



Pandas

created by :
The easylearn academy

Introduction of Pandas

- **Pandas** is an essential library that is frequently used in the field of **Data Science** and **Artificial Intelligence**.
- It is actually an abbreviation of **Panel Data**.
- The **Pandas** library is essential in the python programming language.
- It is a Python library that provides fast, flexible, and impressive data structure for data manipulation and analysis.
- **Panel Data**; in statistics and econometrics, it is multidimensional data that contains specific measurements over a particular period of time.

Key Feature of Pandas

1. Data Structures:

Provides Series (1D) and DataFrame (2D) for handling structured data efficiently.

2. Data Alignment:

Automatic and explicit data alignment with robust handling of missing data.

3. Data Cleaning:

Tools for filtering, sorting, grouping, and handling missing or duplicate data.

4. Merging/Joining:

Flexible methods to merge, join, and concatenate datasets.

5. Time Series:

Robust support for date/time indexing, resampling, and rolling windows.

6. IO Tools:

Read/write data from/to CSV, Excel, JSON, SQL, HDF5, and more.

7. Aggregation:

Efficient group-by operations and pivot tables for summarizing data.

8. Performance:

Optimized for speed with C-based operations via NumPy integration.

9. Visualization:

Integration with Matplotlib for quick plotting of data.

10. Flexibility:

Extensive functions for reshaping, slicing, and indexing data.

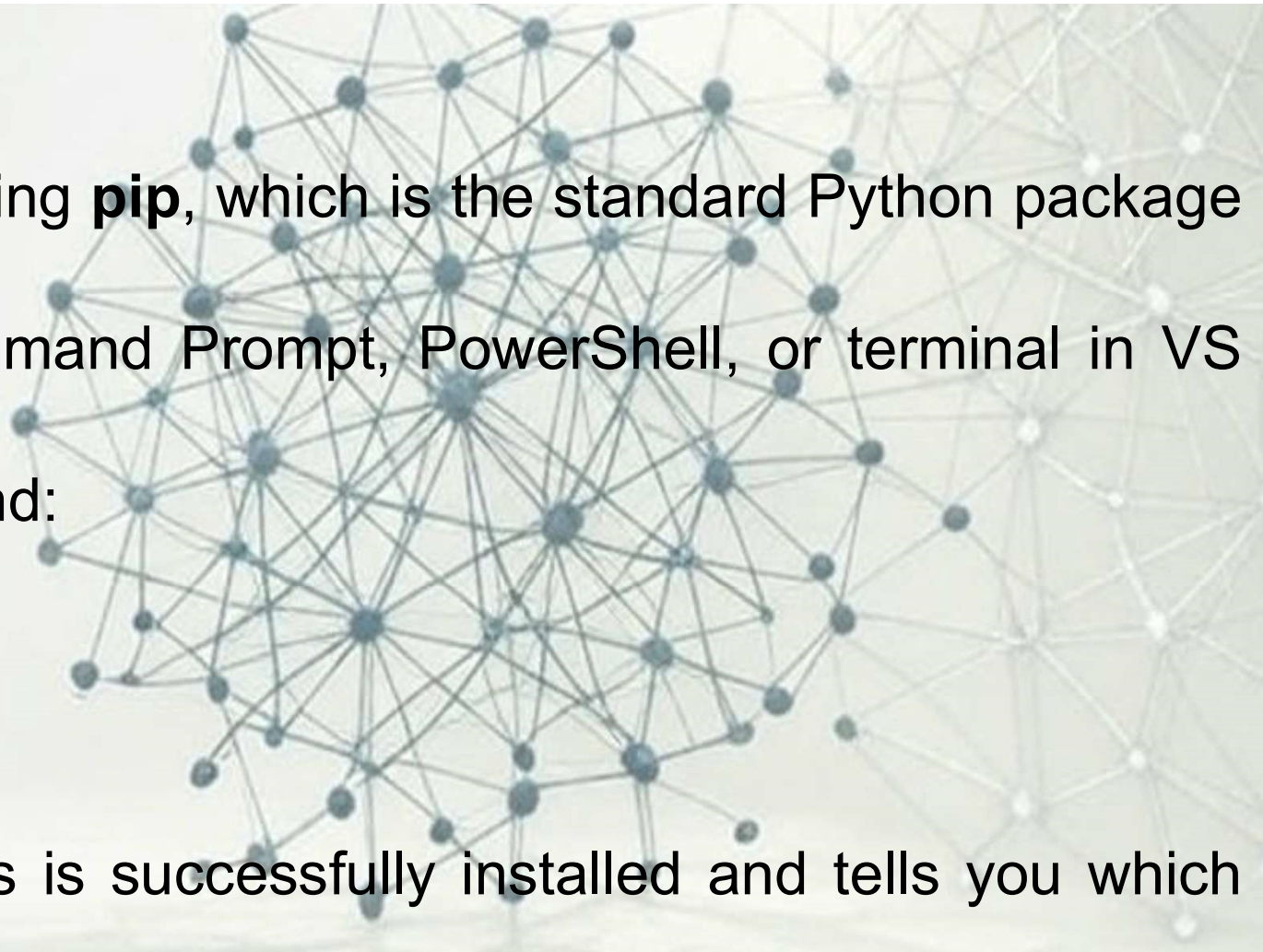
Why is Pandas important?

