Python Libraries: Numpy and Pandas

NIELIT Chandigarh/Ropar

"In God we trust, all others must bring data." – W. Edwards Deming

Numpy

Numpy

Definition: A library for numerical computing in Python.

Key Features:

- Support for multi-dimensional arrays.
- Mathematical functions for fast operations.

Example:

```python
import numpy as np
# Create a 1D array
data = np.array([1, 2, 3, 4])
print(data)
# Perform operations
print(data.mean()) # Mean value
print(data + 5) # Element-wise addition
```

OUTPUT:

```
[1 2 3 4]
2.5
[6 7 8 9]
```

Pandas

Definition: A library for data manipulation and analysis.

- **Key Features**:
  - DataFrames: 2D labeled data structures.
  - Easy handling of missing data.
  - Integration with CSV, Excel, and databases.

Example:

```
import pandas as pd
# Create a DataFrame
data = {'Name': [‘Raju', ‘Priya'], 'Age': [25, 30]}
df = pd.DataFrame(data)
print(df)
# Inspect DataFrame
print(df.head()) # First few rows
print(df.describe()) # Summary statistics
```

Output:

```
    Name  Age
0   Raju   25
1  Priya   30
    Name  Age
0   Raju   25
1  Priya   30
             Age
count   2.000000
mean   27.500000
std     3.535534
min    25.000000
25%    26.250000
50%    27.500000
75%    28.750000
max    30.000000
```

Loading and Inspecting Datasets

Loading CSV Files

```python
# Load a dataset
import pandas as pd
data = pd.read_csv('./data.csv')
print(data)
```

Inspecting Data

- **View First 5 Rows**: data.head()
- **Shape of Data**: data.shape()
- **Column Names**: data.columns()
- **Basic Statistics**: data.describe()

```python
import pandas as pd
data = pd.read_csv('./data.csv')
print(data.head())
```

```
   ID    Name   Age  Gender   Salary Department
0   1   Aarav  29.0    Male  50000.0         IT
1   2   Aditi  34.0  Female  60000.0         HR
2   3  Vikram   NaN    Male  45000.0    Finance
3   4   Sneha  28.0  Female      NaN  Marketing
4   5   Manoj  40.0    Male  75000.0         IT
```

Loading and Inspecting Datasets

- **Loading a CSV file**:

```
data = pd.read_csv('./data.csv')  # Replace 'data.csv' with your file path
```

Inspecting Data:

- **View first few rows**:

```
print(df.head())
```

- **Summary of data**:

```
print(df.info())
```

- **Descriptive statistics**:

```
print(df.describe())
```

- **Check for null values**:

```
print(df.isnull().sum())
```

Data Cleaning and Preprocessing

Handling Missing Values

1. Always check your data for missing values before using dropna():

```
print(data.isnull().sum())
```

2. Use inplace=False if you want to keep the original DataFrame intact.

Why:

- **Missing values can distort analysis and results.**
- **Missing data can skew analysis and lead to incorrect conclusions.**

Methods:

- **Fill Missing Values:**

```
df.fillna(value=0, inplace=True) # Fill missing values with 0
print(df)
data['ColumnName'].fillna(value, inplace=True)
Example: data['Age'].fillna(25, inplace=True)
# Replaces all NaN values in the 'Age' column with 25.
```

Data Cleaning and Preprocessing

```
import pandas as pd
# Load the dataset
df = pd.read_csv('./data.csv’)
# Fill missing values with 0 (create a new modified DataFrame)
df = df.fillna(value=0)
# Print the DataFrame
print(df)
```

- If you prefer to modify the DataFrame in place, you can use:

```
df.fillna(value=0, inplace=True)
print(df)
```

Data Cleaning and Preprocessing

```
import pandas as pd

# Create the DataFrame

df = pd.DataFrame({'Name': ["Ajay", "Vishal", "Raj"],
'Age': [24, None, 19]})

# Modify the DataFrame directly

df.fillna(0, inplace=True)

# Print the DataFrame

print(df)
```

Data Cleaning and Preprocessing

Handle Missing Values

- **Drop Missing Values**: Remove rows or columns with missing data.

- **Drop Rows or Columns**

```
data.dropna(inplace=True)
```

- **Drop columns with missing values**

```
data = data.dropna(axis=1)
```

Data Cleaning and Preprocessing

- **Parameters:**

1. axis (default = 0):

1. Specifies whether to drop rows or columns.
   1. axis=0: Drop rows with missing values.
   2. axis=1: Drop columns with missing values.

Example:

data.dropna(axis=1, inplace=True) # Drops columns with NaN values.

