Velammal College of Engineering and Technology, Madurai

(Autonomous)

Department of Artificial Intelligence &Data Science

Academic Year: 2025-2026

VCET Regulation 2021

LAB MANUAL

21AD303-DATA SCIENCE AND ANALYTICS LABORATORY

Prepared by **Verified by**

Mrs.A.Thirunirai Selvi AP/AI&DS **Dr. S. SASIKALA HOD/AI&DS**

VISION

To emerge and sustain as academic excellence in Artificial Intelligence and Data Science to produce ethical professionals through innovative research and education.

MISSION

- To promote industry ready graduates by acquiring intelligent data analytical skills.
- To empower the graduates towards research and application-oriented knowledge for higher studies.
- To equip the graduates with entrepreneurship skills to serve the needs of society.

Table of Contents

Ex: 1	NumPy ndarray Object

Aim:

To Create an empty and a full NumPy array

Description

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

Create a NumPy ndarray Object

create a NumPy ndarray object by using the array() function.

Eg: arr = np.array([8, 2, 3, 7, 5])

print(arr)

Dimensions in Arrays

- **0-D Arrays**

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Eg: arr = np.array(682)

print(arr)

- **1-D Arrays**

An array that has 0-D arrays as its elements

Eg: arr = np.array([8, 2, 3, 7, 5])
print(arr)

- **2-D Arrays**

An array that has 1-D arrays as its elements used to represent matrix

Eg: arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)

- **3-D arrays**

An array that has 2-D arrays (matrices) as its elements

Eg: arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)

- **Check Number of Dimensions**

Eg: c = np.array([[1, 2, 3], [4, 5, 6]])

print(c.ndim)

- **NumPy Array Indexing**

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1.we can use comma separated integers representing the dimension and the index of the element.

Eg: arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])

- **Access 3-D Arrays**

Eg: arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])

- **Negative Indexing**

To access an array from the end

Eg: arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])

Numpy - ndarray

Attributes of ndarray

1. **ndarray.shape:** Returns a tuple representing the shape (dimensions) of the array.
   arr2 = npy.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

   print(arr1.shape)

2. **ndarray.ndim:** Returns the number of dimensions (axes) of the array.
3. **ndarray.size:** Returns the total number of elements in the array.
4. **ndarray.dtype:** Provides the data type of the array elements.
5. **ndarray.itemsize**: Returns the size in bytes of each element

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Shape:", arr.shape)
print("Dimensions:", arr.ndim)
print("Size:", arr.size)
print("Data type:", arr.dtype)
print("Item size:", arr.itemsize)
```

Array Operations

Element-wise Operations

```
import numpy as np

arr = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

print(arr + arr2)

print(arr * arr2)

print(arr - arr2)

print(arr / arr2)
```

Matrix Operations (Dot product)

```
import numpy as np

matrix1 = np.array([[1, 2], [3, 4]])

matrix2 = np.array([[5, 6], [7, 8]])

print(np.dot(matrix1, matrix2))
```

Reshaping and Flattening

Change the shape of an array while keeping the data same.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

reshaped_arr = arr.reshape(2, 3)

print(reshaped_arr)
```

Arrays filled with zeros:

numpy.zeros(). --This function requires the shape of the array as its primary argument.

Access Array Elements

```
import numpy as np

array_1d = np.zeros(5)

print(array_1d)

array_2d = np.zeros((3, 4))

print(array_2d)
```

Specifying Data Type

```
import numpy as np


# Create an integer zero array
array_int = np.zeros((2, 3), dtype=int)
print(array_int)
```

Result:

Thus the empty and a full NumPy arrays are created.

Ex: 2 a. Remove rows in NumPy array that contains non-numeric Values.

Aim:

To write a program to remove rows in numPy array that contains non numeric values

Description

- **x = np.array(...)**' creates a 2D NumPy array 'x' with the given elements.
- **'np.isnan(x)'** creates a boolean array with the same shape as 'x', where each element is True if the corresponding element in 'x' is NaN and False otherwise.
- **np.isnan(x).any(axis=1):** Use the any() function along axis 1 (rows) to create a 1D boolean array, where each element is True if the corresponding row in 'x' contains at least one NaN value, and False otherwise.
- **~np.isnan(x).any(axis=1):** Apply the ~ bitwise negation operator to invert the boolean values in the 1D boolean array. Returns True if the corresponding row in 'x' does not contain any NaN values, and False otherwise.
- **x[~np.isnan(x).any(axis=1)]:** Use boolean indexing to select only the rows in 'x' for which the corresponding boolean value in the 1D boolean array is True.

  import numpy as np
- **Numpy. all()** is a NumPy library function that returns True if all elements of an array evaluate to True and False else.
- **numpy.any()** function tests whether any array elements along the mentioned axis evaluate to True.