- **Data Handling:** Pandas provides DataFrames and Series, enabling easy storage, cleaning, and transformation of structured data, like spreadsheets or databases, which is often the starting point for ML workflows.
- **Data Preprocessing:** AI/ML models require clean, well-structured data. Pandas excels at handling missing values, filtering, grouping, merging, and reshaping datasets, reducing preprocessing time.
- **Exploratory Data Analysis (EDA):** Pandas supports quick statistical summaries, correlations, and visualizations (with libraries like Matplotlib/Seaborn), helping identify patterns or anomalies before modelling.
- **Integration:** It seamlessly integrates with ML libraries like Scikit-learn, TensorFlow, and PyTorch, allowing data to flow directly from Pandas DataFrames into model training pipelines.
- **Efficiency:** Pandas optimizes operations with vectorized computations (via NumPy), making it faster than manual Python loops for large datasets.
- **Versatility:** It handles diverse data formats (CSV, Excel, JSON, SQL, etc.), making it adaptable to real-world data sources.

NumPy vs Pandas

Numpy	Pandas
NumPy array contains data of same datatype.	Pandas DataFrame can have heterogeneous data.
NumPy has a better performance for 50K rows or less.	Pandas has a better performance for 500K rows or more.
NumPy consumes less memory as compared to Pandas.	Pandas consume large memory as compared to NumPy.
NumPy is preferred while using numbers.	Pandas is preferred in tabular data.

How to Install Pandas

- You can install Pandas using **pip**, which is the standard Python package manager.
 - Open your terminal (Command Prompt, PowerShell, or terminal in VS Code or PyCharm).
 - Run the following command:
 - `pip install pandas`
 - To Verify Installation:
 - `import pandas`
 - `print(pandas.__version__)`
 - This confirms that Pandas is successfully installed and tells you which version is active.
- 

Series

- A **Series** is a one-dimensional array-like object that can hold data of any type (integers, strings, floats, etc.).
- It is similar to a column in a spreadsheet or a single array in NumPy, but with an associated index for labeling.

Key Feature:

- **Structure:** Consists of data and an index (labels for each data point).
- **Index:** By default, a Series has a numeric index (0, 1, 2, ...), but you can assign custom labels.
- **Flexible Data Types:** Can store homogeneous or heterogeneous data.
- **Similar to NumPy Array:** Supports vectorized operations but adds index-based alignment.

How to Create Series

```
import pandas as pd  
X = pd.Series(Data, Index)
```

- **Key Attribute:**

Data: The actual values stored in the Series.

Index: A set of labels (default is 0, 1, 2, ...) associated with the data for easy access.

- **Example:**

- import pandas as pd
- data = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
- print(data)

Creating Series from List:

```
import pandas as pd    # import pandas library  
data = [10, 20, 30]    # list  
s = pd.Series(data)     # pass list as data
```

Creating Series from Dictionary:

```
import pandas as pd  
data = {'a': 1, 'b': 2, 'c': 3} # Dictionary  
s = pd.Series(data)
```

Creating Series from NumPy Array:

```
import pandas as pd    # import pandas library
import numpy as np     # import numpy library
data = np.array([1, 2, 3])    # Numpy Array
s = pd.Series(data, index=['x', 'y', 'z'])    # Pass the numpy array
```

Creating Series with custom index and name:

```
import pandas as pd
s = pd.Series([1, 2, 3], index=['a', 'b', 'c'], name='MySeries')
```


Methods for Loading Data:

- **Loading Data from CSV Files**

- `pd.read_csv(filepath, **kwargs)`

- **Loading Data from Excel Files:**

- `pd.read_excel(filepath, **kwargs)` # only load first sheet

- **Loading Data from JSON Files:**

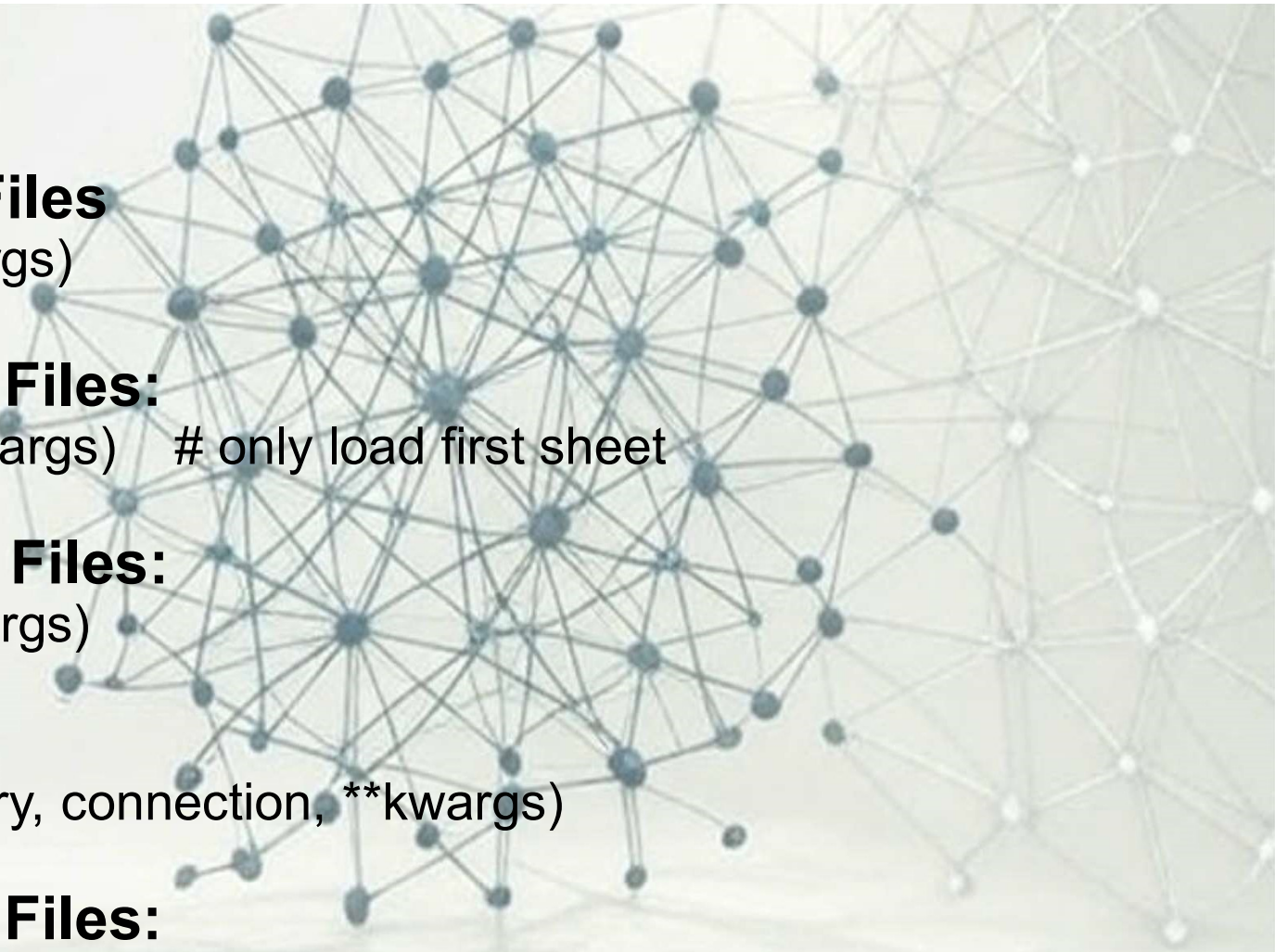
- `pd.read_json(filepath, **kwargs)`

- **Loading Data from SQL:**

- Databases: `pd.read_sql(query, connection, **kwargs)`

- **Loading Data from HDF5 Files:**

- `pd.read_hdf(filepath, key, **kwargs)`



Creating Series from Database:

```
import pandas as pd  
S1 = pd.read_csv('database url', usecols=['column name']).squeeze()  
s1
```

Key Arguments:

usecols: use to call particular column From Database

squeeze(): convert DataFrame into series

Attributes of Series

Attribute	Syntax	Description
index	s.index	Returns the index (labels) of the Series.
Values	s.values	Returns the series data as a NumPy array.
dtype	s.dtype	The data type of the Series (e.g., int64, float64, object, etc.). Returns the data type of the Series' elements.
name	s.name	Returns the name of the Series (if set). Useful when converting to a DataFrame or for identification.
shape	s.shape	Returns a tuple representing the dimensionality of the Series (always (n,)) for a Series, where n is the number of elements).
ndim	s.ndim	Returns the number of dimensions of the Series (always 1 for a Series).
size	s.size	Returns the total number of elements in the Series.
is_unique	s.is_unique	Returns True if all values in the Series are unique , otherwise False.
hasnans	s.hasnans	Returns True if the Series contains any NaN (missing) values, otherwise

Index

- **Description:**

- Returns the index (labels) of the Series.

- **Syntax:**

- `s.index`

- `import pandas as pd`
- `s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])`
- `print(s.index) / s.index`
- **Output:**
 - `Index(['a', 'b', 'c'], dtype='object')`

Values

- **Description:**

- Returns the series data as a NumPy array.

- **Syntax:**

- `s.values`

- `import pandas as pd`
- `s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])`
- `print(s.values) / s.values`
- **Output:**
 - `[10 20 30]`

Dtype

- **Description:**

- The data type of the Series (e.g., int64, float64, object, etc.).
- Returns the data type of the Series' elements.

- **Syntax:**

- s.dtype

- import pandas as pd
- s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
- print(s.dtype)/ s.dtype
- **Output:**
 - int64

Names

- **Description:**

- Returns the name of the Series (if set).
- Useful when converting to a DataFrame or for identification.

- **Syntax:**

- s.names

- import pandas as pd
- s = pd.Series([10, 20, 30], index=['a', 'b', 'c'], name='series1')
- print(s.names) / s.name

- **Output:**

- series1

- Or

- s.name = "My Series" # for change name

- **Output:**

- My Series

Shape

- **Description:**

- Returns a tuple representing the dimensionality of the Series (always (n,) for a Series, where n is the number of elements).

- **Syntax:**

- s.shape

- import pandas as pd
- s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
- print(s.shape) / shape
- **Output:**
 - (3,)

Ndim

- **Description:**

- Returns the number of dimensions of the Series (always 1 for a Series).

- **Syntax:**

- `s.ndim`

- `import pandas as pd`
- `s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])`
- `print(s.ndim) / s.ndim`
- **Output:** 1

Size

- **Description:**

- Returns the total number of elements in the Series.

- **Syntax:**

- `s.size`

- `import pandas as pd`
- `s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])`
- `print(s.size) / s.size`
- **Output:**
 - 3

Is_unique

- **Description:**

- Returns True if all **values** in the Series are **unique**, otherwise False.

- **Syntax:**

- `s.is_unique`

- `s = pd.Series([1, 2, 2])`
- `print(s.is_unique) / s.is_unique`
- **Output:**
 - False

Hasnans

- **Description:**

- Returns True if the Series contains any NaN (missing) values, otherwise

- **Syntax:**

- s.hasnans

- import pandas as pd
- s2 = pd.Series([1, none, 3])
- print(s2.hasnans) / s2.hasnans

- **Output:**

- True

The background of the slide features a complex, abstract network of nodes and lines. The nodes are represented by small, dark blue spheres, and the lines are thin, dark grey or black. These elements are interconnected to form a dense, irregular cluster in the center of the image. This central cluster is surrounded by a more sparse network of nodes and lines that extends towards the edges of the frame. The overall aesthetic is technical and digital, suggesting themes of data, connectivity, or systems. The text "Methods of Series" is overlaid on this background, centered horizontally and slightly below the vertical center.

