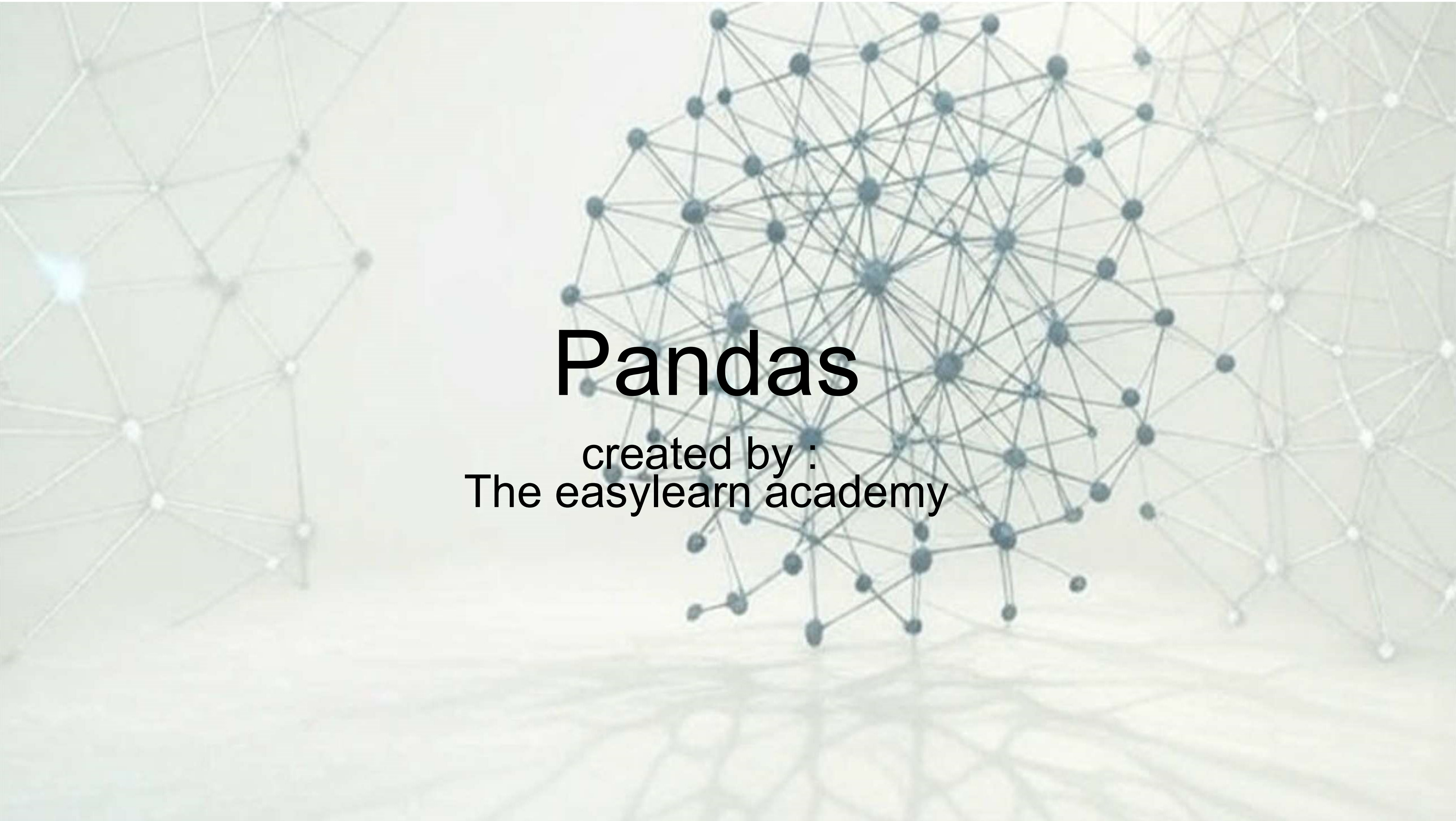
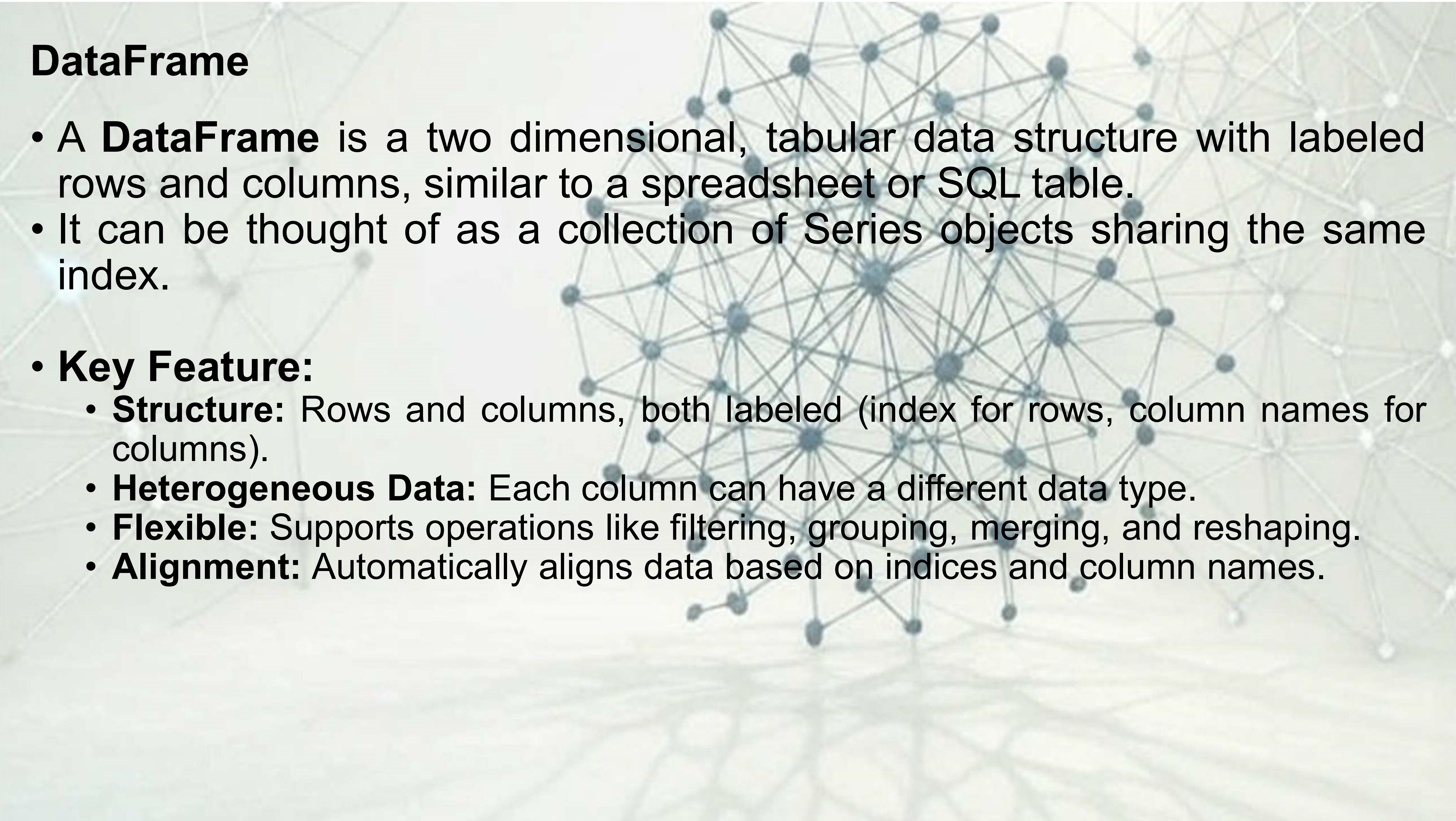


Pandas

created by :
The easylearn academy



DataFrame

- A **DataFrame** is a two dimensional, tabular data structure with labeled rows and columns, similar to a spreadsheet or SQL table.
- It can be thought of as a collection of Series objects sharing the same index.