Algorithm:

1. Take a numpy n-d array.
2. Determine the entire row containing 0's by specifying axis=1.
3. Traverse each row and check for the nan condition given in first parameter.
4. Remove rows that contain non numeric values using numpy.any() function.
5. Print the n-d array.

Program

```
import numpy as np

# Creating a  NumPy array

n_arr = np.array([[10,22,33],
```

```
        [23.45, 50, 78.7],
        [41, np.nan, np.nan],
        [5,15,25],
        [32, np.nan, 55]])
print("Given array:")
print(n_arr)
print("\nRemove all rows containing NaN values:")
cleaned_arr = n_arr[~np.isnan(n_arr).any(axis=1)]
print(cleaned_arr)
```

Result

Thus the program to remove rows in numPy array that contains non numeric values was written and executed successfully

Ex :2b Remove rows in NumPy array that contains value zero

Aim:

To remove rows in numpy array that contains zero values

Algorithm:

1. Take a numpy n-d array.
2. Determine the entire row containing 0's by specifying axis=1.
3. Traverse each row and check for the condition given in first parameter.
4. Remove rows that contain only zeroes using numpy.all() function.
5. Print the n-d array.

Program

```
import numpy as np

# take data

data = np.array([[1, 2, 3], [0, 0, 0], [4, 5, 6],

                 [0, 0, 0], [7, 8, 9], [0, 0, 0]])

# print original data having rows with all zeroes

print("Original Dataset")

print(data)


# remove rows having all zeroes

data = data[~np.all(data == 0, axis=1)]


# data after removing rows having all zeroes

print("After removing rows")

print(data)
```

Result

Thus the rows that contain zero values are removed in numpy arrays

Ex:3	Build all Combinations of two NumPy Arrays

Aim:
To generate an array containing all possible combinations of elements from two NumPy arrays

3.a) Computing combinations of elements of Two NumPy arrays

Algorithm

1. **Take two 2-dimensional numpy arrays**
2. utilize the **np.meshgrid()** function, to create coordinate grids from the arrays.
3. reshape the grids and generate a 2D array that holds every combination of values from the two input arrays.

Description:

np.meshgrid(a, b) -creates coordinate grids from the arrays a and b.

reshape(-1, 1) -results in array with a single column and multiple rows (a column vector).

reshape(1, -1)-results in array with a single row and multiple columns (a row vector).

Program:

```
import numpy as np
a = np.array([1, 2])
b = np.array([4, 6])
print("Array-1")
print(a)
print("\nArray-2")
print(b)
res = np.array(np.meshgrid(a, b)).T.reshape(-1, 2)
print("\nCombine array:")
print(res)
```

Output

```
Array-1
[1 2]
```

```
Array-2
[4 6]
Combine array:
[[1 4]
 [1 6]
 [2 4]
 [2 6]]
```

3.b Computing combinations of elements of Three NumPy arrays

Algorithm

1. Creating three 1D NumPy arrays.
2. Use **np.meshgrid()** to create coordinate grids from these arrays.
3. The result is then reshaped into a 2D array of coordinate by applying. T**.reshape(-1, 3)**.
4. Display all combination of values from the two input arrays.

Program:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 6, 4])
c = np.array([3, 6])
print("Array-1")
print(a)
print("Array-2")
print(b)
print("Array-3")
print(c)
res = np.array(  np.meshgrid(a, b, c)).T.reshape(-1, 3)


print("\nCombine array:")
print(res)
```

```
Array-1
[1 2 3]

Array-2
```

```
[4 6 4]

Array-3
[3 6]

