APPLICATION OF MACHINE LEARNING IN INDUSTRIES

ASSIGNMENT-1

**NAME: AKANKSHA GUPTA**

**SAP ID: 500091659**

**BATCH: 2, AIML(NH)**

**TASK-1**

import pandas as pd

# I. Load the dataset into a Pandas DataFrame from CSV file

file\_path = 'randomdataset.csv'  # Replace with the actual path to your CSV file

df = pd.read\_csv(file\_path)

# II. Display the first 5 rows, check for missing values, and get a summary

print("First 5 rows of the dataset:")

print(df.head())

# Check for missing values

missing\_values = df.isnull().sum()

print("\nMissing values:\n", missing\_values)

# No missing values in this dataset, so no need for handling

# Get a summary of the dataset

summary = df.describe()

print("\nSummary of the dataset:\n", summary)

# III. Select a subset of columns using label-based and position-based indexing

# Label-based indexing

subset\_label = df[['EmployeeID', 'Name', 'Salary']]

print("\nSubset of columns using label-based indexing:\n", subset\_label.head())

# Position-based indexing

subset\_position = df.iloc[:, [0, 1, 3]]

print("\nSubset of columns using position-based indexing:\n", subset\_position.head())

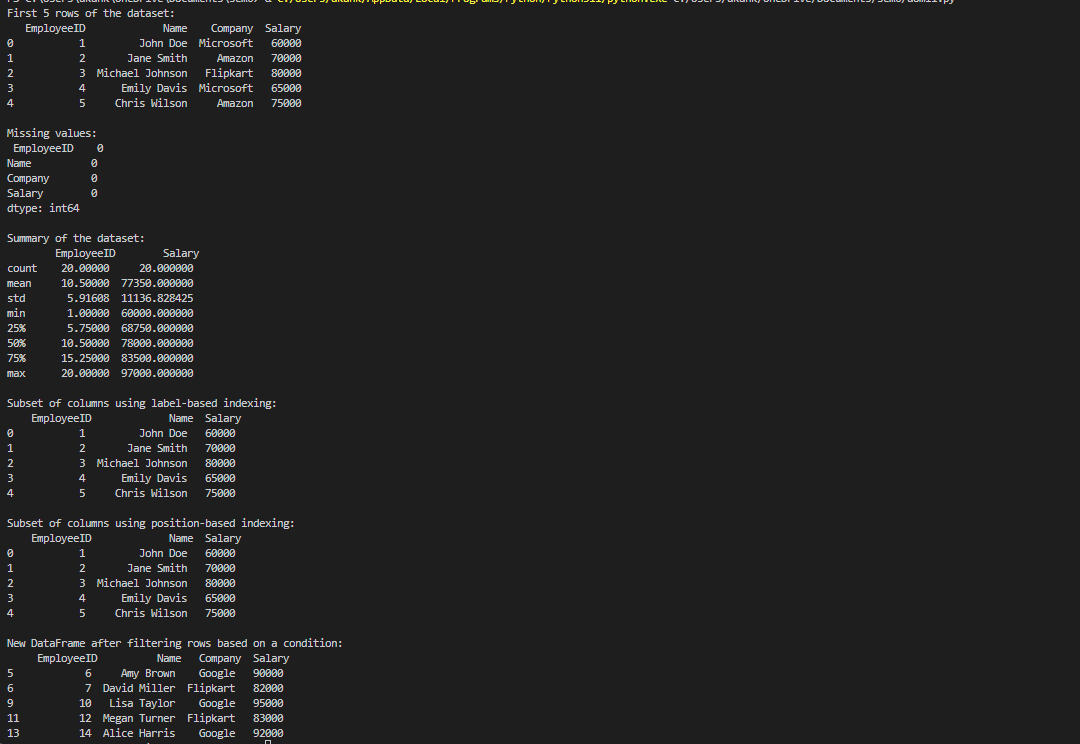
# Create a new DataFrame by filtering rows based on a condition

condition = df['Salary'] > 80000

filtered\_df = df[condition]

print("\nNew DataFrame after filtering rows based on a condition:\n", filtered\_df.head())