- **Key Feature:**
  - **Structure:** Rows and columns, both labeled (index for rows, column names for columns).
  - **Heterogeneous Data:** Each column can have a different data type.
  - **Flexible:** Supports operations like filtering, grouping, merging, and reshaping.
  - **Alignment:** Automatically aligns data based on indices and column names.

Series vs DataFrame

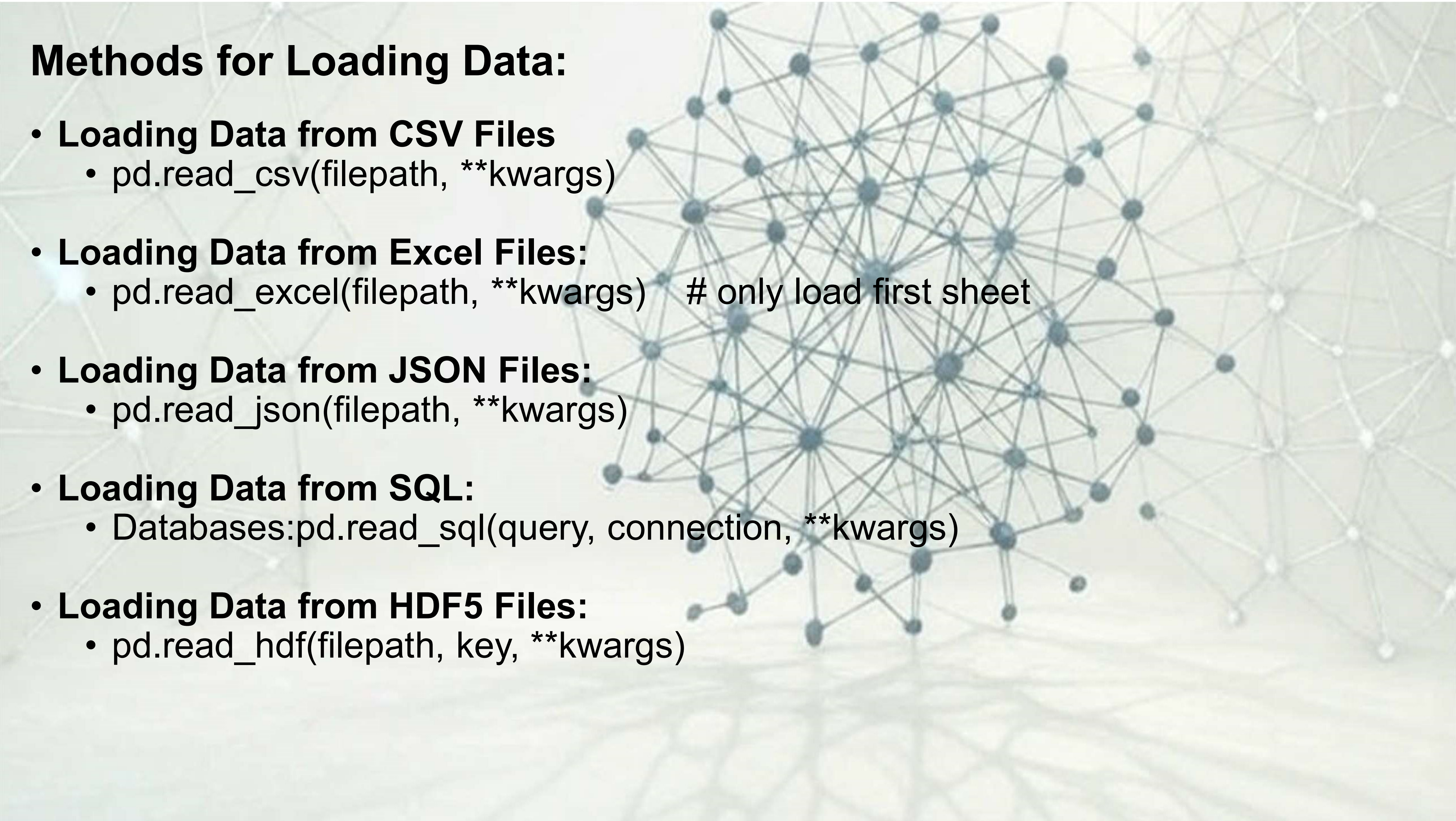
Feature	Series	Dataframe
Definition	A one-dimensional labeled array	A two-dimensional labeled data structure (table)
Structure	Like a single column (or row) of data	Like a full table with rows and columns
Dimensions	1D	2D
Index	Single index	Row and column indexes
Columns	Only one, unnamed or with a name	One or more named columns
Data Type	Homogeneous (same type) usually, but can be mixed	Heterogeneous (each column can be a different type)
Creation Example	pd.Series([10, 20, 30])	pd.DataFrame({'a': [10, 20], 'b': [30, 40]})
Use Case	Ideal for a single column or row of data	Ideal for working with full datasets

Create a sample DataFrame

```
•   import pandas as pd
•   data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30, 35],
            'Salary': [50000, 60000, 75000]
            }
•   df = pd.DataFrame(data)
•   print("Sample DataFrame:")
•   print(df)
```

- **Output:**

```
     Name  Age  Salary
0   Alice   25   50000
1     Bob   30   60000
2 Charlie   35   75000
```



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 DataFrame from Database:

- import pandas as pd
- df = pd.read_csv(filepath_or_buffer, sep=',', header='infer', names=None, index_col=None, usecols=None, dtype=None, na_values=None, parse_dates=None, encoding=None, ...)

- **Key parameter:**
  - **filepath_or_buffer** (required):
    - Specifies the file path (local or URL) or a filelike object (e.g., StringIO) containing the CSV data.
    - **Example:**
      - df = pd.read_csv('data.csv')
      - df = pd.read_csv('https://example.com/data.csv')
  - **sep**(default: ','):
    - Defines the delimiter used in the CSV file (e.g., ',', ';', '\t').Use this when the file uses a delimiter other than a comma.
    - **Example**:
      - df = pd.read_csv('data.txt', sep='\t') # For tabseparated file

Key parameter of read_csv()

- **Delimiter**(alias for sep):
  - Same as sep. Provided for compatibility.
  - **Example**:

```
• df = pd.read_csv('data.csv', delimiter='|')  # For pipeseparated file
```

- **header**(default: 'infer'):
  - Specifies which row(s) to use as column names.
  - **Option:**
    - **0**: Use the first row as headers.
    - **None**: No header; columns are assigned integer indices (0, 1, 2, ...).
    - **List of integers**: Use multiple rows as headers (for MultiIndex).
    - **'infer'**: Automatically detect if the first row is a header.
  - **Example**:

```
• df = pd.read_csv('data.csv', header=None)  # No header in the file
```

- **Names**:
  - List of column names to use. If header=None, this assigns custom column names. If header=0, this overrides the file’s header.
  - **Example**:

```
• df = pd.read_csv('data.csv', header=None, names=['A', 'B', 'C'])
```

