Pandas

Data Science using Python

Avinash A S

Module 1: Introduction to Pandas

1. **Pandas**

Pandas is an open-source Python library used for data manipulation and analysis. It provides data structures and functions needed to work efficiently with structured data, particularly tabular data.

Key Features of Pandas:

- **Fast and efficient** for manipulating data, including missing data handling.
- **Flexible data structures**: It provides the Series and DataFrame structures to hold and manipulate data in various formats (e.g., CSV, Excel).
- **Integrated with other libraries**: Pandas works well with libraries like NumPy, Matplotlib, and Seaborn for numerical computations and visualization.

Install Pandas using pip:

```
pip install pandas
```

To verify if Pandas is installed, you can import it in your Python environment:

```
import pandas as pd
print(pd.__version__)
```

2. **Data Structures in Pandas**

Pandas provides two primary data structures:

- **Series**: 1-dimensional array-like structure (similar to a column in a spreadsheet).
- **DataFrame**: 2-dimensional table-like structure (similar to an entire spreadsheet or SQL table).

Series

Creating a Series from a List:

```
import pandas as pd

data = [10, 20, 30, 40]
series = pd.Series(data)
print(series)
```

Output:

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

Here, 0, 1, 2, 3 are the indices, and 10, 20, 30, 40 are the values.

Creating a Series from a Dictionary:

```
import pandas as pd

data = {'a': 10, 'b': 20, 'c': 30}
series = pd.Series(data)
print(series)
```

Output:

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

In this above, the dictionary keys become the index labels.

Accessing Elements in a Series:

We can access Series elements using their index or by using their key label:

```
import pandas as pd

data = [10, 20, 30, 40]
series_list = pd.Series(data)
print(series_list[0])

data = {'a': 10, 'b': 20, 'c': 30}
series_dict = pd.Series(data)
print(series_dict['b'])
```

Output:

```
10

20
```

Vectorized Operations:

We can able to apply operations directly to the entire Series:

```
import pandas as pd

data = [10, 20, 30, 40]
series_list = pd.Series(data)
print(series_list + 5)  # Adds 5 to each element

data = {'a': 10, 'b': 20, 'c': 30}
series_dict = pd.Series(data)
print(series_dict + 5)  # Adds 5 to each element
```

Output:

```
0    15
1    25
2    35
3    45
dtype: int64

a    15
b    25
c    35
dtype: int64
```

Statistical Methods for Series:

mean(): Calculates the average of all the elements.
min(): Returns the smallest value among the elements.
max(): Returns the largest value among the elements.
std(): Computes the standard deviation, measuring the spread of the data.
sum(): Adds up all the elements in the Series or DataFrame column

```
import pandas as pd

data = [10, 20, 30, 40]
series_list = pd.Series(data)
print(series_list.mean())  # 25.0
print(series_list.sum())  # 100
print(series_list.min())  # 10
print(series_list.max())  # 40
print(series_list.std())  # 12.909944487358056

data = {'a': 30, 'b': 40, 'c': 50}
series_dict = pd.Series(data)
print(series_dict.mean())  # 40.0
print(series_dict.sum())  # 120
print(series_dict.min())  # 30
print(series_dict.max())  # 50
print(series_dict.std())  # 10.0
```

DataFrame

Creating a DataFrame from a Dictionary:

```
import pandas as pd

data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Occupation': ['Engineer', 'Doctor', 'Artist']
}

df = pd.DataFrame(data)
print(df)
```

Output:

```
      Name  Age Occupation
0    Alice   25   Engineer
1      Bob   30     Doctor
2  Charlie   35     Artist
```

Creating a DataFrame from a List of Lists:

```
import pandas as pd

data = [['Alice', 25, "BLR"],
        ['Bob', 30, "TRPL"],
        ['Charlie', 35, "GLB"],
        ['Dread Wing', 30, "CYB"],
        ['Bumble Bee', 40, "CYB"],
        ['Arcee', 45, 'CYB']]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Place'])

print(df)
```

Output:

```
      Name  Age Place
0    Alice   25   BLR
1      Bob   30  TRPL
2  Charlie   35   GLB
```

Viewing data

`head()`: Shows the first few rows (default 5 rows).

`tail()`: Shows the last few rows (default 5 rows).

`info()`: Displays information about the DataFrame including data types and memory usage.