Output:

****

**TASK 2**

import pandas as pd

# I. Load the dataset into a Pandas DataFrame from CSV file

file\_path = 'randomdataset.csv'  # Replace with the actual path to your CSV file

df = pd.read\_csv(file\_path)

# I. Identify missing values and handle them

missing\_values = df.isnull().sum()

print("Missing values before handling:\n", missing\_values)

# For the sake of illustration, let's fill missing values in 'Salary' with the mean salary

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

# Check again after handling missing values

missing\_values\_after\_handling = df.isnull().sum()

print("\nMissing values after handling:\n", missing\_values\_after\_handling)

# II. Create a new column and convert a categorical variable

# Create a new column 'Salary\_Adjusted' by multiplying 'Salary' by 1.1

df['Salary\_Adjusted'] = df['Salary'] \* 1.1

# Convert 'Company' column to numerical representation using one-hot encoding

df\_encoded = pd.get\_dummies(df, columns=['Company'], prefix='Company')

# III. Group the data by the 'Company' column and apply aggregation functions

grouped\_data = df.groupby('Company').agg({

    'Salary': ['sum', 'mean', 'count'],

    'Salary\_Adjusted': 'sum'

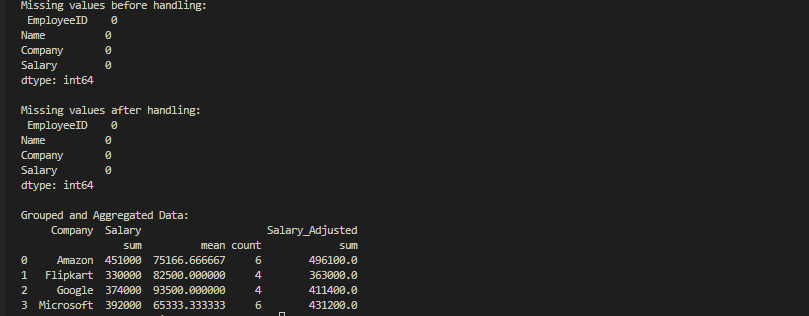
}).reset\_index()

# Present the results in a meaningful way

print("\nGrouped and Aggregated Data:")

print(grouped\_data)

Output:



**TASK-3**

import pandas as pd

# Load datasets from CSV files

df1 = pd.read\_csv('dataset1.csv')

df2 = pd.read\_csv('dataset2.csv')

# Display the first few rows of each dataset

print("Dataset 1:")

print(df1.head())

print("\nDataset 2:")

print(df2.head())

# Check for common columns in both datasets

common\_columns = set(df1.columns) & set(df2.columns)

if 'CommonColumn' not in common\_columns:

    print("\nError: 'CommonColumn' not found in both datasets.")

    print("Please adjust the column names for merging.")

else:

    # Perform different types of joins

    inner\_join = pd.merge(df1, df2, on='ID', how='inner')

    outer\_join = pd.merge(df1, df2, on='ID', how='outer')

    left\_join = pd.merge(df1, df2, on='ID', how='left')

    right\_join = pd.merge(df1, df2, on='ID', how='right')

    # Analyze the impact of each type of join

    print("\nInner Join Result:")

    print(inner\_join)

    print("\nOuter Join Result:")

    print(outer\_join)

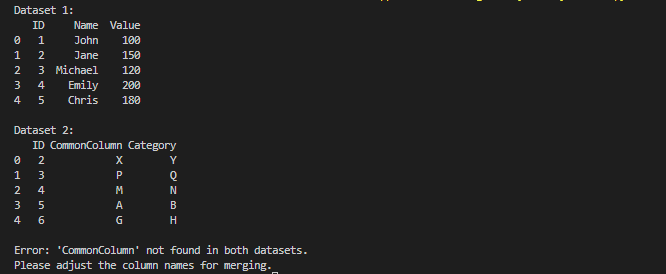
    print("\nLeft Join Result:")

    print(left\_join)

    print("\nRight Join Result:")

    print(right\_join)

**Output:**

****

**TASK-4**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Create a sample DataFrame for visualization

data = {

    'Age': np.random.randint(20, 60, 100),

    'Income': np.random.normal(50000, 10000, 100),

    'Score': np.random.normal(70, 10, 100),

    'Sales': np.random.randint(100, 1000, 100)

}

df = pd.DataFrame(data)

# I. Create a bar plot, line plot, and scatter plot

# Bar Plot

plt.figure(figsize=(10, 5))

df['Age'].value\_counts().sort\_index().plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Bar Plot of Age Distribution')

plt.xlabel('Age')

plt.ylabel('Count')

plt.show()

# Line Plot

plt.figure(figsize=(10, 5))

df['Income'].sort\_values().reset\_index(drop=True).plot(kind='line', color='green')

plt.title('Line Plot of Income Distribution')

plt.xlabel('Index')

plt.ylabel('Income')

plt.show()

# Scatter Plot

plt.figure(figsize=(10, 5))

plt.scatter(df['Score'], df['Sales'], color='orange', alpha=0.7)

plt.title('Scatter Plot of Score vs. Sales')

plt.xlabel('Score')

plt.ylabel('Sales')

plt.show()

# II. Visualize the correlation matrix

correlation\_matrix = df.corr()

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', linewidths=.5)

plt.title('Correlation Matrix')

plt.show()

# III. Create histograms and box plots for numerical columns

# Histograms

df.hist(figsize=(10, 8), bins=20, color='lightblue', edgecolor='black')

plt.suptitle('Histograms of Numerical Columns', y=1.02)

plt.show()

# Box Plots

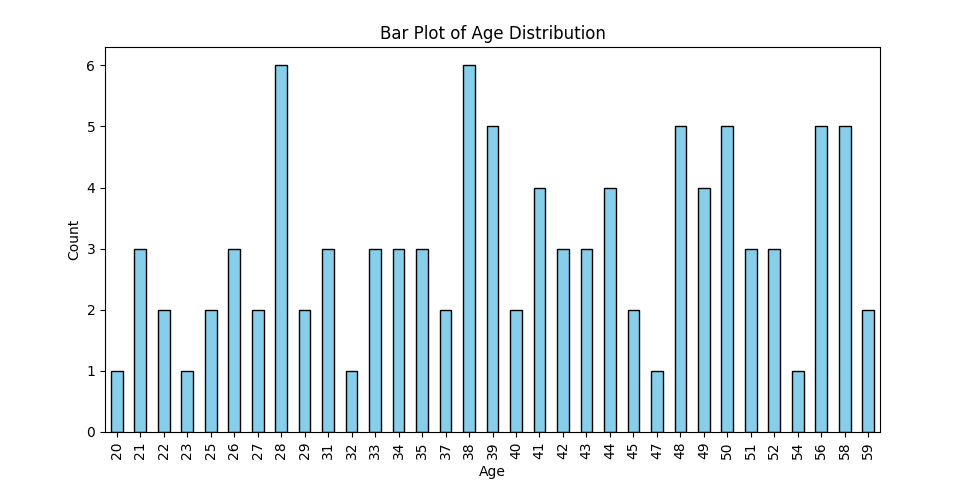
plt.figure(figsize=(10, 6))

