Python Lab Manual: Pandas

Objectives

- To understand and work with Pandas library in Python.
- To learn about Series and DataFrame, the two main Pandas data structures.
- To perform data manipulation including selecting, filtering, adding, and removing data.
- To learn basic data analysis using Pandas functions.
- To practice reading/writing data from/to CSV and Excel files.

1. Introduction to Pandas

- Pandas is a Python library for data analysis and manipulation.
- Pandas allows easy loading, cleaning, and manipulation of large datasets in Python.
- Enables handling of missing values, duplicates, and inconsistent data, which is crucial before AI/ML modeling.
- Can read and write CSV, Excel, SQL, JSON, and more, making it versatile for real-world datasets.
- It is built on top of NumPy.
- Provides Series (1D labeled array) and DataFrame (2D labeled table).
- Think of Pandas like Excel inside Python, where you can manipulate rows and columns programmatically.

2. Pandas Data Structures

Pandas supports two types of data structures.

1. Series : 1-dimensional labeled array

2. DataFrame : 2-dimensional labeled data structure like a spreadsheet

In [ ]:
```
!pip install pandas
```

2.1 Series

A Series is a 1-dimensional labeled array, similar to a list but with an index.

In [14]:
```
import pandas as pd
```

In [2]:
```
# Create a Series from a list
s = pd.Series([10, 20, 30, 40, 50])
print(s)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

- The numbers on the left (0,1,2,...) are indices.
- You can provide custom indices:

In [6]:
```
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(s)
```

```
a    10
b    20
c    30
dtype: int64
```

2.2 DataFrame

A **DataFrame** is a 2-dimensional labeled data structure, like a spreadsheet with rows and columlink textmns.

In [8]:
```
# Create a DataFrame from a dictionary
data = {
    'Name': ['Ali', 'Sara', 'John', "Hanan"],
    'Age': [23, 21, 25, 30],
    'Score': [85, 90, 88, 90]
}

df = pd.DataFrame(data)
```

In [9]:
```
data
```

Out[9]:
```
{'Name': ['Ali', 'Sara', 'John', 'Hanan'],
 'Age': [23, 21, 25, 30],
 'Score': [85, 90, 88, 90]}
```

In [11]:
```
df
```

Out[11]:

	Name	Age	Score
0	Ali	23	85
1	Sara	21	90
2	John	25	88
3	Hanan	30	90

In [12]:
```
print(df)
```

```
    Name  Age  Score
0    Ali   23     85
1   Sara   21     90
2   John   25     88
3  Hanan   30     90
```

- Columns: 'Name', 'Age', 'Score'
- Rows: 0,1,2 (index)

3. Basic DataFrame Operations

3.1 Selecting Columns

In [15]:
```
df["Name"]
```

Out[15]:
```
0      Ali
1     Sara
2     John
3    Hanan
Name: Name, dtype: object
```

In [16]:
```
print(df['Name'])   # Single column (Series)
```

```
0      Ali
1     Sara
2     John
3    Hanan
Name: Name, dtype: object
```

In [17]:
```
print(df[['Name', 'Score']])  # Multiple columns (DataFrame)
```

```
    Name  Score
0    Ali     85
1   Sara     90
2   John     88
3  Hanan     90
```

3.2 Selecting Rows

In [18]:
```
print(df.iloc[2])    # Row by position
```

```
Name     John
Age        25
Score      88
Name: 2, dtype: object
```

In [19]:
```
print(df.loc[0])     # Row by index
```

```
Name     Ali
Age       23
Score     85
Name: 0, dtype: object
```

3.3 Filtering Data

In [20]:
```
# All rows where Age > 22
print(df[df['Age'] > 22])
```

```
    Name  Age  Score
0    Ali   23     85
2   John   25     88
3  Hanan   30     90
```

3.4 Adding a New Column

In [22]:
```
df['MyColumn']= 'Ok'
```

In [23]:
```
df
```

Out[23]:

	Name	Age	Score	MyColumn
0	Ali	23	85	Ok
1	Sara	21	90	Ok
2	John	25	88	Ok

	Name	Age	Score	MyColumn
3	Hanan	30	90	Ok

In [24]:
```
df['Passed'] = df['Score'] > 88
print(df)
```

```
    Name  Age  Score MyColumn  Passed
0    Ali   23     85       Ok   False
1   Sara   21     90       Ok    True
2   John   25     88       Ok   False
3  Hanan   30     90       Ok    True
```

3.5 Removing Columns or Rows

In Pandas, you often need to delete unnecessary data from a DataFrame. This can be done using the .drop() method.

Drop a column

df.drop('column_name', axis=1, inplace=False)

Drop a row

df.drop(row_index, axis=0, inplace=False)

Parameters explained:

- 'column_name' → Name of the column to remove
- row_index → Index (or label) of the row to remove
- axis → Axis along which to drop:
- axis=0 → rows
- axis=1 → columns
- inplace → If True, changes the original DataFrame; if False (default), returns a new DataFrame

In [25]:
```
df
```

Out[25]:

	Name	Age	Score	MyColumn	Passed
0	Ali	23	85	Ok	False
1	Sara	21	90	Ok	True
2	John	25	88	Ok	False
3	Hanan	30	90	Ok	True

In [26]:
```
# Drop column 'Passed'
df = df.drop('MyColumn', axis=1)
df
```

Out[26]:

	Name	Age	Score	Passed
0	Ali	23	85	False
1	Sara	21	90	True
2	John	25	88	False
3	Hanan	30	90	True

In [31]:
```
# Drop row with index 1
df = df.drop(3)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-31-b999c700c089> in <module>
      1 # Drop row with index 1
----> 2 df = df.drop(3)