Methods of Series

Methods of Series

Method	Syntax	Description
Series()	pd.Series()	The Series constructor in Pandas creates a one-dimensional labeled array (Series)
head(n)	s.head(3)	Returns the first n elements of the Series. Defaults to 5.
tail(n)	s.tail(3)	Returns the last n elements of the Series. Defaults to 5.
to_list()	s.to_list()	Convert a Series to a Python list
to_dict()	s.to_dict()	Converts a Series to a Python dictionary , where the index becomes the keys and the values become the dictionary values.
type()	type(s)	Python's built-in type() function to check the type of a Datatable.
describe()	s.describe()	Generates descriptive statistics (count, mean, std, min, quartiles, max) for numeric Series.
astype()	s.astype(dtype)	Converts the Series to a specified data type .

Series()

- **Description:**

- The Series constructor in Pandas creates a one-dimensional labeled array (Series)

- **Syntax:**

- `pd.Series(data, index=None, dtype=None, name=None, copy=False)`

- **Key Parameters:**

- **data:** Input data (e.g., list, dict, ndarray).
- **index:** Optional index labels (defaults to 0, 1, 2, ...).
- **dtype:** Data type for the Series (e.g., int64, float64).
- **name:** Name of the Series (optional).
- **copy:** Whether to copy the input data (default is False).

- `import pandas as pd`
- `s = pd.Series([1, 2, 3], index=['a', 'b', 'c'], name='my_series')`
- `print(s)`
 - **Output:**
 - a 1
 - b 2
 - c 3
 - Name: my_series, dtype: int64

head()

- **Description:**

- Returns the first n rows of the Series (default is 5).

- **Syntax:**

- Series.head(n=5)

- **Key Parameters:**

- **n:** Number of rows to return (integer).

- import pandas as pd
- s = pd.Series([1, 2, 3, 4, 5, 6])
- print(s.head(3))

- **Output:**

- 0 1
- 1 2
- 2 3
- dtype: int64

tail()

- **Description:**

- Returns the last `n` rows of the Series (default is 5).

- **Syntax:**

- `Series.tail(n=5)`

- **Key Parameters:**

- **n:** Number of rows to return (integer).

- `import pandas as pd`
- `s = pd.Series([1, 2, 3, 4, 5, 6])`
- `print(s.tail(3))`

- **Output:**

- 0 1
- 1 2
- 2 3
- dtype: int64

to_list()

- **Description:**

- convert a Series to a Python list using the `.tolist()` method or by passing the Series to the `list()` function.

- **Syntax:**

- `Series.tolist()`

- **Key Parameters:**

- **Series:** series which have to convert into list

- `import pandas as pd`
- `s = pd.Series([1, 2, 3])`
- `print(s.tolist())`

- **Output:**

- `[1, 2, 3]`

- `import pandas as pd`
- `s = pd.Series([1, 2, 3])`
- `print(list(s))`

- **Output:**

- `[1, 2, 3]`

to_dict()

- **Description:**

- Converts a Series to a Python dictionary, where the index becomes the keys and the values become the dictionary values.

- **Syntax:**

- `Series.to_dict(into=<class dict>)`

- **Key Parameters:**

- **into:** The dictionary class to use (default is dict).

- `import pandas as pd`
- `s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])`
- `print(s.to_dict())`

- **Output:**

- `{'a': 1, 'b': 2, 'c': 3}`

type()

- **Description:**

- Python's built-in type() function to check the type of a Series or its elements.

- **Syntax:**

- **type(Series):** Returns the type of the object (pandas.core.series.Series).

- import pandas as pd
- s = pd.Series([1, 2, 3])
- print(type(s))

- **Output:**

- <class pandas.core.series.Series>

describe()

- **Description:**

- Generates descriptive statistics for the Series, including count, mean, std, min, max, and quartiles (for numeric data)

- **Syntax:**

- `Series.describe(percentiles=None, include=None, exclude=None)`

- **Key Parameters:**

- **percentiles:** List of percentiles to include (default is [0.25, 0.5, 0.75]).
- **include/exclude:** Data types to include/exclude.

```
• import pandas as pd
• s = pd.Series([1, 2, 3, 4, 5])
• print(s.describe())
```

- **Output:**

```
• count    5.000000
• mean     3.000000
• std      1.581139
• min      1.000000
• 25%      2.000000
• 50%      3.000000
• 75%      4.000000
• max      5.000000
• dtype: float64
```


astype()

- **Description:**

- Converts the Series to a specified data type.

- **Syntax:**

- `Series.astype(dtype, copy=True, errors='raise')`

- **Key Parameters:**

- **dtype:** Target data type (e.g., int, float, str).
- **copy:** Whether to return a copy (default is True).
- **errors:** 'raise' (default) or 'ignore' (if conversion fails).

```
• import pandas as pd  
• s = pd.Series([1.5, 2.7, 3.2])  
• print(s.astype(int))
```

Output:

- 0 1
- 1 2
- 2 3
- dtype: int32

Methods of Series

Method	Syntax	Description
abs()	s.abs()	Returns the absolute value of each element in the Series.
idxmax()	s.idxmax()	Returns the index label of the maximum value.
idxmin()	s.idxmin()	Returns the index label of the minimum value.
min()	s.min()	Returns the minimum value.
max()	s.max()	Returns the maximum value.
len()	len(s)	Returns the number of elements in the Series. (<i>Python built-in function</i>)
sort_values()	s.sort_values()	Sorts the Series by its values .
sort_index()	s.sort_index()	Sorts the Series by its index .

abs()

- **Description:**

- Returns the absolute value of each element in the Series.