Combine array:
[[1 4 3]
[1 6 3]
[1 4 3]
[2 4 3]
[2 6 3]
[2 4 3]
[3 4 3]
[3 6 3]
[3 4 3]
[1 4 6]
[1 6 6]
[1 4 6]
[2 4 6]
[2 6 6]
[2 4 6]
[3 4 6]
[3 6 6]
[3 4 6]]
```

Result:

Thus the array containing all possible combinations of elements generated from two NumPy arrays

Ex :4a Add a Border Around a NumPy Array

Aim:
To add a border of zeroes around a NumPy array.

Description:

np.pad(x, pad_width=1, mode='constant', constant_values=0): The np.pad function is used to pad the input array x with a border of specified width and values.

pad_width=1 adds a border of width 1 around the array.

mode='constant' specifies that the padding should be done with constant values, and constant_values=0 indicates that the constant value for padding should be 0.

Program:

```
import numpy as np

array = np.ones((3, 3))
bordered = np.pad(array, pad_width=1, mode='constant', constant_values=0)
print("Array with Border:\n", bordered)
```

```
Original array:
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
1 on the border and 0 inside in the array
[[ 0.  0.  0.  0.  0.]
 [ 0.  1.  1.  1.  0.]
 [ 0.  1.  1.  1.  0.]
 [ 0.  1.  1.  1.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Result:

Thus the numpy program to add a border of zeroes executed successfully.

Ex :4 b Create an 8x8 matrix and fill it with a checkerboard pattern.

Aim:

Write a numpy program Create a 8x8 matrix and fill it with a checkerboard pattern

Description

x[1::2,::2] = 1 Assigns the value 1 to every other element in every other row, starting from the second row (index 1).

x[::2,1::2] = 1 Assigns the value 1 to every other element in every other column, starting from the second column (index 1).

print(x): Prints the 8x8 NumPy array.

np.ones((3,3)): Creates a 3x3 NumPy array filled with ones.

np.zeros((8,8),dtype=int): Creates an 8x8 NumPy array filled with zeros, with an integer data type.

Program

```
# Importing the NumPy library with an alias 'np'
import numpy as np
# Creating a 3x3 NumPy array filled with ones
x = np.ones((3, 3))
# Printing the original array 'x'
print("Original array:")
print(x)
# Modifying the array 'x' to set 0s on the border and 1s inside the array using the np.pad function
print("0 on the border and 1 inside in the array")
x = np.pad(x, pad_width=1, mode='constant', constant_values=0)
# Printing the modified array 'x' with 0s on the border and 1s inside
print(x)
```

Result:

Thus an 8x8 matrix filled with a checkerboard pattern generated successfully

Ex :5 **Compare Two NumPy Arrays**

Aim:

To compare two arrays using numpy program.

5.a Element-wise Comparison:

Description:

Element-wise Comparison:

- Comparison Operators (e.g., ==, >, <, >=, <=, !=): These operators perform element-wise comparisons and return a boolean array of the same shape as the input arrays, where True indicates that the condition is met for the corresponding elements.

Program

```
import numpy as np

   arr1 = np.array([1, 2, 3],[4,5,6)
   arr2 = np.array([1,2,3][7,5,6])
print("array1")
print(arr1)
print("array2")
print(arr2)

print(result_equal = arr1 == arr2)
```

Output:

```
array1
[[1 2 3]
 [4 5 6]]
array2
[[1 2 3]
 [7 5 6]]
[[ True  True  True]
 [False  True  True]]
```

- **np.equal(), np.greater():** NumPy provides functions that mirror the comparison operators, offering similar element-wise comparisons.

Program

```
import numpy as np

 arr1 = np.array([101, 99,87])
```

```
arr2 = np.array([897,97,111]
print(arr1 == arr2)

print(arr1 >= arr2)

print(arr1 <=arr2)

print(arr1 <= arr2)
```

```
Array a:  [101  99  87]
Array b:  [897  97 111]
a > b
[False  True False]
a >= b
[False  True False]
a < b
[ True False  True]
a <= b
[ True False  True]
```

5.b Overall Array Equality:

- **np.array_equal():** This function checks if two arrays have the same shape and elements, returning a single boolean value (True or False). It requires exact equality.