~\anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis, index, columns, level, inplace, errors)
   4306                 weight  1.0     0.8
   4307         """
-> 4308         return super().drop(
   4309             labels=labels,
   4310             axis=axis,

~\anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis, index, columns, level, inplace, errors)
   4151         for axis, labels in axes.items():
   4152             if labels is not None:
-> 4153                 obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4154
   4155         if inplace:

~\anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels, axis, level, errors)
   4186                 new_axis = axis.drop(labels, level=level, errors=errors)
   4187             else:
-> 4188                 new_axis = axis.drop(labels, errors=errors)
   4189             result = self.reindex(**{axis_name: new_axis})
   4190

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, errors)
   5589         if mask.any():
   5590             if errors != "ignore":
-> 5591                 raise KeyError(f"{labels[mask]} not found in axis")
   5592             indexer = indexer[~mask]
```

```
5593         return self.delete(indexer)
KeyError: '[3] not found in axis'
```

In [32]:
```
df
```

Out[32]:

	Name	Age	Score	Passed
0	Ali	23	85	False
1	Sara	21	90	True

3.6 Basic Statistics

In [33]:
```
print("Sum: ", df['Score'].sum())  # Sum score
print("Max Value: ", df['Score'].max())   # Maximum score
```

```
Sum:  175
Max Value:  90
```

In [34]:
```
print(df.describe())       # Summary statistics for numeric columns
```

```
             Age      Score
count   2.000000   2.000000
mean   22.000000  87.500000
std     1.414214   3.535534
min    21.000000  85.000000
25%    21.500000  86.250000
50%    22.000000  87.500000
75%    22.500000  88.750000
max    23.000000  90.000000
```

4. Reading and Writing Data

Pandas allows you to import data from files and save your processed data easily.

Reading Data

- CSV files: Use pd.read_csv('file.csv') to load data into a DataFrame.
- Excel files: Use pd.read_excel('file.xlsx') to read Excel spreadsheets.
- Preview data: Use df.head() to see the first few rows and df.tail() for the last rows.

Writing Data

- Save as CSV: Use df.to_csv('output.csv', index=False) to write the DataFrame to a CSV file.
- Save as Excel: Use df.to_excel('output.xlsx', index=False) to export to Excel.

Summary: Pandas makes it easy to load data from different file formats, manipulate it, and save it back for analysis or sharing.

Tip: Always use index=False when you don't want the row numbers to be saved as a separate column.

4.2 Write CSV

In [35]:
```
df
```

Out[35]:

	Name	Age	Score	Passed
0	Ali	23	85	False
1	Sara	21	90	True

In [36]:
```
df.to_csv('output.csv', index=False)
```

4.4 Write Excel

In [37]:
```
df.to_excel('output.xlsx')
```

4.1 Read CSV

In [ ]:
```
df = pd.read_csv('output.csv')
```

In [39]:
```
print(df.head(2))  # Show first 5 rows
```

```
   Name  Age  Score  Passed
0   Ali   23     85   False
1  Sara   21     90    True
```

4.3 Read Excel

In [40]:
```
df = pd.read_excel('output.xlsx')
```

In [41]:
```
df
```

Out[41]:

	Unnamed: 0	Name	Age	Score	Passed
0	0	Ali	23	85	False

	Unnamed: 0	Name	Age	Score	Passed
1	1	Sara	21	90	True

In [42]:
```
print(df.tail(2))  # Show first 5 rows
```

```
   Unnamed: 0  Name  Age  Score  Passed
0           0   Ali   23     85   False
1           1  Sara   21     90    True
```

5. Exercises

1. Create a Series of your 5 favorite numbers and assign custom indices.
2. Create a DataFrame of 4 students with Name, Age, and Marks.
3. Select only the 'Name' column from the DataFrame.
4. Filter students who scored more than 80.
5. Add a column 'Passed' which is True if Marks > 50.
6. Drop the column 'Passed' from the DataFrame.
7. Find the average, minimum, and maximum Marks using Pandas.
8. Save your DataFrame to a CSV file and read it back.
9. Create a DataFrame from a dictionary of lists, then select specific rows and columns using .iloc and .loc.
10. Calculate summary statistics for numeric columns using .describe().
11. Read and Inspect Data

    11.1 Load the students.csv file into a Pandas DataFrame.

    11.2 Display the first 5 rows.

    11.3 Display the last 5 rows.

    11.4 Check the shape of the DataFrame, column names, and data types.
12. Select Columns

    12.1 Select only the Name column.

    12.2 Select the columns Name and Score together.

1. Select Rows

    13.1 Select the first row using iloc.

    13.2 Select the row with index 3 using loc.

1. Filter Data

    14.1 Find all students who scored more than 80.

    14.2 Find all students who passed (Passed == True).

1. Add a New Column

    15.1 Add a column called Grade based on Score:

    ```
    Score ≥ 85 → 'A'
    70 ≤ Score < 85 → 'B'
    Score < 70 → 'C'
    ```

1. Remove Columns or Rows

    16.1 Remove the Passed column.

    16.2 Remove the row with index 2.

1. Sort Data

    17.1 Sort the students by Score in descending order.

    17.2 Sort the students by Age in ascending order.

1. Aggregation and Statistics

    18.1 Calculate the mean, maximum, and minimum of the Score column.

    18.2 Calculate the average age of the students.

    18.3 Display a summary of numeric columns using describe().

1. Save Modified Data

19.1 Save the modified DataFrame to a new CSV file called students_modified.csv.

In [ ]: