**Pandas Series and DataFrame Creation**

1. **Series Creation:** Create a Pandas Series named s\_fruits from the list ['apple', 'banana', 'cherry'] and assign custom indices 'a', 'b', 'c'.
2. **DataFrame from Dictionary:** Create a DataFrame named df\_students from the following dictionary: data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35], 'Score': [90, 85, 95]}.
3. **DataFrame from NumPy Array:** Create a 4×3 NumPy array of random integers between 10 and 20. Use this array to create a DataFrame named df\_random with column names 'X', 'Y', 'Z'.
4. **Display Head and Info:** For the df\_students DataFrame, print the first 2 rows and then print a summary of its structure, including data types and non-null values.

**Indexing, Selection, and Slicing**

1. **Selecting a Column:** From df\_students, select and print the entire 'Name' column as a Pandas Series.
2. **Selecting Multiple Columns:** From df\_students, select and print the 'Name' and 'Score' columns as a new DataFrame.
3. **Selecting Rows by loc:** Use the **.loc** indexer on df\_students to select and print the row corresponding to the student named 'Bob' (you may need to set 'Name' as the index first, or rely on the default integer index if not changed).
4. **Selecting Rows by iloc:** Use the **.iloc** indexer on df\_students to select and print the second row of the DataFrame.
5. **Conditional Selection (Filtering):** From df\_students, use Boolean indexing to select and print only the rows where the 'Score' is greater than 90.
6. **Setting and Resetting Index:** Set the 'Name' column of df\_students as the index **in place**. Then, reset the index back to the default integer index.

**Data Cleaning and Manipulation**

1. **Handling Missing Data (Count):** Given a Series s\_missing = pd.Series([1, np.nan, 3, 4, np.nan]), use a Pandas function to count how many missing (NaN) values it contains.
2. **Dropping Missing Data:** From s\_missing, use a method to create and print a new Series with all the missing values removed.
3. **Filling Missing Data:** Given a Series s\_fill = pd.Series([10, np.nan, 20, np.nan]), use the .fillna() method to replace all missing values with the **mean** of the existing values. Print the result.
4. **Creating a New Column:** Add a new column named 'Passed' to df\_students. Set the value to 'Yes' if the 'Score' is 90 or above, and 'No' otherwise.

**Grouping and Aggregation**

1. **Basic Aggregation:** Calculate and print the **average** of the 'Age' column and the **maximum** value of the 'Score' column in df\_students.
2. **GroupBy and Mean:** Create a DataFrame named df\_sales with columns 'Region' and 'Revenue'. Populate it with sample data (e.g., 3 regions). Use .groupby() to find the **mean revenue** for each region.

**Operations and Series Methods**

1. **Applying a Function (Lambda):** Use the .apply() method on the 'Age' column of df\_students with a lambda function to calculate the age 5 years from now. Store the result in a new Series.
2. **Mapping Values:** Given a Series s\_code = pd.Series([1, 2, 1, 3]), use the .map() method to convert the codes to colors using the dictionary mapping = {1: 'Red', 2: 'Green', 3: 'Blue'}.
3. **Sorting Values:** Sort the df\_students DataFrame by the 'Age' column in **descending** order and print the sorted DataFrame.
4. **Unique Values:** From the 'Name' column of df\_students, use a Series method to print a list of all **unique** names present.