- **Syntax:**

- Series.abs()

- Import pandas as pd
- s = pd.Series([-1, 2, -3])
- print(s.abs())

- **Output:**

- 0 1
- 1 2
- 2 3
- dtype: int64

idxmax()

- **Description:**

- Returns the index of the first occurrence of the maximum value in the Series.

- **Syntax:**

- `Series.idxmax(axis=0, skipna=True)`

- **Key Parameters:**

- **skipna:** Exclude NA/null values (default is True).

- Import pandas as pd
- `s = pd.Series([1, 3, 2], index=['a', 'b', 'c'])`
- `print(s.idxmax())`

- **Output:**

- b

idxmin()

- **Description:**

- Returns the index of the first occurrence of the minimum value in the Series.

- **Syntax:**

- `Series.idxmin(axis=0, skipna=True)`

- **Key Parameters:**

- **skipna:** Exclude NA/null values (default is True).

- Import pandas as pd
- `s = pd.Series([1, 3, 2], index=['a', 'b', 'c'])`
- `print(s.idxmin())`

- **Output:**

- a

min()

- **Description:**

- Returns the minimum value in the Series.

- **Syntax:**

- `Series.min(axis=0, skipna=True)`

- **Key Parameters:**

- **skipna:** Exclude NA/null values (default is True).

- Import pandas as pd
- `s = pd.Series([1, 3, 2])`
- `print(s.min())`

- **Output:**

- 1

max()

- **Description:**

- Returns the maximum value in the Series.

- **Syntax:**

- `Series.max(axis=0, skipna=True)`

- **Key Parameters:**

- **skipna:** Exclude NA/null values (default is True).

- Import pandas as pd
- `s = pd.Series([1, 3, 2])`
- `print(s.max())`

- **Output:**

- 3

len()

- **Description:**

- Returns the number of elements in the Series using Python's len() function.

- **Syntax:**

- len(Series)

- Import pandas as pd
- s = pd.Series([1, 2, 3])
- print(len(s))

- **Output:**

- 3

Sort_values()

- **Description:**

- Sorts the Series by its values.

- **Syntax:**

- `Series.sort_values(ascending=True, inplace=False, key=None, na_position='last')`

- **Key Parameters:**

- **ascending:** Sort in ascending (True) or descending (False) order.
- **inplace:** Modify the Series in place (default is False).
- **key:** Function to apply to values before sorting.
- **na_position:** 'last' (default) or 'first' for NA values.

- Import pandas as pd
- `s = pd.Series([3, 1, 2])`
- `print(s.sort_values())`

- **Output:**

- 1 1
- 2 2
- 0 3
- dtype: int64

Sort_index()

- **Description:**

- Sorts the Series by its index.

- **Syntax:**

- `Series.sort_index(ascending=True, inplace=False, key=None)`

- **Key Parameters:**

- **ascending:** Sort in ascending (True) or descending (False) order.
- **inplace:** Modify the Series in place (default is False).
- **key:** Function to apply to index before sorting.

- Import pandas as pd
- `s = pd.Series([1, 2, 3], index=['c', 'a', 'b'])`
- `print(s.sort_index())`

- **Output:**

- a 2
- b 3
- c 1
- dtype: int64

Methods of Series

Method	Syntax	Description
add()	s.add(x)	Performs element-wise addition of the Series with a scalar, Series, or array-like object, handling missing values with an optional fill value.
sub()	s.sub(x)	Performs element-wise subtraction of a scalar, Series, or array-like from the Series.
mul()	s.mul(x)	Performs element-wise multiplication of the Series with a scalar, Series, or array-like.
div()	s.div(x)	Performs element-wise division of the Series by a scalar, Series, or array-like.
pow()	s.pow(other)	Raises the Series values to the power of a scalar, Series, or array-like.
sum()	s.sum()	Computes the sum of all non-NaN values in the Series.
mean()	s.mean()	Computes the arithmetic mean of non-NaN values.
median()	s.median()	Computes the median of non-NaN values (middle value when sorted).

add()

- **Description:**

- Performs element-wise addition of the Series with a scalar, Series, or array-like object, handling missing values with an optional fill value.

- **Syntax:**

- `Series.add(other, fill_value=None)`

- **Key Parameters:**

- **other:** Scalar, Series, or array-like to add.
- **fill_value:** Value to replace NaN before addition (default None).

```
• import pandas as pd
• s1 = pd.Series([1, 2, None], index=['a', 'b', 'c'])
• s2 = pd.Series([4, 5, 6], index=['a', 'b', 'c'])
• result = s1.add(s2, fill_value=0)
```

- **Output:**

- a 5.0
- b 7.0
- c 6.0

sub()

- **Description:**

- Performs element-wise subtraction of a scalar, Series, or array-like from the Series.

- **Syntax:**

- `Series.sub(other, fill_value=None)`

- **Key Parameters:**

- **other:** Scalar, Series, or array-like to subtract.
- **fill_value:** Value to replace NaN before subtraction.

- Import pandas as pd
- `s1 = pd.Series([1, 2, None], index=['a', 'b', 'c'])`
- `s2 = pd.Series([4, 5, 6], index=['a', 'b', 'c'])`
- `result = s1.sub(s2, fill_value=0)`

- **Output:**

- a -3.0
- b -3.0
- c -6.0

mul()

- **Description:**

- Performs element-wise multiplication of the Series with a scalar, Series, or array-like.

