Lab – 24
Student Name: Navneet P
Student ID: AF0411619

Topic-Pandas Library

- What is Pandas?

Pandas is an open-source data manipulation and analysis library for the Python programming language that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series and for working with structured data, making it one of the most popular and widely used libraries for data analysis and manipulation in the Python ecosystem.

- Installation
  - Pip install pandas
- Data Cleaning
  - Pandas includes functions for handling missing data, removing duplicates, and transforming data to make it suitable for analysis.
- Dataframes
  - The DataFrame is one of the central data structures in Pandas. It is a two-dimensional table with rows and columns, similar to a spreadsheet or a SQL table.
- Series
  - A Series is a one-dimensional array-like object in Pandas. It can be thought of as a single column of data within a DataFrame, with an associated index
- Grouping
  - Pandas allows you to group data by one or more columns and perform aggregation operations on the groups.

- Why Pandas?
  - Fast and efficient for manipulating data. Data from different file objects can be easily loaded. Flexible reshaping, time-series functionality.

Functions used in this assignment:-

1. **Pandas.Series** – A one-dimensional labeled array capable of holding any data type, similar to a column in a DataFrame.

2. **Pandas.DataFrame** – A two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).

3. **to_string()** – Converts a DataFrame or Series to a string representation for more readable console output.

4. **pandas.read_excel()** – Reads an Excel file into a DataFrame, allowing you to load data from spreadsheets.

5. **head()** – Returns the first few rows of a DataFrame (default is 5 rows).

6. **tail()** – Returns the last few rows of a DataFrame (default is 5 rows).

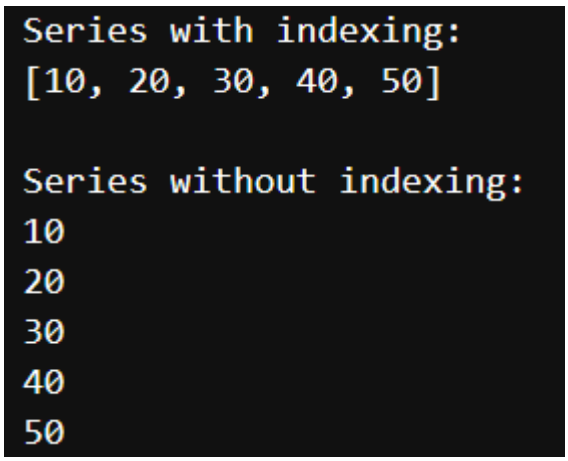
7. **loc[]** – Accesses a group of rows and columns by labels or a boolean array in a DataFrame.

Q1. Create Pandas series [10,20,30,40,50] without index

Solution:

```python
import pandas as pd  # Importing the pandas library

# Creating a simple list of data
data = [10, 20, 30, 40, 50]

# Converting the list into a Pandas Series (with default indexing 0, 1, 2, ...)
series = pd.Series(data)

# Display the Series with default indexing
print("Series with indexing:")
print(data)

# Converting the Series to a string, but without the index
unindexed = series.to_string(index=False)

# Display the Series without the index
print("\nSeries without indexing:")
print(unindexed)

```

Output:



Q2. Create Pandas DataFrame example.

Solution:

```python
import pandas as pd  # Importing the pandas library

# Creating a dictionary with two keys: 'Batsman' and 'Averages'
cricket = {
    "Batsman": ["Rohit Sharma", "Suryakumar Yadav", "Virat Kohli"],  # List of batsmen
    "Averages": [46, 89, 56]  # Corresponding batting averages
}

# Converting the dictionary into a Pandas DataFrame
df = pd.DataFrame(cricket)

# Converting the DataFrame to a string representation without the index
result = df.to_string(index=False)

# Printing the DataFrame in string format, without index
print(result)

# Printing the type of the 'df' variable, which is a Pandas DataFrame
print(type(df))

```

Output:

```
          Batsman  Averages
     Rohit Sharma        46
Suryakumar Yadav        89
      Virat Kohli        56
<class 'pandas.core.frame.DataFrame'>
```

Q3. Example: Monthly Sales Data Imagine you are a sales manager for a retail company, and you want to analyze the monthly sales performance of a particular product in a given year. You have recorded the monthly sales figures for that product, and you want to represent this data using a Pandas Series.

