Ex. No: 7 INTRODUCTION TO PYTHON LIBRARIES - NUMPY, PANDAS, MATPLOTLIB, SCIKIT

AIM

The aim to study the Python Libraries such as Numpy for numerical operations, Pandas for data manipulation and analysis, Matplotlib for data visualization, Scikit – Learn for machine learning tasks.

PROCEDURE

1. **Numpy**
    - Numpy is used for numerical operations in Python.
    - It provides support for large, multi-dimensional arrays, along with mathematical functions to operate on these arrays.

    Program

    ```
    import numpy as np
    arr = np. array ([1, 2, 3, 4, 5])
    print (“Numpy Array:”, arr)
    ```

    Output

    ```
    Numpy Array: [1 2 3 4 5]
    ```

2. **Pandas**
    - Pandas is a powerful library for data manipulation and analysis.
    - It introduces two primary data structures: Series (1D labeled array) and DataFrame (2D labeled table).
    - Procedures including loading data, exploring data using methods like info (), head (), and performing operations like dropping null values.

    Program

    ```
    import pandas as pd
    data = {‘Name’: [‘Alice’, ‘Bob’, ‘Charlie’], ‘Age’: [25, 30, 35]}
    df = pd. DataFrame(data)
    print (“Pandas DataFrame:”)
    ```

print(df)

Output

```
Pandas DataFrame:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
```

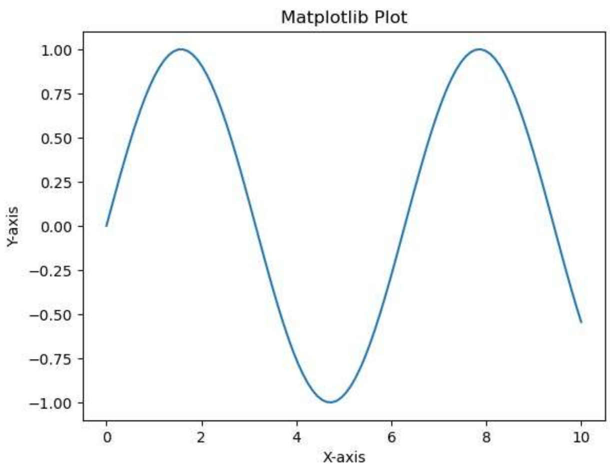
3. Matplotlib

- Matplotlib is a popular plotting library for creating static, interactive, and animated visualizations.
- The focus is on using Matplotlib to create basic plots like line plots, scatter plots, and bar plots.

Program

```
import numpy as np
import matplotlib. pyplot as plt
x = np. linspace (0, 10, 100)
y = np. sin (x)
plt. plot (x, y)
plt. title ("Matplotlib Plot")
plt. xlabel ("X-axis")
plt. ylabel ("Y-axis")
plt. show ()
```

Output



4. **Scikit-Learn**
    - Scikit-Learn is a machine learning library that provides simple and efficient tools for data analysis and modeling.
    - Procedures involve splitting data into training and testing sets using **train_test_split,** initializing and training a machine learning model (e.g., Linear regression), making predictions, and evaluating model performance.

Program

 # Dataset
x = np. array ([1, 2, 3, 4, 5]). reshape (-1, 1)
y = np. array ([2, 4, 5, 4, 5])

 # Split the training and testing sets
from sklearn. model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split (x, y, test_size = 0.2, random_state = 42)

 # Initialize the Linear Regression Model
from sklearn. linear_model import LinearRegression
model = LinearRegression ()

 # Train the Linear regression Model
model.fit (x_train, y_train)

Output
LinearRegression ()

 # Make Predictions
prediction = model. predict(x_test)
print (“Scikit-Learn Prediction:”, prediction)

 # Evaluate the model
from sklearn. metrics import mean_absolute_error, mean_squared_error, root_mean_sqerr, R_squared
mae = mean_absolute_error (y_test, y_pred)
print (f'Mean Absolute Error: {mae}')

```
mse = mean_squared_error (y_test, y_pred)
print (f'Mean Squared Error: {mse}')
```

Output

Mean Absolute Error: 85.71428571428572
Mean Squared Error: 73.46938775510206

Result

Thus, the basic usage of Numpy, Pandas, Matplotlib, and Scikit-Lear for machine learning tasks was studied successfully.

