Program 1

Student Performance Analysis using Numpy

Design a Python program to analyse the student performance data across multiple subjects. The program should allow users to get the input data for a specified number of students, including their names, ages, and scores in Math, Science, Physics, and Chemistry. Based on this data, the program will calculate and display various statistics:

Data Analysis Objectives:

- **Overall Average Score:** Computes the average score across all students.
- **Top Students:** Finds and displays the top-performing students.
- **Filtering:** Allows filtering of students by age and subject scores.

```
import numpy as np  # Importing numpy library for numerical operations


# Function to initialize student data interactively
def initialize_student_data(num_students):
    student_data = []  # Initialize an empty list to store student data
    for i in range(num_students):  # Loop to gather data for each student
        name = input(f"Get the student's name for Student {i+1}: ")  # Get the student's name
        age = int(input(f"Get the age for {name}: "))  # Get the age for the student
        math_score = float(input(f"Get the Math score for {name}: "))  # Get the Math score
        science_score = float(input(f"Get the Science score for {name}: "))  # Get the Science score
        physics_score = float(input(f"Get the Physics score for {name}: "))  # Get the Physics score
        chemistry_score = float(input(f"Get the Chemistry score for {name}: "))  # Get the Chemistry score


        student_data.append([name, age, math_score, science_score, physics_score, chemistry_score])  # Append student data to list


    student_data = np.array(student_data)  # Convert list of lists to a numpy array
    return student_data  # Return numpy array containing student data


# Function to calculate overall average score
def calculate_overall_average(student_data):
    scores = student_data[:, 2:].astype(float)  # Extract scores from student data and convert to float
    overall_avg = np.mean(scores)  # Calculate mean of all scores
```

```
    return overall_avg  # Return overall average score

# Function to find top N students based on overall average score
def top_students_overall(student_data, n):
    scores = student_data[:, 2:].astype(float)  # Extract scores from student data and convert to float
    overall_avg_scores = np.mean(scores, axis=1)  # Calculate mean score for each student
    top_indices = np.argsort(overall_avg_scores)[::-1][:n]  # Get indices of top N students based on scores
    top_students = student_data[top_indices]  # Get top students based on indices

    return top_students  # Return top N students

# Function to filter students based on criteria (age and score in a specific subject)
def filter_students(student_data, min_age, min_score, subject='Math'):
    subject_index = {'Math': 2, 'Science': 3, 'Physics': 4, 'Chemistry': 5}[subject]  # Determine index of subject based on input
    filtered_students = student_data[(student_data[:, 1].astype(int) >= min_age) &
                                     (student_data[:, subject_index].astype(float) >= min_score)]  # Filter students
    return filtered_students  # Return filtered students

# Example usage with interactive input
num_students = int(input("Get the number of students: "))  # Get the number of students
student_data = initialize_student_data(num_students)  # Initialize student data interactively

print("\nInitial Student Data:")  # Print header for initial student data
print(student_data)  # Print initial student data
print() # Output an empty line (newline) for better readability

overall_avg = calculate_overall_average(student_data)  # Calculate overall average score of students
print(f"Overall Average Score of Students: {overall_avg:.2f}")  # Print overall average score
print() # Output an empty line (newline) for better readability

top_n = int(input("Get the number of top students to display: "))  # Get the number of top students
top_students = top_students_overall(student_data, top_n)  # Find top N students based on overall average score
print(f"\nTop {top_n} Students based on Overall Average Score:")  # Print header for top students
print(top_students)  # Print top students
```

```
print() # Output an empty line (newline) for better readability


min_age_filter = int(input("Get the minimum age to filter students: "))  # Get the minimum age to filter students

min_score_filter = float(input("Get the minimum score in Physics to filter students: "))  # Get the minimum score in Physics

filtered_students = filter_students(student_data, min_age_filter, min_score_filter, subject='Physics')  # Filter students based on criteria

print(f"\nStudents aged {min_age_filter} or older with at least {min_score_filter} in Physics:")  # Print header for filtered students

print(filtered_students)  # Print filtered students


min_score_filter_chem = float(input("Get the minimum score in Chemistry to filter students: "))  # Get the minimum score in Chemistry

filtered_students_chem = filter_students(student_data, min_age_filter, min_score_filter_chem, subject='Chemistry')  # Filter students based on criteria

print(f"\nStudents aged {min_age_filter} or older with at least {min_score_filter_chem} in Chemistry:")  # Print header for filtered students

print(filtered_students_chem)  # Print filtered students
```

Program 1b

Develop a machine learning model that accurately classifies iris flowers into one of three species based on their sepal and petal measurements using pandas library

Objective:

To build a Decision Tree classifier using the Iris dataset that achieves high accuracy in predicting the species of iris flowers.

The model should be able to:

- **Accurately Classify Iris Species:** Develop a classifier that correctly identifies the species of iris flowers based on their sepal length, sepal width, petal length, and petal width.
- **Evaluate Model Performance:** Measure the performance of the classifier using metrics such as accuracy score and confusion matrix.

```
# Import necessary libraries and modules

from sklearn.datasets import load_iris  # To load the Iris dataset

import pandas as pd  # For data manipulation with DataFrames

from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
```

```
from sklearn.tree import DecisionTreeClassifier  # Import Decision Tree classifier
from sklearn.metrics import accuracy_score, confusion_matrix  # Import metrics for evaluation

# Load the dataset
iris = load_iris()  # Load Iris dataset from sklearn

# iris is a dictionary-like object with data, target, and other attributes
X = iris.data  # Features (independent variables)
y = iris.target  # Target (dependent variable)

# Convert to DataFrame for easier manipulation and analysis
df = pd.DataFrame(data=X, columns=iris.feature_names)  # Create a DataFrame with feature names as columns
df['target'] = y  # Add a target column to the DataFrame
missing_values = df.isnull().sum()  # Check for missing values in the DataFrame
print("Missing Values =", missing_values)  # Print the count of missing values

# Summary statistics of the dataset
summary_stats = df.describe()  # Generate summary statistics of the DataFrame
print(summary_stats)  # Print the summary statistics

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Split the data into training (80%) and testing (20%) sets using a fixed random state for reproducibility

# Initialize the Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)  # Create a Decision Tree classifier object

# Fit the classifier on the training data
clf.fit(X_train, y_train)  # Train the Decision Tree classifier using the training data

# Predictions on the test data
y_pred = clf.predict(X_test)  # Use the trained classifier to predict labels on the test set

# Evaluate the model performance
```

```
accuracy = accuracy_score(y_test, y_pred)  # Compute the accuracy score of the model
print(f'Accuracy: {accuracy}')  # Print the accuracy score

# Compute and display the Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)  # Generate confusion matrix
print('Confusion Matrix:')  # Print header for confusion matrix
print(conf_matrix)  # Print the confusion matrix itself
```