Pawan Chaudhary

1. Understanding Numpy

NumPy slicing creates a view of the same memory rather than a copy. A change in the slice also changes the original array. Python lists handle slicing differently because they usually make a copy which is why small edits in NumPy slices can directly affect the data and cause problems during analysis.

2. Establishing Analytical Reproductibility

In [2]:
```
import numpy as np
```

In [3]:
```
np.random.seed(2023)
```

Setting a random seed initializes the random number generator to a fixed starting point, ensuring the same sequence of random values is produced each time. This is important for reproducible data analysis because it allows experiments and results to be consistently replicated.

3. Array Operations

In [20]:
```
# nested sequence representing initial budget allocation

budget_data = [
    [1200, 1300, 1250, 1400, 1350, 1500, 1600, 1700, 1650, 1750, 1800, 1900],
    [1000, 1100, 1150, 1200, 1250, 1300, 1400, 1450, 1500, 1550, 1600, 1650],
    [800, 900, 850, 950, 1000, 1050, 1100, 1150, 1200, 1250, 1300, 1350],
    [1500, 1600, 1550, 1650, 1700, 1750, 1800, 1850, 1900, 1950, 2000, 2100],
    [900, 950, 1000, 1050, 1100, 1150, 1200, 1250, 1300, 1350, 1400, 1450]
]

print(f"{budget_data}")
```

```
[[1200, 1300, 1250, 1400, 1350, 1500, 1600, 1700, 1650, 1750, 1800, 1900], [1000, 11
00, 1150, 1200, 1250, 1300, 1400, 1450, 1500, 1550, 1600, 1650], [800, 900, 850, 95
0, 1000, 1050, 1100, 1150, 1200, 1250, 1300, 1350], [1500, 1600, 1550, 1650, 1700, 1
750, 1800, 1850, 1900, 1950, 2000, 2100], [900, 950, 1000, 1050, 1100, 1150, 1200, 1
250, 1300, 1350, 1400, 1450]]
```

In [21]:
```
# converting into NumPy array

budget_array = np.array(budget_data)
```

In [22]:
```
# budget data in thousands
```

```
budget_array = budget_array * 1000
print(f"{budget_array}")
```

```
[[1200000 1300000 1250000 1400000 1350000 1500000 1600000 1700000 1650000
  1750000 1800000 1900000]
 [1000000 1100000 1150000 1200000 1250000 1300000 1400000 1450000 1500000
  1550000 1600000 1650000]
 [ 800000  900000  850000  950000 1000000 1050000 1100000 1150000 1200000
  1250000 1300000 1350000]
 [1500000 1600000 1550000 1650000 1700000 1750000 1800000 1850000 1900000
  1950000 2000000 2100000]
 [ 900000  950000 1000000 1050000 1100000 1150000 1200000 1250000 1300000
  1350000 1400000 1450000]]
```

In [23]:
```
# shape
print(f"Shape: {budget_array.shape}")
```

```
Shape: (5, 12)
```

In [24]:
```
# data type
print(f"Data Type: {budget_array.dtype}")
```

```
Data Type: int64
```

In [25]:
```
# creating array of years 2022-2026
ArrYears = np.arange(2022, 2027)
print(f"Array Years: {ArrYears}")
```

```
Array Years: [2022 2023 2024 2025 2026]
```

In [26]:
```
# subsetting to first 3 years
threeYears = ArrYears[:3]
print("Three Years:", threeYears)
```

```
Three Years: [2022 2023 2024]
```

In [28]:
```
# further subset to get 2023 and 2024
selected_years = threeYears[1:3]
print(f"Selected Years: {selected_years}")
```

```
Selected Years: [2023 2024]
```

4. Impact Analysis

In [30]:
```
# generating 5x4 random array for unforeseen additional costs with values between 0
add_costs = np.random.rand(5,4) * 0.2
print(f"Unforseen Additional Costs: \n{add_costs}")
```

```
Unforseen Additional Costs:
[[0.11297233 0.0406923  0.06412089 0.07531276]
 [0.03681083 0.02079037 0.09098544 0.03917277]
 [0.07570508 0.18610639 0.15203194 0.15415285]
 [0.11934011 0.15832423 0.16206766 0.19611145]
 [0.17695705 0.02196023 0.16394215 0.06152258]]
```

In [32]:
```
# replacing values less than 0.05
# 0 if < 0.5 and 1 if >= 0.05
```

```
neg_impact = np.where(add_costs < 0.05, 0, 1)
print(f"Impact: \n{neg_impact}")
```

```
Impact:
[[1 0 1 1]
 [0 0 1 0]
 [1 1 1 1]
 [1 1 1 1]
 [1 0 1 1]]
```

