Practical 4 : NumPy Arrays, Pandas DataFrames & Variance–Covariance Analysis

Name : *Deepak Kumar Bharti*

Roll No. : 25056765013

Group : *A*

Problem Statement

Solve the following problems using Python with NumPy and Pandas.

Given Dataset

The following bivariate dataset represents observations on two variables **X** and **Y**.

Observation	X	Y
1	4	7
2	6	9
3	8	10
4	5	8
5	7	11

Problem 1 : Creating and Writing a Dataset Using Pandas

(a) Create a Pandas DataFrame for the given bivariate data.

(b) Write the DataFrame to a CSV file named **bivariate_data.csv**.

(c) Display the stored data using `head` .

In [1]:
```
# Importing required libraries
import numpy as np
import pandas as pd

# Given bivariate data
X = [4, 6, 8, 5, 7]
Y = [7, 9, 10, 8, 11]

# (a) Creating DataFrame
data = pd.DataFrame(
    {
    'X': X,
```

```
    'Y': Y
    },
    index = range(1,len(X)+1)
)

# Display the DataFrame
data
```

Out[1]:

	X	Y
1	4	7
2	6	9
3	8	10
4	5	8
5	7	11

In [2]:
```
# (b) Writing DataFrame to CSV
data.to_csv('bivariate_data.csv', index=False)
```

In [3]:
```
# (c) Reading and displaying the stored data
data.head()
```

Out[3]:

	X	Y
1	4	7
2	6	9
3	8	10
4	5	8
5	7	11

Problem 2 : Reading Data from File and Creating Structures

(a) Read the CSV file **bivariate_data.csv** using Pandas.

(b) Convert the DataFrame into a two-dimensional NumPy array.

(c) Display the DataFrame and NumPy array.

In [4]:
```
# (a) Reading CSV file bivariate_data.csv using Pandas.
df = pd.read_csv('bivariate_data.csv')

# Display DataFrame
df
```

Out[4]:

	X	Y
0	4	7
1	6	9
2	8	10
3	5	8
4	7	11

In [5]:
```
# (b) Converting the DataFrame into a two-dimensional NumPy array.
data_array = df.to_numpy()

# Display NumPy array
data_array
```

Out[5]:
```
array([[ 4,  7],
       [ 6,  9],
       [ 8, 10],
       [ 5,  8],
       [ 7, 11]])
```

In [6]:
```
# (c) Displaying the DataFrame and NumPy array.
print("Pandas DataFrame:\n")
print(df)

print("\nNumPy Array:\n")
print(data_array)
```

```
Pandas DataFrame:

   X   Y
0  4   7
1  6   9
2  8  10
3  5   8
4  7  11

NumPy Array:

[[ 4  7]
 [ 6  9]
 [ 8 10]
 [ 5  8]
 [ 7 11]]
```

Problem 3 : Extracting Variables from the Dataset

Using the NumPy array obtained in **Problem 2**:

(a) Extract a one-dimensional NumPy array for variable **X**.

(b) Extract a one-dimensional NumPy array for variable **Y**.

(c) Display both arrays.

In [7]:
```
# (a) Extracting a one-dimensional NumPy array for variable X.
X_array = data_array[:, 0]
```

In [8]:
```
# (b) Extracting a one-dimensional NumPy array for variable Y.
Y_array = data_array[:, 1]
```

In [9]:
```
# (c) Displaying both arrays.
print("X Array:", X_array)
print("Y Array:", Y_array)
```

```
X Array: [4 6 8 5 7]
Y Array: [ 7  9 10  8 11]
```

Problem 4 : Matrix Representation and Matrix Operations

Using the two-dimensional NumPy array obtained earlier:

(a) Treat the dataset as a **data matrix** with observations as rows and variables as columns.

(b) Compute the **transpose** of the data matrix.

(c) Perform **matrix addition** of the data matrix with its transpose (where defined).

(d) Compute the **matrix multiplication**.

(e) Compute the **inverse**, if it exists.

(f) Compute:

- Row sums of the data matrix
- Column sums of the data matrix

(g) Compute the **eigenvalues and eigenvectors** of the matrix.

In [10]:
```
# Treating the dataset as a data matrix with observations as rows and variables as
columns.
A = data_array
A
```

Out[10]:
```
array([[ 4,  7],
       [ 6,  9],
       [ 8, 10],
       [ 5,  8],
       [ 7, 11]])
```

In [11]:
```
# (b) Computing the transpose of the data matrix.
A_transpose = A.T
A_transpose
```

Out[11]:
```
array([[ 4,  6,  8,  5,  7],
       [ 7,  9, 10,  8, 11]])
```

In [12]:
```
# (c) Performing matrix addition of the data matrix with its transpose (where
defined).
# Matrix addition only possible if shapes match
```

```
# A is (5x2) and A.T is (2x5), so addition is NOT defined
print("Matrix addition is not defined because shapes are different.")
```

```
Matrix addition is not defined because shapes are different.
```

