1, Consider the following NumPy array:

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

What will be the output of:

a) arr[1:, :2]

b) arr[::-1, ::2]

c) arr.reshape(1, -1)

d) np.where(arr > 5)

solution

Given the array:

```python
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

a) `arr[1:, :2]`

This performs array slicing:

- `1:` selects rows from index 1 to the end (rows 1 and 2)
- `:2` selects columns from start up to (but not including) index 2 (columns 0 and 1)

Output:

```text
array([[4, 5],
       [7, 8]])
```

b) `arr[::-1, ::2]`

This uses step-based slicing:

- `::-1` reverses the row order (step of -1)
- `::2` selects every second column starting from 0 (columns 0 and 2)

Output:

```text
array([[7, 9],
       [4, 6],
       [1, 3]])
```

c) `arr.reshape(1, -1)`

Reshapes the array:

- `1` creates 1 row
- `-1` automatically calculates the needed number of columns (9 in this case)

Output:

```text
array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

d) `np.where(arr > 5)`

Returns indices where the condition is true:

- Returns a tuple of arrays (one for each dimension)
- First array contains row indices, second contains column indices

Output:

```text
(array([1, 2, 2, 2]), array([2, 0, 1, 2]))
```

This means elements >5 are at positions: (1,2), (2,0), (2,1), (2,2)

2.Given a pandas DataFrame containing student records:

data = { 'Name': ['John', 'Alice', 'Bob'],

'Score1': [85, 92, 78],

'Score2': [90, 88, 85]}

Write expressions for:

a) Calculate mean score per student

b) Filter students with average score > 85

c) Sort by Score1 in descending order

d) Add a new column for grade based on average score

solution

Given the DataFrame:

python Copy Download

```python
import pandas as pd
data = {
    'Name': ['John', 'Alice', 'Bob'],
    'Score1': [85, 92, 78],
    'Score2': [90, 88, 85]
}
df = pd.DataFrame(data)
```

a) Calculate mean score per student

python Copy Download

```python
df['Mean_Score'] = df[['Score1', 'Score2']].mean(axis=1)
```

b) Filter students with average score > 85

python Copy Download

```python
df[df['Mean_Score'] > 85]
```

d) Add a new column for grade based on average score

python Copy Download

```python
def assign_grade(score):
    if score >= 90: return 'A'
    elif score >= 80: return 'B'
    elif score >= 70: return 'C'
    else: return 'D'

df['Grade'] = df['Mean_Score'].apply(assign_grade)
```

3. Analyze the time complexity and space complexity of:

a) NumPy array broadcasting

b) Pandas DataFrame merging

c) GroupBy operations

d) Sorting in pandas

solution

a) NumPy array broadcasting

- **Time Complexity:** O(n) where n is the size of the resulting array after broadcasting
- **Space Complexity:** O(n) for the output array
- Broadcasting avoids explicit loops but creates temporary arrays in memory

b) Pandas DataFrame merging

- **Time Complexity:** O(n log n) for hash-based joins (most common case)
- **Space Complexity:** O(n) for the resulting DataFrame
- Complexity can increase to $O(n^2)$ for non-indexed or cross joins

c) GroupBy operations

- **Time Complexity:** O(n) for simple aggregations with hash-based grouping
- **Space Complexity:** O(k) where k is the number of groups
- More complex operations (like transform) can be O(n log n) or $O(n^2)$

d) Sorting in pandas

- **Time Complexity:** O(n log n) for most cases (using quicksort or mergesort)
- **Space Complexity:** O(n) for the sorted DataFrame
- In-place sorting can reduce space complexity to O(1)

4. Design a data processing pipeline using pandas and NumPy to:

- **Load and clean a large dataset**
- **Handle missing values**
- **Perform feature engineering**
- **Create visualization Explain the rationale behind each step and potential**
- **optimization techniques.**

Solution

1. Load and clean a large dataset

- **Rationale:** Efficient loading prevents memory issues
- **Approach:**
  - Use `pd.read_csv()` with `chunksize` for large files
  - Specify `dtypes` to reduce memory usage
  - Filter unnecessary columns early
- **Optimization**: Use efficient data types (category for strings, downcast numbers)

2. Handle missing values

- **Rationale:** Missing data affects analysis quality
- **Approach:**
  - Identify patterns with `isna().sum()` and visualization
  - Decide strategy: imputation (mean/median), deletion, or prediction
- **Optimization**: Use `fillna()` with `inplace=True` to save memory

3. Feature engineering

- **Rationale:** Create meaningful predictors
- **Approach:**
  - Create interaction terms, polynomial features
  - Date/time decomposition
  - Text feature extraction
- **Optimization:** Vectorized operations with NumPy

4. Create visualization

- **Rationale:** Exploratory data analysis
- **Approach:**
  - Use `matplotlib` or `seaborn`
  - Start with distributions, correlations
  - Progress to more complex visualizations
- **Optimization:** Sample large datasets before plotting

5. Three factories A, B, C of an electric bulb manufacturing company produces 45%, 35% and 20% of the total output. Approximately 1.5%, l% and 2% of the bulbs produced by these factories are known to be defective. If a randomly selected bulb manufactured by the company was found to be defective, what is the probability that the bulb was manufactured in factory B?

Solution

Given:

- P(A) = 0.45 (Factory A production)
- P(B) = 0.35 (Factory B production)
- P(C) = 0.20 (Factory C production)
- P(D|A) = 0.015 (Defect rate for A)
- P(D|B) = 0.01 (Defect rate for B)
- P(D|C) = 0.02 (Defect rate for C)

We need to find P(B|D) - probability bulb was from B given it's defective.

Using Bayes' Theorem:
P(B|D) = [P(D|B) * P(B)] / P(D)

First calculate total probability of defect P(D):
P(D) = P(D|A)P(A) + P(D|B)P(B) + P(D|C)P(C)
= (0.015*0.45) + (0.01*0.35) + (0.02*0.20)
= 0.00675 + 0.0035 + 0.004
= 0.01425

Now apply Bayes' Theorem:
P(B|D) = (0.01 * 0.35) / 0.01425
= 0.0035 / 0.01425
≈ 0.2456 or 24.56%