- **Syntax:**

- `Series.mul(other, fill_value=None)`

- **Key Parameters:**

- **other:** Scalar, Series, or array-like to multiply.
- **fill_value:** Value to replace NaN before multiplication.

- Import pandas as pd
- `s1 = pd.Series([1, 2, None], index=['a', 'b', 'c'])`
- `s2 = pd.Series([4, 5, 6], index=['a', 'b', 'c'])`
- `result = s1.mul(2, fill_value=0)`

- **Output:**

- a 2.0
- b 4.0
- c 0.0

div()

- **Description:**

- Performs element-wise division of the Series by a scalar, Series, or array-like.

- **Syntax:**

- `Series.div(other, fill_value=None)`

- **Key Parameters:**

- **other:** Scalar, Series, or array-like to divide by.
- **fill_value:** Value to replace NaN before division.

- Import pandas as pd
- `s1 = pd.Series([1, 2, None], index=['a', 'b', 'c'])`
- `s2 = pd.Series([4, 5, 6], index=['a', 'b', 'c'])`
- `result = s1.div(s2, fill_value=1)`

- **Output:**

- a 0.25
- b 0.4
- c 0.166667

pow()

- **Description:**

- Raises the Series values to the power of a scalar, Series, or array-like.

- **Syntax:**

- `Series.pow(other, fill_value=None)`

- **Key Parameters:**

- **other:** Scalar, Series, or array-like for exponent.
- **fill_value:** Value to replace NaN before exponentiation.

- Import pandas as pd
- `s1 = pd.Series([1, 2, None], index=['a', 'b', 'c'])`
- `result = s1.pow(2, fill_value=0)`

- **Output:**

- a 1.0
- b 4.0
- c 0.0

sum()

- **Description:**

- Computes the sum of all non-NaN values in the Series.

- **Syntax:**

- `Series.sum(skipna=True)`

- **Key Parameters:**

- **skipna:** If True, excludes NaN from calculation (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 3, None])`
- `result = s.sum()`

- **Output:**

- 6

mean()

- **Description:**

- Computes the arithmetic mean of non-NaN values.

- **Syntax:**

- `Series.mean(skipna=True)`

- **Key Parameters:**

- **skipna:** If True, excludes NaN (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.mean()`

- **Output:**

- 2 # $(1+2+3)/3$

median()

- **Description:**

- Computes the median of non-NaN values (middle value when sorted).

- **Syntax:**

- `Series.median(skipna=True)`

- **Key Parameters:**

- **skipna:** If True, excludes NaN (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.median()`

- **Output:**

- 2.0

Methods of Series

Method	Syntax	Description
mode()	s.mode()	Returns the most frequent value(s) in the Series.
count()	s.count()	Counts the number of non-NaN values in the Series.
std()	s.std()	Computes the standard deviation of non-NaN values.
var()	s.var()	Computes the variance of non-NaN values.
isnull()	s.isnull()	Returns a boolean Series with True for NaN/None values.
notnull()	s.notnull()	Returns a boolean Series with True for non-NaN/None values.
dropna()	s.dropna()	Removes NaN/None values from the Series.
fillna()	s.fillna(value)	Replaces NaN/None with a specified value or interpolation method.

mode()

- **Description:**

- Returns the most frequent value(s) in the Series.

- **Syntax:**

- `Series.mode(dropna=True)`

- **Key Parameters:**

- **dropna:** If True, ignores NaN when finding mode (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 2, 3])`
- `result = s.mode()`

- **Output:**

- 0 2

count()

- **Description:**

- Counts the number of non-NaN values in the Series.

- **Syntax:**

- Series.count()

- Import pandas as pd
- s = pd.Series([1, 2, 2, 3])
- result = s.count()

- **Output:**

- 4

std()

- **Description:**

- Computes the standard deviation of non-NaN values.

- **Syntax:**

- `Series.std(skipna=True)`

- **Key Parameters:**

- **skipna:** If True, excludes NaN (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.std()`

- **Output:**

- 1.0

var()

- **Description:**
 - Computes the variance of non-NaN values.
- **Syntax:**
 - `Series.var(skipna=True)`
- **Key Parameters:**
 - **skipna:** If True, excludes NaN (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.var()`

- **Output:**
 - 1.0

isnull()

- **Description:**

- Returns a boolean Series with True for NaN/None values.

- **Syntax:**

- Series.isnull()

- Import pandas as pd
- s = pd.Series([1, None, 3])
- result = s.isnull()

- **Output:**

- 0 False
- 1 True
- 2 False

notnull()

- **Description:**

- Returns a boolean Series with True for non-NaN/None values.

- **Syntax:**

- Series.notnull()

- Import pandas as pd
- s = pd.Series([1, None, 3])
- result = s.notnull()

- **Output:**

- 0 True
- 1 False
- 2 True

dropna()

- **Description:**

- Removes NaN/None values from the Series.

- **Syntax:**

- `Series.dropna(inplace=False)`

- **Key Parameters:**

- `inplace`: If True, modifies Series directly (default False).

- Import pandas as pd
- `s = pd.Series([1, None, 3])`
- `result = s.dropna()`

- **Output:**

- 0 1.0
- 2 3.0

fillna()

- **Description:**

- Replaces NaN/None with a specified value or interpolation method.

- **Syntax:**

- `Series.fillna(value=None, method=None, inplace=False)`

- **Key Parameters:**

- **value:** Scalar to replace NaN.
- **method(optional):** 'ffill' (forward fill) or 'bfill' (backward fill).
- **inplace:** If True, modifies Series directly.

