Module 3: Python Libraries- Pandas

1. Introduction to Pandas:

Pandas is a powerful open-source library for data manipulation and analysis in Python.

Key Components:

- Series: A one-dimensional array with labeled data, similar to a column in a spreadsheet.
- DataFrame: A two-dimensional table, an organized collection of Series, representing a dataset.

2. Series and DataFrames:

2.1 Series:

The pd.Series() constructor is used to create a Pandas Series, which is a one-dimensional labeled array. It can hold any data type, such as integers, floats, strings, or even more complex objects like Python dictionaries.

Creation:

- Constructed using the pd.Series() constructor.
- Contains data with associated labels (index).

Example:

```
import pandas as pd
data = [10, 20, 30, 40, 50]
series = pd.Series(data, name='Numbers')
print(series)
```

Key Attributes and Properties:

Values: Retrieves the underlying NumPy array containing the values of the Series.

```
series_values = series.values
```

Index: Retrieves the index of the Series. If not specified, Pandas automatically generates a default integer index.

```
series_index = series.index
```

Name: Retrieves the name of the Series, if assigned.

```
series_name = series.name
```

Accessing Elements: Elements can be accessed in a Series using indexing, similar to a NumPy array or a Python list.

```
first_element = series[0]  # Retrieves the first element of the Series
```

Operations and Methods:

Mathematical Operations: Applies a mathematical operation to each element of the Series.

```
series_squared = series ** 2
```

Descriptive Statistics: Calculates the mean of the Series.

```
mean_value = series.mean()
```

Filtering: Creates a new Series with only those elements that satisfy a given condition.

```
filtered_series = series[series > 20]
```

Methods: unique() Returns an array of unique values in the Series.

```
unique_values = series.unique()
```

Example using series

```
import pandas as pd

data = [10, 20, 30, 40, 50]
series = pd.Series(data, name='Numbers')

# Accessing attributes
print("Values:", series.values)
print("Index:", series.index)
print("Name:", series.name)

# Performing operations
squared_series = series ** 2
print("Squared Series:", squared_series)

# Calculating mean
mean_value = series.mean()
print("Mean Value:", mean_value)
```

2.2 DataFrames:

A Pandas DataFrame is a two-dimensional tabular data structure, and the pd.DataFrame() constructor is used to create it. It can be constructed using various data types, such as dictionaries, lists, NumPy arrays, or other DataFrames.

Creation:

- Constructed using the pd.DataFrame() constructor.
- Composed of multiple Series, each representing a column.

Example:

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 22],
        'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)

print(df)
```

Key Parameters:

- data: The main argument that provides the data for the DataFrame. It can be a dictionary, list of dictionaries, NumPy array, or another DataFrame.
- index: Specifies the row labels. If not specified, Pandas automatically generates a default integer index.
- columns: Specifies the column labels. If not specified, the keys of the dictionary (or the columns of the array) are used as column labels.
- dtype: Specifies the data type for elements in the DataFrame. It can be a single data type or a dictionary mapping column names to data types.

Accessing Attributes:

Values:

```
df_values = df.values
```

- Retrieves the underlying NumPy array containing the values of the DataFrame.

Index and Columns:

```
df_index = df.index

df_columns = df.columns
```

- Retrieves the row and column labels, respectively.

Accessing Columns:

- Columns in a DataFrame are essentially Pandas Series. You can access them using square brackets or the dot notation

```
name_column = df['Name']  # Using square brackets

age_column = df.Age       # Using dot notation
```

Operations and Methods:

Descriptive Statistics:

```
mean_age = df['Age'].mean()
```

- Calculates the mean of a specific column.

Filtering:

```
filtered_df = df[df['Age'] > 25]
```

- Creates a new DataFrame with rows that satisfy a given condition.

Sorting:

```
sorted_df = df.sort_values(by='Age')
```

- Sorts the DataFrame based on a specific column.

Example dataframe:

```
import pandas as pd

# Creating a DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 22],
        'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)

# Accessing attributes
print("Values:\n", df.values)
print("Index:", df.index)
print("Columns:", df.columns)

# Accessing columns
print("\nName Column:\n", df['Name'])
print("\nAge Column:\n", df.Age)

# Performing operations
mean_age = df['Age'].mean()
print("\nMean Age:", mean_age)

# Filtering data
filtered_df = df[df['Age'] > 25]
print("\nFiltered DataFrame:\n", filtered_df)
```

3. Grouping, Aggregating, and Applying:

3.1 Grouping and Aggregating:

Grouping:

- Process of splitting data into groups based on some criteria.
- Accomplished using the groupby() function.

Aggregating:

- Computing a summary statistic (or transformation) for each group.
- Common aggregations include mean, sum, count, etc.

Example:

```
import pandas as pd

data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
    'Value': [10, 15, 20, 25, 30, 35]}

df = pd.DataFrame(data)

# Grouping by 'Category' and calculating the mean
grouped_df = df.groupby('Category').mean()
```

3.2 Applying Custom Functions:

Applying Functions:

- Applying custom functions to data, often using the apply() method.
- Allows for complex operations on Series or DataFrame elements.

Example:

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 22]}

df = pd.DataFrame(data)

# Applying a custom function to the 'Age' column
df['Age Doubled'] = df['Age'].apply(lambda x: x * 2)
```