Data Science Assignment (3)

Full Name: Ibrahim Qahtan Adnan
Theoretical Group: B

---

1. numpy.sum() — Total Measurement

Scenario:
You are analyzing the total monthly energy consumption (in kWh) of a smart building.
Dataset: [350, 420, 390, 410, 460, 480, 500, 490, 470, 450, 430, 400]
Task:
Compute the total annual energy consumption using numpy.sum().
Interpret what this total tells you about the building's energy usage.
Solution:

```
import numpy as np

# Dataset: Monthly energy consumption (in kWh) [cite: 5]
consumption = np.array([350, 420, 390, 410, 460, 480, 500, 490, 470, 450, 430,
400])

# 1. Compute the total annual consumption
total_consumption = np.sum(consumption)

print(f"Monthly Consumption (kWh): {consumption}")
print(f"Total Annual Consumption: {total_consumption} kWh")
```

2. numpy.mean(), numpy.median(), numpy.var(), numpy.std()

Scenario:
A company tracks the salaries (in $1,000s) of its 12 employees:
[40, 42, 43, 45, 46, 47, 50, 52, 54, 55, 60, 65]
Task:
Find mean, median, variance, and standard deviation.
Discuss what these measures tell you about income distribution and consistency.
Solution:

```
import numpy as np

# Dataset: Salaries (in $1,000s) [cite: 10]

salaries = np.array([40, 42, 43, 45, 46, 47, 50, 52, 54, 55, 60, 65])

# 1. Calculate statistics [cite: 12]
mean_salary = np.mean(salaries)
median_salary = np.median(salaries)
variance_salary = np.var(salaries)
std_dev_salary = np.std(salaries)

print(f"Salaries ($1,000s): {salaries}")
print(f"Mean: {mean_salary:.2f}")
print(f"Median: {median_salary:.2f}")
print(f"Variance: {variance_salary:.2f}")
print(f"Standard Deviation: {std_dev_salary:.2f}")
```

3. numpy.average() — Weighted Average

Scenario:
A student’s final grade depends on: Assignments (20%), Midterm (30%), and Final Exam (50%).
Scores: [80, 75, 90], Weights: [0.2, 0.3, 0.5]
Task:
Use numpy.average() to calculate the weighted grade.
Explain why weighting gives a more accurate result than a simple mean.
Solution:

```python
import numpy as np

# Scores: [Assignments, Midterm, Final Exam] [cite: 17]
scores = np.array([80, 75, 90])
# Weights: [20%, 30%, 50%] [cite: 17]
weights = np.array([0.2, 0.3, 0.5])

# 1. Calculate the weighted average
weighted_grade = np.average(scores, weights=weights)

print(f"Scores: {scores}")
print(f"Weights: {weights}")
print(f"Final Weighted Grade: {weighted_grade:.2f}")
```

4. numpy.ptp(), numpy.min(), numpy.max() — Data Range

Scenario:
You are monitoring temperature (°C) for 7 days: [18, 20, 22, 21, 19, 23, 25]
Task:
Find the minimum, maximum, and range (peak-to-peak).
Interpret what the range tells you about weather stability during the week.
Solution:

```python
import numpy as np

# Dataset: Temperatures (°C) for 7 days [cite: 23]
temperatures = np.array([18, 20, 22, 21, 19, 23, 25])

# 1. Find min, max, and range
min_temp = np.min(temperatures)
max_temp = np.max(temperatures)
temp_range = np.ptp(temperatures) # ptp = "peak-to-peak"

print(f"Temperatures (°C): {temperatures}")
print(f"Minimum Temperature: {min_temp}°C")
print(f"Maximum Temperature: {max_temp}°C")
```

5. **numpy.percentile(), numpy.quantile() — Distribution Spread**
Task:
Compute 25th, 50th, and 75th percentiles, and 0.9 quantile.
Interpret what these values say about student performance distribution.

Solution:
```python
import numpy as np

# Dataset: Students' exam scores [cite: 30]
scores = np.array([45, 50, 55, 60, 65, 70, 75, 80, 85, 90])

# 1. Compute percentiles
p_25 = np.percentile(scores, 25)
p_50 = np.percentile(scores, 50) # This is the median
p_75 = np.percentile(scores, 75)

# 2. Compute quantile (0.9 is the same as 90th percentile)
q_90 = np.quantile(scores, 0.9)

print(f"Scores: {scores}")
print(f"25th Percentile (Q1): {p_25}")
print(f"50th Percentile (Median/Q2): {p_50}")
print(f"75th Percentile (Q3): {p_75}")
print(f"0.9 Quantile (90th Percentile): {q_90}")
```

6. **numpy.random.normal() — Simulating Real Data**

Scenario:
You want to simulate the weights (in kg) of 200 individuals with mean = 70, std = 10.

Task:
Generate the dataset using numpy.random.normal(70, 10, 200).
Compute mean and std of your generated data.
Explain why we use the normal distribution in data simulation.