Key parameter of read_csv()

- **index_col**(default: 'None'):
  - Column(s) to use as the DataFrame’s index. Can be a column name, index (integer), or list for MultiIndex.
  - **Example**:
    - df = pd.read_csv('data.csv', index_col='ID') # Use 'ID' column as index
- **usecols**(default: 'None'):
  - Specifies a subset of columns to load (by name or index). Reduces memory usage for large files.
  - **Example**:
    - df = pd.read_csv('data.csv', usecols=['Name', 'Age']) # Load only these columns
- **dtype**(default: 'None'):
  - Specifies the data type for columns (dictionary or single type). Useful for optimizing memory or ensuring correct types.
  - **Example**:
    - df = pd.read_csv('data.csv', dtype={'Age': int, 'Score': float})
- **na_values(**default: 'None'):
  - Defines additional strings to treat as missing values (NaN). Pandas already recognizes values like '', 'NA', 'NaN', etc.
  - **Example**:
    - df = pd.read_csv('data.csv', na_values=['missing', 'N/A'])

Key parameter of read_csv()

- **keep_default_na**(default: 'True'):
  - If `False`, prevents default NaN values (e.g., ``'NA'`, ``'NaN'`) from being parsed as missing.
  - **Example**:
    - `df = pd.read_csv('data.csv', na_values=['missing'], keep_default_na=False)`

- **missing_values(**default: 'None'):
  - Deprecated in newer versions; use `na_values` instead.

- **skiprows(**default: 'None'):
  - Skips specified rows (integer, list of integers, or callable). Useful for skipping metadata or corrupted rows.
  - **Example:**
    - `df = pd.read_csv('data.csv', skiprows=2)  # Skip first two rows`

- **nrows**(default: 'None'):
  - Limits the number of rows to read. Useful for large files when only a sample is needed.
  - **Example**:
    - `df = pd.read_csv('data.csv', nrows=100)    # Read only 100 rows`

Key parameter of read_csv()

- **encoding(**default: 'None'):
  - Specifies the file encoding (e.g., `'utf-8'`, `'latin1'`) for non-standard text files.
  - **Example**:

```
•   df = pd.read_csv('data.csv', encoding='latin1')         #encoding="utf-8"
```

- **parse_dates**(default: 'False'):
  - Columns to parse as datetime. Can be `True`, a list of column names, or a list of lists for combining columns.
  - **Example**:

```
•   df = pd.read_csv('data.csv', parse_dates=['Date'])  # Parse 'Date' as datetime
```

- **date_format(**default: 'None'):
  - Specifies the format for parsing dates (used with 'parse_dates').
  - **Example**:

```
•   df = pd.read_csv('data.csv', parse_dates=['Date'], date_format='%Y-%m-%d')
```

- **chunksize**(default: 'None'):
  - Reads the file in chunks, returning a `TextFileReader` object for iteration. Useful for very large files.
  - **Example**:

```
•   for chunk in pd.read_csv('data.csv', chunksize=1000):
•           process_chunk(chunk)  # Process 1000 rows at a time
```

Key parameter of read_csv()

- **compression(**default: 'infer'):
  - Handles compressed files (e.g., `'gzip'`, `'bz2'`, `'zip'`, `'xz'`). If `'infer'`, detects compression from file extension.
  - **Example**:

    ```
    • df = pd.read_csv('data.csv.gz', compression='gzip')
    ```

- **skip_blank_lines**(default: 'True'):
  - If `True`, skips blank lines; if `False`, treats them as rows with NaN values.
  - **Example**:

    ```
    • df = pd.read_csv('data.csv', skip_blank_lines=False)
    ```

- **low_memory(**default: 'True'):
  - Processes the file in chunks internally to save memory. Set to `False` for faster reading if memory is not a constraint.
  - **Example**:

    ```
    • df = pd.read_csv('data.csv', low_memory=False)
    ```

Attributes of DataFrame

Attribute	Syntax	Description
index	df.index	Returns the index (row labels) of the DataFrame.
Values	df.values	Returns the underlying data as a NumPy array.
dtype	df.dtype	Returns the data type of each column.
columns	df.columns	Returns the column labels of the DataFrame.
shape	df.shape	Returns a tuple representing the dimensions of the DataFrame (rows, columns).
ndim	df.ndim	Returns the number of dimensions of the DataFrame.
size	df.size	Returns the total number of elements in the DataFrame (rows × columns).
empty	df.empty	Indicates whether the DataFrame is empty (no rows or columns).
axes	df.axes	Returns a list of the row and column axis labels (`[index, columns]`).
T(Transpose)	df.T	Returns the transpose of the DataFrame (swaps rows and columns).

Index

- **Description**:
  - Returns the index (row labels) of the DataFrame.
- **Syntax**:
  - df.index

```
import pandas as pd
data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
          'Age': [25, 30, 35],
          'Salary': [50000, 60000, 75000]
          }
df = pd.DataFrame(data)
print(df.index)
```

- **Output**:
  - RangeIndex(start=0, stop=3, step=1)

Values

- **Description**:
  - Returns the underlying data as a NumPy array.
- **Syntax**:
  - df.values

```
import pandas as pd
data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30, 35],
            'Salary': [50000, 60000, 75000]
        }
df = pd.DataFrame(data)
print(df.values)
```

- **Output**:

```
[['Alice' 25 50000]
 ['Bob' 30 60000]
 ['Charlie' 35 75000]]
```

Dtype

- **Description**:
  - Returns the data types of each column.
- **Syntax**:
  - df.dtype

```
•  import pandas as pd
•  data = {
               'Name': ['Alice', 'Bob', 'Charlie'],
               'Age': [25, 30, 35],
               'Salary': [50000, 60000, 75000]
          }
•  df = pd.DataFrame(data)
•  print(df.dtype)
```

- **Output**:
  - Name object
  - Age int64
  - Salary int64
  - dtype: object

columns

- **Description**:
  - Returns the column labels of the DataFrame.
- **Syntax**:
  - df.columns

```
•  import pandas as pd
•  data = {
              'Name': ['Alice', 'Bob', 'Charlie'],
              'Age': [25, 30, 35],
              'Salary': [50000, 60000, 75000]
          }
•  df = pd.DataFrame(data)
•  print(df.columns)
```

- **Output**:
  - Index(['Name', 'Age', 'Salary'], dtype='object')

Shape

- **Description**:
  - Returns the number of dimensions of the DataFrame.
- **Syntax**:
  - df.shape

```
•  import pandas as pd
•  data = {
              'Name': ['Alice', 'Bob', 'Charlie'],
              'Age': [25, 30, 35],
              'Salary': [50000, 60000, 75000]
          }
•  df = pd.DataFrame(data)
•  print(df.shape)
```

- **Output**:
  - (3, 3)

Ndim

- **Description**:
  - Returns the number of dimensions of the DataFrame (always 1 for a DataFrame).
- **Syntax**:
  - df.ndim

```
•  import pandas as pd
•  data = {
             'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
         }
•  df = pd.DataFrame(data)
•  print(df.ndim)
```

- **Output**:
  - 2

Size

- **Description**:
  - Returns the total number of elements in the DataFrame (rows × columns).
- **Syntax**:
  - df.size

```
import pandas as pd
data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30, 35],
            'Salary': [50000, 60000, 75000]
        }
df = pd.DataFrame(data)
print(df.size)
```

