Descriptive Statistics

```
import numpy as np
data = np.array([5, 15, 10, 25, 20, 30, 25, 35])

# Mean
mean = np.mean(data)
print("Mean:", mean)
#
# median
median = np.median(data)
print("Median:", median)

# mode
from scipy import stats
mode = stats.mode(data)
print("Mode:", mode.mode[0])

# SD
std_dev = np.std(data)
print("Standard Deviation:", std_dev)

#
# var
variance = np.var(data)
print("Variance:", variance)

# min, max
print("Minimum:", np.min(data))
print("Maximum:", np.max(data))

# range
```

```
range_val = np.max(data) - np.min(data)
print("Range:", range_val)

# pwecentile
q25 = np.percentile(data, 25)
q50 = np.percentile(data, 50)  # Same as median
q75 = np.percentile(data, 75)
print("25th percentile:", q25)
print("50th percentile (Median):", q50)
print("75th percentile:", q75)
#

# skewness, kurtosis
import numpy as np
from scipy.stats import skew, kurtosis

data2 = np.array([4, 5, 6, 7, 8, 8, 9, 10, 12, 15])

skewness = skew(data2)
print("Skewness:", skewness)

## or
mean = np.mean(data2)
std_dev = np.std(data2, ddof=1)
n = len(data2)
skew_manual = (np.sum(((data2 - mean) / std_dev)**3)) * (n / ((n - 1)*(n - 2)))
print("Manual Skewness:", skew_manual)

### kurtosis
kurt = kurtosis(data2)
print("Kurtosis:", kurt)
```

```
#######

# correlation
# Independent variable (X) and dependent variable (Y)
X = np.array([1, 2, 3, 4, 5])
Y = np.array([2, 4, 5, 4, 5])

correlation_matrix = np.corrcoef(X, Y)
correlation = correlation_matrix[0, 1]  # [[1.0, r],  correlation_matrix[0, 1], extracts the off-diagonal value
#   [r, 1.0]]

print("Pearson Correlation Coefficient:", correlation)



## regression

# Mean values
mean_x = np.mean(X)
mean_y = np.mean(Y)

# Slope (a)
slope = np.sum((X - mean_x) * (Y - mean_y)) / np.sum((X - mean_x)**2)

# Intercept (b)
intercept = mean_y - slope * mean_x

print("Slope:", slope)
print("Intercept:", intercept)

# predicted value
Y_pred = slope * X + intercept
```

```
print("Predicted Y values:", Y_pred)


# R-squared (Coefficient of Determination)
ss_total = np.sum((Y - mean_y) ** 2)
ss_residual = np.sum((Y - Y_pred) ** 2)
r_squared = 1 - (ss_residual / ss_total)
print("R-squared:", r_squared)



############################################################################
# Create Matrices
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])


print(A)
print(B)


## Identity Matrix


I = np.eye(4)
print("Identity Matrix:\n", I)


##### Addition & Subtraction
add = A + B
sub = A - B
print("Addition:\n", add)
print("Subtraction:\n", sub)
```

```
#Element-wise Multiplication

elementwise = A * B
print("Element-wise multiplication:\n", elementwise)


#Matrix Multiplication (Dot Product)
dot_product = A @ B
print("Matrix multiplication:\n", dot_product)

# OR
dot_product2 = np.dot(A, B)
print("Matrix multiplication:\n", dot_product2)


# Transpose of a Matrix
transpose = A.T
print("Transpose of A:\n", transpose)


##Determinant
det_A = np.linalg.det(A)
print("Determinant of A:", det_A)


### Inverse of a Matrix
inv_A = np.linalg.inv(A)
print("Inverse of A:\n", inv_A)


# Rank of a Matrix
```

```
rank_A = np.linalg.matrix_rank(A)
print("Rank of A:", rank_A)



## Eigenvalues and Eigenvectors

eigenvalues, eigenvectors = np.linalg.eig(A)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)




# Solve a System of Linear Equations
#Solve: AX = b
A = np.array([[1, 5, 3], [3, 4, 7], [5, 8, 2]])
b = np.array([2, 5, 11])
X = np.linalg.solve(A, b)
print("Solution X:", X)



#####################################################################################
#Probability Distributions

from scipy.stats import binom, poisson, norm



# Toss a coin 10 times, find P(X = 4) and cumulative probability where X = number of heads
n = 10      # number of trials
p = 0.5     # probability of success
```

```
# P(X = 4)
prob = binom.pmf(k=4, n=n, p=p)
print("P(X=4):", prob)

# Cumulative P(X ≤ 4)
cum_prob = binom.cdf(k=4, n=n, p=p)
print("P(X ≤ 4):", cum_prob)

###  Poisson Distribution
# Average 3 calls per hour. Find P(X = 2)

lam = 3  # average rate λ

# P(X = 2)
prob2 = poisson.pmf(k=2, mu=lam)
print("P(X=2):", prob2)

# Cumulative P(X ≤ 2)
cum_prob2 = poisson.cdf(k=2, mu=lam)
print("P(X ≤ 2):", cum_prob2)


############ Normal Distribution
# Heights are normally distributed with μ = 170, σ = 10. Find P(X < 180)

mu = 170
sigma = 10

# P(X < 180)
prob3 = norm.cdf(x=180, loc=mu, scale=sigma)
print("P(X < 180):", prob3)
```

```
# P(X > 180)
prob_right = 1 - prob3
print("P(X > 180):", prob_right)

# P(160 < X < 180)
prob_between = norm.cdf(180, mu, sigma) - norm.cdf(160, mu, sigma)
print("P(160 < X < 180):", prob_between)


# Load Dataset & Compute Summary Statistics
pip install pandas


import pandas as pd

## Load a CSV Dataset

data_load = pd.read_csv("D:\\Universities\\JU\\Teaching_JU\\2025\\LAB_Programming and Data Analysis using Python\\Lecture\\score data.csv")
print(data_load.head())  # Show the first 5 rows

## or
data_load2 = pd.read_csv(r"D:\Universities\JU\Teaching_JU\2025\LAB_Programming and Data Analysis using Python\Lecture\score data.csv")

print(data_load2.head())  # Show the first 5 rows

print(data_load.columns)  # to see all column names

# Frequency for Gender

gender_freq = data_load['Gender'].value_counts()
print("Frequency of Gender:\n", gender_freq)
```

```
### Relative frequency
gender_prop = data_load['Gender'].value_counts(normalize=True)
print("Proportion of Gender:\n", gender_prop)

# As pandas Series
age_series = data_load['Age']

# As NumPy array
age_array = data_load['Age'].values

np.mean(age_series)
np.mean(age_array)


### Summary stat
age_summary = data_load['Age'].describe()
print("Summary Statistics for Age:\n", age_summary)


variance = data_load['Age'].var()
print("Variance:", variance)


####################
d2 = data_load[['Age', 'Math_Score', 'Reading_Score', 'Science_Score']]

correlation_matrix = d2.corr()
print("Correlation Matrix:\n", correlation_matrix)


##########################
```

```
from sklearn.linear_model import LinearRegression

# Define X and Y
X = data_load[['Age', 'Science_Score']]
Y = data_load['Math_Score']



# Fit model
reg = LinearRegression().fit(X, Y)

# Print coefficients
print("Intercept:", reg.intercept_)
print("Coefficients:", reg.coef_)
```