`describe()`: Generates descriptive statistics like mean, min, max, and quartiles for numeric columns.

```
import pandas as pd

data = [['Alice', 25, "BLR"],
        ['Bob', 30, "TRPL"],
        ['Charlie', 35, "GLB"],
        ['Dread Wing', 30, "CYB"],
        ['Bumble Bee', 40, "CYB"],
        ['Arcee', 45, 'CYB']]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Place'])
print("Head")
print(df.head(), '\n')
print("Tail")
print(df.tail(), '\n')
print("Description")
print(df.describe(),'\n')
print("Info")
df.info()  # Just Call the method here
```

Output:

```
Head
         Name  Age Place
0       Alice   25   BLR
1         Bob   30  TRPL
2     Charlie   35   GLB
3  Dread Wing   30   CYB
4  Bumble Bee   40   CYB

Tail
         Name  Age Place
1         Bob   30  TRPL
2     Charlie   35   GLB
3  Dread Wing   30   CYB
4  Bumble Bee   40   CYB
5       Arcee   45   CYB
```

```
Description
             Age
count   6.000000
mean   34.166667
std     7.359801
min    25.000000
25%    30.000000
50%    32.500000
       38.750000
max    45.000000

Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    6 non-null      object
 1   Age     6 non-null      int64
 2   Place   6 non-null      object
dtypes: int64(1), object(2)
memory usage: 272.0+ bytes
```

Module 2: Data Manipulation with Pandas

Basic DataFrame Operations

Selecting Columns

```
import pandas as pd

data = [['Alice', 25, "BLR"],
        ['Bob', 30, "TRPL"],
        ['Charlie', 35, "GLB"],
        ['Dread Wing', 30, "CYB"],
        ['Bumble Bee', 40, "CYB"],
        ['Arcee', 45, 'CYB']]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Place'])

print(df['Name'])  # Returns as a Series

print("\n")

print(df[['Name', 'Age']])  # Returns as a DataFrame
```

Output:

```
0         Alice
1           Bob
2       Charlie
3    Dread Wing
4    Bumble Bee
5         Arcee
Name: Name, dtype: object


          Name  Age
0       Alice   25
1         Bob   30
2     Charlie   35
3  Dread Wing   30
4  Bumble Bee   40
5       Arcee   45
```

Selecting Rows (Using loc or iloc)

```
import pandas as pd

# Data as a dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Dread Wing', 'Bumble Bee', 'Arcee'],
    'Age': [25, 30, 35, 30, 40, 45],
    'Place': ['BLR', 'TRPL', 'GLB', 'CYB', 'CYB', 'CYB']
}

# Create DataFrame from dictionary
df = pd.DataFrame(data)

# Select the first row using iloc (position-based)
print(df.iloc[0],"\n")

# Select the first and third rows using iloc
print(df.iloc[[0, 2]],"\n")

# Select the first row using loc (label-based)
print(df.loc[0],"\n")

# Select rows where Age > 30
print(df[df['Age'] > 30],"\n")

# Select rows where Age > 30 and Place is 'CYB'
print(df[(df['Age'] > 30) & (df['Place'] == 'CYB')],"\n")
```

Output:

```
Name     Alice
Age         25
Place      BLR
Name: 0, dtype: object

      Name  Age Place
0    Alice   25   BLR
2  Charlie   35   GLB

Name     Alice
Age         25
Place      BLR
Name: 0, dtype: object

          Name  Age Place
2      Charlie   35   GLB
4  Bumble Bee   40   CYB
5       Arcee   45   CYB

          Name  Age Place
4  Bumble Bee   40   CYB
5       Arcee   45   CYB
```

Adding, deleting, and renaming columns

Adding Column

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Rahul', 'Ananya', 'Vikram'],
        'Age': [25, 30, 22]}

df = pd.DataFrame(data)

# Adding a new column
df['City'] = ['Delhi', 'Mumbai', 'Bangalore']

print(df)

print('\n')
# Adding a new column based on existing column
df['Age+5'] = df['Age'] + 5

print(df)
```

Output:

```
     Name  Age       City
0   Rahul   25      Delhi
1  Ananya   30     Mumbai
2  Vikram   22  Bangalore


     Name  Age       City  Age+5
0   Rahul   25      Delhi     30
1  Ananya   30     Mumbai     35
2  Vikram   22  Bangalore     27
```

Deleting Columns

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Rahul', 'Ananya', 'Vikram'],
        'Age': [25, 30, 22],
        'Age+5': [30, 35, 27]}

df = pd.DataFrame(data)

# Deleting the 'Age+5' column using drop()
df = df.drop('Age+5', axis=1)

print(df)

print('\n')
del df['Name']

print(df)
```