- **Output**:
  - 9

empty

- **Description**:
  - Indicates whether the DataFrame is empty (no rows or columns).
- **Syntax**:
  - df.empty

```
import pandas as pd
data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30, 35],
            'Salary': [50000, 60000, 75000]
        }
df = pd.DataFrame(data)
print(df.empty)
```

- **Output**:
  - False

```
import pandas as pd
empty_df = pd.DataFrame()
print(empty_df)
```

- **Output**:

```
Empty DataFrame
Columns: []
Index: []
```

axes

- **Description**:
  - Returns a list of the row and column axis labels (`[index, columns]`).
- **Syntax**:
  - df.axes

```
•  import pandas as pd
•  data = {
              'Name': ['Alice', 'Bob', 'Charlie'],
              'Age': [25, 30, 35],
              'Salary': [50000, 60000, 75000]
          }
•  df = pd.DataFrame(data)
•  print(df.axes)
```

- **Output**:
  - [RangeIndex(start=0, stop=3, step=1), Index(['Name', 'Age', 'Salary'], dtype='object')]

T(Transpose)

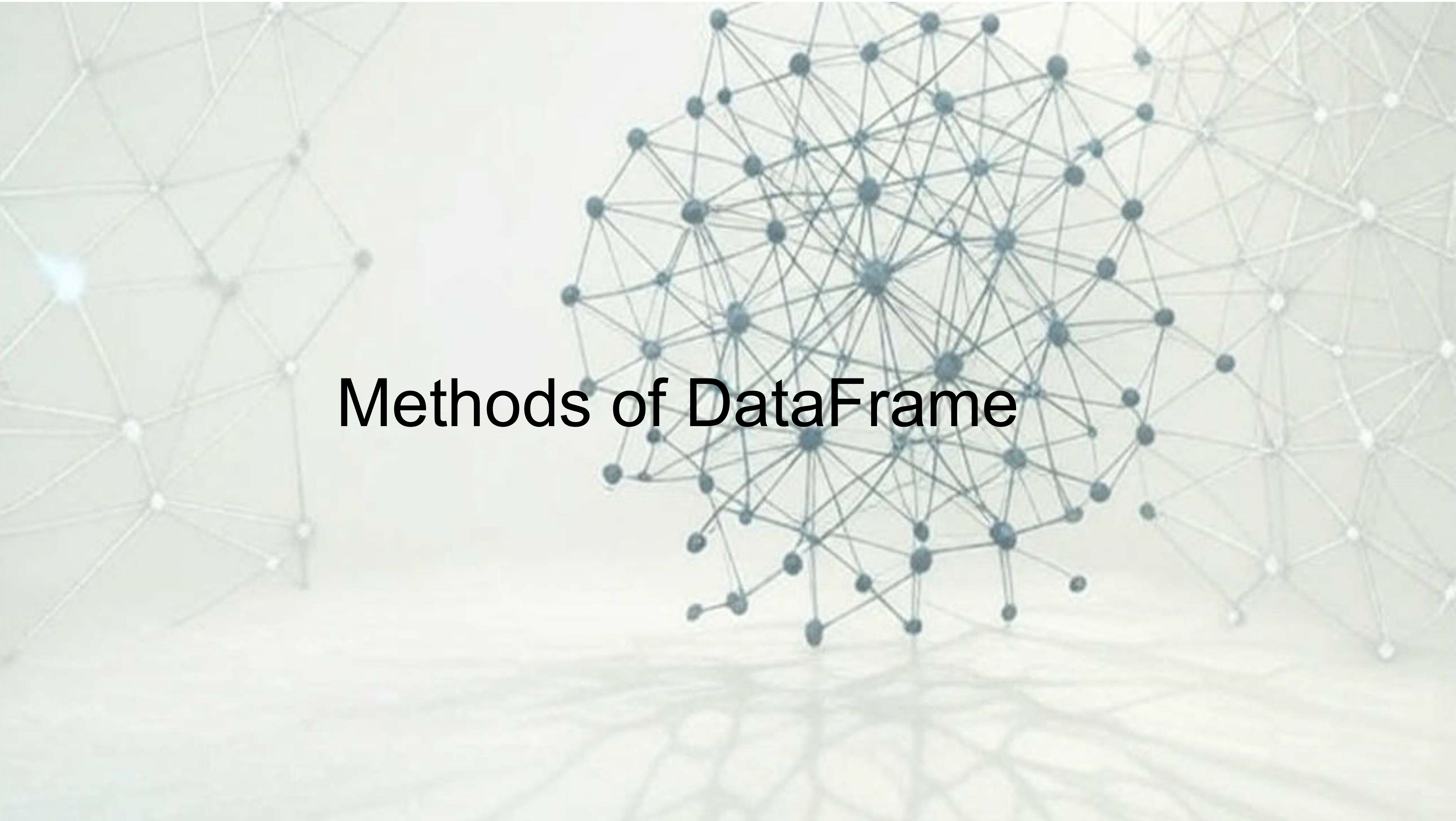
- **Description**:
  - Returns the transpose of the DataFrame (swaps rows and columns).
- **Syntax**:
  - df.T

```
import pandas as pd
data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30, 35],
            'Salary': [50000, 60000, 75000]
        }
df = pd.DataFrame(data)
print(df.T)
```

- **Output**:

```
              0      1       2
Name     Alice    Bob  Charlie
Age         25     30       35
Salary   50000  60000    75000
```

Methods of DataFrame



Methods of DataFrame: Data Inspection and Summary

Method	Syntax	Description
DataFrame()	pd.DataFrame()	Create a **DataFrame** from a dictionary, list, or array.
head(n)	df.head(3)	Returns the **first n elements** of the DataFrame. (Defaults to 5.)
tail(n)	df.tail(3)	Returns the **last n elements** of the DataFrame. (Defaults to 5.)
type()	type(df)	Python’s builtin type() function to check the **type** of a Datatable.
describe()	df.describe()	Generates **descriptive statistics** (count, mean, std, min, quartiles, max) for **numeric** DataFrame.
info()	df.info()	Provides a summary of the DataFrame, including column names, data types, and non-null counts.
value_counts()	df. value_counts()	Counts unique combinations (typically on Series).

DataFrame()

- **Description**:
  - Create a DataFrame from a dictionary, list, or array.
- **Syntax**:
  - pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
- **Key Parameters**:
  - **data**: Input data (e.g., list, dict, ndarray).
  - **index**: Optional index labels (defaults to 0, 1, 2, ...).
  - **columns:** The column labels for the DataFrame. Can be a list or array of labels.
  - **dtype**: Data type for the Series (e.g., int64, float64).
  - **copy**: Whether to copy the input data (default is False).

```
•   import pandas as pd
•   import numpy as np
•   data = {
                'Name': ['Alice', 'Bob', 'Charlie'],
                'Age': [25, 30, 35],
                'Salary': [50000, 60000, 75000]
           }
•   df = pd.DataFrame(data)
•   print(df)
```

- **Output:**

```
•        Name  Age  Salary
•   0   Alice   25   50000
•   1     Bob   30   60000
•   2  Charlie   35   75000
```

