EX : 1

```
def is_prime(n):
if n <= 1:
return False
for i in range(2, n):
if n % i == 0:
return False
return True
number = int(input("Enter a number: "))
if is_prime(number):
print(f"{number} is a prime number.")
else:
print(f"{number} is not a prime number.")
```

EX : 2

```
# Step 1: Input scores
scores = list(map(int, input("Enter scores separated by space: ").split()))
n = len(scores)
# Step 2: Mean
total = 0
for score in scores:
total += score
mean = total / n
# Step 3: Sorting (bubble sort for median)
for i in range(n):
for j in range(0, n - i - 1):
if scores[j] > scores[j + 1]:
scores[j], scores[j + 1] = scores[j + 1], scores[j]
# Step 4: Median
if n % 2 == 0:
median = (scores[n // 2 - 1] + scores[n // 2]) / 2
```

```
else:
median = scores[n // 2]
# Step 5: Maximum and Minimum
maximum = scores[0]
minimum = scores[0]
for score in scores:
if score > maximum:
maximum = score
if score < minimum:
minimum = score
# Step 6: Standard Deviation
sum_sq_diff = 0
for score in scores:
sum_sq_diff += (score - mean) ** 2
variance = sum_sq_diff / n
std_dev = variance ** 0.5
# Step 7: Grade Classification
def assign_grade(score):
if score >= 90:
return 'A'
elif score >= 80:
return 'B'
elif score >= 70:
return 'C'
elif score >= 60:
return 'D'
else:
return 'F'
grades = [assign_grade(score) for score in scores]
# Step 8: Display Results
print("\n--- Statistical Results ---")
print(f"Mean: {mean:.2f}")
print(f"Median: {median}")
print(f"Maximum: {maximum}")
print(f"Minimum: {minimum}")
```

```
print(f"Standard Deviation: {std_dev:.2f}")
print("\n--- Grade Classification ---")
for i in range(n):
print(f"Student {i+1}: Score = {scores[i]}, Grade = {grades[i]}")
```

EX : 3

Image invertion:

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
# Step 1: Load the image and convert it to grayscale
img = Image.open('example.jpg').convert('L') # 'L' for grayscale
img_array = np.array(img)
# Step 2: Perform image inversion
inverted_img = 255 - img_array
# Step 3: Display original and inverted images
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img_array, cmap='gray')
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(inverted_img, cmap='gray')
plt.title('Inverted Image')
plt.axis('off')
plt.show()
```

threshold:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
# Step 1: Create a simulated grayscale image (gradient pattern)
img_array = np.tile(np.arange(0, 256, dtype=np.uint8), (100, 1)) # shape: (100, 256)
# Step 2: Apply thresholding using NumPy
threshold = 150
binary_img = np.where(img_array > threshold, 255, 0).astype(np.uint8)
# Step 3: Display original and thresholded images
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(img_array, cmap='gray')
plt.title("Original Image")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(binary_img, cmap='gray')
plt.title(f"Thresholded Image (T = {threshold})")
plt.axis('off')
plt.show()
```

EX 4:

Linear system:

2 x 2

```
import numpy as np
A = np.array([[2, 3], [1, -1]])
B = np.array([8, 2])
X = np.linalg.solve(A, B)
print("Solution:", X)
```

3 x 3

```
import numpy as np
A = np.array([[2, 3, 4], [1, -1, 5], [2, 3, 5]])
B = np.array([8, 2, 10])
X = np.linalg.solve(A, B)
print("Solution:", X)
```

4 x 4

```
import numpy as np
A = np.array([[2, 3, 4,5], [1, -1, 5,1], [2, 3, 5,7], [1,2,3,4]])
B = np.array([8, 2, 10,12])
X = np.linalg.solve(A, B)
print("Solution:", X)
```

speed test:

```
import numpy as np
# Create the system of equations using the given data points
# Each row: [t^2, t, 1] for t = 3, 6, 9
A = np.array ([[3**2, 3, 1],[6**2, 6, 1],[9**2, 9, 1]])
# Corresponding speeds at t = 3, 6, 9
B = np.array([64, 133, 208])
# Solve for a, b, c
coefficients = np.linalg.solve(A, B)
a, b, c = coefficients
# Calculate speed at t = 15
t = 15
v_15 = a*t**2 + b*t + c
# Output
print(f"Coefficients: a = {a:.2f}, b = {b:.2f}, c = {c:.2f}")
print(f"Speed at t = 15 seconds: {v_15:.2f} miles/sec")
```

