CONTENTS

Ex. No. 01

Programs Using Numpy and Pandas

Aim:

To implement python programs using Numpy and Pandas libraries for data collections, Statistical analysis and classification using a machine learning model on real-world and user-defined datasets.

Procedure:

step 01: Import the required libraries, including Numpy for performing numerical and statistical operations and Pandas for managing and analysing structured tabular data.

step 02: Create a method to collect structured data from the user, such as names, ages and scores. Organise this data using Numpy arrays to enable efficient numeric computations.

step 03: Use Numpy to calculate overall statistical measures like averages or totals by processing only the numerical portions of the dataset.

Step 04: Identify top-performing records based on calculated scores by sorting the Numpy array in descending order and selecting the highest values.

Step 05: Apply conditional filtering to select records that meet specific criteria, such as a minimum age or subject score, using logical indexing techniques.

Step 06: Load a predefined dataset for analysis and model development. Convert this dataset into a Pandas DataFrame to facilitate structured data operations and easy column access.

Step 07: Rename the columns of the Pandas DataFrame appropriately and include an additional column to represent categorical class labels or output values.

Step 08: Explore the dataset using Pandas by checking for any missing data and summarising the distribution of values through descriptive statistics.

Step 09: Divide the dataset into training and testing portions for the purpose of building and evaluating a machine learning model. Ensure consistency by using a fixed method for the split.

Step 10: Select and train a classification model using the training portion of the data. Use the trained model to make predictions on the testing data.

step 11: Evaluate the performance of the model by comparing predicted values with actual outcomes. Interpret the results using standard accuracy and error analysis methods.

Program 1a:

```
imposk numpy as np
def initialize_student_data(num_students):
  student_data=[]
  for i in range(num_students):
     name=input(f"Get the Student's name
                    for student {i+1}:")
     age=int(input(f"Get the age for
                    {name}:"))
     math_score= float(input(f"Get the Math
                    score for {name}:"))
```

```
        science_score = float(input(f"Get the
                Science score for {name}: "))
        physics_score = float(input(f"Get the
                Physics score for {name}: "))
        chemistry_score = float(input(f"Get the
                Chemistry score for {name}: "))

        student_data.append([name, age,
            math_score, science_score, physics_score,
                        chemistry_score])

    student_data = np.array(student_data)
    return student_data

def calculate_overall_average(student_data):
    scores = student_data[:, 2:].astype(float)
    overall_avg = np.mean(scores)
    return overall_avg

def top_students_overall(student_data, n):
    scores = student_data[:, 2:].astype(float)
    overall_avg_scores = np.mean(scores, axis=1)
    top_indices = np.argsort(overall_avg_scores)
                                [::-1][:n]
    top_students = student_data[top_indices]

    return top_students
```

Output 1a:

Get the number of students: 5

Get the student's name for Student 1: Nive

Get the age for Nive: 22

Get the Math Score for Nive: 85

Get the Science score for Nive: 72

Get the Physics score for Nive: 73

Get the Chemistry score for Nive: 63

Initial Student Data:

[['Nive' '22' '85.0' '72.0' '73.0' '63.0']]

Overall Average Score of Students: 73.25

Get the number of top students to display: 2

Top 2 students based on Overall Average score:

```
def filter_students(student_data, min_age,
                    min_score, subject='Math'):
    subject_index = {'Math': 2, 'Science': 3,
        'Physics': 4, 'Chemistry': 5}[subject]
    filtered_students = student_data[(student_
        data[:,1].astype(int) >= min_age) &
        (student_data[:, subject_index].
            astype(float) >= min_score)]
    return filtered_students

num_students = int(input("Get the no. of students:
                         "))
student_data = initialize_student_data(num_
                                       students)
print("\nInitial Student Data:")
print(student_data)
print()

overall_avg = calculate_overall_average(student_data)
print(f"Overall Average Score of Students:
                {overall_avg:.2f}")
print()

top_n = int(input("Get the number of top
            students to display:"))
top_students = top_students_overall(student_data,
                                    top_n)
print(f"\nTop {top_n} Students based on
            Overall Average Score:")
```

[['Nive' '22' '85.0' '72.0' '73.0' '63.0']]

Get the minimum age to filter students: 21

Get the minimum score in Physics to filter students: 70

Students aged 21 or older with at least 70.0 in Physics: None

Get the minimum Score in Chemistry to filter Students: 60

Students aged 21 or older with at least 60.0 in Chemistry: None

```
print(top_students)
print()

min_age_filter = int(input("Get the minimum
        age to filter students:"))
min_score_filter = float(input("Get the
        minimum score in physics to
                filter students:"))
filtered_students = filter_students(student_data,
        min_age_filter, min_score_filter,
                subject='Physics')
print(f"\nStudents aged {min_age_filter} or
        older with at least {min_score_filter}
                in Physics:")

min_score_filter_chem= float(input("Get the
        minimum score in Chemistry to
                filter students:"))
filtered_students_chem = filter_students
        (student_data, min_age_filter, min_score_filter
        _chem, subject = 'Chemistry')
print(f"\nStudents aged {min_age_filter} or
        older with at least {min_score_filter
                chem} in Chemistry:")
print(filtered_students_chem)
```

Output 1b:

Missing Values : Sepal length (cm) 0

Sepal width (cm) 0

petal length (cm) 0

petal width (cm) 0

target

dtype: int64

	Sepal length (cm)	Sepal width (cm)	petal length (cm)
Count	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667
std	0.828066	0.433594	1.764420
min	4.300000	2.000000	1.000000
25%	5.100000	2.800000	1.600000
50%	5.800000	3.000000	4.350000
75%	6.400000	3.300000	5.100000
max	7.900000	4.400000	6.900000

Program 1b:

```
from sklearn.datasets import load_iris
import pandas as pd
from sklearn.model_selection import
                                train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
                            confusion_matrix

iris = load_iris()
X = iris.data
y = iris.target
df = pd.DataFrame(data = X, columns = iris.feature
                                    _names)

df['target'] = y
missing_values = df.isnull().sum()
print("Missing Values = ", missing_values)
summary_stats = df.describe()
print(summary_stats)
X_train, X_test, y_train, y_test = train_test_split
        (X, y, test_size = 0.2, random_state = 42)

clf = DecisionTreeClassifier(random_state = 42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

	petal width (cm)	target
Count	150.000000	150.000000
mean	1.198667	1.000000
std	0.763161	0.819232
min	0.100000	0.000000
25%	0.300000	0.000000
50%	1.300000	1.000000
75%	1.800000	2.000000
max	2.500000	2.000000

Accuracy: 1.0

Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
conf_matrix = confusion_matrix (y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

Result:

The programs were successfully implemented using Numpy and Pandas libraries. Numpy was used to handle and analyse structured numerical data efficiently, while Pandas was used to organise and explore tabular data.