Output:

```
     Name  Age
0   Rahul   25
1  Ananya   30
2  Vikram   22


   Age
0   25
1   30
2   22
```

Renaming Columns

```
import pandas as pd

# Sample DataFrame
data = {'Name': ['Rahul', 'Ananya', 'Vikram'],
        'Age': [25, 30, 22]}

df = pd.DataFrame(data)

# Renaming the 'Age' column to 'Years'
df = df.rename(columns={'Age': 'Years'})
print(df)

print('\n')

# Renaming all columns at once
df.columns = ['Full Name', 'Age in Years']
print(df)
```

Output:

```
     Name  Years
0   Rahul     25
1  Ananya     30
2  Vikram     22


  Full Name  Age in Years
0     Rahul            25
1    Ananya            30
2    Vikram            22
```

Handling missing data (isnull(), dropna(), fillna())

isnull() and notnul()

```
import pandas as pd

# Sample DataFrame
data = {'A': [1, 2, None],
        'B': [4, None, None]}
df = pd.DataFrame(data)

# Detect missing values
print(df.isnull(),'\n')
print(df.notnull(),'\n')
print(df.isnull().sum(),'\n')
print(df.notnull().sum(),'\n')
```

Output:

```
       A      B
0  False  False
1  False   True
2   True   True

       A      B
0   True   True
1   True  False
2  False  False

A    1
B    2
dtype: int64

A    2
B    1
dtype: int64
```

dropna()

Remove Rows with Any Missing Values

```
import pandas as pd

# Sample DataFrame with missing values
data = {
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': [1, 2, 3, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

df_dropped_any = df.dropna()
print("\nDataFrame after dropping rows with any missing values:")
print(df_dropped_any)
```

Output:

```
Original DataFrame:
     A    B  C
0  1.0  NaN  1
1  2.0  2.0  2
2  NaN  3.0  3
3  4.0  4.0  4
DataFrame after dropping rows with any missing values:
     A    B  C
1  2.0  2.0  2
3  4.0  4.0  4
```

Remove Rows Where All Values Are Missing

```
import pandas as pd

# Sample DataFrame with missing values
data = {
    'A': [None, None, None, 4],
    'B': [None, None, None, 4],
    'C': [None, None, None, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Drop rows where all values are missing
df_dropped_all = df.dropna(how='all')
print("\nDataFrame after dropping rows where all values are missing:")
print(df_dropped_all)
```

Output:

```
Original DataFrame:
     A    B    C
0  NaN  NaN  NaN
1  NaN  NaN  NaN
2  NaN  NaN  NaN
3  4.0  4.0  4.0

DataFrame after dropping rows where all values are missing:
     A    B    C
3  4.0  4.0  4.0
```

Remove Columns with Any Missing Values

```
import pandas as pd

data = {
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': [1, 2, 3, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

df_dropped_cols_any = df.dropna(axis=1)
print("\nDataFrame after dropping columns with any missing values:")
print(df_dropped_cols_any)
```

Output:

```
Original DataFrame:
     A    B  C
0  1.0  NaN  1
1  2.0  2.0  2
2  NaN  3.0  3
3  4.0  4.0  4

DataFrame after dropping columns with any missing values:
   C
0  1
1  2
2  3
3  4
```

Remove Rows with Missing Values in Specific Columns

```
import pandas as pd

data = {
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': [1, 2, 3, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

df_dropped_specific = df.dropna(subset=['A', 'B'])
print("\nDataFrame after dropping rows with missing values in columns A and B:")
print(df_dropped_specific)
```

Output:

```
Original DataFrame:
     A    B  C
0  1.0  NaN  1
1  2.0  2.0  2
2  NaN  3.0  3
3  4.0  4.0  4

DataFrame after dropping rows with missing values in columns A and B:
     A    B  C
1  2.0  2.0  2
3  4.0  4.0  4
```

fillna()

Fill Missing Values with a Constant Value

```
import pandas as pd

# Sample DataFrame with missing values
data = {
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, None],
    'C': [1, 2, None, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Fill missing values with a constant value (e.g., 0)
df_filled_constant = df.fillna(0)
print("\nDataFrame after filling missing values with 0:")
print(df_filled_constant)
```

Output:

Original DataFrame:

```
     A    B    C
0  1.0  NaN  1.0
1  2.0  2.0  2.0
2  NaN  3.0  NaN
3  4.0  NaN  4.0

DataFrame after filling missing values with 0:
     A    B    C
0  1.0  0.0  1.0
1  2.0  2.0  2.0
2  0.0  3.0  0.0
3  4.0  0.0  4.0
```

Forward and Backward fille

```
import pandas as pd

# Creating the DataFrame
data = {'A': [1.0, None, 3.0, 4.0],
        'B': [None, 2.0, None, 4.0],
        'C': [1.0, 2.0, 3.0, None]}

df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Forward filling the missing values
ffill = df.ffill()

print("DataFrame after forward filling:")
print(ffill)

# Backward filling the missing values
bfill = df.bfill()

print("DataFrame after backward filling:")
print(bfill)
```

Output:

```
Original DataFrame:
     A    B    C
0  1.0  NaN  1.0
1  NaN  2.0  2.0
2  3.0  NaN  3.0
3  4.0  4.0  NaN
DataFrame after forward filling:
     A    B    C
0  1.0  NaN  1.0
1  1.0  2.0  2.0
2  3.0  2.0  3.0
3  4.0  4.0  3.0
DataFrame after backward filling:
     A    B    C
0  1.0  2.0  1.0
1  3.0  2.0  2.0
2  3.0  4.0  3.0
3  4.0  4.0  NaN
```

Fill Missing Values with the Mean of a Column

```
import pandas as pd

# Sample DataFrame with missing values
data = {
    'A': [1, 2, None, 4],
    'B': [None, 2, 3, 4],
    'C': [1, 2, 3, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Fill missing values in column 'A' with its mean
mean_A = df['A'].mean()
df['A'] = df['A'].fillna(mean_A)

print("\nDataFrame after filling missing values in column A with its mean:")
print(df)
```

Output:

```
Original DataFrame:
     A    B  C
0  1.0  NaN  1
1  2.0  2.0  2
2  NaN  3.0  3
3  4.0  4.0  4

DataFrame after filling missing values in column A with its mean:
          A    B  C
0  1.000000  NaN  1
1  2.000000  2.0  2
2  2.333333  3.0  3
3  4.000000  4.0  4
```

Fill Missing Values with Different Values for Each Column

```
import pandas as pd

# Sample DataFrame with missing values
data = {
    'A': [1, None, 3, None],
    'B': [None, 2, None, 4],
    'C': [1, None, 3, 4]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Fill missing values with different values for each column
fill_values = {'A': 0, 'B': 99, 'C': 5}
df_filled_diff = df.fillna(value=fill_values)
print("\nDataFrame after filling missing values with different values for each
column:")
print(df_filled_diff)
```

Output:

```
Original DataFrame:
     A    B    C
0  1.0  NaN  1.0
1  NaN  2.0  NaN
2  3.0  NaN  3.0
3  NaN  4.0  4.0

DataFrame after filling missing values with different values for each column:
     A     B    C
0  1.0  99.0  1.0
1  0.0   2.0  5.0
2  3.0  99.0  3.0
3  0.0   4.0  4.0
```

Filling Missing Values with Values from Another Series or DataFrame

```
import pandas as pd

# Sample DataFrames
df1 = pd.DataFrame({
    'A': [None, 2, None, 4],
    'B': [1, None, 3, None]
})

df2 = pd.DataFrame({
    'A': [5, 6, 7, 8],
    'B': [9, 10, 11, 12]
})

print("Original DataFrame:")
print(df1)

print("\nFilling missing values using another DataFrame:")
df_filled_from_other = df1.fillna(df2)
print(df_filled_from_other)
```

Output:

```
Original DataFrame:
     A    B
0  NaN  1.0
1  2.0  NaN
2  NaN  3.0
3  4.0  NaN

Filling missing values using another DataFrame:
     A     B
0  5.0   1.0
1  2.0  10.0
2  7.0   3.0
3  4.0  12.0
```

Filling Missing Values with a Dictionary of Functions

```
import pandas as pd

# Sample DataFrame
data = {
    'A': [None, 2, None, 4],
    'B': [1, None, 3, None]
}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Fill missing values using different functions for each column
df_filled = df.fillna({'A': df['A'].mean(), 'B': df['B'].median()})
print("\nDataFrame after filling missing values with dict methods")
print(df_filled)
```