EX : 5

T-Test:

```
from scipy import stats
n = int(input("Enter number of elements in each group: "))
group1 = [float(input(f"Group1 value {i+1}: ")) for i in range(n)]
group2 = [float(input(f"Group2 value {i+1}: ")) for i in range(n)]
t_stat, p_value = stats.ttest_ind(group1, group2)
print("T-statistic:", t_stat)
print("P-value:", p_value)
if p_value < 0.05:
print("Reject H0: Significant difference.")
else:
print("Fail to reject H0: No significant difference.")
```

Chi-Square Test Using Python:

```
import numpy as np
from scipy.stats import chi2_contingency
rows = int(input("Enter number of rows: "))
cols = int(input("Enter number of columns: "))
observed = []
for i in range(rows):
row = [int(input(f"Value at ({i+1},{j+1}): ")) for j in range(cols)]
observed.append(row)
observed_array = np.array(observed)
chi2, p, dof, expected = chi2_contingency(observed_array)
print("Chi-Square Value:", chi2)
print("P-value:", p)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:\n", expected)
if p < 0.05:
print("Reject H0: Variables are dependent.")
else:
```

```
print("Fail to reject H0: Variables are independent.")
```

EX : 6

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
data = {
    'Hours_Studied': [5, 3, 8, 2, 7, 4],
    'Attendance': [1, 1, 1, 0, 1, 0],
    'Passed': [1, 0, 1, 0, 1, 0]
}
X = pd.DataFrame({'Hours_Studied': data['Hours_Studied'], 'Attendance':
data['Attendance']})
y = pd.Series(data['Passed'])
model = LogisticRegression().fit(X, y)
hrs = float(input("Hours studied: "))
att = int(input("Attendance (1=Present, 0=Absent): "))
pred = model.predict([[hrs, att]])[0]
print("PASS" if pred == 1 else "FAIL")
```

EX : 7

```
import pandas as pd

df = pd.read_csv("sales.csv")
df.fillna(0, inplace=True)
report = df.pivot_table(index='Region', values='Sales', aggfunc='sum')
```

```
print(report)
```

EX : 8

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("covid.csv", parse_dates=['Date'], index_col='Date')
df['Cases'].plot(label='Daily Cases')
df['Cases'].rolling(7).mean().plot(label='7-Day Average')
plt.legend()
plt.title("COVID-19 Trend")
plt.show()
```

EX : 9

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=200,centers=3,random_state=42)
kmeans = KMeans(n_clusters=3).fit(X)
labels = kmeans.predict(X)
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], c='red',
s=150, marker='X')
plt.show()
```

EX : 10

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import numpy as np
data = {
   'Category': ['A', 'B', 'C', 'D', 'E'],
   'Value1': [25, 40, 30, 35, 20],
   'Value2': [15, 20, 25, 10, 30]
}
df = pd.DataFrame(data)
plt.figure(figsize=(8, 6))
sns.barplot(x='Category', y='Value1', data=df)
plt.title('Bar Chart')
plt.show()
plt.figure(figsize=(8, 8))
plt.pie(df['Value2'], labels=df['Category'], autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart')
plt.show()
plt.figure(figsize=(8, 6))
plt.scatter(df['Value1'], df['Value2'], c='blue', marker='o')
plt.title('Scatter Plot')
plt.xlabel('Value1')
plt.ylabel('Value2')
plt.show()
plt.figure(figsize=(8, 6))
plt.hist(df['Value1'], bins=np.arange(15, 45, 5), edgecolor='black')
plt.title('Histogram')
plt.xlabel('Value1')
plt.ylabel('Frequency')
plt.show()
```