Ex. No: 8 PERFORM DATA EXPLORATION AND PREPROCESSING IN PYTHON

AIM

The aim is to perform the data exploration and preprocessing techniques in Python using a real-world dataset.

PROCEDURE

1. **Import necessary libraries**

   Importing the necessary libraries, including **'pandas'** for data manipulation, and **'numpy'** for numerical operations.

   Program

   ```
   import numpy as np
   import pandas as pd
   ```

2. **Load the dataset**

   Load the real-world sample dataset to explore and preprocess the data for better understanding. For Example, Data.CSV dataset.

   Program

   ```
   dataset = pd. read_csv ("C:\\Users\\HP\\Documents\\DSC LAB\\Dataset\\Data.csv")
   dataset
   ```

   Output

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

3. Display the few rows to understand the data structure

Display the few rows to understand the data structure of the data using **head ()** method.

Program

dataset. head (10)

Output

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

4. Analyze and view the information in tha data

To view the information of the data using **info ()** method.

Program

dataset.info ()

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Country    10 non-null     object
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object
dtypes: float64(2), object(2)
memory usage: 452.0+ bytes
```

5. **Analyze and view the statistics information of the data**

To view the statistics information of the data using **describe ()** method.

Program

```
dataset. describe ()
```

Output

	Age	Salary
count	9.000000	9.000000
mean	38.777778	63777.777778
std	7.693793	12265.579662
min	27.000000	48000.000000
25%	35.000000	54000.000000
50%	38.000000	61000.000000
75%	44.000000	72000.000000
max	50.000000	83000.000000

6. **Split the dataset into features (x)**

From the dataset, split the independent variable (x).

Program

```
x = dataset. iloc [:, :-1].values
print(x)
```

Output

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

7. **Split the dataset into target variable (y)**

From the dataset, split the dependent variable (y).

Program

```
y = dataset. iloc [:, 3].values
print(y)
```

Output

['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']

8. **Handling missing Data**

Identify missing values using **'dataset. isnull (). sum ()'**.

Program

```
print (dataset. isnull (). sum ())
```

Output

```
Country      0
Age          1
Salary       1
Purchased    0
dtype: int64
```

9. **Drop missing values**

Drop missing values from the dataset using **'dataset. dropna ()'**.

Program

```
dataset. dropna (inplace = True)
print(dataset)
```

Output

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
5   France  35.0  58000.0       Yes
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

10. **Replace missing values**

Use **'SimpleImputer'** from scikit-learn to replace missing values with th e mean of the respective columns.

Program

```
# Taking care of missing data (replacing with the mean)
from sklearn. impute import SimpleImputer
imputer = SimpleImputer (missing_values = np.nan, strategy = 'mean')
```

 # Fitting the imputer object to the matrix of features X
imputer.fit(x[:,1:3])

 # Replacing the missing data by the mean of the column
x[:,1:3] = imputer.transform(x[:,1:3])
print(x[:,1:3])

Output

```
[[44.0 72000.0]
 [27.0 48000.0]
 [30.0 54000.0]
 [38.0 61000.0]
 [40.0 63777.77777777778]
 [35.0 58000.0]
 [38.77777777777778 52000.0]
 [48.0 79000.0]
 [50.0 83000.0]
 [37.0 67000.0]]
```

11. Encoding the categorical data

- Apply one-hot encoding to the **'Country'** column using **'ColumnTransfomer'** and **'OneHotEncoder'**.

Program

```
from sklearn. compose import ColumnTransformer
from sklearn. preprocessing import OneHotEncoder
ct = ColumnTransformer (transformers = [('encoder', OneHotEncoder (), [0])], remainder = "passthrough")
x = np. array(ct.fit_transform(x))
print(x)
```

Output

```
[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

- Apply Label encode to the **'Purchased'** column using **'LabelEncoder'**.

Program

```
from sklearn. preprocessing import LabelEncoder
le = LabelEncoder ()
y=le.fit_transform(y)
print(y)
```

Output

[0 1 0 0 1 1 0 1 0 1]

12.Splitting the Dataset

Use **'train_test_split'** from scikit-learn to split the dataset into training a nd testing sets.

Program

```
from sklearn. model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split (x, y, test_size = 0.25, ran
dom_state = 1)
print(x_train)
```

Output

```
[[0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
```

Program

```
print(x_test)
```

Output

```
[[0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]]
```

Program

```
print(y_train)
```

Output

[1 0 0 1 1 0 1]

Program

```
print(y_test)
```

Output
[0 1 0]

13. Feature Scaling

Standardize the numerical features using **‘StandardScaler’** from scikit-learn.

