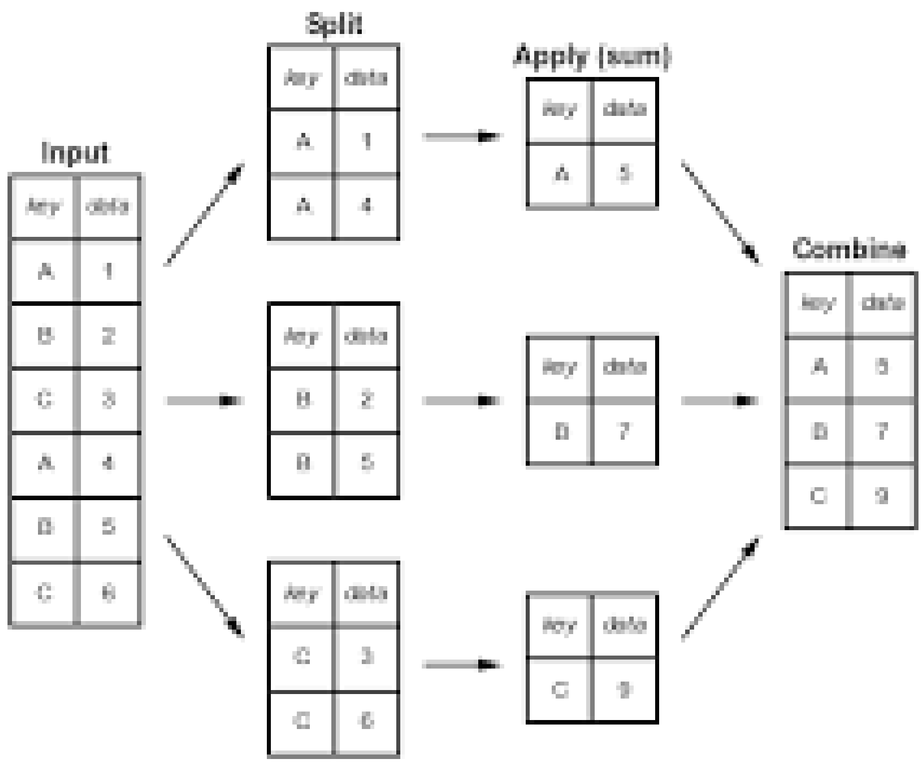
Clean and transform your data to prepare it for analysis.

Instructor : Fawad Bahadur

Data Aggregation & Grouping

GroupBy Basics

- Split data into groups based on column values.
- Apply aggregation functions.



Common Aggregation Functions

- .sum()
- .mean()
- .count()
- .min()
- .max()

Example

```
grouped = df.groupby('AgeGroup')
print(grouped['Income'].mean())
```

Multiple Aggregations

```
grouped['Income'].agg(['mean', 'max', 'min'])
```

Tip

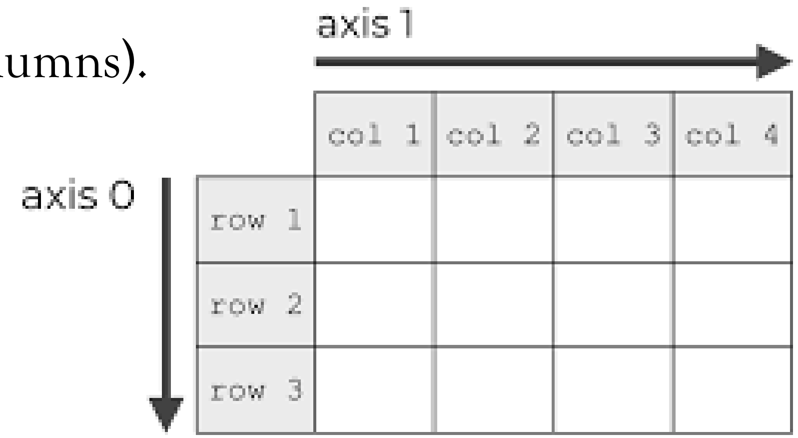
GroupBy is powerful for summarizing data by categories.

Instructor : Fawad Bahadur

Concatenation in Pandas

Purpose
Used to stack DataFrames or Series vertically or horizontally along an axis.

Functionality
- Combines DataFrames by appending them along a specified axis (rows or columns).
- Does not perform any relational operations or key-based joining.
- Can handle DataFrames with different columns or indices.
- Can result in duplicate indices if not handled properly.



Use Case
When you want to combine DataFrames with similar structures or append new data to existing DataFrames without considering common keys.

Code
```
pd.concat([df1, df2], axis=0)  # Vertical (rows)
pd.concat([df1, df2], axis=1)  # Horizontal (columns)
```

Merge in Pandas

Purpose

Used to combine DataFrames based on common columns or indices, similar to SQL joins.

Join Type	Syntax	Returns
Inner	how='inner'	Only matching rows
Left	how='left'	All left rows + matching right
Right	how='right'	All right rows + matching left
Outer	how='outer'	All rows from both

Functionality

- Merges two DataFrames based on specified columns or indices.
- Supports different types of joins: inner, outer, left, right.
- Can handle complex merging scenarios with multiple keys.
- Can result in duplicate columns if not handled properly.

Use Case

When you want to combine DataFrames based on related columns, similar to joining tables in a database.

Code

```
pd.merge(df1, df2, on='key', how='inner')  # Inner join
pd.merge(df1, df2, on='key', how='left')   # Left join
```

Tip

Use merging and joining to combine related datasets effectively.

Instructor : Fawad Bahadur

Join in Pandas

Purpose

A specialized version of merge() optimized for joining DataFrames based on their indices or key columns.