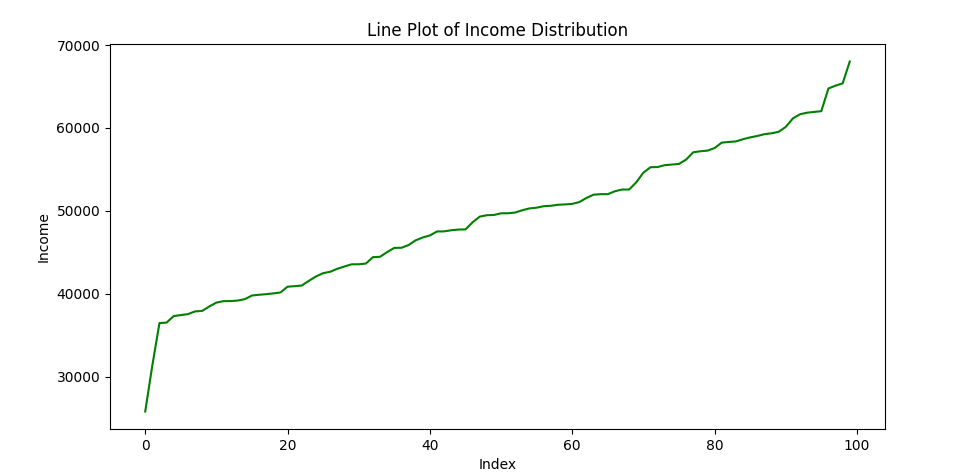
sns.boxplot(data=df, palette='pastel')

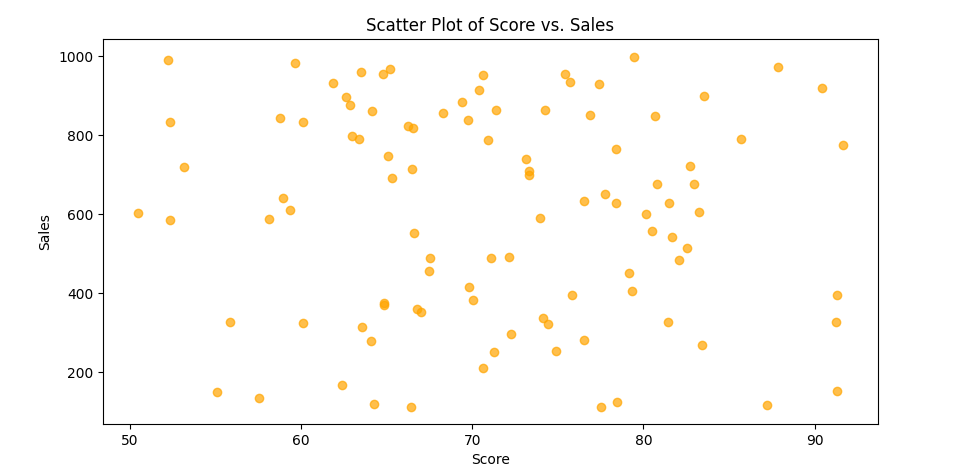
plt.title('Box Plots of Numerical Columns')

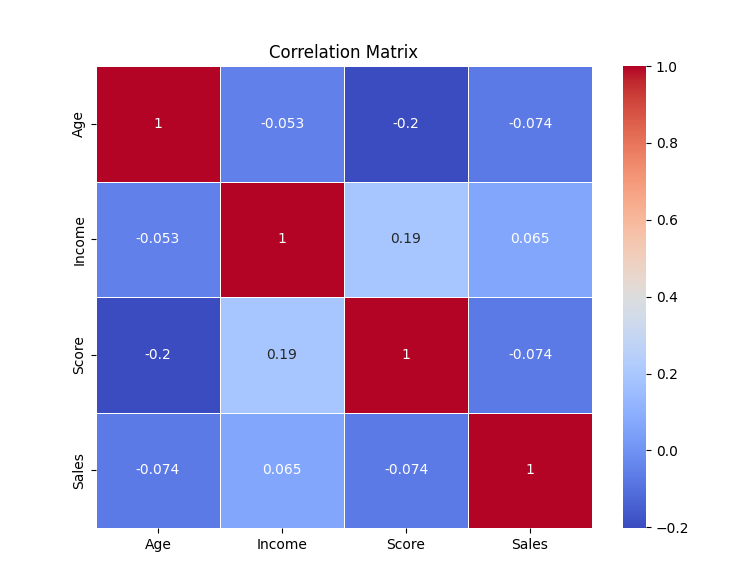
plt.show()

