Pandas

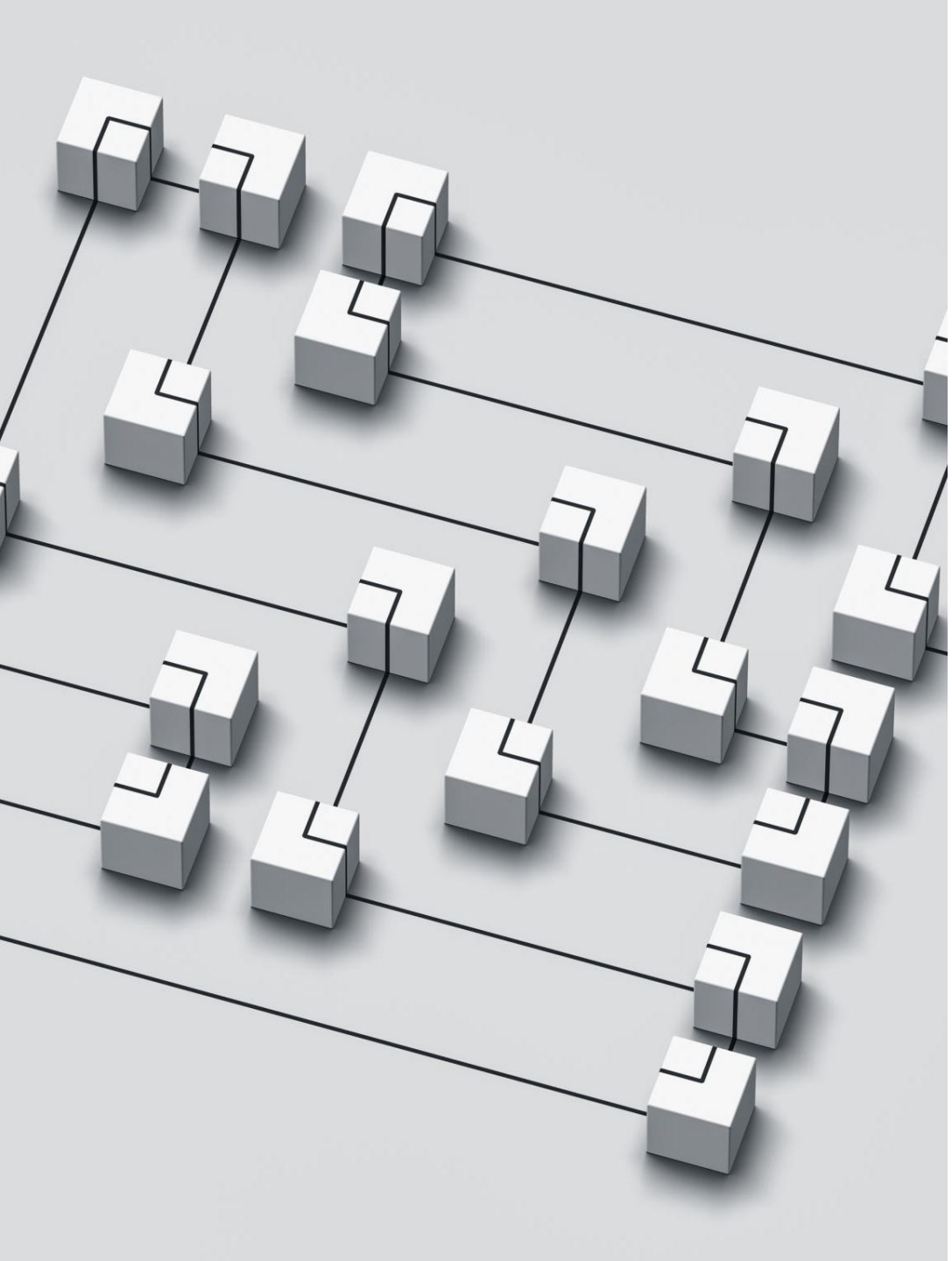
- The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language

Installing pandas

- pip install pandas

- %pip install pandas

- import pandas as pd
- print(pd.__version__)



Pandas Data Structures

- Series (1D labeled array)
- DataFrame (2D table with rows and columns)

Pandas series

A one-dimensional labeled array capable of holding any data type

```
s = pd.Series([3, -5, 7, 4], index=[ ‘a’, ’b’ , ‘c’, ’d’ ])
```

question

- Tracking Monthly Sales
- sales = pd.Series([5000, 7000, 8000, 6500, 7200], index=['Jan', 'Feb', 'Mar', 'Apr', 'May’])
- Find February month sale
- Find the average sale
- Days sale above 6000
- Day with the highest sales
- New Series with 10% increase in sales

Dataframe

- A two-dimensional labelled data structure with columns of potentially different types

- data = {
- 'Name': ['Alice', 'Bob', 'Charlie', 'David'],
- 'Age': [25, 30, 35, 40],
- 'Salary': [50000, 60000, 70000, 80000],
- 'Department': ['HR', 'IT', 'IT', 'Finance']
- }