Program

```
import numpy as np

    arr1 = np.array([1, 2, 3])
    arr2 = np.array([1, 2, 3])
    arr3 = np.array([1, 2, 4])

    are_equal = np.array_equal(arr1, arr2) # Returns True
    are_equal_false = np.array_equal(arr1, arr3) # Returns False
```

Output:

```
array1
[[1 2 3]
 [4 5 6]]
array2
[[1 2 3]
 [7 2 6]]
```

```
array2
[[1 2 3]
 [4 5 6]]
arr1==arr2
False
arr1==arr3
True
```

5.c Set Operations:

Description

- **np.intersect1d():** Finds common elements between two arrays.
- **np.setdiff1d():** Finds elements present in one array but not in another.
- **np.union1d():** Finds all unique elements from both arrays.

```
import numpy as np

 arr_a = np.array([1, 2, 3],[4, 5, 6])
 arr_b = np.array([1,2,3], [7,2,6])

arr_c = np.array([1,2,3], [7,8,6])

diff_a_b = np.setdiff1d(arr_a, arr_b) # Returns array([1, 2])
   diff_b_a = np.setdiff1d(arr_b, arr_a) # Returns array([6, 7])
```

Output:

```
array1
[[1 2 3]
 [4 5 6]]
array2
[[1 2 3]
 [7 2 6]]
array2
[[1 2 3]
 [7 8 6]]
common Element in arr1, arr2
[1 2 3 6]
Common Elements in arr1,arr3
[4 5]
```

Exercise

1. **Write a NumPy program to append values to the end of an array.**

2. Write a NumPy program to convert Centigrade degrees into Fahrenheit degrees. Centigrade values are stored in a NumPy array.

Ex :6	Data Frame from Dictionary Data

Aim:

Write a Pandas program to create and display a Data Frame from a specified dictionary data which has the index labels.

Algorithm

1. **Import the pandas library:** .
2. **Define dictionary data:** The keys of the dictionary will typically become the column names, and the values will be lists or arrays representing the data for each column.
3. **Define index labels:** Create a list or array containing the desired labels for the rows of your DataFrame.
4. **Create the DataFrame:** Use pd.DataFrame() and pass both the dictionary data and the index labels.
5. **Display the DataFrame:** Print the DataFrame object to see its contents.

Program

```
import pandas as pd

# 1. Define the dictionary data
data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David'],
   'Age': [25, 30, 35, 28],
   'City': ['New York', 'London', 'Paris', 'Tokyo']
}

# 2. Define the index labels
index_labels = ['Person_A', 'Person_B', 'Person_C', 'Person_D']

# 3. Create the DataFrame with specified index
df = pd.DataFrame(data, index=index_labels)

# 4. Display the DataFrame
print(df)
```

Result:

Thus the Pandas program to create and display a Data Frame from a specified dictionary data which has the index labels executed successfully.

Ex: 7 Get the first 3 rows of a given DataFrame

Aim:

To Write a Pandas program to get the first 3 rows of a given DataFrame.

Algorithm

1. Creates a Pandas DataFrame named df with columns 'name', 'score', 'attempts', and 'qualify', and a custom index 'labels'.
2. Selects and print the first three rows of the DataFrame using the .iloc indexing method.

Program

```
import pandas as pd
import numpy as np

exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam_data , index=labels)
print("First three rows of the data frame:")
print(df.iloc[:3])
```

Output:

```
First three rows of the data frame:
   attempts      name  qualify score
a         1     Anastasia     yes   12.5
b         3      Dima          no    9.0
c         2      Katherine     yes   16.5
```

Result:

Thus the Pandas program to get the first 3 rows of a given DataFrame executed successfully.

Ex: 8 Draw a line with Title, X axis, Y axis

Aim:

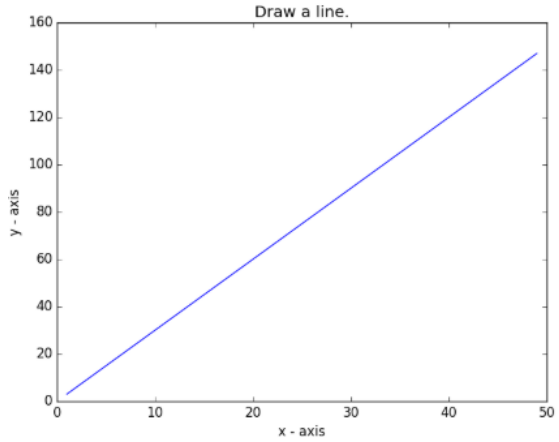
To Write a Python program to draw a line with suitable label in the x axis, y axis and a title.