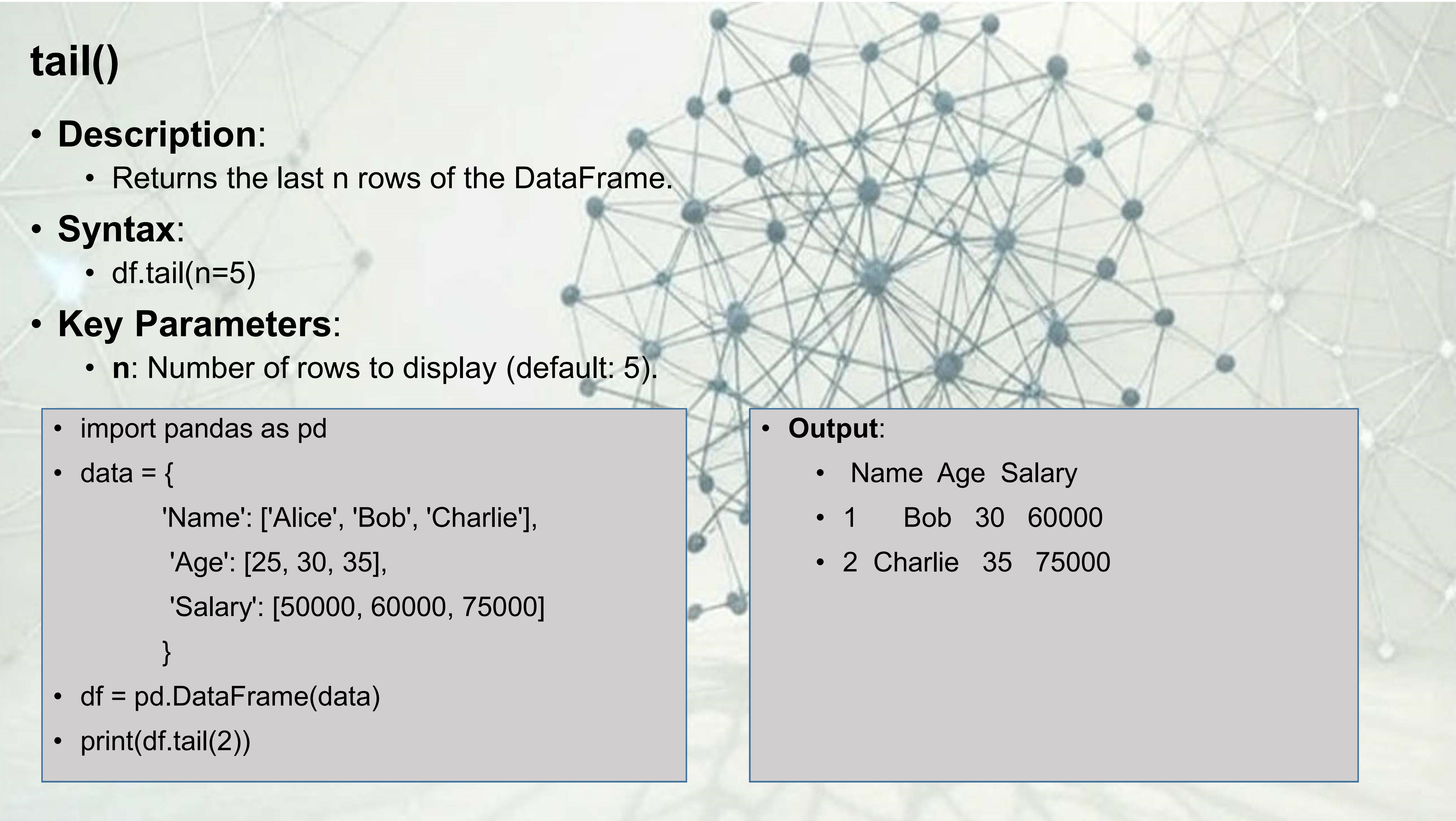
head()

- **Description**:
  - Returns the **first n elements** of the DataFrame. (Defaults to 5.)
- **Syntax**:
  - df.head(n=5)
- **Key Parameters**:
  - **n**: Number of rows to display (default: 5).

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• print(df.head())
```

- **Output**:

```
•      Name  Age  Salary
• 0    Alice   25   50000
• 1      Bob   30   60000
• 2  Charlie   35   75000
```



tail()

- **Description**:
  - Returns the last n rows of the DataFrame.
- **Syntax**:
  - df.tail(n=5)
- **Key Parameters**:
  - **n**: Number of rows to display (default: 5).

```
• import pandas as pd
• data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
• df = pd.DataFrame(data)
• print(df.tail(2))
```

- **Output**:

```
•   Name  Age  Salary
•  1      Bob   30   60000
•  2  Charlie   35   75000
```

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.frame.DataFrame).

```
• import pandas as pd
• data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
• df = pd.DataFrame(data)
• print(type(df))
```

- **Output**:
  - <class 'pandas.core.frame.DataFrame'>

describe()

- **Description**:
  - Generates descriptive statistics (count, mean, std, min, max, quartiles) for numeric columns.
- **Syntax:**
  - df.describe(include='all')
- **Key Parameters**:
  - **include**:Columns to include ('all' for all columns, or specify types like `np.number` or `object`).

```
•  import pandas as pd
•  data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
•  df = pd.DataFrame(data)
•  print(df.describe())
```

- **Output**:

```
•            Age        Salary
•  count   3.0      3.000000
•  mean   30.0  61666.666667
•  std     5.0       12583.057392
•  min     25.0    50000.000000
•  25%     27.5  55000.000000
•  50%     30.0  60000.000000
•  75%     32.5  67500.000000
•  max    35.0  75000.000000
```

info()

- **Description**:
  - Provides a summary of the DataFrame, including column names, data types, and non-null counts.
- **Syntax**:
  - df.info()

```
• import pandas as pd
• data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
• df = pd.DataFrame(data)
• print(df.info())
```

- **Output**:

```
• <class 'pandas.core.frame.DataFrame'>
• RangeIndex: 3 entries, 0 to 2
• Data columns (total 3 columns):
•  #   Column  Non-Null Count  Dtype
• ---  ------  --------------  -----
•  0   Name    3 non-null      object
•  1   Age     3 non-null      int64
•  2   Salary  3 non-null      int64
• dtypes: int64(2), object(1)
• memory usage: 204.0+ bytes
• None
```

value_counts()

- **Description**:
  - Returns a Series containing counts of unique values in a DataFrame column or Series. Often used for frequency analysis.
- **Syntax**:
  - Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)
- **Key Parameters**:
  - **Normalize:** If True, returns relative frequencies (proportions) instead of counts.
  - **Sort:** If True, sorts by counts in descending order.
  - **Ascending:** If True, sorts in ascending order.
  - **Bins:** Groups numeric data into bins (used for continuous data).
  - **Dropna**: If True, excludes NaN values from counts.

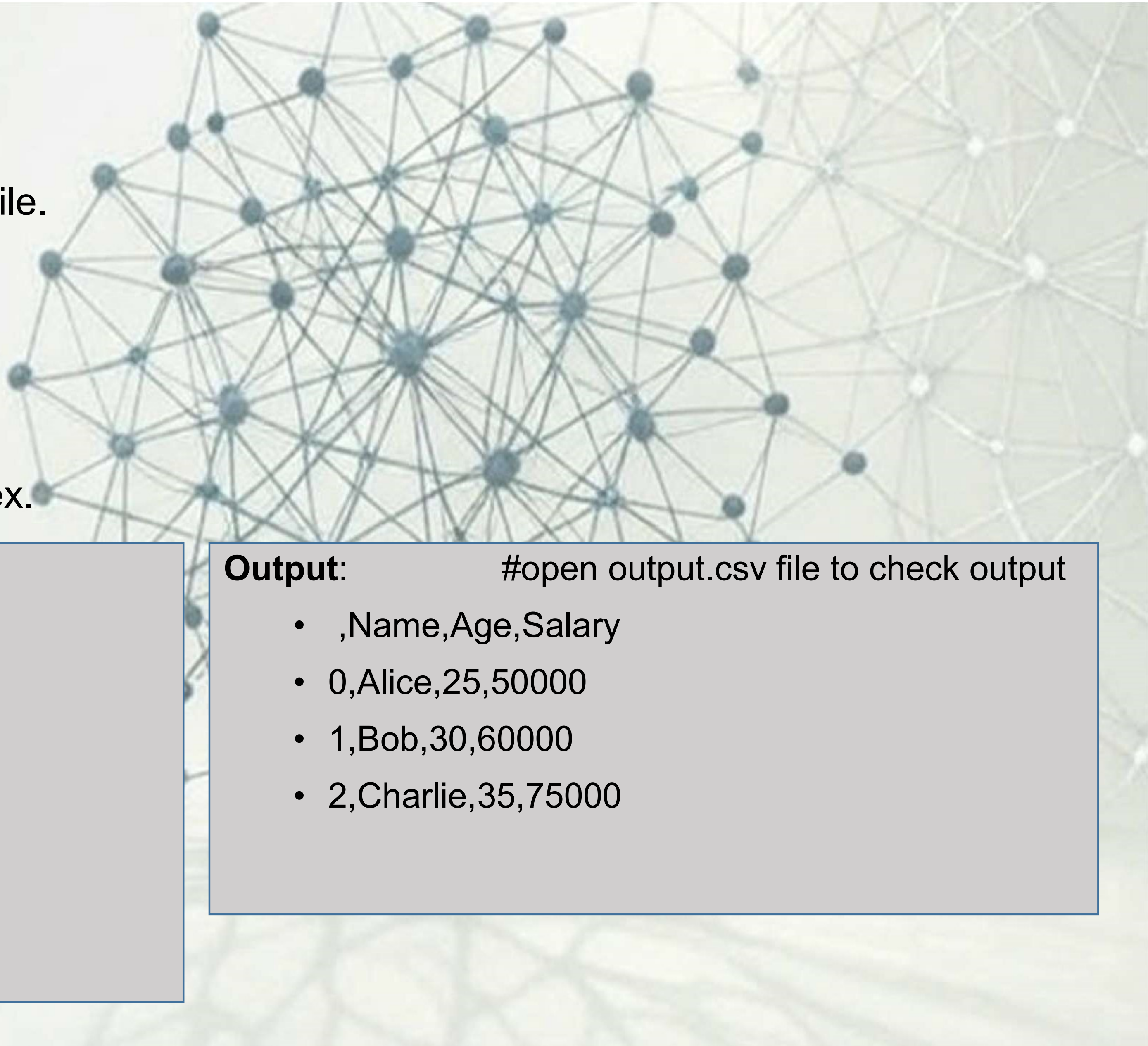
```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• print(df.value_counts())
```

Output:

```
• Name   Age   Salary
• Alice  25.0  50000     1
• Bob    30.0  60000     1
• Name: count, dtype: int64
```

Methods of DataFrame

Method	Syntax	Description
to_csv()	df.to_csv(path, index=True)	Writes the DataFrame to a CSV file.
to_excel()	df.to_excel(path, sheet_name='Sheet1', index=True)	Writes the DataFrame to an Excel file.
read_csv()	pd.read_csv(path, sep=',', encoding='utf-8')	Reads a CSV file into a DataFrame.
read_excel()	pd.read_excel(path, sheet_name=0)	Reads an Excel file into a DataFrame.



to_csv()

- **Description**:
  - Writes the DataFrame to a CSV file.
- **Syntax**:
  - df.to_csv(path, index=True)
- **Key Parameters**:
  - **path**: File path or object.
  - **index**: If `True`, includes the index.

```
import pandas as pd
data = {
        'Name': ['Alice', 'Bob', 'Charlie'],
         'Age': [25, 30, 35],
         'Salary': [50000, 60000, 75000]
        }
df = pd.DataFrame(data)
df.to_csv('output.csv')
```

Output: #open output.csv file to check output

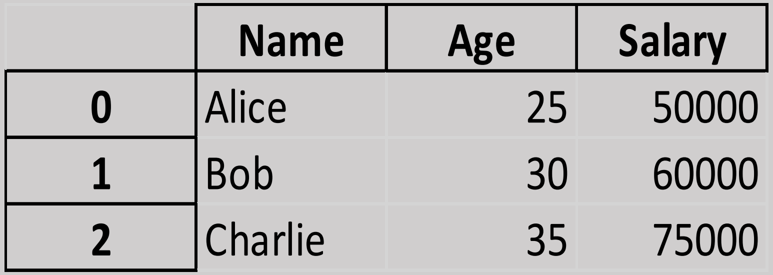
```
,Name,Age,Salary
0,Alice,25,50000
1,Bob,30,60000
2,Charlie,35,75000
```

to_excel()

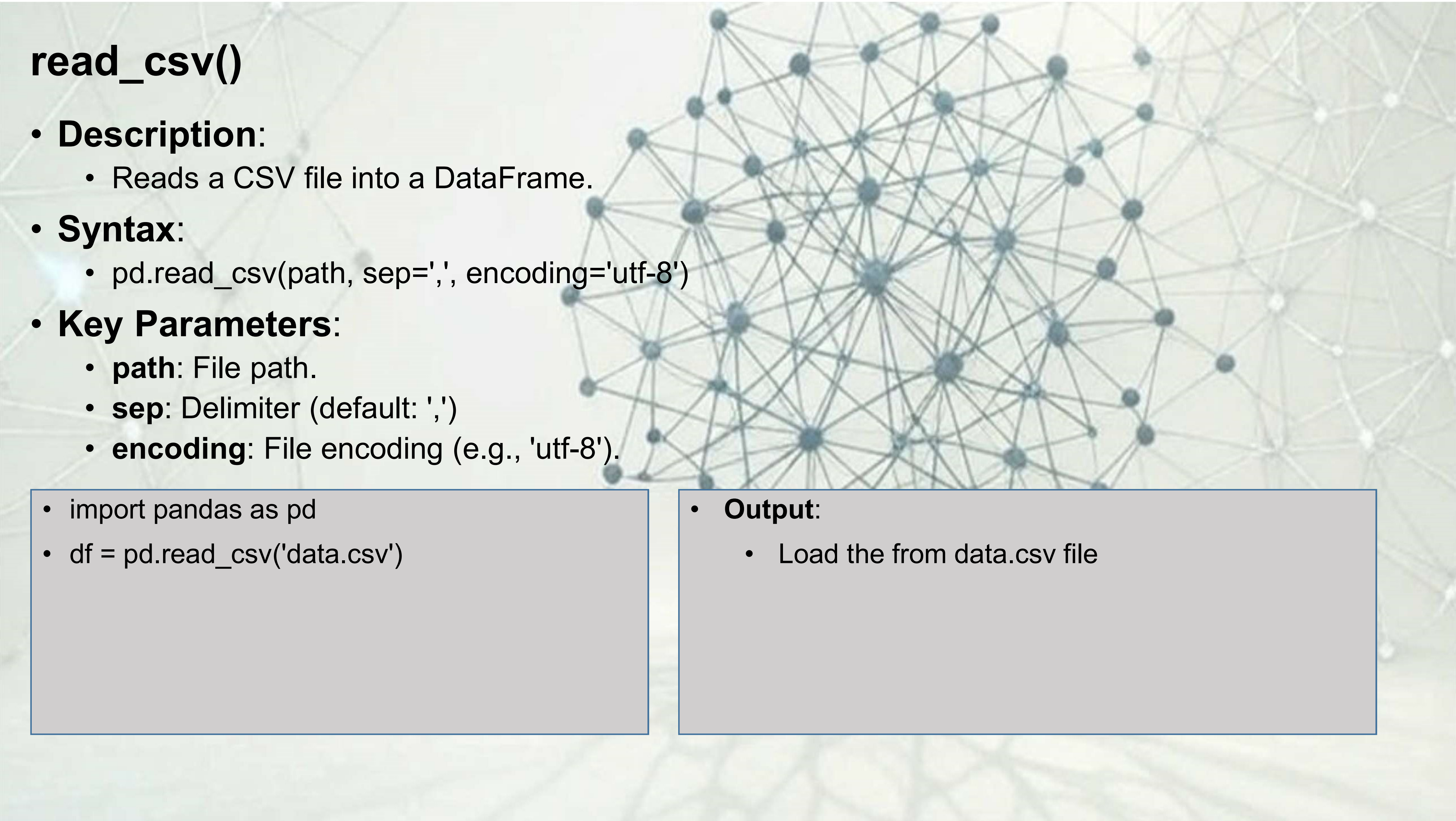
- **Description**:
  - Writes the DataFrame to an Excel file.
- **Syntax**:
  - df.to_excel(path, sheet_name='Sheet1', index=True)
- **Key Parameters**:
  - **path**: File path.
  - **sheet_name**: Name of the sheet.
  - **index**: If True, includes the index.