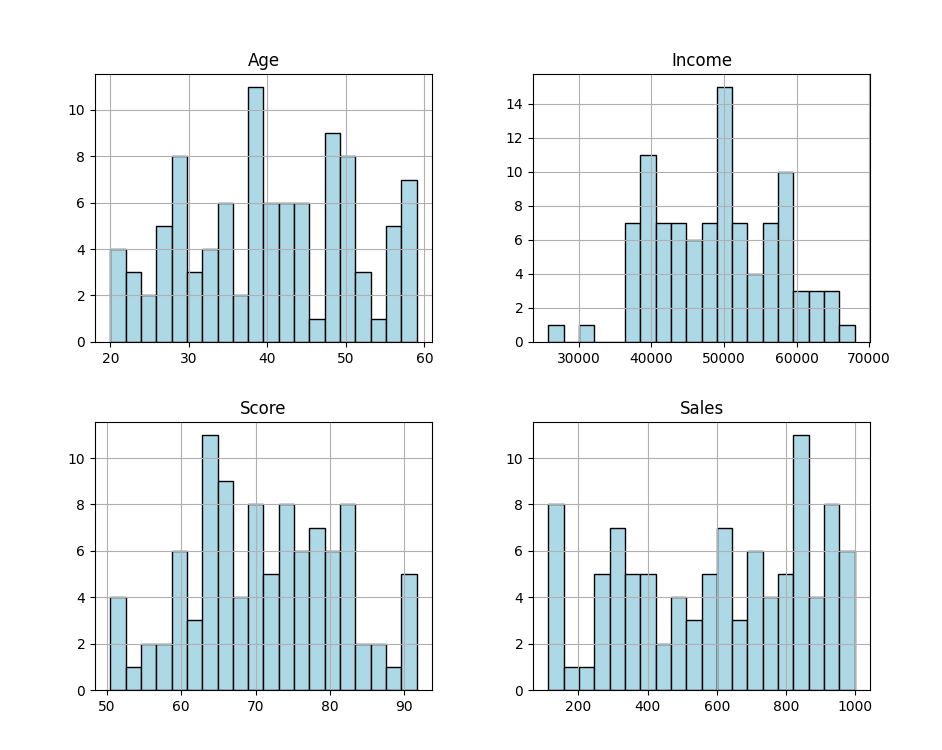
**Output:**

****

****

****

****

****

**TASK-5**

import numpy as np

# 1. Create a NumPy array 'arr' with values from 1 to 10.

arr = np.arange(1, 11)

# 2. Create another NumPy array 'arr2' with values from 11 to 20.

arr2 = np.arange(11, 21)

# 3. Add, subtract, multiply, and divide 'arr' and 'arr2'. Print the results.

addition\_result = arr + arr2

subtraction\_result = arr - arr2

multiplication\_result = arr \* arr2

division\_result = arr / arr2

# Print the results

print("Array 'arr':", arr)

print("Array 'arr2':", arr2)

print("\nAddition Result:")

print(addition\_result)

print("\nSubtraction Result:")

print(subtraction\_result)

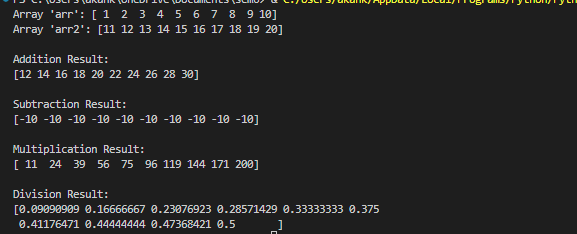
print("\nMultiplication Result:")

print(multiplication\_result)

print("\nDivision Result:")

print(division\_result)

**Output**

****

**TASK-6**

import numpy as np

# Given arrays

arr = np.arange(1, 11)

arr2 = np.arange(11, 21)

# 1. Reshape 'arr' into a 2x5 matrix.

reshaped\_arr = arr.reshape(2, 5)