Join Type	Syntax	Returns
Inner	how='inner'	Only matching rows
Left	how='left'	All left rows + matching right
Right	how='right'	All right rows + matching left
Outer	how='outer'	All rows from both

Functionality

- Combines two DataFrames based on indices or specified key columns.
- By default, performs a left join, but other types can be specified
- Can be more efficient than merge() when joining on indices.

Use Case

When you want to combine DataFrames based on their indices or specific key columns, similar to joining tables in a database but with a focus on index-based joins.

Code

```
df1.join(df2, how='outer')
```

Tip

Use merging and joining to combine related datasets effectively.

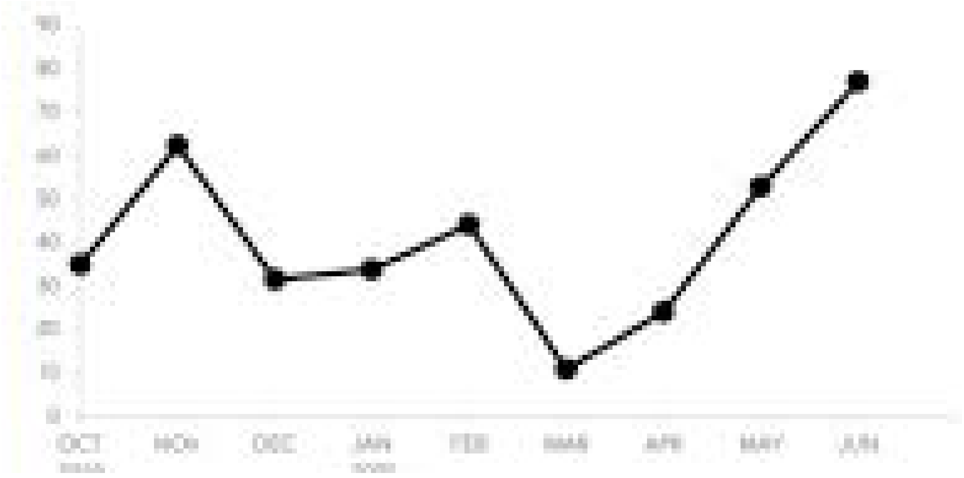
Instructor : Fawad Bahadur

Working with Time Series Data

DateTime Index

Use pd.to_datetime() to convert columns to datetime.

```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
```



Resampling

Aggregate time series data at different frequencies.

```
df.resample('M').mean()  # Monthly average
```

Time-based Selection

```
df.loc['2023-01-01':'2023-01-31']
```

Shifting Data

```
df['PrevDay'] = df['Value'].shift(1)
```

Tip

Pandas makes time series manipulation intuitive and powerful.

Instructor : Fawad Bahadur

Handling Text Data in Pandas

String Methods

- Use .str accessor for vectorized string operations.

```
df['Name'].str.lower()        # Convert to lowercase
df['Name'].str.contains('a') # Check if 'a' in string
```

Extracting Substrings

```
df['Initial'] = df['Name'].str[0]
```

Splitting Strings

```
df['Name'].str.split(' ')
```

Replacing Text

```
df['Name'] = df['Name'].str.replace('Alice', 'Alicia')
```

Tip

Text processing in Pandas is efficient and easy with .str methods.

Instructor : Fawad Bahadur

Reading & Writing Data with Pandas

Reading Data

CSV files

```
df = pd.read_csv('data.csv’)
```

Excel files

```
df = pd.read_excel('data.xlsx', sheet_name='Sheet1’)
```

JSON files

```
df = pd.read_json('data.json’)
```

Writing Data

To CSV

```
df.to_csv('output.csv', index=False)
```

To Excel

```
df.to_excel('output.xlsx', index=False)
```

Tip

Pandas supports many file formats for easy data import/export.

Instructor : Fawad Bahadur

Optimizing Pandas Performance

Use Vectorized Operations

Avoid loops
use built-in Pandas/NumPy functions.

Use Categorical Data Type

Reduce memory and speed up operations on repeated values.

Downcast Numeric Types

df['Age'] = pd.to_numeric(df['Age'], downcast='integer')

Use .query() and .eval()

Faster filtering and evaluation.
df.query('Age > 30 & Income > 50000')

Avoid Chained Indexing

Use .loc instead for safe and efficient assignment.

Tip
Efficient coding drastically improves speed on large data..

Instructor : Fawad Bahadur

Pandas Integration with Other Libraries

NumPy

Pandas is built on NumPy arrays, enabling fast numerical operations.

```
import numpy as np
arr = np.array([1, 2, 3])
df = pd.DataFrame(arr)
```

Matplotlib & Seaborn

For data visualization.

```
df['Age'].plot(kind='hist’)
import seaborn as sns
sns.boxplot(data=df, x='Age')
```

Scikit-learn

For machine learning pipelines.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df.drop('target', axis=1), df['target'])
```

SQLAlchemy

Read/write data from SQL databases.

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///:memory:’)
name', engine)
```

Tip

Pandas works seamlessly with the Python data ecosystem.

Instructor : Fawad Bahadur

Data Visualization Using Pandas

Basic Plotting

```
df['Age'].plot(kind='hist', bins=20, title='Age Distribution')
```

Line Plot

```
df.plot(x='Date', y='Sales', kind='line')
```

Bar Plot

```
df['Category'].value_counts().plot(kind='bar')
```

Scatter Plot

```
df.plot.scatter(x='Age', y='Income')
```

Using matplotlib and seaborn for Enhanced Visualization

```
sns.boxplot(x='Category', y='Income', data=df)
```

Tip

Pandas integrates well with visualization libraries for exploratory analysis.

Instructor : Fawad Bahadur

THANK YOU