Algorithm:

1. Import matplotlib library
2. Specify the ranges for visualization in X axis and Y axis
3. Plot the x axis, y axis along with labels
4. Set the title and display the output

Program

```
import matplotlib.pyplot as plt

X = range(1, 50)

Y = [value * 3 for value in X]

print("Values of X:")

print(*range(1,50))

print("Values of Y (thrice of X):")

print(Y)

# Plot lines and/or markers to the Axes.

plt.plot(X, Y)

# Set the x axis label of the current axis.

plt.xlabel('x - axis')

# Set the y axis label of the current axis.

plt.ylabel('y - axis')

# Set a title

plt.title('Draw a line.')

# Display the figure.

plt.show()
```



Result:

Thus the Python program to draw a line with suitable label in the x axis, y axis and a title was executed.

Ex: 9 Draw Line Charts Of The Financial Data

Aim:

To Write a Python program to draw line charts of the financial data of Alphabet Inc. between October 3, 2024 to October 7, 2024.

Algorithm

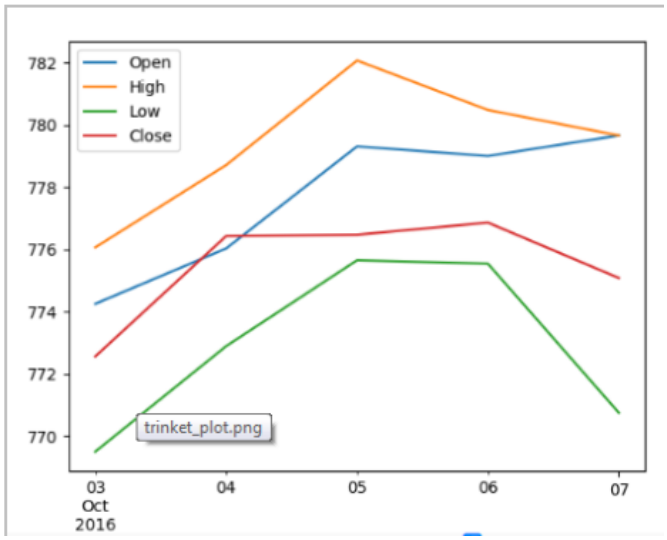
1. Import Matplotlib and Pandas library
2. Read the csv file
3. Plot the data using Plot function
4. Show the Line chart

Sample Financial data (fdata.csv):

Date,Open,High,Low,Close
10-03-16,774.25,776.065002,769.5,772.559998
10-04-16,776.030029,778.710022,772.890015,776.429993
10-05-16,779.309998,782.070007,775.650024,776.469971
10-06-16,779,780.47998,775.539978,776.859985
10-07-16,779.659973,779.659973,770.75,775.080017

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('fdata.csv', sep=',', parse_dates=True, index_col=0)
df.plot()
plt.show()
```

Output:



Result:

Thus, the Python program to draw line charts of the financial data of Alphabet Inc. between October 3, 2024 to October 7, 2024 was executed successfully.

Ex: 10 Cumulative frequency distribution

Aim:

To Write a python program for the following table that gives the values of runs scored by Virat Kohli in last 25 T-20 matches. Represent the data in the form of less than type cumulative frequency distribution:

45	34	50	75	22
56	63	70	49	33
08	14	39	86	52
92	88	70	56	50
57	45	42	12	39

Program:

```
def less_than_cumulative_frequency(data):
    """
Calculates the less than type cumulative frequency distribution.

    Args:     data: A list of integers representing the runs scored.

    Returns:        A dictionary where keys are "less than" values and values are cumulative
frequencies.

    """      # Sort the data in ascending order
    data.sort()
    # Create a dictionary to store the cumulative frequencies
    cumulative_frequencies = {}
    # Iterate through the sorted data and calculate cumulative frequencies
    cumulative_count = 0
    for i, score in enumerate(data):
        cumulative_count += 1
        cumulative_frequencies[f"Less than {score + 1}"] = cumulative_count
    return cumulative_frequencies

runs_scored = [45, 34, 50, 75, 22, 56, 63, 70, 49, 33, 0, 8, 14, 39, 86, 92, 88, 70, 56, 50, 57, 45, 42,
12, 39]
cumulative_distribution = less_than_cumulative_frequency(runs_scored)