- df = pd.DataFrame(data)

Manipulate: Add column

- import pandas as pd

- # Mock data: Coffee shop customers
- data = {
-     'Name': ['Alice', 'Bob', 'Charlie'],
-     'Age': [25, 30, 35],
-     'Spend': [45.0, 30.0, 50.0]
- }

- df['Loyalty'] = df['Spend'] > 40 # Boolean
- print("\nUpdated:\n", df)

Reading and Writing Files

- Loading Sales Data from CSV
- df = pd.read_csv('sales.csv’)
- print(df.head()) # Display first 5 rows

Writing csv

- df.to_csv('new_sales.csv', index=False)

Reading file in Numpy

- With no missing values
- Use **numpy.loadtxt**

- With missing values
- Use **numpy.genfromtxt.**

Pandas Indexing:

- .loc
- .loc (label-based)
- Row/column names (labels)
- df.loc[row_label, col_label]

iloc

- .iloc (position-based)

- Row/column integer positions
- df.iloc[row_pos, col_pos]

Sample data

- import pandas as pd

- data = {
- 'Customer': ['Alice', 'Bob', 'Charlie', 'Diana'],
- 'Drink' : ['Latte', 'Espresso', 'Cappuccino', 'Americano'],
- 'Price' : [4.50, 3.00, 5.00, 2.50],
- 'Qty' : [2, 3, 1, 4]
- }
- df = pd.DataFrame(data, index=['A1', 'A2', 'A3', 'A4'])

- 1. From csv take Single row(Drink) by label
- 2. Multiple rows from Drink to Price

.loc

- # 1. Single row by label
- df.loc['A2’]

- # 2. Multiple rows (inclusive)

- df.loc['A1':'A3’]

- # 3. Specific cells
- df.loc['A1', 'Price'] # → 4.50
- df.loc[['A1','A4'], ['Drink','Qty']]
- # 4. Boolean mask with labels
- df.loc[df['Price'] > 3.5, 'Customer']

iloc

- # 1. First 2 rows (positions 0,1)
- df.iloc[:2]
- # 2. Row 1 to 3 (exclusive end)
- df.iloc[1:4]
- # 3. 2nd & 4th rows, 1st & 3rd columns
- df.iloc[[1,3], [0,2]]
- # 4. Last column only
- df.iloc[:, -1]

dropping

- s.drop([‘jan’]) #Drop values from rows

- Df.drop(“name”) #Drop values from columns(axis=1)

Data Cleaning

- Handling Missing Data

- df = pd.DataFrame({
- 'Customer': ['A', 'B', 'C', 'D'],
- 'Age': [25, None, 35, None],
- 'Purchase': [500, 1000, None, 700]
- })
- df.fillna(df.select_dtypes(include=['number']).mean(), inplace=True)
- # Fill missing values with column mean
- print(df)

Data Filtering

- Find Customers Who Spent More Than 800

- high_spenders = df[df['Purchase'] > 800]
- print(high_spenders)

select elements from an array based on multiple conditions

- np.where()
- np.where(condition, x, y)
- x: The value (or array) to use where the condition is True
- y: The value (or array) to use where the condition is False

Solve Day 1

- test_RecallPandasDataFrame
- **test_WorkwithNumPy2DArray**

Convert Pandas DataFrame to NumPy Array

- Why convert?
  Use NumPy for fast math, machine learning input, or plotting.

- .to_numpy()

- df.to_numpy()

function converts Series to DataFrame

.to_frame()

- Series.**to_frame()**

Solve

- Practice question 1 and 3
- test_UnderstandStructuredData1/2
- test_SliceandModify2DArrays

Grouping and Aggregation

- Grouping and aggregation allow us to summarize data based on certain categories.
- This is useful for reports, performance analysis, and trend identification.

```
# Creating a sample employee dataset
df = pd.DataFrame({
    'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve',
'Frank'],
    'Department': ['HR', 'IT', 'IT', 'c', 'HR', 'Finance'],
    'Salary': [50000, 70000, 80000, 60000, 52000, 65000]
})

# Average salary per department
avg_salary = df.groupby('Department')['Salary'].mean()
print(avg_salary)

```

More Aggregations

- Finding Total Salary per Department
- Finding Number of Employees per Department

Task

- df = pd.DataFrame({
- 'City': ['New York', 'Los Angeles', 'Chicago', 'New York', 'Chicago', 'Los Angeles'],
- 'Sales': [200, 300, 250, 400, 350, 500],
- 'Profit': [20, 40, 35, 50, 45, 60]
- })

- **Write code to calculate:**

1. Total sales per city.
2. Average profit per city.
3. Number of transactions per city.

Merging and Joining in Pandas

- Merging allows you to combine multiple datasets based on a common column.

example

- A company wants to **combine employee details with performance ratings**.
- df1 = pd.DataFrame({
- 'ID': [1, 2, 3, 4],
- 'Name': ['Alice', 'Bob', 'Charlie', 'David'],
- 'Department': ['HR', 'IT', 'IT', 'Finance']
- })

- df2 = pd.DataFrame({
- 'ID': [1, 2, 3, 4],
- 'Performance': ['A', 'B', 'A', 'C'],
- 'Bonus': [1000, 500, 1200, 300]
- })

- merged_df = pd.merge(df1, df2, on='ID')
- print(merged_df)

Types of Joins

- **Inner Join (Default)**
- Only keeps matching rows.