Program
```
from sklearn. preprocessing import StandardScaler
scaler = StandardScaler ()
x_train[:,2:]=scaler.fit_transform(x_train[:,2:])
x_test[:,2:]=scaler.fit_transform(x_test[:,2:])
print(x_train)
print(x_test)
```

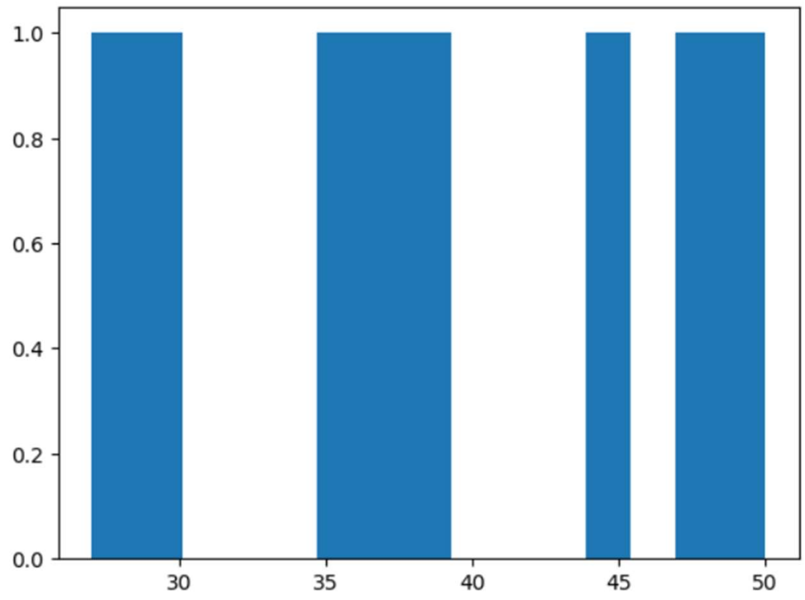
Output
```
[[0.0 1.0 -0.6324555320336758 -0.038910211282047996 -0.22960023388015188]
 [1.0 0.0 -0.6324555320336758 0.5058327466666259 0.49120534884662787]
 [0.0 0.0 1.5811388300841895 -0.3112816902563849 -0.4731156334500103]
 [0.0 0.0 1.5811388300841895 -1.809324824615238 -1.6127677034369463]
 [1.0 0.0 -0.6324555320336758 1.0505757046152997 1.1048641557626704]
 [0.0 1.0 -0.6324555320336758 1.3229471835896367 1.455526331143266]
 [1.0 0.0 -0.6324555320336758 -0.7198389087178904 -0.736112264985457]]
[[0.0 1.0 -0.7071067811865475 -1.3880272079128577 -0.5513801778287937]
 [1.0 0.0 -0.7071067811865475 0.4594174561401711 1.40351317992784]
 [0.0 0.0 1.4142135623730951 0.9286097517726866 -0.8521330020990451]]
```

14. Data Visualization

Visualize the distribution of the ‘Age’ column using a histogram.

Program
```
import matplotlib. pyplot as plt
plt. hist(dataset['Age'], bins=15)
plt. show ()
```

Output



15. Identify outliers

Identify the outliers using the quantile method.

Program

lowerLimit=dataset['Age']. quantile (0.05)
print(lowerLimit)

Output

28.05

Program

print(dataset[dataset['Age’] <lowerLimit])

Output

```
  Country   Age   Salary Purchased
1   Spain  27.0  48000.0       Yes
```

Program

upperLimit=dataset['Age']. quantile (0.95)
print(upperLimit)

Output

49.3

Program

```
print(dataset[dataset['Age']>upperLimit])
```

Output

```
  Country   Age   Salary Purchased
8  Germany  50.0  83000.0        No
```

16.Remove outliers from the dataset

Remove the outliers from the dataset using quantile method.

Program

```
dataset = dataset[(dataset['Age']>lowerLimit) & (dataset['Age’] <upperLimit)]
print(dataset)
```

Output

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
5   France  35.0  58000.0       Yes
7   France  48.0  79000.0       Yes
9   France  37.0  67000.0       Yes
```

Result

Thus, the performance of the data exploration and preprocessing techniques in Python using a real-world dataset was successfully completed.