Output:

```
Original DataFrame:
     A    B
0  NaN  1.0
1  2.0  NaN
2  NaN  3.0
3  4.0  NaN

DataFrame after filling missing values with mean for 'A' and median for 'B':
     A    B
0  3.0  1.0
1  2.0  2.0
2  3.0  3.0
3  4.0  2.0
```

Data Alignment and Operations

Arithmetic operations on DataFrames and Series

```
import pandas as pd

# Creating two DataFrames
df1 = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6]
}, index=['X', 'Y', 'Z'])

df2 = pd.DataFrame({
    'A': [7, 8, 9],
    'B': [10, 11, 12]
}, index=['X', 'Y', 'Z'])

# Arithmetic operation
print(df1 + df2, '\n')
print(df1 - df2, '\n')
print(df1 * df2, '\n')
print(df1 / df2)
```

Output:

```
    A   B
X   8  14
Y  10  16
Z  12  18

   A  B
X -6 -6
Y -6 -6
Z -6 -6

    A   B
X   7  40
Y  16  55
Z  27  72

          A         B
X  0.142857  0.400000
Y  0.250000  0.454545
Z  0.333333  0.500000
```

Data alignment during operations

Pandas performs **alignment** automatically when performing arithmetic operations on objects that may not have the same labels (for both Series and DataFrames). This means that Pandas will match on the row and column labels, and if the labels don't match, it will fill missing data with NaN.

```
import pandas as pd

df1 = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6]
}, index=['X', 'Y', 'Z'])
print(df1)

df2 = pd.DataFrame({
    'A': [7, 8, 9],
    'B': [10, 11, 12]
}, index=['X', 'Y', 'Z'])
print(df2)

df3 = pd.DataFrame({
    'A': [7, 8, 9],
    'C': [10, 11, 12]
}, index=['X', 'Y', 'Z'])
print(df3)
# Addition with alignment
df_sum_misaligned = df1 + df3
print(df_sum_misaligned)
```

Output:

```
   A  B
X  1  4
Y  2  5
Z  3  6
   A   B
X  7  10
Y  8  11
Z  9  12
   A   C
X  7  10
Y  8  11
Z  9  12
    A   B   C
X   8 NaN NaN
Y  10 NaN NaN
Z  12 NaN NaN
```

Broadcasting and handling mismatched data

Broadcasting in Pandas refers to performing operations between objects of different dimensions, such as a Series and a DataFrame

```
import pandas as pd

df1 = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6]
}, index=['X', 'Y', 'Z'])
print(df1)

# Series for broadcasting
series = pd.Series([10, 20, 30],
                   index=['A', 'B',
'C'])

# Broadcasting addition
df_broadcast = df1 + series
print(df_broadcast)
```

Output:

```
   A  B
X  1  4
Y  2  5
Z  3  6
    A   B   C
X  11  24 NaN
Y  12  25 NaN
Z  13  26 NaN
```

Sorting Data

Sorting by index (sort_index()) and Sorting by values (sort_values())

```
import pandas as pd

# Creating a DataFrame
data = {'Name': ['Ravi', 'Anita', 'Priya', 'Amit'],
        'Age': [28, 22, 25, 32]}
df = pd.DataFrame(data, index=['b', 'a', 'd', 'c'])

# Sorting by index (row labels)
sorted_df = df.sort_index()

print(sorted_df)
print('\n')

# Sorting by the 'Age' column
sorted_by_values = df.sort_values(by='Age')

print(sorted_by_values)
```

Output:

```
    Name  Age
a  Anita   22
b   Ravi   28
c   Amit   32
d  Priya   25


    Name  Age
a  Anita   22
d  Priya   25
b   Ravi   28
c   Amit   32
```

Module 3: Reading and Writing Data Into files

Working With CSV file

Reading a CSV File

Let's say we have a CSV file called *employees.csv* with the following content in the same folder where we have the python file:

```
Name,Department,Salary
Vijay,Sales,50000
Nisha,HR,60000
Anil,Finance,70000
```

Example

```
import pandas as pd

df = pd.read_csv('employees.csv')

print(df.head())
```

Output

```
    Name Department  Salary
0  Vijay      Sales   50000
1  Nisha         HR   60000
2   Anil    Finance   70000
```

Breakdown

Step 1: Importing Pandas We import the Pandas library

Step 2: Reading CSV File *pd.read_csv()* is used to read a CSV file and store its contents in a DataFrame.