# 2. Transpose the matrix obtained in the previous step.

transposed\_matrix = reshaped\_arr.T

# 3. Flatten the transposed matrix into a 1D array.

flattened\_array = transposed\_matrix.flatten()

# 4. Stack 'arr' and 'arr2' vertically. Print the result.

stacked\_arrays = np.vstack((arr, arr2))

# Print the results

print("Original Array 'arr':", arr)

print("Original Array 'arr2':", arr2)

print("\n1. Reshaped Array:")

print(reshaped\_arr)

print("\n2. Transposed Matrix:")

print(transposed\_matrix)

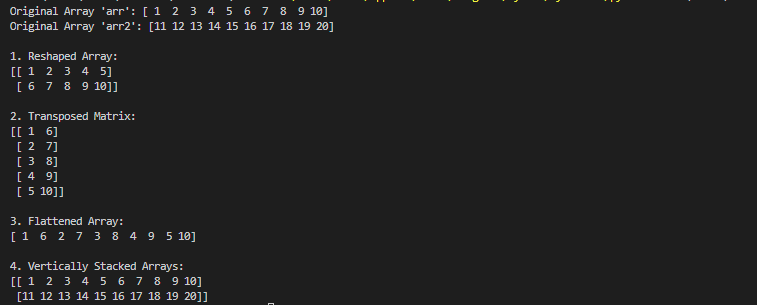
print("\n3. Flattened Array:")

print(flattened\_array)

print("\n4. Vertically Stacked Arrays:")

print(stacked\_arrays)

**Output**

****

**TASK- 7**

import numpy as np

# Given array

arr = np.arange(1, 11)

# 1. Calculate the mean, median, and standard deviation of 'arr'.

mean\_value = np.mean(arr)

median\_value = np.median(arr)

std\_deviation = np.std(arr)

# 2. Find the maximum and minimum values in 'arr'.

max\_value = np.max(arr)

min\_value = np.min(arr)

# 3. Normalize 'arr' (subtract the mean and divide by the standard deviation).

normalized\_arr = (arr - mean\_value) / std\_deviation

# Print the results

print("Original Array 'arr':", arr)

print("\n1. Statistical Measures:")

print("   Mean:", mean\_value)

print("   Median:", median\_value)

print("   Standard Deviation:", std\_deviation)

print("\n2. Maximum and Minimum Values:")

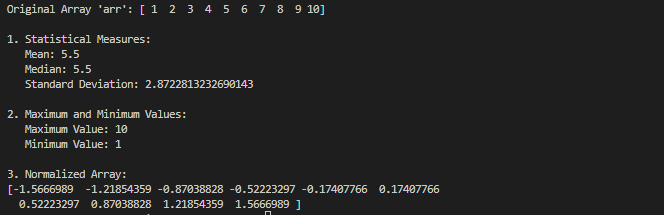
print("   Maximum Value:", max\_value)

print("   Minimum Value:", min\_value)

print("\n3. Normalized Array:")

print(normalized\_arr)

**Output:**

****

**TASK-8**

import numpy as np

# Given array

arr = np.arange(1, 11)

# 1. Create a boolean array 'bool\_arr' for elements in 'arr' greater than 5.

bool\_arr = arr > 5

# 2. Use 'bool\_arr' to extract the elements from 'arr' that are greater than 5.

filtered\_elements = arr[bool\_arr]

# Print the results

print("Original Array 'arr':", arr)

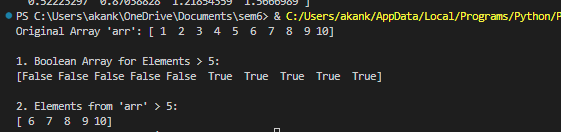
print("\n1. Boolean Array for Elements > 5:")

print(bool\_arr)

print("\n2. Elements from 'arr' > 5:")

print(filtered\_elements)

**Output:**

****

**TASK-9**

import numpy as np

# 1. Generate a 3x3 matrix with random values between 0 and 1.

random\_matrix = np.random.rand(3, 3)

