Pandas Data Handling & Operations

To work with tabular data using the pandas library — one of the most important libraries in Python for data analysis.

1. Installation and Importing

Before working with any external Python library, we must install and import it.

➤pip install pandas

This command installs the pandas library from the Python Package Index (PyPI). It's a one-time installation per environment.

➤ import pandas as pd

Imports the installed pandas library and gives it the alias pd, which is a common convention. Using pd instead of typing pandas repeatedly makes the code shorter and cleaner.

1.What is a Series?

A **Series** is a **one-dimensional labeled array** capable of holding any data type (integers, strings, floats, etc.). It has:

- **Only one column**
- An **index** (like row labels)

Key Points:

- Acts like a **column** from a spreadsheet or database.
- Created from list, array, dictionary, scalar value, etc.

```
import pandas as pd

data = [10, 20, 30, 40, 50]

s = pd.Series(data)

print("Series:")

print(s)

print("Type:", type(s))
```

2. Creating and Saving DataFrame to CSV

➤ Creating a DataFrame

A **dictionary dis** is defined with keys "a", "b", and "c" and list values.

```
dis = {"a":[1,23,4,5,6,7], "b":[8,4,3,2,11,3], "c":[78,44,22,11,4,8]}
```

d = pd.DataFrame(dis)

d

Explanation:

- A **DataFrame** is like an Excel sheet or a SQL table — rows and columns of data.
- The keys of the dictionary become column headers.
- Each list is a column's data.

➤ **Saving DataFrame to CSV**

d.to_csv("batch530.csv", header=[10,20,30], index=False)

Explanation:

- Saves the d DataFrame as a .csv file.
- header=[10, 20, 30] replaces the actual column names with these numbers.
- index=False prevents pandas from adding a default 0,1,2... row index.

Feature	Series	DataFrame
Dimension	1D (One-dimensional)	2D (Two-dimensional)
Structure	Like a single column	Like a table with rows & columns
Data Type	Homogeneous (mostly)	Heterogeneous (mixed column types)
Index	Single axis (only row index)	Two axes (row index & column names)
Use Case	For a single data column	For datasets with multiple fields

3. Reading CSV and Excel Files

➤ **Reading a CSV File**

b = pd.read_csv("book.csv")

Loads a file named book.csv into DataFrame b.

➤ **Reading an Excel File**

c = pd.read_excel("dist.xlsx")

Loads the dist.xlsx Excel file into a DataFrame named c.

➤ **Reading Limited Rows**

s = pd.read_csv("batch530.csv", nrows=3)

Reads only the **first 3 rows** of the CSV.

➤ **Reading Specific Columns**

s1 = pd.read_csv("batch530.csv", usecols=["20"])

Reads only the column named "20" from the CSV file.

4. Exporting Data to Other Formats

➤ **Export to HTML**

c.to_html("batch530.html")

Converts DataFrame c to a webpage format (.html file).

➤ **Export to JSON**

c.to_json("batch530.json")

Saves the data in JSON format, useful for APIs and JavaScript-based systems.

➤ **Export to XML**

c.to_xml("batch5330.xml")

Stores data in .xml format, commonly used in data transfer.

5. Basic Data Exploration

➤ **.info()**

c.info()

Displays:

- Number of rows and columns
- Column names
- Data types
- Memory usage
- Number of non-null values

➤ **.describe()**

c.describe()

Gives **summary statistics** for numerical columns:

- Count
- Mean
- Standard deviation

- Min, Max
- Percentiles (25%, 50%, 75%)

➤ **.head() and .tail()**

c.head(3) # First 3 rows

c.tail(2) # Last 2 rows

Used to **preview** data quickly.

6. Handling Missing Data

➤ **Detecting Missing Data**

c.isnull()

Returns True for every missing (NaN) value.

c.isnull().sum()

Returns the count of missing values in each column.

➤ **Filling Missing Values**

c.fillna(2)

Replaces all NaN values with 2.

➤ **Dropping Missing Data**

c.dropna()

Removes rows containing any missing values.

➤ **Dropping Duplicates**

c.drop_duplicates()

Removes repeated/duplicate rows.

7. Advanced Data Selection and Filtering

Here we create a new dataset data related to car brands.

```
data = pd.DataFrame({
   'Brand': ['Maruti', 'Hyundai', 'Tata', 'Mahindra', 'Maruti', 'Hyundai', 'Renault', 'Tata', 'Maruti'],
   'Year': [2012, 2014, 2011, 2015, 2012, 2016, 2014, 2018, 2019],
   'Kms Driven': [50000, 30000, 60000, 25000, 10000, 46000, 31000, 15000, 12000],
   'City': ['Gurgaon', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Chennai',  'Ghaziabad'],
```

```
    'Mileage': [28, 27, 25, 26, 28, 29, 24, 21, 24]
})
```

Using loc[] (Label-based indexing)

```
data.loc[(data.Brand=="Maruti") & (data.Mileage>25)]
```

- Filters rows where Brand is **Maruti** AND Mileage is **greater than 25**.

```
data.loc[2:5]
```

- Returns rows from **index 2 to 5** (inclusive).

```
data.loc[(data.Year<2015), ['Mileage']] = 22
```

- Changes Mileage to 22 wherever Year is less than 2015.

Using iloc[] (Integer-based indexing)

```
data.iloc[[0,2,4,7]]
```

- Selects **specific rows** using integer positions.

```
data.iloc[1:5, 2:5]
```

- Selects rows from index 1 to 4 (exclusive of 5) and columns 2 to 4.

Operation	**Function**
Create DataFrame	pd.DataFrame()
Read CSV/Excel	pd.read_csv(), pd.read_excel()
Save as File	.to_csv(), .to_html(), etc.
Preview Rows	.head(), .tail()
Summary & Info	.describe(), .info()
Null Handling	.isnull(), .fillna(), .dropna()
Duplicates	.drop_duplicates()
Filter Rows	.loc[], .iloc[]

Working with Kaggle Dataset:

Practice Questions

1. Creating and Saving a DataFrame

1. Create a dictionary with student names and their scores in three subjects. Convert it into a DataFrame and save it as students.csv without the index.
2. Create a DataFrame with columns "Roll No", "Name", and "Marks" for 5 students. Save it to CSV with custom headers [101, 102, 103].

2. Reading Files

3. Read only the first 3 rows of a CSV file named data.csv.
4. Read only the "Name" and "Marks" columns from a CSV file named students.csv.
5. Read an Excel file named records.xlsx and print its first 5 rows.

3. Exporting Data

6. Convert an Excel file data.xlsx into:
    - HTML file (data.html)
    - JSON file (data.json)
    - XML file (data.xml)

4. Exploring Data

7. Load a CSV file and display:
    - First 3 rows
    - Last 2 rows
    - Summary statistics
    - Data types of each column

5. Handling Missing Data

8. Add 2 missing (NaN) values to a DataFrame manually. Then:
    - Show total missing values per column
    - Replace all missing values with 0
    - Drop all rows with missing values
9. Create a DataFrame with duplicate rows. Remove all duplicates using pandas.

6. Data Filtering with loc and iloc

10. Create a DataFrame with columns: "Brand", "Year", "Mileage", "City". Then:

- Filter all rows where Brand == "Maruti" and Mileage > 25 using loc
- Change Mileage = 22 where Year < 2015 using loc

11. Using iloc, select:

- The 1st, 4th, and 6th rows
- The 2nd to 5th rows and 2nd to 4th columns