```
import pandas as pd
data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
df = pd.DataFrame(data)
df.to_excel('output.xlsx')
```

- **Output**: #open output.xlsx file to check output



	Name	Age	Salary
0	Alice	25	50000
1	Bob	30	60000
2	Charlie	35	75000



read_csv()

- **Description**:
  - Reads a CSV file into a DataFrame.
- **Syntax**:
  - pd.read_csv(path, sep=',', encoding='utf-8')
- **Key Parameters**:
  - **path**: File path.
  - **sep**: Delimiter (default: ',')
  - **encoding**: File encoding (e.g., 'utf-8').

- import pandas as pd
- df = pd.read_csv('data.csv')

- **Output**:
  - Load the from data.csv file

read_excel()

- **Description**:
  - Reads an Excel file into a DataFrame.
- **Syntax**:
  - pd.read_excel(path, sheet_name=0)
- **Key Parameters**:
  - **path**: File path.
  - **sheet_name**: Sheet to read (name or index).

- import pandas as pd
- df = pd.read_excel('data.xlsx')

Output:

- Load the from exal file

Methods of DataFrame: Data Selection and Filtering

Method	Syntax	Description
loc[]	df.loc[rows, columns]	Access rows and columns by labels or boolean arrays.
iloc[]	df.iloc[rows, columns]	Access rows and columns by integer positions.
at[]	df.at[row_label, column_label]	Fast access to a single value by label.
iat[]	df.iat[row_index, column_index]	Fast access to a single value by integer position.
query(expr)	df.query(expr)	Query the DataFrame using a boolean expression.
filter(items/like/regex)	df.filter(items=None, like=None, regex=None, axis=0)	Subset columns based on names or patterns.
select_dtypes(include/exclude)	df.select_dtypes(include=None, exclude=None)	Select columns by data type.
duplicated()	df.duplicated(subset=None, keep='first')	Identifies duplicate rows.
drop_duplicates()	df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)	Removes duplicate rows.

loc[]

- **Description**:
  - Access rows and columns by labels or boolean arrays.
- **Syntax**:
  - df.loc[rows, columns]
- **Key Parameters**:
  - **rows**: Row labels or boolean array.
  - **columns**: Column labels or list of labels.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• print(df.loc[df['Age'] > 30, ['Name', 'Age']])
```

- **Output**:

```
•       Name  Age
•    2  Charlie   35
```

iloc[]

- **Description**:
  - Access rows and columns by integer positions.
- **Syntax**:
  - df.iloc[rows, columns]
- **Key Parameters**:
  - **rows**: Integer indices or slice.
  - **columns**: Integer indices or slice.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• df.iloc[0:3, 1:3]
```

- **Output**:

```
•       Age  Salary
•   0   25   50000
•   1   30   60000
•   2   35   750000
```

at[]

- **Description**:
  - Access a single value by row and column label (faster than `loc`).
- **Syntax**:
  - df.at[row_label, column_label
- **Key Parameters**:
  - **row_label**: Row label.
  - **column_label**: Column label.

```
• import pandas as pd
• data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
• df = pd.DataFrame(data)
• print(df.at[0, 'Name'])
```

- **Output**:
  - 'Alice'

iat[]

- **Description**:
  - Access a single value by integer position (faster than `iloc`)
- **Syntax**:
  - df.iat[row_index, column_index]
- **Key Parameters**:
  - **row_index**: Integer row position.
  - **column_index**: Integer column position.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• print(df.iat[0, 1])
```

- **Output**:
  - 25

query()

- **Description**:
  - Filters rows using a string expression.
- **Syntax**:
  - df.query(expr)
- **Key Parameters**:
  - **expr**: String expression (e.g., 'age > 30').

```
•  import pandas as pd
•  data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
•  df = pd.DataFrame(data)
•  print(df.query('Age > 30 and Salary == 75000'))
```

- **Output**:

```
•        Name  Age  Salary
•    2  Charlie   35   75000
```

filter(items/like/regex)

- **Description**:
  - Subset the DataFrame rows or columns based on the specified index or column labels.
- **Syntax**:
  - df.filter(items=None, like=None, regex=None, axis=0)
- **Key Parameters**:
  - **items**: List of column or index labels to select (exact matches).
  - **like**: String to match in column or index labels (partial matches).
  - **regex**: Regular expression to match in column or index labels.
  - **axis**: Axis to filter on (0 for index, 1 for columns; default is 0).

filter(items/like/regex)

```
• import pandas as pd
• data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30,np.nan],
            'Salary': [50000, 60000, 75000]
        }
• df = pd.DataFrame(data)
• print(df.filter(items=['Age'], axis=1))
                    • or
• print(df.filter(items=[0], axis=0))
```

- **Output**:

```
•    Age
•   0  25.0
•   1  30.0
•   2   NaN
```

- **or**

- **Output**:

```
•       Name   Age  Salary
•   0  Alice  25.0   50000
```

select_dtypes(include/exclude)

- **Description**:
  - Select columns from a DataFrame based on specified data types.
- **Syntax**:
  - df.select_dtypes(include=None, exclude=None)
- **Key Parameters**:
  - **include**: Data types to include (e.g., 'int64', 'float64', 'object', or numpy dtype).
  - **exclude**: Data types to exclude (e.g., 'int64', 'float64', 'object', or numpy dtype).