- Import pandas as pd
- `s = pd.Series([1, None, 3])`
- `result = s.fillna(0)`

- **Output:**

- 0 1.0
- 1 0.0
- 2 3.0

Methods of Series

Method	Syntax	Description
<code>uplicated()</code>	<code>s.duplicated()</code>	Identifies duplicate values, marking them as True.
<code>drop_duplicates()</code>	<code>s.drop_duplicates()</code>	Removes duplicate values, keeping the first, last, or none.
<code>copy()</code>	<code>s.copy()</code>	Creates a deep or shallow copy of the Series to avoid modifying the original.
<code>replace()</code>	<code>s.replace()</code>	Replaces specified values with new values.
<code>clip()</code>	<code>s.clip()</code>	Limits values to a specified minimum and maximum range.
<code>cumsum()</code>	<code>s.cumsum()</code>	Computes the cumulative sum of Series values.
<code>cumprod()</code>	<code>s.cumprod()</code>	Computes the cumulative product of Series values.

deduplicated()

- **Description:**

- Identifies duplicate values, marking them as True.

- **Syntax:**

- Series.duplicated(keep='first')

- **Key Parameters:**

- **keep:** 'first' (mark duplicates except first), 'last', or False (mark all duplicates).

- Import pandas as pd
- s = pd.Series([1, 2, 2, 3])
- result = s.duplicated()

- **Output:**

- 0 False
- 1 False
- 2 True
- 3 False

Drop_duplicates()

- **Description:**

- Removes duplicate values, keeping the first, last, or none.

- **Syntax:**

- `Series.drop_duplicates(keep='first', inplace=False)`

- **Key Parameters:**

- **keep:** 'first' (keep first occurrence), 'last', or False (drop all duplicates).
- **inplace:** If True, modifies Series directly.

- Import pandas as pd
- `s = pd.Series([1, 2, 2, 3])`
- `result = s.drop_duplicates()`

- **Output:**

- 0 1
- 1 2
- 3 3

copy()

- **Description:**

- Creates a deep or shallow copy of the Series to avoid modifying the original.

- **Syntax:**

- `Series.copy(deep=True)`

- **Key Parameters:**

- **deep:** If True, creates a deep copy; if False, shallow copy.

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `s_copy = s.copy()`
- `s_copy[0] = 10` # change value of index 0

- **Output:**

- `s_copy: [10, 2, 3]`
- `s: [1, 2, 3]`

replace()

- **Description:**

- Replaces specified values with new values.

- **Syntax:**

- `Series.replace(to_replace, value, inplace=False)`

- **Key Parameters:**

- **to_replace:** Value(s) to replace (scalar, list, dict, etc.).
- **value:** Replacement value.
- **inplace:** If True, modifies Series directly.

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.replace(1, 10)`

- **Output:**

- 0 10
- 1 2
- 2 3

clip()

- **Description:**

- Limits values to a specified minimum and maximum range.

- **Syntax:**

- `Series.clip(lower=None, upper=None, inplace=False)`

- **Key Parameters:**

- **lower:** Minimum threshold.
- **upper:** Maximum threshold.
- **inplace:** If True, modifies Series directly.

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.clip(1, 2)`

- **Output:**

- 0 1
- 1 2
- 2 2

cumsum()

- **Description:**

- Computes the cumulative sum of Series values.

- **Syntax:**

- `Series.cumsum(skipna=True)`

- **Key Parameters:**

- **skipna:** If True, excludes NaN (default True).

```
• Import pandas as pd
• s = pd.Series([1, 2, 3])
• result = s.cumsum()
• print(result)
```

- **Output:**

```
• 0    1    #1
• 1    3    # 1+2
• 2    6    # 1+2+3
```

cumprod()

- **Description:**

- Computes the cumulative product of Series values.

- **Syntax:**

- `Series.cumprod(skipna=True)`

- **Key Parameters:**

- **skipna:** If True, excludes NaN (default True).

- Import pandas as pd
- `s = pd.Series([1, 2, 3])`
- `result = s.cumprod()`
- `print(result)`

- **Output:**

- 0 1 #1
- 1 2 # 1*2
- 2 6 # 1*2*3

Methods of Series

Method	Syntax	Description
<code>str.replace()</code>	<code>s.str.replace(old, new)</code>	Replaces occurrences of a pattern or substring with a new value.
<code>str.lower()</code>	<code>s.str.lower()</code>	Converts all characters in string Series to lowercase.
<code>str.upper()</code>	<code>s.str.upper()</code>	Converts all characters in string Series to uppercase.
<code>str.contains()</code>	<code>s.str.contains('subsring')</code>	Checks if each string contains a specified pattern or substring.
<code>str.len()</code>	<code>s.str.len()</code>	Computes the length of each string in the Series.
<code>str.strip()</code>	<code>s.str.strip()</code>	Removes leading and trailing characters (default: whitespace) from strings.

str.replace()

- **Description:**

- Replaces occurrences of a pattern or substring with a new value.

- **Syntax:**

- Series.str.replace(pat, repl, regex=False)

- **Key Parameters:**

- pat: Pattern or substring to replace.
- repl: Replacement string.
- regex: If True, treats `pat` as regex (default False).

- Import pandas as pd
- s = pd.Series(['ABC', 'Def'])
- result = s.str.replace('A', 'X')

- **Output:**

- 0 XBC
- 1 Def

Str.lower()

- **Description:**

- Converts all characters in string Series to lowercase.