Solution:
```python
import numpy as np

# 1. Generate the dataset [cite: 38]
# Mean = 70, Std Dev = 10, N = 200
generated_weights = np.random.normal(70, 10, 200)

# 2. Compute mean and std of the generated data [cite: 39]
gen_mean = np.mean(generated_weights)
gen_std = np.std(generated_weights)

print(f"Generated Mean: {gen_mean:.2f} kg (Target was 70 kg)")
print(f"Generated Std Dev: {gen_std:.2f} kg (Target was 10 kg)")
```

7. numpy.random.seed() — Reproducibility

Scenario:
Two analysts are testing random sampling results for product quality.
They need identical samples for fair comparison.
Task:
Set a seed (np.random.seed(42)) before generating random numbers.
Show that both analysts get identical results.
Explain why reproducibility is critical in data science research.
Solution:

```python
import numpy as np

# 1. Analyst 1's work
np.random.seed(42) # Set the seed [cite: 45]
analyst1_samples = np.random.rand(5) # Generate 5 random numbers

# 2. Analyst 2's work
np.random.seed(42) # Set the *same* seed [cite: 45]
analyst2_samples = np.random.rand(5) # Generate 5 random numbers

print(f"Analyst 1's samples: {analyst1_samples}")
print(f"Analyst 2's samples: {analyst2_samples}")

# 3. Check for identical results [cite: 46]
are_identical = np.array_equal(analyst1_samples, analyst2_samples)
print(f"\nAre the results identical? {are_identical}")
```

8. numpy.corrcoef(), numpy.cov() — Correlation & Covariance

Scenario:
You're studying the relationship between **advertising spending** and **sales growth** for 6 months.
Spending: [1000, 1500, 2000, 2500, 3000, 3500]
Sales: [20, 25, 28, 35, 40, 45]
Task:
Compute covariance and correlation using NumPy.
Explain how correlation helps predict sales performance.
Solution:

```python
import numpy as np
# Dataset [cite: 51]
spending = np.array([1000, 1500, 2000, 2500, 3000, 3500])
sales = np.array([20, 25, 28, 35, 40, 45])
# 1. Compute covariance matrix
cov_matrix = np.cov(spending, sales)
covariance = cov_matrix[0, 1] # Get the specific covariance
# 2. Compute correlation matrix
# The diagonal is 1 (correlation with itself)
# The off-diagonal is the correlation coefficient
corr_matrix = np.corrcoef(spending, sales)
correlation = corr_matrix[0, 1] # Get the specific correlation
print(f"--- Covariance Matrix --- \n{cov_matrix}\n")
print(f"--- Correlation Matrix --- \n{corr_matrix}\n")
print(f"Covariance: {covariance:.2f}")
print(f"Correlation: {correlation:.2f}")
```

9. numpy.histogram() — Data Distribution
Scenario:
You collected 100 random test scores between 0 and 100.
Task:
Use numpy.histogram() and matplotlib.pyplot.hist() to plot score distribution.
Describe whether the distribution is normal, skewed, or uniform.
Solution:

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

# 1. Generate 100 random test scores between 0 and 100 [cite: 58]
# We use randint, which creates a *uniform* distribution
scores = np.random.randint(0, 101, 100)

# 2. Use numpy.histogram()
counts, bin_edges = np.histogram(scores, bins=10)

print(f"Counts per bin: {counts}")
print(f"Bin edges: {bin_edges}")

# 3. Use matplotlib.pyplot.hist() to plot
plt.hist(scores, bins=10, edgecolor='black')
plt.title('Distribution of 100 Random Test Scores')
plt.xlabel('Score')
plt.ylabel('Number of Students')
plt.show()
```

10. numpy.bincount() — Counting Frequencies

Scenario:
A customer satisfaction survey collected ratings (1–5 stars).
Ratings: [5, 4, 3, 5, 4, 5, 2, 3, 4, 5, 1, 5, 4]
Task:
Use numpy.bincount() to count how many times each rating occurs.
Which rating was most frequent, and what does it indicate?
Solution:

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

# Dataset: Customer satisfaction ratings (1-5 stars) [cite: 63]
ratings = np.array([5, 4, 3, 5, 4, 5, 2, 3, 4, 5, 1, 5, 4])
# 1. Use bincount to count frequencies
# bincount starts from 0, so index 0 is for "0 stars"
# We'll get counts for 0, 1, 2, 3, 4, 5
rating_counts = np.bincount(ratings)

# 2. Get the most frequent rating
# We skip index 0 as it's not a valid rating
most_frequent_rating = np.argmax(rating_counts[1:]) + 1
print(f"--- Rating Counts ---")
for i in range(1, len(rating_counts)):
    print(f"{i}-Star: {rating_counts[i]} times")

print(f"\nMost Frequent Rating: {most_frequent_rating} stars")
```

11. numpy.random.randint(), numpy.random.choice(), numpy.random.uniform()

Scenario:
You are simulating 50 random product prices between $10–$100.
Then, randomly pick 5 products to apply a discount.
Task:

1. Generate prices using numpy.random.randint(10, 100, 50).
2. Use numpy.random.choice() to select 5 random items.
3. Apply a 10% discount to them and print the new list.

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

# Set a seed for reproducible results
np.random.seed(99)

# 1. Generate 50 random prices between $10-$100 [cite: 70]
# Note: randint(10, 100) generates numbers from 10 up to 99
prices = np.random.randint(10, 100, 50)
print(f"Original Prices (first 10): {prices[:10]}...")

# 2. Use choice() to select 5 *indices* of products to discount [cite: 71]
# We set replace=False so we don't pick the same item twice
discount_indices = np.random.choice(range(len(prices)), 5, replace=False)
print(f"Indices chosen for discount: {discount_indices}")

# 3. Apply a 10% discount to them [cite: 72]
# Create a copy so we don't change the original
discounted_prices = np.copy(prices).astype(float) # Use float for decimals

for index in discount_indices:
    discounted_prices[index] = discounted_prices[index] * 0.90

# 4. Print the new list
print("\n--- Price Changes ---")
for index in discount_indices:
    print(f"Item at index {index}: Original Price ${prices[index]} -> New Price
${discounted_prices[index]:.2f}")
```