# Print the results in the specified format
for key, value in cumulative_distribution.items():
    print(f"Runs scored by Virat Kohli: {key} | Cumulative Frequency: {value}")
```

Result:

Thus the cumulative Frequency Distribution generated using Python program

Ex No: 11 Find the sum and average of n integer numbers.

Aim:

To Write a Program to find the sum and average of n integer numbers.

Algorithm

1. Import numpy library
2. Read the required number of inputs to find sum and average .
3. Read the list of numbers
4. Convert list to numpy array
5. Calculate sum and average using sum and mean function
6. Display the sum and average

Program

```
import numpy as np

# Input: number of elements

n = int(input("Enter how many numbers you want to input: "))

# Input: list of numbers

numbers = []

for i in range(n):

    num = int(input(f"Enter number {i+1}: "))

    numbers.append(num)

# Convert list to NumPy array

arr = np.array(numbers)

# Calculate sum and average

total = np.sum(arr)

average = np.mean(arr)


# Output

print(f"\nSum of the numbers: {total}")

print(f"Average of the numbers: {average}")
```

Input:

Enter how many numbers you want to input: 4

Enter number 1: 10

Enter number 2: 20

Enter number 3: 30

Enter number 4: 40

Output:

Sum of the numbers: 100

Average of the numbers: 25.0

Result:

Thus the python Program to find the sum and average of n integer numbers was executed successfully.

Ex :12 Find the variance and standard deviation of set of elements.

Aim

To find the variance and standard deviation of set of elements using python program.

Description:

- **import numpy as np:**

  This line imports the NumPy library, which provides efficient numerical operations, including statistical functions.

- **data = np.array([...]):**

  This creates a NumPy array named data containing the set of elements for which you want to calculate the variance and standard deviation.

- **np.var(data):**

  This function calculates the variance of the elements in the data array. By default, it calculates the population variance (dividing by N). If you need the sample variance (dividing by N-1, Bessel's correction), you can use np.var(data, ddof=1).

- **np.std(data):**

  This function calculates the standard deviation of the elements in the data array. Similar to np.var(), it calculates the population standard deviation by default. For sample standard deviation, use np.std(data, ddof=1)

Program

```
import numpy as np

# Define a set of elements (as a NumPy array)
data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Calculate the variance
variance = np.var(data)

# Calculate the standard deviation
standard_deviation = np.std(data)
# Print the results
print(f"The given data set is: {data}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {standard_deviation}")
```

Result:

Thus, the variance and standard deviation of set of elements was calculated using python program.

Ex 13 Program to plot a normal distribution in python.

Aim:

To write a python program to plot a normal distribution.

Description

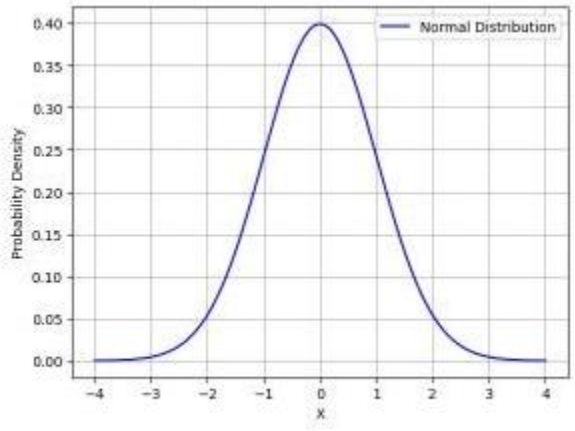
scipy.stats.norm.pdf() :

calculates the probability density function (PDF) for a normal distribution. This method allows for direct computation and visualization of the standard normal curve (mean = 0, standard deviation = 1). It is useful for precise statistical modeling and probability calculations**.**

Using scipy.stats.norm.pdf()

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

x = np.linspace(-4, 4, 1000)
plt.plot(x, norm.pdf(x), 'b-', label='Normal Distribution')
plt.xlabel('X')
plt.ylabel('Probability Density')
plt.legend()
plt.grid(True)
plt.show()
```

Output



Standard Normal Distribution

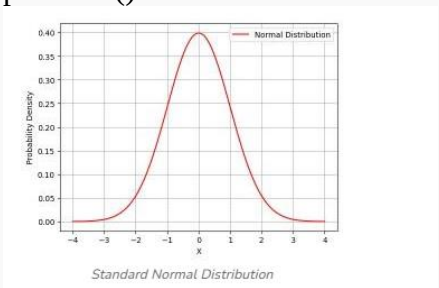
Using numpy

Algorithm

1. Generate 1000 evenly spaced x-values from -4 to 4 with np.linspace()
2. Compute the probability density function (PDF) using the Gaussian formula $\exp(-x^2/2) / \sqrt{(2\pi)}$.
3. The distribution is plotted as a red line, with labeled axes, a legend and a grid for clarity.

