Exercise 1: Library Management System

Create a basic Library Management System with the following requirements:

1. **Class: Book**
    - Attributes:
        - title (string)
        - author (string)
        - copies (integer)
    - Methods:
        - __init__(self, title, author, copies) - Initializes a book with the given title, author, and number of copies.
        - borrow_book(self) - Decreases the number of copies by 1 if available; otherwise, prints "No copies left."
        - return_book(self) - Increases the number of copies by 1.
2. **Class: Library**
    - Attributes:
        - books (list of Book objects)
    - Methods:
        - __init__(self) - Initializes the library with an empty list of books.
        - add_book(self, book) - Adds a Book object to the library's collection.
        - list_books(self) - Prints a list of all books in the library along with the number of available copies.
        - find_book(self, title) - Searches for a book by its title and returns the Book object if found; otherwise, prints "Book not found."
3. **Test the System**
    - Add a few books to the library.
    - Borrow and return books.

- List all books in the library.

4. Example Output

Adding books to the library...

Listing all books:

1. Harry Potter by J.K. Rowling - Copies: 5

2. The Great Gatsby by F. Scott Fitzgerald - Copies: 3

Borrowing "Harry Potter"...

Harry Potter borrowed successfully!

Borrowing "Harry Potter" again...

Harry Potter borrowed successfully!

Listing all books:

1. Harry Potter by J.K. Rowling - Copies: 3

2. The Great Gatsby by F. Scott Fitzgerald - Copies: 3

Returning "Harry Potter"...

Harry Potter returned successfully!

Listing all books:

1. Harry Potter by J.K. Rowling - Copies: 4

2. The Great Gatsby by F. Scott Fitzgerald - Copies: 3

Exercise 2: Bank Account Management System

Problem Statement

Design a program that simulates basic banking operations. Implement the following:

1. **Class: BankAccount**
    - Attributes:
        - account_holder (string)
        - balance (float, default is 0)
    - Methods:
        - __init__(self, account_holder, initial_balance) - Initializes the account with the holder's name and an optional initial balance.
        - deposit(self, amount) - Adds the specified amount to the balance.
        - withdraw(self, amount) - Subtracts the specified amount from the balance if sufficient funds exist; otherwise, prints "Insufficient funds."
        - display_balance(self) - Prints the current balance.
2. **Class: SavingsAccount (inherits BankAccount)**
    - Attributes:
        - interest_rate (float, e.g., 0.03 for 3%)
    - Methods:
        - apply_interest(self) - Adds interest to the balance based on the interest_rate.
3. **Test the System**
    - Create a savings account for a user.
    - Deposit and withdraw money.
    - Apply interest and display the updated balance.

Example Output

Creating a savings account for Alice...

Initial balance: $500.00

Depositing $200.00 into Alice's account...

New balance: $700.00

Withdrawing $100.00 from Alice's account...

New balance: $600.00

Applying interest...

Interest added: $18.00

New balance: $618.00

Exercise 3: Data Cleaning with Pandas

Problem Statement

You are provided with a dataset of employees in a company, which contains some issues such as missing values, incorrect formats, and duplicates. Perform the following tasks to clean the data.

Dataset Overview (employee_data.csv):

Employee ID	Name	Age	Department	Salary	Joining Date
101	John Doe	28	HR	50000	2020-03-15
102	Jane Smith	NaN	IT	60000	2020/07/10
103	Emily Brown	32	Marketing	70000	2021-05-21
104	Mark White	45	HR	NaN	2022-01-10
105	Sarah Black	29	IT	65000	2021/10/13
106	Sarah Black	29	IT	65000	2021/10/13

Tasks:

1. Load the Data

- Load the dataset into a Pandas DataFrame.

2. Clean Empty Cells

- Identify and handle missing data in columns like Age, Salary, and Joining Date. Fill or drop rows/columns based on your analysis.

3. Clean Wrong Format

- The Joining Date column contains dates in two different formats (2020-03-15 and 2020/07/10). Convert the column to a consistent format YYYY-MM-DD.

4. Clean Wrong Data

- The Age column should be between 18 and 70. Replace any value outside this range with NaN and then handle those NaN values appropriately.

5. Remove Duplicates

- Identify and remove duplicate rows, especially for Employee ID and Name.

Steps Breakdown:

Load the Data: The CSV file is loaded into a DataFrame for easy manipulation.

Clean Empty Cells: The missing values in Age, Salary, and Joining Date are handled. For Age and Salary, we fill missing values with the mean and median, respectively. We drop rows where Joining Date is missing.

Clean Wrong Format: The Joining Date column, which has inconsistent date formats, is standardized to YYYY-MM-DD using pd.to_datetime().

Clean Wrong Data: The Age column is cleaned by replacing values outside the reasonable range (18 to 70) with NaN, which is then filled with the mean value.

Remove Duplicates: Duplicate rows based on Employee ID and Name are removed using drop_duplicates().

Expected Outputs:

Missing Data: You'll see the changes in missing data handling, particularly for Age, Salary, and Joining Date.

Date Format: The Joining Date column will be in a consistent format (YYYY-MM-DD).

Age Data: Any out-of-range Age values are cleaned and handled.

Duplicates: Duplicates based on Employee ID and Name are removed.

You can save the cleaned data into a new file, cleaned_employee_data.csv.

Exercise 4 : Scenario: Heights of Adults in a Population

Imagine you're conducting a study on the heights of adult women in a country. Based on previous research, you know that the heights of adult women follow a **normal distribution** with the following parameters:

- **Mean height (μ\muμ)**: 160 cm (average height)
- **Standard deviation (σ\sigmaσ)**: 10 cm (measure of variation in height)

You want to simulate the heights of **1000 adult women** and analyze the distribution of heights in your sample.

Steps:

1. **Define the Parameters**:
    - **Mean height**: 160 cm
    - **Standard deviation**: 10 cm
    - **Sample size**: 1000 women
2. **Use Normal Distribution to Simulate Heights**: You can generate **1000 random heights** from a normal distribution with a mean of **160 cm** and a standard deviation of **10 cm**.
3. **Objective**:
    - Simulate the heights of 1000 women.
    - Plot the distribution of these heights.
    - Analyze the proportion of women within certain height ranges (e.g., how many are between 150 cm and 170 cm?).