# 2. Create an array of 10 random integers between 1 and 100.

random\_integers = np.random.randint(1, 101, 10)

# Given array for task 3

arr = np.arange(1, 11)

# 3. Shuffle the elements of 'arr' randomly.

shuffled\_arr = np.random.permutation(arr)

# Print the results

print("1. 3x3 Matrix with Random Values:")

print(random\_matrix)

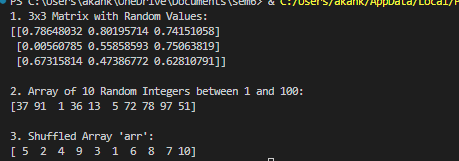
print("\n2. Array of 10 Random Integers between 1 and 100:")

print(random\_integers)

print("\n3. Shuffled Array 'arr':")

print(shuffled\_arr)

**Output:**

****

**TASK-10**

import numpy as np

# Given array

arr = np.arange(1, 11)

# 1. Apply the square root function to all elements in 'arr'.

sqrt\_arr = np.sqrt(arr)

# 2. Use the exponential function to calculate e^x for each element in 'arr'.

exp\_arr = np.exp(arr)

# Print the results

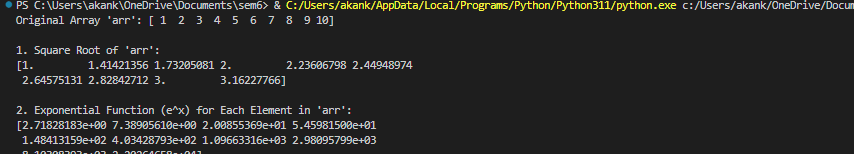
print("Original Array 'arr':", arr)

print("\n1. Square Root of 'arr':")

print(sqrt\_arr)

print("\n2. Exponential Function (e^x) for Each Element in 'arr':")

print(exp\_arr)

**Output:  
**

**TASK-11**

import numpy as np

# 1. Create a 3x3 matrix 'mat\_a' with random values.

mat\_a = np.random.rand(3, 3)

# 2. Create a 3x1 matrix 'vec\_b' with random values.

vec\_b = np.random.rand(3, 1)

# 3. Multiply 'mat\_a' and 'vec\_b' using the dot product.

result = np.dot(mat\_a, vec\_b)

# Print the results

print("1. 3x3 Matrix 'mat\_a':")

print(mat\_a)

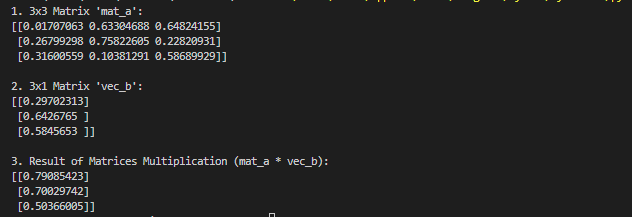
print("\n2. 3x1 Matrix 'vec\_b':")

print(vec\_b)

print("\n3. Result of Matrices Multiplication (mat\_a \* vec\_b):")

print(result)

**Output:**

****

**TASK-12**

import numpy as np

# 1. Create a 2D array 'matrix' with values from 1 to 9.

matrix = np.arange(1, 10).reshape(3, 3)

# 2. Subtract the mean of each row from each element in that row.

row\_means = np.mean(matrix, axis=1, keepdims=True)

result = matrix - row\_means

# Print the results

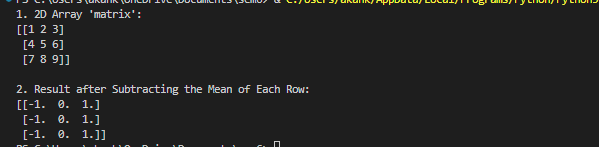
print("1. 2D Array 'matrix':")

print(matrix)

print("\n2. Result after Subtracting the Mean of Each Row:")

print(result)

**Output:**

****