Program

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-4, 4, 1000)
y = np.exp(-x**2 / 2) / np.sqrt(2 * np.pi)

plt.plot(x, y, 'r-', label='Normal Distribution')
plt.xlabel('X')
plt.ylabel('Probability Density')
plt.legend()
plt.grid()
plt.show()
```



Standard Normal Distribution

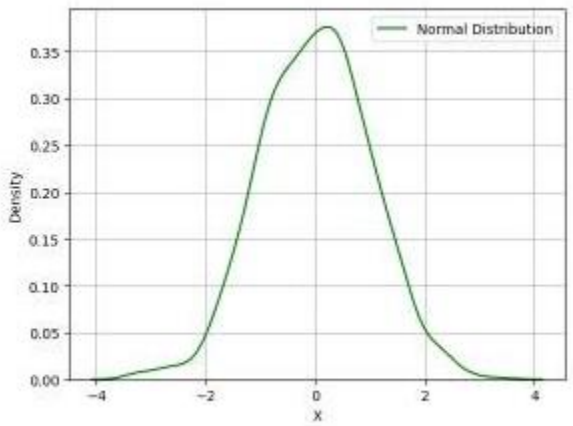
Using Seaborn sns.kdeplot()

Description

sns.kdeplot() function --takes a randomly generated normal dataset and plots its density curve, providing a smoothed representation of the underlying distribution

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = np.random.normal(loc=0, scale=1, size=1000)

sns.kdeplot(data, color='green', label='Normal Distribution')

# Labels and title
plt.xlabel('X')
plt.ylabel('Density')
```

```
plt.legend()
plt.grid()
plt.show()
```



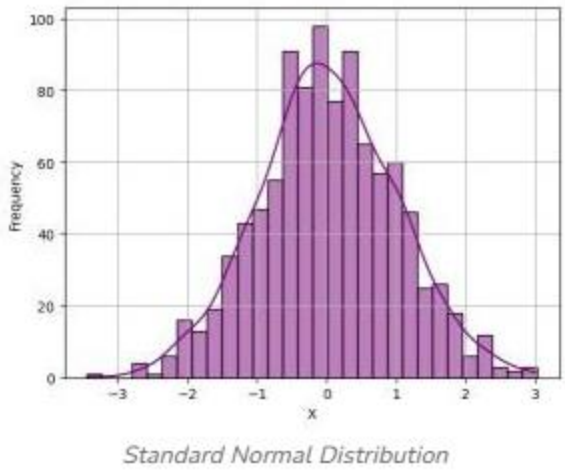
Standard Normal Distribution

Using histogram

Description

Seaborn's sns.histplot() with kde=True, visualize both the histogram and its corresponding KDE curve, offering insights into data distribution patterns

```
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

data = np.random.normal(loc=0, scale=1, size=1000)

sns.histplot(data, kde=True, bins=30, color='purple')

# Labels and title

plt.xlabel('X')

plt.ylabel('Frequency')

plt.grid()

plt.show()
```



Result:

Thus, python program to plot a normal distribution executed successfully.

Ex 14 Plot Correlation and scatter plots.

Aim:

To write python Program to plot a Correlation and scatter plots.

Algorithm:

1. Import matplotlib and numpy libraries
2. Input the number of data points
3. Enter X values and Y values
4. Convert to numpy arrays.
5. Calculate Coorelation coefficient
6. Plotter scatterplot and best fit line

Program for Correlation

```
import numpy as np
import matplotlib.pyplot as plt


# Input: number of data points
n = int(input("Enter the number of data points: "))


# Input: x values
x = []
print("\nEnter X values:")
for i in range(n):
    val = float(input(f"X[{i+1}]: "))
    x.append(val)


# Input: y values
y = []
print("\nEnter Y values:")
for i in range(n):
    val = float(input(f"Y[{i+1}]: "))
    y.append(val)
# Convert to numpy arrays
x = np.array(x)
```

```
y = np.array(y)
# Calculate correlation coefficient
correlation = np.corrcoef(x, y)[0, 1]
print(f"\nCorrelation coefficient: {correlation:.2f}")
# Plotting the scatter plot
plt.scatter(x, y, color='green', label='Data Points')
plt.title('Scatter Plot with Correlation')
plt.xlabel('X values')
plt.ylabel('Y values')
# Plot best-fit line (regression)
m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x + b, color='red', label='Best Fit Line')
plt.legend()
plt.grid(True)
plt.show()
```

