📊 Pandas Series

🔹 What is Pandas?

Pandas is a powerful Python library used for **data analysis and data manipulation**.
It helps us work easily with structured data like tables, columns, and rows.

In [ ]:
```
# import major libraries
import pandas as pd
import numpy as np
```

🔹 Main Data Structures in Pandas

Pandas mainly provides **two core data types**:

- **Series** → One-dimensional data (single column)
- **DataFrame** → Two-dimensional data (rows + columns, like a table)

Two data types:

- Series --> column
- DataFrames --> Table

🔹 Creating a Series from a List

When a list is converted into a Series:

- Pandas automatically assigns **numeric indexes starting from 0**
- All values are stored in a single column-like structure

This is useful when you want to convert simple data into an analyzable format.

In [ ]:
```
#series -- > column
#list
l1=[10,20,30,40]
pd.Series(l1)
```

🔹 Creating a Series from Strings

A Series can also store **text data** like country names.

- Data type becomes `object`
- Indexes are automatically generated
- Useful for categorical data

In [ ]:

```
# countries
countries = ['India','China','USA','Japan','Russia']
countries = pd.Series(countries)
countries
```

🔹 Creating a Series from a Dictionary

When a dictionary is used:

- **Keys become indexes**
- **Values become data**
- This is called **labeled indexing**

This is very useful when data already has meaningful labels.

🔹 Custom Index in Series

You can assign your **own index names** instead of default numbers.

- Helpful in marksheets, subject-wise data, or named records
- Makes data more readable and meaningful

In [ ]:

```
dict1={
    'Dunki':'SRK',
    'Sultan':'SK',
    'Sanju':'Ranbir kapoor',
    'PK':'AK',
    'Holiday':'Akshay Kumar'
}
movies = pd.Series(dict1)
movies
#labelled indexes
```

🔹 Handling Missing Values (NaN)

- `NaN` represents **missing or undefined data**

- Pandas automatically converts numeric data with NaN to `float`
- Very common in real-world datasets

In [ ]:

```
sub = ['Hindi','Englis','SST','Science']
marks=[np.nan,78,56,np.nan]
std=pd.Series(marks,index=sub,name='Vipul_Marks')
std
```

🔹 Saving Series to CSV

A Series can be saved as a **CSV file**.

- CSV means *Comma Separated Values*
- Used for data sharing and storage
- Can be opened in Excel or Google Sheets

In [ ]:

```
#(saving Table in home of jupyter CSV=Comma Separated Values)
std.to_csv('std.csv')
```

🔹 Creating Series from NumPy Arrays

NumPy arrays can be converted into Series easily.

- Useful for numerical and statistical analysis
- Supports large datasets efficiently
- Indexes can be customized

In [ ]:

```
#numpy arrays --> series
marks = pd.Series(np.random.randint(0,101,100),index=range(1,101,1)) #index --
marks
```

🔹 Important Attributes of Series

▶ Index

- Shows labels of each value
- Can be numeric or text-based

In [ ]:

```
# attributes
# basic attributes
# index
print(marks.index)
print(countries.index)
print(movies.index)
```

▶ Values

- Returns only the data stored in the Series
- Does not include indexes

In [ ]:

```
# values
marks.values
movies.values
```

▶ Data Type (`dtype`)

- Tells the type of data stored
- Important for calculations and memory usage

In [ ]:

```
#dtype
marks.dtype
std.dtype
print(countries.dtype)
```

▶ Name

- A Series can have a name
- Useful for identification

In [ ]:

```
#name
marks.name
std.name
```

▶ Shape

- Shows number of elements
- Always one-dimensional

In [ ]:

```
#shape
marks.shape
```

▶ Size

- Total number of elements including missing values

In [ ]:

```
#size
marks.size
std.size
```

▶ Count

- Counts only **non-missing values**
- Ignores NaN values

In [ ]:

```
#count function
marks.count()
std.count()
```

▶ Dimensions (`ndim`)

- Always `1` for Series

In [ ]:

```
#ndim
marks.ndim
```

▶ Is Unique

- Checks whether all values are unique

In [ ]:

```
# isunique
countries.is_unique
marks.is_unique
```

▶ Empty

- Checks if the Series has no data

In [ ]:

```
#empty
marks.empty
```

🔹 String Operations on Series

Series containing text data support **string methods**.