- **Syntax:**

- Series.str.lower()

- Import pandas as pd
- s = pd.Series(['ABC', 'Def'])
- result = s.str.lower()

- **Output:**

- 0 abc
- 1 def

str.upper()

- **Description:**

- Converts all characters in string Series to uppercase.

- **Syntax:**

- Series.str.upper()

- Import pandas as pd
- s = pd.Series(['ABC', 'Def'])
- result = s.str.upper()

- **Output:**

- 0 ABC
- 1 DEF

str.contains()

- **Description:**

- Checks if each string contains a specified pattern or substring.

- **Syntax:**

- `Series.str.contains(pat, case=True, regex=True, na=False)`

- **Key Parameters:**

- `pat`: Pattern or substring to search.
- `case`: If True, case-sensitive (default True).
- `regex`: If True, treats `pat` as regex (default True).
- `na`: Value to return for NaN (default False).

- Import pandas as pd
- `s = pd.Series(['ABC', 'Def'])`
- `result = s.str.contains('A', case=True)`

- **Output:**

- 0 True
- 1 False

str.len()

- **Description:**

- Computes the length of each string in the Series.

- **Syntax:**

- Series.str.len()

- Import pandas as pd
- s = pd.Series(['ABC', 'Def'])
- result = s.str.len()

- **Output:**

- 0 3
- 1 3

str.strip()

- **Description:**

- Removes leading and trailing characters (default: whitespace) from strings.

- **Syntax:**

- `Series.str.strip(to_strip=None)`

- **Key Parameters:**

- **to_strip:** Specific characters to strip; default is whitespace.

- Import pandas as pd
- `s = pd.Series([' ABC ', ' Def'])`
- `result = s.str.strip()`

- **Output:**

- 0 ABC
- 1 Def

Methods of Series

Method	Syntax	Description
dt.year	s.dt.year	Extracts the year from datetime values in the Series.
dt.month	s.dt.month	Extracts the month (1–12) from datetime values.
dt.day	s.dt.day	Extracts the day of the week (0=Monday, 6=Sunday) from datetime values.
dt.weekday	s.dt.weekday	Extracts the day of the week (0=Monday, 6=Sunday) from datetime values.
dt.strftime	s.dt.strftime('%Y-%m-%d')	Formats datetime values as strings using a specified format.
loc[]	s.loc[label]	Accesses or sets values by label-based indexing.
iloc[]	s.iloc[position]	Accesses or sets values by integer position-based indexing.
get()	s.get(key, default)	Safely retrieves a value by index label, returning a default if not found.

dt.year

- **Description:**

- Extracts the year from datetime values in the Series.

- **Syntax:**

- Series.dt.year

- import pandas as pd
- s = pd.Series([pd.Timestamp('2023-01-01'),
pd.Timestamp('2024-01-01')])
- result = s.dt.year
- Print(result)

- **Output:**

- 0 2023
- 1 2024

dt.month

- **Description:**

- Extracts the month (1–12) from datetime values.

- **Syntax:**

- Series.dt.month

- import pandas as pd
- s = pd.Series([pd.Timestamp('2023-01-01'),
pd.Timestamp('2024-02-01')])
- result = s.dt.month
- print(result)

- **Output:**

- 0 1
- 1 2

dt.day

- **Description:**

- Extracts the day of the week (0=Monday, 6=Sunday) from datetime values.

- **Syntax:**

- Series.dt.day

- import pandas as pd
- s = pd.Series([pd.Timestamp('2023-01-01'),
pd.Timestamp('2024-02-10')])
- result = s.dt.day
- print(result)

- **Output:**

- 0 1
- 1 10
- dtype: int32

dt.weekday

- **Description:**

- Extracts the day of the week (0=Monday, 6=Sunday) from datetime values.

- **Syntax:**

- Series.dt.weekday

- Import pandas as pd
- s = pd.Series([pd.Timestamp('2023-01-01'),
pd.Timestamp('2024-02-01')])
- result = s.dt.weekday
- print(weekday)

- **Output:**

- 0 6 #Sunday
- 1 1 # Monday

Dt.strftime

- **Description:**

- Formats datetime values as strings using a specified format.

- **Syntax:**

- `Series.dt.strftime(format)`

- **Key Parameters:**

- **format:** String format (e.g., '%Y-%m' for year-month).

- Import pandas as pd
- `s = pd.Series([pd.Timestamp('2023-01-01'), pd.Timestamp('2024-02-01')])`
- `result = s.dt.strftime('%Y-%m')`
- `Print(result)`

- **Output:**

- 0 2023-01
- 1 2024-01

loc[]

- **Description:**

- Accesses or sets values by label-based indexing.

- **Syntax:**

- `Series.loc[indexer]`

- **Key Parameters:**

- **indexer:** Label, list of labels, or boolean array.

- Import pandas as pd
- `s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])`
- `result = s.loc['a']`
- `print(result)`

- **Output:**

- 1

iloc[]

- **Description:**

- Accesses or sets values by integer position-based indexing.

- **Syntax:**

- `Series.iloc[indexer]`

- **Key Parameters:**

- **indexer:** Integer, list of integers, or slice.

- Import pandas as pd
- `s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])`
- `result = s.iloc[0]`
- `result1 = s.iloc[0:2]`
- `print(result, result1)`

- **Output:**

- 1
- Or
- [1,2]

get()

- **Description:**

- Safely retrieves a value by index label, returning a default if not found.

- **Syntax:**

- `Series.get(key, default=None)`

- **Key Parameters:**

- **key:** Index label to access.
- **default:** Value to return if key not found (default None).

- Import pandas as pd
- `s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])`
- `result = s.get('a', 0)`
- `print(result)`

- **Output:**

- 1