PANDAS

Intro & Basics of Pandas

– What is Pandas?

- Python library for **data manipulation & analysis**
- Works best with **tabular data** (rows & columns)
- Built on **NumPy**, integrates with **Matplotlib, Scikit-learn**
- Think of Pandas like an **Excel inside Python** but more powerful

💡 *Example:* Excel file → read into Pandas → clean → analyze → visualize.

Why Pandas?

- Easy handling of CSV/Excel/SQL data
- Rich built-in functions → no need to write loops
- Handles **missing values**, **duplicates**, **grouping**, **sorting**
- Industry use: finance, HR analytics, healthcare, marketing, ML

Getting Started

```
pip install pandas  # install library
import pandas as pd  #import library
```

- By convention, imported as pd.
- Works on top of NumPy (import numpy as np).

Pandas Data Structures

- **Series** → 1D labeled array (like a column in Excel).
- **DataFrame** → 2D table with rows & columns.

🧠 *Analogy:*

- Series → one column in Excel
- DataFrame → the full Excel sheet

Pandas Series

```
import pandas as pd
import numpy as np

data = np.array([10, 20, 30, 40])
ser = pd.Series(data)
print(ser)
```

👉 Explanation:

- Left side → index (like row numbers in Excel)
- Right side → values

📌 Output:

```
0  10
1  20
2  30
3  40
dtype: int64
```

Creating Series (Different Ways)

```
# From list
s1 = pd.Series([1, 2, 3, 4])

# From dictionary
s2 = pd.Series({'a':10, 'b':20, 'c':30})

# With custom index
s3 = pd.Series([100, 200, 300], index=['x','y','z'])
```

🧠 Which way looks most familiar? (Dict → key-value, List → simple Excel column)

Pandas DataFrame

```
data = {
   'Name': ['Alice', 'Bob', 'Charlie'],
   'Age': [25, 30, 35],
   'City': ['Delhi', 'Mumbai', 'Pune']
}
df = pd.DataFrame(data)
print(df)
```

📌 Output:

```
   Name  Age   City
0   Alice   25  Delhi
1     Bob   30  Mumbai
2 Charlie   35   Pune
```

👉 Rows (0,1,2) = Index | Columns = Name, Age, City

Indexing & Selecting Data

```
# Select one column
df['Age']


# Select multiple columns
df[['Name','City']]


# Select row by index
df.loc[0]    # by label
df.iloc[0]   # by position
```

👉 Doubt-clearing: loc vs iloc

- loc → label (row name)
- iloc → index number

Reading CSV/Excel

```
df = pd.read_csv("students.csv")
df.head()   # First 5 rows

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

💡 Tip: Always check df.head() and df.info() when loading new data.

Basic Operations

```
df['Age'].mean()     # average age
df['Age'].max()      # max age
df['Age'].min()      # min age
df['City'].unique()  # unique cities
```

Pandas Data Cleaning

Why Data Cleaning?

- Real-world data = *messy*
- Missing values (NaN), wrong formats, duplicates
- Pandas helps clean data efficiently

Checking for Missing Values

```
import pandas as pd
import numpy as np

data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David'],
   'Age': [25, None, 30, None],
   'City': ['Delhi', 'Mumbai', np.nan, 'Pune']
}
df = pd.DataFrame(data)
print(df)
```

📌 **Output:**

```
  Name   Age   City
0   Alice  25.0  Delhi
1     Bob   NaN Mumbai
2 Charlie  30.0    NaN
3   David   NaN   Pune
```

Detecting NaN

```
df.isnull()       # Boolean table (True if NaN)
df.isnull().sum()  # Count missing values in each column
df.notnull()      # Opposite of isnull
```

👉 Common trick: always check .isnull().sum() before analysis.

Dropping Missing Values

```
df.dropna()           # Drop rows with NaN
df.dropna(axis=1)       # Drop columns with NaN
df.dropna(thresh=2)     # Keep rows with at least 2 non-NaN values
```

🧠 Question: *"What will happen if Age column has only NaN?" → drop entire column with axis=1.*

Filling Missing Values

