SUMMARY-DAY8

Name: Tejaswini Gokanakonda

Roll no:DE142

Date:15-11-2024

Detailed Notes on NumPy and Pandas

Introduction to NumPy and Pandas :

NumPy and Pandas are powerful Python libraries essential for data analysis and manipulation. Here's a breakdown:

1. NumPy (Numerical Python):

- A library for numerical computations.
- Provides support for large multi-dimensional arrays and matrices.
- Includes a collection of mathematical functions to operate on these arrays.

2. Pandas:

- A library designed for data manipulation and analysis.
- Offers data structures like Series and DataFrame.
- Facilitates operations like merging, reshaping, selecting, and cleaning data.

NumPy Basics

1. Creating Arrays:

- `numpy.array()` is used to create arrays.
- Supports multi-dimensional arrays, such as 2D matrices and 3D tensors.

Example:

```
import numpy as np
array_1d = np.array([1, 2, 3])
array_2d = np.array([[1, 2], [3, 4]])
```

2. Array Operations:

- Arithmetic operations are element-wise.
- **Example:**

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = arr1 + arr2  # [5, 7, 9]
```

3. Indexing and Slicing:

- Access elements using indices or slices.
- **Example:**

```
arr = np.array([10, 20, 30, 40])
print(arr[1])        # 20
print(arr[1:3])      # [20, 30]
```

4. Statistical Functions:

- NumPy provides functions like `mean()`, `median()`, and `std()` for statistical analysis.
- **Example:**

```
np.mean(arr)  # Average of array elements.
np.std(arr)   # Standard deviation.
```

Pandas Basics

1. Data Structures:

- **Series:** A one-dimensional labeled array.

  Example:

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

- **DataFrame:** A two-dimensional labeled data structure.

  Example:

  ```
  df = pd.DataFrame({
      'Name': ['Alice', 'Bob'],
      'Age': [24, 27]
  })
  ```

2. DataFrame Operations:

- Access rows/columns using `.loc` or `.iloc`.

  Example:

  ```
  df.loc[0, 'Name']  # Access specific cell
  df.iloc[0, 1]      # Access using integer-based indexing
  ```

3. Basic Functions:

- **`head()`, `tail()`:** View top or bottom rows.
- **`describe()`:** Summary statistics.
- **`info()`:** Detailed overview of the DataFrame.

Advanced Operations:

1. Data Cleaning:

- **Handle missing values using:**
  - `fillna()`: Fill missing data.
  - `dropna()`: Remove rows/columns with missing values.

Example:

```
df['Age'].fillna(df['Age'].mean(), inplace=True)
```

2. Data Aggregation:

- Group and aggregate data using `groupby()` and `agg()`.

Example:

```
grouped = df.groupby('Category')['Sales'].sum()
```

3. Data Visualization:

- Use Matplotlib or Seaborn with Pandas for visualizations.

Example:

```
df['Sales'].plot(kind='bar')
```

NumPy and Pandas Integration

- NumPy arrays can be converted to Pandas DataFrames and vice versa.
- **Example:**

```
np_array = np.array([[1, 2], [3, 4]])
df = pd.DataFrame(np_array, columns=['A', 'B'])
```

Applications and Use Cases:

- **NumPy**: Best for mathematical computations, linear algebra, Fourier transforms, and random number generation.
- **Pandas**: Ideal for data cleaning, analysis, and transformation.