Python Pandas – I

Class 12 CBSE | Sections 1.2 to 1.6 | Study Notes with Examples

1.2 Using Pandas

Pandas is an open-source Python library used for **data manipulation and analysis**. It provides powerful, easy-to-use data structures and data analysis tools.

Installing Pandas

Open Command Prompt / Terminal and type:

```
pip install pandas
```

Importing Pandas

Before using Pandas in a program, it must be imported. The conventional alias is **pd**:

```
import pandas as pd
```

■ **Note:** Using 'pd' as alias is a standard convention followed worldwide.

1.3 Pandas Data Structures

Pandas provides two primary data structures:

Data Structure	Dimensions	Description
Series	1-D	A one-dimensional labeled array (like a column in Excel)
DataFrame	2-D	A two-dimensional labeled table (like a full Excel sheet)

1.4 Series Data Structure

A **Series** is a one-dimensional array-like object that can hold any data type (integers, strings, floats, etc.). Every element has an associated **label called index**.

1.4.1 Creating Series Objects

Syntax:

```
pandas.Series(data, index=index_values, dtype=dtype, name=name)
```

Example 1 – Series from a List:

```
import pandas as pd

marks = [85, 90, 78, 92]
s = pd.Series(marks)
print(s)

# Output:
```

```
# 0 85
# 1 90
# 2 78
# 3 92
# dtype: int64
```

■ **Note:** When no index is given, Pandas automatically assigns 0, 1, 2, … as default index.

Example 2 – Series with Custom Index:

```
import pandas as pd

marks = [85, 90, 78, 92]
s = pd.Series(marks, index=['Aman', 'Bina', 'Chetan', 'Dia'])
print(s)

# Output:
# Aman 85
# Bina 90
# Chetan 78
# Dia 92
# dtype: int64
```

Example 3 – Series from a Dictionary:

```
import pandas as pd

d = {'Maths': 95, 'Science': 88, 'English': 76}
s = pd.Series(d)
print(s)

# Output:
# Maths 95
# Science 88
# English 76
# dtype: int64
```

■ **Note:** When a dictionary is used, keys become the index labels automatically.

1.4.3 Series Object Attributes

Commonly used attributes of a Series:

Attribute	Description	Example
`s.index`	Returns the index labels	`Index(['Aman','Bina',...])`
`s.values`	Returns the data as NumPy array	`array([85, 90, 78, 92])`
`s.dtype`	Data type of elements	`int64`
`s.size`	Total number of elements	`4`
`s.shape`	Tuple representing dimensions	`(4,)`
`s.name`	Name of the Series	`None (unless set)`

`s.ndim`	Number of dimensions	`1`

1.5 Accessing a Series Object and its Elements

1.5.1 Accessing Individual Elements

Elements can be accessed using **index label** or **position number**:

```
import pandas as pd

s = pd.Series([85, 90, 78, 92], index=['Aman','Bina','Chetan','Dia'])

# By label (index name)
print(s['Bina']) # Output: 90

# By position (like a list)
print(s[1]) # Output: 90

# Multiple elements
print(s[['Aman', 'Dia']])
# Output:
# Aman 85
# Dia 92
# dtype: int64
```

1.5.2 Extracting Slices from Series Object

Slicing works similarly to Python lists using **start:stop:step** notation:

```
import pandas as pd

s = pd.Series([10, 20, 30, 40, 50], index=['a','b','c','d','e'])

print(s[1:4]) # Position-based slice (stop is excluded)
# Output:
# b 20
# c 30
# d 40

print(s['b':'d']) # Label-based slice (stop is INCLUDED)
# Output:
# b 20
# c 30
# d 40
```

■ **Note:** In label-based slicing, the stop label IS included. In position-based slicing, the stop index is NOT included.

1.6 Operations on Series Object

1.6.1 Modifying Elements of Series Object

```
import pandas as pd

s = pd.Series([85, 90, 78], index=['Aman','Bina','Chetan'])

# Modify an existing element
s['Bina'] = 95
print(s)
# Aman 85
# Bina 95 ← updated
# Chetan 78

# Add a new element
s['Dia'] = 88
print(s)
# Aman 85
# Bina 95
# Chetan 78
# Dia 88 ← new entry
```

1.6.2 Renaming Indexes

```
import pandas as pd

s = pd.Series([85, 90, 78], index=['A','B','C'])

# Rename using rename()
s = s.rename(index={'A':'Aman', 'B':'Bina', 'C':'Chetan'})
print(s)
# Output:
# Aman 85
# Bina 90
# Chetan 78
```

1.6.3 The head() and tail() Functions

head(n) – returns first n rows | **tail(n)** – returns last n rows (default n = 5)

```
import pandas as pd

s = pd.Series([10, 20, 30, 40, 50, 60, 70])

print(s.head(3)) # First 3 elements
# 0 10
# 1 20
# 2 30

print(s.tail(3)) # Last 3 elements
# 4 50
# 5 60
# 6 70
```

1.6.4 Vector Operations on Series Objects

Operations are applied **element-wise** (no need for loops):

```
import pandas as pd

s = pd.Series([10, 20, 30, 40])

print(s + 5) # Add 5 to each: [15, 25, 35, 45]
print(s * 2) # Multiply each: [20, 40, 60, 80]
print(s > 15) # Compare each: [False, True, True, True]
```

1.6.5 Arithmetic on Series Objects

When two Series are added, values with **matching index labels** are added together. Non-matching indices get **NaN** (Not a Number):

```
import pandas as pd

s1 = pd.Series([10, 20, 30], index=['a','b','c'])
s2 = pd.Series([1, 2, 3], index=['a','b','d'])

print(s1 + s2)
# Output:
# a 11.0 ← 10 + 1 (matched)
# b 22.0 ← 20 + 2 (matched)
# c NaN ← 30 + ? (no match in s2)
# d NaN ← ? + 3 (no match in s1)
```

■ **Note:** NaN stands for 'Not a Number'. It appears when an operation cannot be performed due to a missing matching index.

1.6.6 Filtering Entries in Series Objects

Use **Boolean conditions** to filter elements:

```
import pandas as pd

marks = pd.Series([85, 45, 90, 38, 72], index=['A','B','C','D','E'])

# Filter students who passed (marks >= 50)
passed = marks[marks >= 50]
print(passed)
# Output:
# A 85
# C 90
# E 72
```

1.6.7 Sorting Series Values

Two main methods for sorting:

```
import pandas as pd

s = pd.Series([30, 10, 50, 20], index=['d','b','a','c'])

# Sort by VALUES
```

```
print(s.sort_values())
# b 10
# c 20
# d 30
# a 50

# Sort by INDEX labels
print(s.sort_index())
# a 50
# b 10
# c 20
# d 30

# Sort in descending order
print(s.sort_values(ascending=False))
# a 50
# d 30
# c 20
# b 10
```

■ **Note:** sort_values() sorts by data values. sort_index() sorts by index labels. Use ascending=False for descending order.

■ Quick Revision – Important Points

Topic	Key Point
Import Pandas	`import pandas as pd`
Create Series	`pd.Series(data, index=[...])`
Default Index	`Starts from 0 if not specified`
Dict → Series	`Keys become index labels`
Attributes	`.index, .values, .dtype, .size, .shape`
Slicing (position)	`Stop index NOT included`
Slicing (label)	`Stop label IS included`
head() / tail()	`First/Last n elements (default 5)`
Vector Ops	`Applied element-wise automatically`
NaN	`Appears when index labels do not match in arithmetic`
sort_values()	`Sorts by data values`
sort_index()	`Sorts by index labels`
Filter	`series[series > value]`