Solution:

```python
import pandas as pd  # Importing the pandas library

# List of months
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
          'September', 'October', 'November', 'December']

# Corresponding sales data for each month
sales_data = [12000, 13500, 14200, 12800, 14000, 15500, 16200, 15800, 16500,
              17800, 18500, 17200]

# Combining the two lists into a dictionary where each month is mapped to sales
data = dict(zip(months, sales_data))

# Creating a DataFrame from the dictionary, where each dictionary item (month-sales pair) becomes a row
df = pd.DataFrame(data.items(), columns=["Month", "Sales"])

# Converting the DataFrame to a string, excluding the index, for a cleaner output
result = df.to_string(index=False)

# Printing the DataFrame in string format
print(result)

# Printing the type of 'df' to confirm it's a DataFrame
print(type(df))

```

Output:

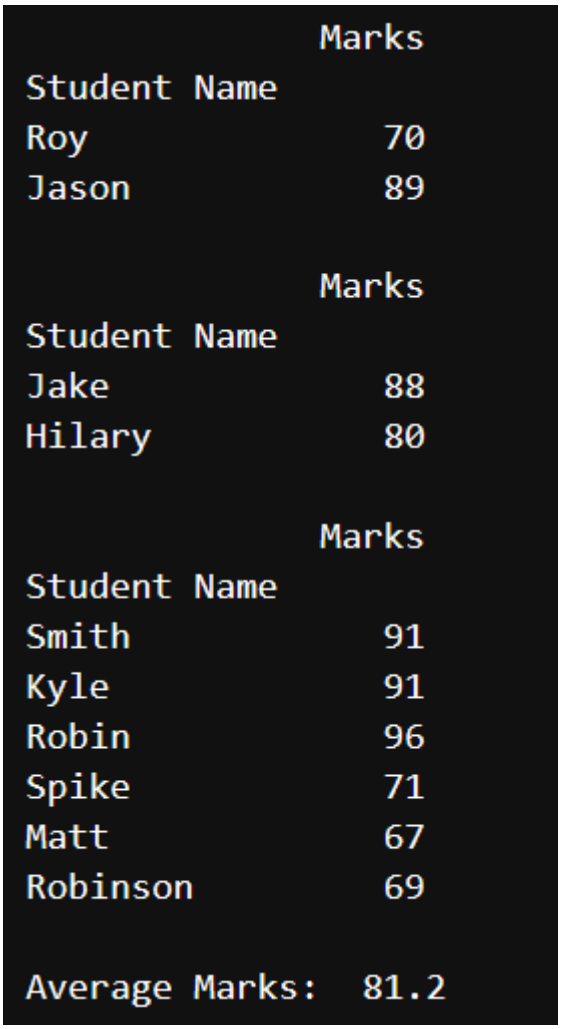
```
     Month  Sales
   January  12000
  February  13500
     March  14200
     April  12800
       May  14000
      June  15500
      July  16200
    August  15800
 September  16500
   October  17800
  November  18500
  December  17200
<class 'pandas.core.frame.DataFrame'>
```

Example: Read an excel file and demonstrate the use of loc slicing, head and tail methods.

Solution:

```python
import pandas as pd  # Importing pandas for data handling
import numpy as np   # Importing numpy for numerical operations

# Reading data from an Excel file into a Pandas DataFrame
data = pd.read_excel("Excelfile.xlsx")

# Converting the 'Marks' column into a NumPy array for numerical operations
marks = np.array(data['Marks'])

# Calculating the average of the 'Marks' column using NumPy's average function
average = np.average(marks)

# Extracting the first 2 rows (top of the DataFrame)
top = data.head(2)

# Extracting the last 2 rows (bottom of the DataFrame)
bottom = data.tail(2)

# Extracting the middle rows based on specific row indices (rows from index 2 to 7)
middle = data.loc[2:7]

# Setting 'Student Name' as the index for the 'top' DataFrame
top.set_index("Student Name", inplace=True)

# Setting 'Student Name' as the index for the 'bottom' DataFrame
bottom.set_index("Student Name", inplace=True)

# Setting 'Student Name' as the index for the 'middle' DataFrame
middle.set_index("Student Name", inplace=True)

# Printing the top 2 rows
print(top,  "\n")

# Printing the bottom 2 rows
print(bottom, "\n")

# Printing the middle rows (rows from index 2 to 7)
print(middle)

# Printing the average of the 'Marks' column
print("\nAverage Marks: ", average)

```

Output:



Student Name	Marks
Roy	70
Jason	89
Smith	91
Kyle	91
Robin	96
Spike	71
Matt	67
Robinson	69
Jake	88
Hilary	80