```
df.fillna(0)              # Replace NaN with 0
df['Age'].fillna(df['Age'].mean(), inplace=True)  # Replace with mean
df['City'].fillna('Unknown')  # Replace with custom value
```

👉 Good practice: numeric → mean/median, categorical → mode.

Forward & Backward Fill

```
df.fillna(method='ffill')   # Forward fill (previous value)
df.fillna(method='bfill')   # Backward fill (next value)
```

```
  Name   Age   City
0   Alice  25.0  Delhi
```

1 Bob 25.0 Mumbai

2 Charlie 30.0 Mumbai ← ffill copied from above

Replace Method

df['City'].replace(np.nan, 'Bangalore', inplace=True)

df['Age'].replace({25: "Twenty Five", 30: "Thirty"})

💡 Replace is useful when cleaning categorical text data.

🔄 **Data Transformations (apply, map, lambda)**

Using apply()

df['Age_plus_5'] = df['Age'].apply(lambda x: x+5)

- apply() → runs a function on each element of a Series or row/column of a DataFrame.

Using map()

df['Name_upper'] = df['Name'].map(str.upper)

- map() works element-wise (good for Series).
- Lighter than apply().

Using lambda Functions

df['Age_Category'] = df['Age'].apply(lambda x: 'Adult' if x>=18 else 'Child')

👉 Quick way to apply conditions.

🔹 **What is lambda in Python?**

- A **lambda function** is a *small anonymous function*.
- "Anonymous" = it doesn't need a name like def my_function().
- It's often used when you need a function **only once**.

✅ **Example 1: Normal Function vs Lambda**

```
# Normal function
def add5(x):
    return x + 5
```

```
print(add5(10))   # Output: 15


# Lambda function (one-line)
add5_lambda = lambda x: x + 5
print(add5_lambda(10))   # Output: 15
```

👉 Both do the same thing! But lambda is shorter.

🔍 **Sorting, Filtering & Boolean Masking**

Sorting

```
df.sort_values(by='Age')            # Sort ascending
df.sort_values(by='Age', ascending=False)  # Descending
df.sort_values(by=['City','Age'])    # Multiple columns
```

Filtering Rows

```
df[df['Age'] > 25]          # Age greater than 25
df[df['City'] == 'Delhi']    # City is Delhi
```

Boolean Masking

```
mask = (df['Age'] > 20) & (df['City'] == 'Delhi')
df[mask]
```

📌 Output = only rows where both conditions are True.

Quick Recap

- dropna() → remove missing
- fillna() → replace missing
- apply(), map(), lambda → transformations
- sort_values(), filtering & masking → selecting data

👉 Exercise:

1. Replace missing Age with mean.
2. Create new column Age*2.
3. Show all people from Mumbai older than 25.

Lab exercise/tasks:

1. Load data in pandas dataframe
2. Show first 5 rows.
3. Find the number of rows and columns.
4. Show data types of each column.
5. Get summary statistics of numeric columns.
6. What is the average passenger age?
7. **Data Exploration and Preparation (Learn about data)**

- Are there missing data?
- Is it categorical? if not, min , max, avg values? if yes, what are the categories?
- distribution of variables
- Duplicate entry

8. Handle missing values

a. Count missing values in each column.

b. Replace missing Age with the mean age.

c. Fill missing Embarked with "Unknown".

9. Filtering Data

- Show all passengers older than 30.
- Show all **female passengers** who survived (Survived = 1).

10. Indexing & Selecting

- Select only the Name and Age columns.
- Show the first 10 names of passengers in Pclass = 1.

11. Sorting

- Sort passengers by Age (ascending).
- Sort by Fare (descending) and show top 5.

12. Unique & Counts

- How many unique values are in the Sex column?
- Count how many males and females are onboard.

13. Apply & Lambda

- Create a new column "AgeGroup":
    - Child (<18), Adult (18–50), Senior (50+).

- Use apply with a lambda function.

14. New Column

- Create a new column "FamilySize" = SibSp + Parch + 1.
- Show top 5 passengers with the largest family size.

15. Boolean Masking

- Show passengers who paid more than 50 fare **and** survived.
- Show all children (Age < 18) who did not survive.

16. NumPy Operations

- Convert the Age column into a NumPy array and calculate:
    - Mean, Median, Standard Deviation.
- Find max fare using NumPy.

Pandas Advanced

1. GroupBy Operations

👉 groupby() lets you **split → apply → combine** data.

🔹 **Common use-cases:**

- Summarizing data by categories
- Aggregating with mean, sum, count, etc.
- Applying multiple functions at once

✅ **Examples**