- Example: converting text to uppercase
- Very useful for text cleaning

In [ ]:

```
#str
countries.str.upper()
```

🔹 Viewing Data

▶ Head

- Shows first few values
- Helps quickly inspect data

▶ Tail

- Shows last few values

▶ Sample

- Returns random values
- Useful for large datasets

In [ ]:

```
# function
# head
# tail
# sample
marks
# formula for all value shown 1 to 100(consider not you use it)  --> pd.setopt
marks.head(10) #top 5
marks.tail() #last 5
marks.sample(5) # random 5
```

In [ ]:

```
marks.head(10) #top 10
```

In [ ]:

```
marks.tail() #last 5
```

🔹 Info Method

Provides:

- Total entries
- Data types
- Memory usage
- Non-null count

Gives a **summary of Series structure**.

In [ ]:

```
#info
marks.info()
```

🔹 Describe Method

Gives **statistical summary**:

- Count
- Mean
- Standard deviation
- Minimum and maximum
- Quartiles (25%, 50%, 75%)

Used mainly for numerical analysis.

In [ ]:
```
#describe()
marks.describe()
```

SELECTION AND FILTERATION --> IMPORTANT

🔹 Selection and Filtering

▶ Indexing

Accessing data using index positions.

▶ Slicing

Extracting a range of values.

▶ Label-based Selection

Accessing values using index labels.

▶ Condition-based Filtering

Selecting values based on conditions.

In [ ]:
```
marks[3] #indexing
```

In [ ]:
```
marks[3:5] # slicing
```

In [ ]:
```
#loc --> labelled indexing
movies
```

```
In [ ]: movies.loc['Dunki':'PK']
```

```
In [ ]: #iloc --> index
        countries[1:4:2] # 2 stpe size or gap
```

```
In [ ]: #condition based
        marks[marks<10]
```

```
In [ ]: # sorting methods
        #sort_values
        #sort_index
```

🔹 Sorting Series

▶ Sort by Values

- Arranges data in ascending or descending order

▶ Sort by Index

- Orders data based on index labels

```
In [ ]: marks.sort_values()
```

```
In [ ]: marks=marks.sort_values(ascending=False)
        marks
```

```
In [ ]: marks=marks.sort_index()
        marks
```

🔹 Aggregate Functions

- **Sum** – Adds all values
- **Mean** – Average value
- **Median** – Middle value
- **Mode** – Most frequent value
- **Variance** – Data spread
- **Standard Deviation** – Data variation
- **Min / Max** – Lowest and highest values
- **Quantiles** – Divides data into equal parts

```
In [ ]: #aggregate functions
        #sum
        marks.sum()
```

```
std.sum()
```

In [ ]:
```
#mean
marks.mean()
```

In [ ]:
```
#median
marks.median()
```

In [ ]:
```
#mode
marks.mode()
```

In [ ]:
```
#value_counts()
marks.value_counts().head()
```

In [ ]:
```
#Variance
marks.var()
```

In [ ]:
```
#std
marks.std()
```

In [ ]:
```
#min/max
print(marks.min())
print(marks.max())
```

In [ ]:
```
#count
marks.count()
```

In [ ]:
```
#quantile
print(marks.quantile(0.25))
print(marks.quantile(0.50))
print(marks.quantile(0.75))
```

🔹 Value Frequency Analysis

- Counts how many times each value appears
- Useful for categorical data analysis

🔹 Data Cleaning Operations

▶ Replace

- Replaces specific values

▶ Type Conversion

- Changes data type

▶ Round

- Rounds numeric values

▶ Clip

- Limits values within a range

In [ ]:
```
#replace and clean
#replace
countries.replace('USA','SOUTH KORIA')
```

In [ ]:
```
#astype
marks.astype(float)
```

In [ ]:
```
#round
marks.round(2)
```

In [ ]:
```
#clip
marks.clip(10,60).head(20) #--> 10 se niche 10 ho jaye ge values
```

🔹 Unique and Duplicate Values

▶ Unique

- Returns unique values

▶ Duplicated

- Finds repeated values

▶ Drop Duplicates

- Removes duplicate values

In [ ]:
```
#unique
#duplicated
#value_counts
#to_dict
```

In [ ]:
```
marks.unique()
```

In [ ]:

```
marks[marks.duplicated()].head(15)
```

In [ ]:

```
marks.drop_duplicates().head(15)
```

In [ ]:

```
movies.value_counts()
```

In [ ]:

```
movies.value_counts().to_dict()
```

🔹 Handling Missing Data

▶ Is Null

- Detects missing values

▶ Drop NA

- Removes missing values

▶ Fill NA

- Replaces missing values

In [ ]:

```
#filling
#dropna
#isnull
std.isnull().sum()
```

In [ ]:

```
#dropna --> NA VALUE HATA DIYA
std.dropna()
```

In [ ]:

```
#filling
std.fillna(10)
```

In [ ]:

```
std[std.isnull()]
```

🔹 Final Note

Pandas Series is the **foundation of data analysis**.
Mastering Series makes DataFrames and advanced analysis much easier.