NumPy

Exercise1:

Scenario: Imagine you have temperature data for 5 cities over 3 days.

1. Create a 2D NumPy array to store the temperature data. You can use np.random.randint to generate random integer values between a specific range (e.g., 10 to 30 degrees Celsius).
2. Print the original array to see the temperatures for each city across the days.
3. Calculate some basic statistics for the entire dataset:
    - Find the average temperature across all cities and days.
    - Find the city with the highest average temperature.
4. Select the days where the temperature was above 25 degrees in any city.

Objective: Using Random modules, NumPy array functions for calculations and extractions

Exercise2:

Scenario: You're working with a dataset containing sales figures for different products over a month. The data is represented as a 2D NumPy array where each row represents a product and each column represents a day of the month.

1. **Import NumPy and create a sample sales data array.** You can use np.array() to create a 2D array with random sales values.
2. **Calculate the total sales for each product.** Use summation functions like np.sum() along the desired axis (axis=1 for summing each row).
3. **Find the day of the month with the highest total sales.** Sum the sales across all products for each day (axis=0) and find the index of the maximum value using np.argmax().
4. **Identify the product that had the most sales overall.** Similar to step 3, sum across all days (axis=1) and find the index of the maximum value.
5. **Calculate the average daily sales.** Calculate the total sales for the month (using np.sum() on the entire array) and divide it by the number of days in the month.

Objective: Using multidimensional arrays, performing axis-wise operations, and extracting information for specific rows or columns.

Scenario: Simulate the spread of a disease in a population over time.

Instructions:

1. **Import libraries:**

2. **Define parameters:**

- population_size: Total number of individuals (e.g., 1000)
- initial_infected: Number of initially infected individuals (e.g., 10)
- infection_rate: Probability of a susceptible individual getting infected upon contact with an infected individual (e.g., 0.1)
- simulation_days: Number of days to simulate the disease spread (e.g., 30)

3. **Initialize arrays:**

- Create a NumPy array population of size population_size filled with ones (representing susceptible individuals).
- Set the first initial_infected elements of population to zero (representing infected individuals).

4. **Simulate disease spread for each day:**

- Iterate for simulation_days:
    - Randomly select a subset of individuals proportional to the infection_rate from the infected individuals (use np.random.choice).
    - Set the corresponding elements in the population array to zero (representing infected) for the selected individuals who come in contact with infected ones.

5. **Track daily infected counts:**

- Create another NumPy array daily_infected of size simulation_days to store the number of infected individuals each day.
- Within the loop from step 4, calculate the number of infected individuals (elements with value 0) in the population array for each day and store it in the daily_infected array.

6. **Plot the results:**

- Use plt.plot() to plot the daily_infected array vs. the simulation days (x-axis).
- Label the axes and title the plot (e.g., "Disease Spread Simulation").

Objective: Using NumPy array manipulation, loops, and conditional operations for simulating a process. By plotting with matplotlib, you can visualize the disease spread dynamics.

Pandas

Exercise 4:

Scenario: You're analyzing a dataset containing information about customer purchases. The data is provided in a CSV file named "customer_purchases.csv".

Instructions:

1. **Import pandas and read the CSV data** "customer_purchases.csv"
2. **Identify missing values (if any) and handle them:**
    - Decide on a strategy to handle missing values (e.g., remove rows with missing values, impute missing values with a specific value like the mean).
3. **Find unique category names.**
    - For columns with categorical data (e.g., product category), use df["column_name"].unique() to see the unique values.
4. **Calculate purchase statistics:**
    - Group the data by customer ID (df.groupby("customer_id")).
    - Calculate descriptive statistics like total purchase amount, average purchase value, or most frequently purchased product category for each customer using appropriate aggregation functions.
5. **Identify top N customers by total purchase amount:**
    - Use .sort_values() on the grouped data by total purchase amount (descending order) and select the top N customers.

Visualize the results using libraries like matplotlib to create charts showing trends or identify patterns in customer behavior.

Objective: Essential pandas operations for data cleaning, manipulation, aggregation, and working with grouped data. It also provides a foundation for customer analysis tasks.

Exercise 5:

Scenario: You're analyzing a dataset containing sales information for a clothing store. The data includes columns like CustomerID, ItemDescription, Price, Quantity, and SaleDate.

Instructions:

1. **Import pandas and read the data:**

2. **Identify the most popular items (by total quantity sold):**
    - Group the DataFrame by ItemDescription and calculate the sum of Quantity for each item using the groupby() and sum() methods.
    - Sort the resulting DataFrame by the sum of Quantity in descending order.
    - Select the top N items (e.g., N=10) to identify the most popular items based on total quantity sold.
3. **Analyze sales by month:**
    - Extract the month from the SaleDate column and add a new column named Month to the DataFrame. You can use datetime functions in pandas to achieve this.
    - Group the DataFrame by Month and calculate the following for each month:
        - Total sales amount (sum of Price multiplied by Quantity)
        - Average sales amount per transaction (total sales amount divided by the number of transactions - count of rows in the group)
        - Number of unique customers (count of unique values in CustomerID)
    - Print or visualize the resulting DataFrame to analyze sales trends across months.
4. **Find customers who spent above a certain amount:**
    - Calculate the total spending per customer by grouping by CustomerID and summing the product of Price and Quantity.
    - Filter the DataFrame to select customers whose total spending is greater than a specific threshold (e.g., $100).

5. **Visualizations**: Create visualizations (using libraries like matplotlib) to represent your findings, such as bar charts for total spending per customer or pie charts for item category distribution.

Objective: Using DataFrames in pandas. Applying data manipulation, filtering, grouping, aggregation, and basic data analysis techniques.