Input:

Enter the number of data points: 5

Enter X values:

X[1]: 10

X[2]: 20

X[3]: 30

X[4]: 40

X[5]: 50

Enter Y values:

Y[1]: 15

Y[2]: 25

Y[3]: 35

Y[4]: 40

Y[5]: 55

Program for Scatter plots

```
import numpy as np
import matplotlib.pyplot as plt

# Input number of data points
n = int(input("Enter the number of data points: "))

# Input x and y values
x = []
y = []

print("\nEnter the X values:")
for i in range(n):
    x.append(float(input(f"X[{i+1}]: ")))

print("\nEnter the Y values:")
for i in range(n):
    y.append(float(input(f"Y[{i+1}]: ")))

# Convert to numpy arrays
x = np.array(x)
y = np.array(y)

# Linear regression using least squares
x_mean = np.mean(x)
y_mean = np.mean(y)

numerator = np.sum((x - x_mean) * (y - y_mean))
denominator = np.sum((x - x_mean) ** 2)
m = numerator / denominator
```

```
c = y_mean - m * x_mean
# Prediction
y_pred = m * x + c
# Output
print(f"\nLinear Regression Equation: y = {m:.2f}x + {c:.2f}")
# Plot
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, y_pred, color='red', label='Best Fit Line')
plt.title("Linear Regression (NumPy + Input)")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid(True)
plt.show()
```

Result:

Thus, the python Program to plot a Correlation and scatter plots was executed successfully.

Ex 15: Linear Regression and Logistic Regression

Aim:

To write a python program for Linear Regression and Logistic Regression

Algorithm:

1. Input number of data points for logistic regression
2. Enter X values , Y values,
3. Train the model using sigmoid function
4. Predict for plotting using linspace () function
5. Display the final weights and Plot the linear regression and logistic regression model

Program for Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
def compute_loss(y, y_pred):
    return -np.mean(y * np.log(y_pred + 1e-9) + (1 - y) * np.log(1 - y_pred + 1e-9))
def logistic_regression(x, y, lr=0.1, epochs=1000):
    m = len(x)
    w = 0.0
    b = 0.0

    for i in range(epochs):
        z = w * x + b
        y_pred = sigmoid(z)

        dw = np.dot((y_pred - y), x) / m
        db = np.sum(y_pred - y) / m

        w -= lr * dw
```

```
        b -= lr * db

        if i % 200 == 0:
            loss = compute_loss(y, y_pred)
            print(f"Epoch {i}: Loss = {loss:.4f}")

    return w, b

# Input number of data points
n = int(input("Enter number of data points: "))

# Input x and y values
x = []
y = []

print("\nEnter X values:")
for i in range(n):
    x.append(float(input(f"X[{i+1}]: ")))

print("\nEnter Y values (0 or 1):")
for i in range(n):
    y.append(int(input(f"Y[{i+1}]: ")))
x = np.array(x)
y = np.array(y)

# Train model
w, b = logistic_regression(x, y)

# Predict for plotting
x_test = np.linspace(min(x)-1, max(x)+1, 100)
y_pred = sigmoid(w * x_test + b)
```

```
# Output
print(f"\nFinal weights: w = {w:.4f}, b = {b:.4f}")
# Plot
plt.plot(x_test, y_pred, color='green', label='Predicted Probability')
plt.scatter(x, y, color='blue', label='Input Data')
plt.title("Logistic Regression (NumPy + Input)")
plt.xlabel("X")
plt.ylabel("Probability")
plt.legend()
plt.grid(True)
plt.show()
```

For linear regression:

Enter the number of data points: 4

X[1]: 1

X[2]: 2

X[3]: 3

X[4]: 4

Y[1]: 2

Y[2]: 3

Y[3]: 5

Y[4]: 7

For logistic regression:

Enter number of data points: 6

X[1]: 1

X[2]: 2

X[3]: 3

X[4]: 4

X[5]: 5

X[6]: 6