```
import pandas as pd


# Sample dataset
data = {'Department': ['HR','HR','IT','IT','Finance','Finance'],
        'Employee': ['A','B','C','D','E','F'],
        'Salary': [40000, 42000, 50000, 52000, 60000, 62000],
        'Bonus': [2000, 2500, 3000, 3500, 4000, 4500]}
df = pd.DataFrame(data)
print(df)
```

1. Group by department → get mean salary

```
df.groupby('Department')['Salary'].mean()
```

2. Count employees in each department

```
df.groupby('Department')['Employee'].count()
```

3. Multiple aggregations

```
df.groupby('Department').agg({'Salary':['mean','max'],
                              'Bonus':'sum'})
```

4. Group by multiple columns

```
df.groupby(['Department','Employee'])['Salary'].sum()
```

5. Custom function with groupby

df.groupby('Department')['Salary'].apply(lambda x: x.max()-x.min())

2. Merging, Joining, Concatenating

👉 In real datasets, data often comes from **multiple tables**. Pandas helps combine them.

✅ **Examples**

```
students = pd.DataFrame({
   'ID':[1,2,3],
   'Name':['Alice','Bob','Charlie']
})

scores = pd.DataFrame({
   'ID':[1,2,4],
   'Math':[85,90,95]
})
```

1. Merge on common column (ID)

pd.merge(students, scores, on='ID', how='inner')

2. Left join (keep all students, even if no score)

pd.merge(students, scores, on='ID', how='left')

3. Right join (keep all scores, even if student missing)

pd.merge(students, scores, on='ID', how='right')

4. Outer join (union of both)

pd.

3. Pivot Tables in Pandas

👉 A **pivot table** reshapes data like Excel Pivot Tables.

✅ **Examples**

```
sales = pd.DataFrame({
   'Region':['North','South','East','West','North','South'],
   'Product':['A','A','B','B','A','B'],
   'Sales':[100,150,200,250,300,350],
   'Profit':[10,20,30,40,50,60]
})
merge(students, scores, on='ID', how='outer')
```

1. Simple pivot: Region vs Product

```
pd.pivot_table(sales, values='Sales', index='Region', columns='Product')
```

2. Aggregation function (mean, sum)

```
pd.pivot_table(sales, values='Sales', index='Region', aggfunc='sum')
```

3. Multiple values

```
pd.pivot_table(sales, values=['Sales','Profit'], index='Region', aggfunc='mean')
```

4. Pivot with multiple indexes

```
pd.pivot_table(sales, values='Sales', index=['Region','Product'], aggfunc='sum')
```

5. Fill missing values

```
pd.pivot_table(sales, values='Sales', index='Region', columns='Product', fill_value=0)
```

Mini Project – Student Gradebook

👉 Goal: Build a **gradebook** using Pandas.

Dataset

```
grades = pd.DataFrame({
   'Student':['Alice','Bob','Charlie','Alice','Bob','Charlie'],
   'Subject':['Math','Math','Math','Science','Science','Science'],
   'Score':[85,90,78,88,76,95]})
```

Steps:

1. Average score per student
2. Subject-wise average
3. Pivot table (students × subjects)
4. Add grade (A/B/C)
5. Merge with student info

1. Average score per student

```
grades.groupby('Student')['Score'].mean()
```

2. Subject-wise average

```
grades.groupby('Subject')['Score'].mean()
```

3. Pivot table (students × subjects)

```
pd.pivot_table(grades, values='Score', index='Student', columns='Subject')
```

4. Add grade (A/B/C)

```
grades['Grade'] = grades['Score'].apply(lambda x: 'A' if x>=85 else 'B' if x>=75 else 'C')
```

5. Merge with student info

```
student_info = pd.DataFrame({'Student':['Alice','Bob','Charlie'],
                             'Age':[20,21,22]})

pd.merge(grades, student_info, on='Student')
```