In [33]:
```
# summing total additional costs for each department
dept_add = neg_impact.sum(axis=1)
print(f"Total additional costs for each department: \n{dept_add}")
```

```
Total additional costs for each department:
[3 1 4 4 3]
```

5. Data Validation

In [34]:
```
departments = ['Infrastructure', 'Development', 'QA', 'Cybersecurity', 'Consulting'
```

In [35]:
```
# checking if important departments are in the list

if "Cybersecurity" in departments and "Consulting" in departments:
    print("Both departments are included.")
else:
    print("Department missing.")
```

```
Both departments are included.
```

6. Financial Performance Evaluation

In [ ]:
```
# calculating descriptive statistics


# mean
print(f"Each quarter's mean: \n{budget_array.mean(axis=0)}")
```

```
This quarter's mean:
[1080000. 1170000. 1160000. 1250000. 1280000. 1350000. 1420000. 1480000.
 1510000. 1570000. 1620000. 1690000.]
```

In [ ]:
```
# standard deviation
print(f"Each quarter's standard deviation: \n{budget_array.std(axis=0)}")
```

```
This quarter's standard deviation:
[248193.47291982 256124.96949731 237486.84174076 250998.00796022
 242074.3687382  250998.00796022 256124.96949731 263818.11916546
 249799.91993594 256124.96949731 256124.96949731 278208.55486487]
```

In [ ]:
```
# variance
print(f"Each quarter's variance: \n{budget_array.var(axis=0)}")
```

```
This quarter's variance:
[6.16e+10 6.56e+10 5.64e+10 6.30e+10 5.86e+10 6.30e+10 6.56e+10 6.96e+10
 6.24e+10 6.56e+10 6.56e+10 7.74e+10]
```

In [40]:
```
# minimum
print(f"Each quarter's minimum: \n{budget_array.min(axis=0)}")
```

```
Each quarter's minimum: 
[ 800000  900000  850000  950000 1000000 1050000 1100000 1150000 1200000
 1250000 1300000 1350000]
```

In [41]:
```
# maximum
print(f"This quarter's mean: \n{budget_array.max(axis=0)}")
```

```
This quarter's mean: 
[1500000 1600000 1550000 1650000 1700000 1750000 1800000 1850000 1900000
 1950000 2000000 2100000]
```

In [42]:
```
# total spending for each department

dept_total = budget_array.sum(axis=1)
print(f"Department Totals: {dept_total}")
```

```
Department Totals: [18400000 16150000 12900000 21350000 14100000]
```

In [ ]:
```
# finding the highest total spending for each department
highest_spending = departments[np.argmax(dept_total)]
print(f"The highest spending department: {highest_spending}")
```

```
The highest spending department: Cybersecurity
```

In [44]:
```
# finding the lowest total spending for each department
lowest_spending = departments[np.argmin(dept_total)]
print(f"The lowest spending department: {lowest_spending}")
```

```
The lowest spending department: QA
```

7. Advanced Array Operations

In [48]:
```
# array representing the years and quarters of the project from 2021Q1 to 2023Q4

years_and_quarters = []
for y in range(2021, 2024):
    for q in range(1, 5):
        years_and_quarters.append(f"{y}Q{q}")

years_and_quarters = np.array(years_and_quarters)
print(f"The years and quarters of the project from 2021Q1 to 2023Q4: {years_and_qua
```

```
The years and quarters of the project from 2021Q1 to 2023Q4: ['2021Q1' '2021Q2' '202
1Q3' '2021Q4' '2022Q1' '2022Q2' '2022Q3' '2022Q4'
 '2023Q1' '2023Q2' '2023Q3' '2023Q4']
```

In [51]:
```
# for first two quarters of 2022, there is increase in budget by 10% across all dep
# using [:, 4:6] because it selects all departments (rows) and only 2022 Q1-Q2 (col

budget_array[:, 4:6] = budget_array[:, 4:6] * 1.10
print(f"Updated budget array: {budget_array}")
```

```
Updated budget array: [[1200000 1300000 1250000 1400000 1633500 1815000 1600000 1700
000 1650000
  1750000 1800000 1900000]
 [1000000 1100000 1150000 1200000 1512500 1573000 1400000 1450000 1500000
  1550000 1600000 1650000]
 [ 800000  900000  850000  950000 1210000 1270500 1100000 1150000 1200000
  1250000 1300000 1350000]
 [1500000 1600000 1550000 1650000 2057000 2117500 1800000 1850000 1900000
  1950000 2000000 2100000]
 [ 900000  950000 1000000 1050000 1331000 1391500 1200000 1250000 1300000
  1350000 1400000 1450000]]
```