Step 3: Displaying Data The *.head()* method displays the first 5 rows of the DataFrame to quickly inspect the

Writing CSV files (to_csv())

Let's create and write the following data to a new CSV file *new_employees.csv.*

Example

```
import pandas as pd

data = {
    'Name': ['Ramesh', 'Sunita', 'Amit'],
    'Department': ['IT', 'Marketing', 'Operations'],
    'Salary': [80000, 65000, 72000]
}

df = pd.DataFrame(data)

df.to_csv('new_employees.csv', index=False)

df_new = pd.read_csv('new_employees.csv')
print(df_new)
```

Breakdown

Step 0: Import library to the code

Step 1: Creating Data The sample data is stored in a dictionary format where each key represents a column name.

Step 2: Creating DataFrame We convert the dictionary into a Pandas DataFrame.

Step 3: Writing CSV File The .to_csv() method is used to write the DataFrame to a CSV file named new_employees.csv. We set index=False to avoid writing row indices into the CSV.

Step 4: Verifying Data We read the newly created CSV file back to verify its content.

Working With XLSX file

Reading Excel Files*(read_excel()).(make sure openpyxl module installed)*

Let's assume we have an Excel file *students.xlsx* with the following sheet:

Name	Grade
Amit	A
Priya	B
Sohan	A

Example

```
import pandas as pd

# Reading an Excel file
df = pd.read_excel('students.xlsx')

# Display the first 5 rows
print(df.head())
```

Output

```
   Student  Marks
0    Ravi     85
1   Kiran     90
2  Anjali     88
```

Breakdown

Step 1: Import Pandas We import Pandas to handle the Excel file.

Step 2: Reading Excel File The *pd.read_excel()* function reads the students.xlsx file.

Step 3: Displaying Data We use *.head()* to inspect the first few rows of the DataFrame.

Writing Excel files (*to_excel()*)

Let's write a DataFrame to an Excel file called *marks.xlsx*

Example

```
import pandas as pd

# Creating sample data
data = {
    'Student': ['Ravi', 'Kiran', 'Anjali'],
    'Marks': [85, 90, 88]
}

# Creating DataFrame from the data
df = pd.DataFrame(data)

# Writing DataFrame to Excel file
df.to_excel('marks.xlsx', index=False)

# Verifying by reading the newly created Excel file
df_new = pd.read_excel('marks.xlsx')
print(df_new)
```

Breakdown

Step 1: Creating Data A dictionary is used to store the student names and their marks.

Step 2: Creating DataFrame The dictionary is converted into a Pandas DataFrame.

Step 3: Writing Excel File The DataFrame is written to an Excel file named *marks.xlsx*. The *index=False* option prevents Pandas from writing row numbers into the Excel file.

Step 4: Verifying Data The newly created Excel file is read back into a DataFrame for verification.

Working With JSON file

Reading a JSON File: Let's assume we have a JSON file data.json with the following content:

```
[
    {"Name": "Nikhil", "Age": 23},
    {"Name": "Sonal", "Age": 25},
    {"Name": "Pooja", "Age": 22}
]
```

Example

```
import pandas as pd

# Reading a JSON file
df = pd.read_json('data.json')

# Display the first 5 rows
print(df.head())
```

Output:

```
     Name  Age
0  Nikhil   23
1   Sonal   25
2   Pooja   22
```

Breakdown

Step 1: Import Pandas We import Pandas to handle the JSON file.

Step 2: Reading JSON File The *pd.read_json()* function reads the JSON file and loads it into a DataFrame.

Step 3: Displaying Data The *.head()* method is used to display the first few rows of the DataFrame.

Writing a JSON File

Let's write a DataFrame to a JSON file named cities.json

Example

```
import pandas as pd

# Creating sample data
data = {
    'City': ['Delhi', 'Bangalore', 'Hyderabad'],
    'Population': [3000000, 9000000, 6000000]
}

# Creating DataFrame from the data
df = pd.DataFrame(data)

# Writing DataFrame to JSON file
df.to_json('cities.json')

# Verifying by reading the newly created JSON file
df_new = pd.read_json('cities.json')
print(df_new)
```

Breakdown

Step 1: Creating Data A dictionary is used to store city names and their population.

Step 2: Creating DataFrame The dictionary is converted into a Pandas DataFrame.

Step 3: Writing JSON File The DataFrame is written to a JSON file named cities.json.

Step 4: Verifying Data The JSON file is read back to ensure it was written correct.