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
          'Age': [25, 30,np.nan],
          'Salary': [50000, 60000, 75000]
       }
• df = pd.DataFrame(data)
• print(df.select_dtypes(include=int, exclude=None))
```

- **Output**:

```
•      Salary
•  0   50000
•  1   60000
•  2   75000
```

duplicated()

- **Description**:
  - Identifies duplicate rows in a DataFrame and returns a boolean Series where True indicates a duplicate row.
- **Syntax**:
  - df.duplicated(subset=None, keep='first')
- **Key Parameters**:
  - **subset**: Column label(s) to consider for identifying duplicates (default: None, uses all columns).
  - **keep**: 'first': Mark duplicates as True except for the first occurrence (default).
  - **last**: Mark duplicates as True except for the last occurrence.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'Bob'],
          'Age': [25, 30, 35, 25, 30],
          'Salary': [50000, 60000, 70000, 50000, 60000]
}
• df = pd.DataFrame(data)
• print(df.duplicated( keep='first'))
```

- **Output**:

```
  • 0    False
  • 1    False
  • 2    False
  • 3     True
  • 4     True
  • dtype: bool
```

drop_duplicated()

- **Description**:
  - Removes duplicate rows from a DataFrame and returns a new DataFrame with unique rows.
- **Syntax**:
  - df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
- **Key Parameters**:
  - **subset**: Column label(s) to consider for identifying duplicates (default: None, uses all columns).
  - **keep**: 'first': Keep the first occurrence of each duplicate (default).
  - **'last'**: Keep the last occurrence of each duplicate.
  - **False**: Drop all duplicates.
  - **inplace**: If True, modifies the DataFrame in place (default: False). ignore_index: If True, resets the index after dropping duplicates

```
•  import pandas as pd
•  data = {
           'Name': ['Alice', 'Bob', 'Charlie'],
            'Age': [25, 30, 35],
            'Salary': [50000, 60000, 75000]
           }
•  df = pd.DataFrame(data)
```

```
•   print(df.drop_duplicates(keep='first'))
•   Output:
    •         Name  Age  Salary
    •   0    Alice   25   50000
    •   1      Bob   30   60000
    •   2  Charlie   35   70000
```

Methods of DataFrame: Data Manipulation

Method	Syntax	Description
copy()	DataFrame.copy(deep=True)	Creates a deep copy of the DataFrame.
assign(**kwargs)	DataFrame.assign(**kwargs)	Adds new columns or modifies existing ones.
pop(column)	DataFrame.pop(item)	Removes and returns a column.
drop(labels, axis)	df.drop(labels, axis=0, inplace=False)	Drops specified rows or columns.
rename(columns/index)	df.rename(columns=None, index=None, inplace=False)	Renames columns or index labels.
replace(to_replace, value)	df.replace(to_replace, value, inplace=False)	Replaces values in the DataFrame.
astype(dtype)	df.astype(dtype)	Converts data types of columns.
mask(cond, other)	DataFrame.mask(cond, other=None, inplace=False)	Replaces values where condition is True.

copy()

- **Description**:
  - Creates a deep or shallow copy of the DataFrame.
- **Syntax**:
  - df.copy(deep=True)
- **Key Parameters**:
  - **Deep:** If True, creates a deep copy (default). If False, creates a shallow copy.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• a=df.copy()
• print(a)
```

- **Output:**

```
•    Name  Age  Salary
•  0   Alice   25   50000
•  1     Bob   30   60000
•  2  Charlie   35   75000
```

assign()

- **Description**:
  - Adds new columns or modifies existing ones in a DataFrame, returning a new DataFrame.
- **Syntax**:
  - df.assign(**kwargs)
- **Key Parameters**:
  - **Kwargs:** Column names and values (can be scalars, lists, or functions).

```
import pandas as pd
data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
df = pd.DataFrame(data)
print(df.assign(number=102106478))
```

Output:

```
      Name  Age  Salary     number
0    Alice   25   50000  102106478
1      Bob   30   60000  102106478
2  Charlie   35   75000  102106478
```

pop()

- **Description**:
  - Removes and returns a column from the DataFrame.
- **Syntax**:
  - df.pop(item)
- **Key Parameters**:
  - **Item:** Label of the column to remove.

```
import pandas as pd
data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            'number':[102106478, 102106478, 102106478]
            }
df = pd.DataFrame(data)
df.pop('number')
print(df)
```

- **Output:**

```
      Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   75000
```

drop()

- **Description**:
  - Removes specified rows or columns.
- **Syntax**:
  - df.drop(labels, axis=0, inplace=False)
- **Key Parameters**:
  - **labels**: Row or column labels to drop.
  - **axis**: 0 for rows, 1 for columns.
  - **inplace**: If `True`, modifies the DataFrame in place.

```
• import pandas as pd
• data = {
            'Name': ['Alice', 'Bob', 'Charlie'],
             'Age': [25, 30, 35],
             'Salary': [50000, 60000, 75000]
            }
• df = pd.DataFrame(data)
• df.drop('Age',axis=1,inplace=True)
• print(df)
```

- **Output**:

```
•        Name  Salary
•  0    Alice   50000
•  1      Bob   60000
•  2  Charlie   75000
```

rename()

- **Description**:
  - Renames columns or index labels.
- **Syntax**:
  - df.rename(columns=None, index=None, inplace=False)
- **Key Parameters**:
  - **columns**: Dict of old to new column names.
  - **index**: Dict of old to new index labels.
  - **inplace**: If `True`, modifies in place.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• print(df.rename(columns={'Age': 'age'}))
```

- **Output:**

```
•        Name  age  Salary
•   0   Alice   25   50000
•   1     Bob   30   60000
•   2  Charlie   35   75000
```

replace()

- **Description**:
  - Replaces values in the DataFrame.
- **Syntax**:
  - df.replace(to_replace, value, inplace=False)
- **Key Parameters**:
  - **to_replace**: Value(s) to replace (scalar, list, dict).
  - **value**: Replacement value(s).
  - **inplace**: If `True`, modifies in place.

```
•  import pandas as pd
•  data = {
         'Name': ['Alice', 'Bob', 'Charlie'],
          'Age': [25, 30, 35],
          'Salary': [50000, 60000, 75000]
         }
•  df = pd.DataFrame(data)
•  print(df.replace(25,20)
```

- **Output:**

```
•        Name  Age  Salary
•  0    Alice   20   50000
•  1      Bob   30   60000
•  2  Charlie   35   75000
```

astype()

- **Description**:
  - Converts data types of columns.
- **Syntax**:
  - df.astype(dtype)
- **Key Parameters**:
  - **dtype**: Data type or dict of column-to-type mappings.

```
• import pandas as pd
• data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
• df = pd.DataFrame(data)
• print(df.astype('float64'))
```

- **Output:**

```
•  0   25.0
•  1   30.0
•  2   35.0
•  Name: Age, dtype: float64
```

mask()

- **Description**:
  - Replaces values where a condition is True with a specified value.
- **Syntax**:
  - df.mask(cond, other=None, inplace=False)
- **Key Parameters**:
  - **Cond:** Boolean condition or callable.
  - **Other:** Value to replace where condition is True.
  - **Inplace**: If True, modifies the DataFrame in place.

```
import pandas as pd
data = {
          'Name': ['Alice', 'Bob', 'Charlie'],
           'Age': [25, 30, 35],
           'Salary': [50000, 60000, 75000]
          }
df = pd.DataFrame(data)
mask_df = df.mask(df['Age'] < 28)
print(mask_df)
```

- **Output:**

```
        Name   Age   Salary
0       NaN   NaN      NaN
1       Bob  30.0  60000.0
2   Charlie  35.0  75000.0
```