2. how (default = 'any'):

- Defines the condition to drop rows or columns:
  - 'any': Drops rows/columns if **any** value is missing.
  - 'all': Drops rows/columns only if **all** values are missing.

Example:

data.dropna(how='all', inplace=True) # Drops rows where all values are NaN.

Data Cleaning and Preprocessing

3. thresh:

- Requires a minimum number of **non-NaN** values to retain the row/column.

Example:

data.dropna(thresh=3, inplace=True) # Keeps rows with at least 3 non-NaN values.

4. subset:

- Allows specifying columns to check for missing values instead of the entire DataFrame.

Example:

data.dropna(subset=['Column1', 'Column2'], inplace=True) # Drops rows based on NaNs in specified columns.

5. inplace (default = False):

- If True, makes changes directly to the original DataFrame.
- If False, returns a new DataFrame with rows/columns dropped.

Data Cleaning and Preprocessing

- **Handle Missing Values**

Fill Missing Values

- With a constant value:

```
data['ColumnName'] = data['ColumnName'].fillna('Value’)
```

- With the mean, median, or mode:

```
data['ColumnName'] = data['ColumnName'].fillna(data['ColumnName'].mean())

data['ColumnName'] = data['ColumnName'].fillna(data['ColumnName'].median())

data['ColumnName'] = data['ColumnName'].fillna(data['ColumnName'].mode()[0])
```

Encoding Categorical Data

- **Why**: Machine learning models work with numerical data.
- Categorical data must be converted into numeric values for most machine learning models. There are two common encoding techniques:
- **How**:
  1. **Label Encoding** (Simple Integer Mapping):
  2. **One-Hot Encoding**

1. Label Encoding

- Assigns a unique integer to each category.
- Suitable for ordinal (ranked) categories.

Encoding Categorical Data

EXAMPLE:

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Sample Data
data = {'Name': ['Ajay', 'Vishal', 'Raj'], 'Gender': ['Male', 'Male', 'Female']}
df = pd.DataFrame(data)

# Encode Gender
encoder = LabelEncoder()
df['Gender'] = encoder.fit_transform(df['Gender'])

print(df)
```

OUTPUT: (Here, Male is encoded as 1, and Female as 0.)

```
     Name  Gender
0    Ajay       1
1  Vishal       1
2     Raj       0
```

Encoding Categorical Data

2. One-Hot Encoding

- Creates binary columns for each category.
- Suitable for nominal (unordered) categories.
- **Example:**

```
# One-hot encoding using pandas

df = pd.DataFrame({'Name': ['Ajay', 'Vishal', 'Raj'],
'Department': ['IT', 'HR', 'Finance']})

df_encoded = pd.get_dummies(df, columns=['Department'])

print(df_encoded)
```

```
     Name  Department_Finance  Department_HR  Department_IT
0    Ajay               False          False           True
1  Vishal               False           True          False
2     Raj                True          False          False
```

Scaling Data

- **Why**: Models converge faster and perform better when data is scaled.
- Scaling ensures all features are in a similar range, which helps improve model performance.

1. Min-Max Scaling: Scales data to a fixed range, typically [0, 1].

- Formula: $X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$

```
from sklearn.preprocessing import MinMaxScaler
# Sample Data
df = pd.DataFrame({'Age': [24, 19, 30], 'Salary': [50000, 40000, 70000]})
# Scale data
scaler = MinMaxScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])
print(df)
```

Output:

```
        Age    Salary
0  0.454545  0.333333
1  0.000000  0.000000
2  1.000000  1.000000
```

Scaling Data

2. Standard Scaling Standardizes data to have a mean of 0 and a standard deviation of 1.

Formula: $Z = \frac{X - \mu}{\sigma}$

```
from sklearn.preprocessing import StandardScaler

# Sample Data
df = pd.DataFrame({'Age': [24, 19, 30], 'Salary': [50000, 40000,
70000]})

# Scale data
scaler = StandardScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])
print(df)
```

Output:

```
        Age    Salary
0 -0.074125 -0.267261
1 -1.185999 -1.069045
2  1.260124  1.336306
```

Summary

- **Numpy** and **Pandas** are essential Python libraries for data analysis.
- Loading and inspecting datasets is the first step in any data science workflow.
- Data cleaning ensures quality, while preprocessing prepares data for machine learning.

Encoding Categorical Data:

- Label Encoding: Use for ordinal data.
- One-Hot Encoding: Use for nominal data.

Scaling Data:

- Min-Max Scaling: Scales between a range (e.g., [0, 1]).
- Standard Scaling: Standardizes to a mean of 0 and a standard deviation of 1.

Thank You! ☺