- pd.merge(df1, df2, on='ID', how='inner')

Types of Joins

- Left Join (Keep all from Left)

- pd.merge(df1, df2, on='ID', how='left')

Types of Joins

- Right join (keeps all from right)

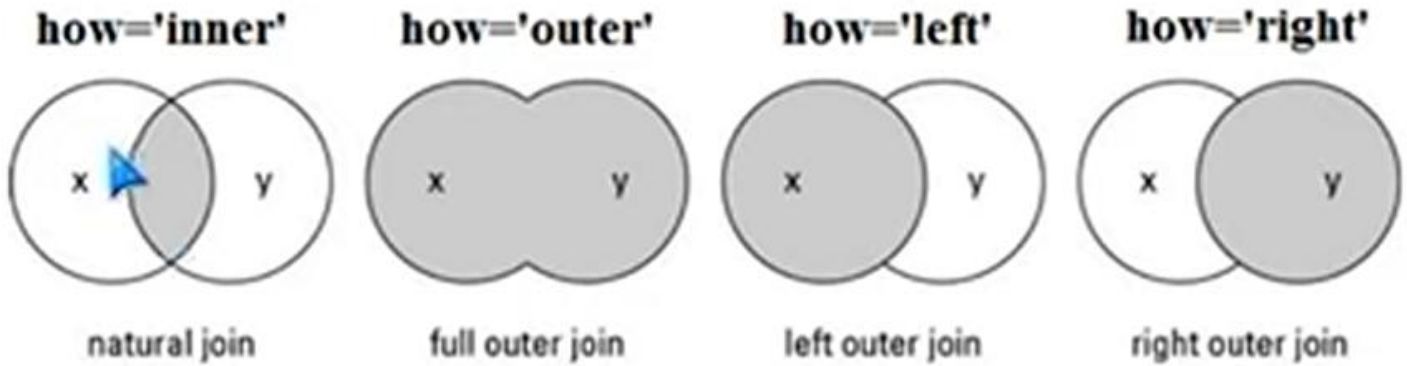
- pd.merge(df1, df2, on='ID', how='left')

Types of Joins

- Outer join (keeps everything)

- pd.merge(df1, df2, on='ID', how=outer')

Merge, Join, and Concatenate



Merge, Join, and Concatenate

Method	Use When
`pd.concat()`	Stack vertically/horizontally
`df.merge()`	SQL-like join on columns
`df.join()`	Join on index

Stacking Rows

- df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})
- df2 = pd.DataFrame({'A': ['A2', 'A3'], 'B': ['B2', 'B3']})

- # 1. Stacking Rows (axis=0 is default)
- row_concat = pd.concat([df1, df2])

- row_concat_fixed = pd.concat([df1, df2], ignore_index=True)

Stacking Columns (axis=1)

- df3 = pd.DataFrame({'C': ['C0', 'C1'], 'D': ['D0', 'D1']})
- col_concat = pd.concat([df1, df3], axis=1)
- print(f"Column concat (axis=1):\n{col_concat}")

df.join()

- Join columns with *other* DataFrame either on index or on a key column.
- Efficiently join multiple DataFrame objects by index at once by passing a list

- >>> df = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3', 'K4', 'K5'],
- ... 'A': ['A0', 'A1', 'A2', 'A3', 'A4', 'A5']})

- >>> other = pd.DataFrame({'key': ['K0', 'K1', 'K2'],
- ... 'B': ['B0', 'B1', 'B2']})

- df.join(other, lsuffix='_caller', rsuffix='_other')

Hands-On Code

```
# List of orders
df_orders = pd.DataFrame({
    'order_id': [101, 102, 103, 104],
    'customer_id': [1, 2, 1, 3]
})
```

```
# List of customers
df_customers = pd.DataFrame({
    'customer_id': [1, 2, 3, 4],
    'name': ['Alice', 'Bob', 'Charlie', 'David']
})
```

Task 1:Merge df_orders (left) with df_customers (right) to get the name for each order_id.(Hint: The key is customer_id. Use a left join).

Task 2 (Bonus):Which customer placed no orders?(Hint: Use how='outer' and look for NaNs).

Task

- products = pd.DataFrame({
- 'Product_ID': [101, 102, 103, 104],
- 'Product_Name': ['Laptop', 'Phone', 'Tablet', 'Monitor']
- })

- sales = pd.DataFrame({
- 'Product_ID': [101, 103, 104, 105],
- 'Units_Sold': [10, 5, 8, 3]
- })
- Perform an inner join to find which products were sold.
- Perform a left join to keep all products, even if they have no sales.
- Perform an outer join to keep all products and sales, even if they don't match.