In [13]:
```
# (d) Computing the matrix multiplication.
# Matrix multiplication (A^T * A)
matrix_product = np.dot(A_transpose, A)
print('The product of matrix "A_transpose" & "A" is :\n',matrix_product)
```

```
The product of matrix "A_transpose" & "A" is :
 [[190 279]
 [279 415]]
```

In [14]:
```
# (e) Computing the inverse, if it exists.
# Check if inverse exists
try:
    inverse_matrix = np.linalg.inv(matrix_product)
    print('The inverse of matrix "matrix_product" :\n', inverse_matrix)
except np.linalg.LinAlgError:
    print("Inverse does not exist.")
```

```
The inverse of matrix "matrix_product" :
 [[ 0.41129832 -0.2765114 ]
 [-0.2765114   0.18830525]]
```

In [15]:
```
# (f) Computing :

# Row sums of the data matrix
row_sums = np.sum(A, axis=1)
print("Row Sums:", row_sums)

# Column sums of the data matrix
column_sums = np.sum(A, axis=0)
print("Column Sums:", column_sums)
```

```
Row Sums: [11 15 18 13 18]
Column Sums: [30 45]
```

In [16]:
```
# (g) Computing the eigenvalues and eigenvectors of the matrix.
# Eigenvalues and Eigenvectors of (A^T * A)
eigenvalues, eigenvectors = np.linalg.eig(matrix_product)

print("Eigenvalues of (A_transpose * A) :\n", eigenvalues)
print("\nEigenvectors of (A_transpose * A) :\n", eigenvectors)
```

```
Eigenvalues of (A_transpose * A) :
 [  1.67239156 603.32760844]

Eigenvectors of (A_transpose * A) :
 [[-0.82884508 -0.55947818]
 [ 0.55947818 -0.82884508]]
```

Problem 5 : Computing Sample Mean Vector and Variance–Covariance Matrix

(a) Compute the **sample mean** of **X** and **Y** using NumPy.

(b) Using NumPy, compute the **variance–covariance matrix** of **X** and **Y**.

In [17]:
```
# (a) Computing the sample mean of X and Y using NumPy.

mean_X = np.mean(X_array)
mean_Y = np.mean(Y_array)

print("Mean of X:", mean_X)
print("Mean of Y:", mean_Y)
```

```
Mean of X: 6.0
Mean of Y: 9.0
```

In [18]:
```
# (b) Using NumPy, computing the variance-covariance matrix of X and Y.
# Variance of X
var_X = np.var(X_array, ddof=1)   # ddof=1 for sample variance

# Variance of Y
var_Y = np.var(Y_array, ddof=1)

print("Variance of X:", var_X)
print("Variance of Y:", var_Y)

# Covariance matrix
cov_matrix = np.cov(X_array, Y_array)

print("\nCovariance Matrix:\n", cov_matrix)
```

```
Variance of X: 2.5
Variance of Y: 2.5

Covariance Matrix:
 [[2.5  2.25]
 [2.25 2.5 ]]
```

Problem 6 : Statistical Interpretation

(a) Interpret the **covariance value**.

(b) Comment on the **relationship between X and Y**.

(c) Briefly explain the role of **eigenvalues and eigenvectors** in multivariate analysis.

Interpretations :

(a) Interpreting the Covariance Value

From the output, the variance–covariance matrix is:

$$\begin{pmatrix} 2.5 & 2.25 \\ 2.25 & 2.5 \end{pmatrix}$$

- The **covariance between X and Y is 2.25**, which is **positive**.
- A positive covariance indicates that **X and Y tend to increase together**.
- The relatively large magnitude of the covariance (close to the variances 2.5) suggests a **strong linear association** between X and Y.

Conclusion :

There is a **strong positive covariance** between X and Y.

(b) Commenting on the Relationship Between X and Y

- Since the covariance is **positive**, X and Y have a **direct (positive) linear relationship**.
- As the values of **X increase**, the values of **Y also tend to increase**.
- This indicates that X and Y are **positively correlated**, although covariance alone does not provide a standardized measure of strength.

Conclusion :

X and Y show a **strong positive linear relationship**.

(c) Explaining the Role of Eigenvalues and Eigenvectors in Multivariate Analysis

From the computation of eigenvalues and eigenvectors of (A^T A):

- **Eigenvalues** represent the **amount of variance** captured along the directions defined by the eigenvectors.
    - A **larger eigenvalue** (≈ 603.33) indicates the **principal direction of maximum variance**.
    - A **smaller eigenvalue** (≈ 1.67) corresponds to a direction with **much less variation**.
- **Eigenvectors** represent the **directions** along which the data varies the most.
    - They define new, **uncorrelated axes** known as **principal components**.
    - Eigenvalues and eigenvectors are fundamental in **Principal Component Analysis (PCA)**.

Conclusion :

Eigenvalues quantify **how much variance** exists in each principal direction, while eigenvectors define the **directions of maximum variance**, aiding in **dimensionality reduction, data interpretation, and feature extraction**.