AIML Programs

1. a. Develop a program to read the student details like Name, USN, and Marks in three subjects. Display the student details, total marks and percentage with suitable messages.

```
def calculate_total_and_percentage(marks1, marks2, marks3):
    total = marks1 + marks2 + marks3
    percentage = (total / 300) * 100  # Assuming the maximum marks per subject is 100
    return total, percentage
def display_student_details(name, usn, marks1, marks2, marks3, total, percentage):
    print("\nStudent Details:")
    print(f"Name       : {name}")
    print(f"USN        : {usn}")
    print(f"Marks in Subject 1: {marks1}")
    print(f"Marks in Subject 2: {marks2}")
    print(f"Marks in Subject 3: {marks3}")
    print(f"Total Marks    : {total}")
    print(f"Percentage     : {percentage:.2f}%")
def main():
    try:
        name = input("Enter student's name: ")
        usn = input("Enter student's USN: ")
        marks1 = float(input("Enter marks in subject 1: "))
        marks2 = float(input("Enter marks in subject 2: "))
        marks3 = float(input("Enter marks in subject 3: "))
 if not (0 <= marks1 <= 100 and 0 <= marks2 <= 100 and 0 <= marks3 <= 100):
            raise ValueError("Marks should be between 0 and 100")
total, percentage = calculate_total_and_percentage(marks1, marks2, marks3)
display_student_details(name, usn, marks1, marks2, marks3, total, percentage)
except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
```

```
        print(f"Unexpected error: {e}")
    main()
```

OUTPUT:

```
        Enter student's name: John Doe
Enter student's USN: 1BCS2345
Enter marks in subject 1: 85
Enter marks in subject 2: 90
Enter marks in subject 3: 88

Student Details:
Name       : John Doe
USN        : 1BCS2345
Marks in Subject 1: 85.0
Marks in Subject 2: 90.0
Marks in Subject 3: 88.0
Total Marks    : 263.0
Percentage     : 87.67%
```

b. Develop a program to read the name and year of birth of a person. Display whether the person is a senior citizen or not.

```
from datetime import datetime
def check_senior_citizen(age):
    if age >= 60:
        return "Yes, the person is a senior citizen."
    else:
        return "No, the person is not a senior citizen."
def main():
    try:
        name = input("Enter the person's name: ")
        year_of_birth = int(input("Enter the year of birth: "))
```

```
    current_year = datetime.now().year
    age = current_year - year_of_birth
   result = check_senior_citizen(age)
     print(f"\nName: {name}")
    print(f"Age: {age}")
    print(result)
  except ValueError:
    print("Invalid input! Please enter a valid year of birth.")
 main()
```

OUTPUT:

```
      Enter the person's name: Alice
Enter the year of birth: 1955

Name: Alice
Age: 69
Yes, the person is a senior citizen.
```

2. a. Develop a program to generate Fibonacci sequence of length (N). Read N from the console.

```
def generate_fibonacci(N):
  fibonacci_sequence = []
     a, b = 0, 1
  for _ in range(N):
   fibonacci_sequence.append(a)
    a, b = b, a + b
  return fibonacci_sequence
   def main():
  try:
      N = int(input("Enter the length of the Fibonacci sequence: "))
          if N <= 0:
        print("Please enter a positive integer for the length of the sequence.")
sequence = generate_fibonacci(N)
print(f"The Fibonacci sequence of length {N} is:")
```

```
        print(sequence)
        except ValueError:
        print("Invalid input! Please enter a valid integer.")
         main()
```

OUTPUT:

```
        Enter the length of the Fibonacci sequence: 5
The Fibonacci sequence of length 5 is:
[0, 1, 1, 2, 3]
```

b. Write a function to calculate factorial of a number. Develop a program to compute binomial coefficient (Given N and R).

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
def binomial_coefficient(N, R):
    if R > N:
        return 0  # Binomial coefficient is 0 if R > N
    return factorial(N) // (factorial(R) * factorial(N - R))

def main():
    try:
        N = int(input("Enter the value of N: "))
        R = int(input("Enter the value of R: "))
              if N < 0 or R < 0:
            print("Please enter non-negative integers for N and R.")
            return
        result = binomial_coefficient(N, R)
        print(f"The binomial coefficient C({N}, {R}) is: {result}")
    except ValueError:
```

```
        print("Invalid input! Please enter valid integers for N and R.")
    main()
```

output:

Enter the value of N: 5

Enter the value of R: 2

The binomial coefficient C(5, 2) is: 10

3. Read N numbers from the console and create a list. Develop a program to print mean, variance and standard deviation with suitable messages.

```
import math
def calculate_mean(numbers):
    return sum(numbers) / len(numbers)
def calculate_variance(numbers, mean):
    return sum((x - mean) ** 2 for x in numbers) / len(numbers)
def calculate_standard_deviation(variance):
    return math.sqrt(variance)
def main():
    try:
        # Read N numbers from the user
        N = int(input("Enter the number of elements: "))
        if N <= 0:
            print("Please enter a positive integer for the number of elements.")
            return

        numbers = []
        print(f"Enter {N} numbers:")
        for i in range(N):
            number = float(input(f"Number {i+1}: "))
            numbers.append(number)
```

```
        mean = calculate_mean(numbers)
        variance = calculate_variance(numbers, mean)
        standard_deviation = calculate_standard_deviation(variance)
        print(f"\nMean: {mean:.2f}")
        print(f"Variance: {variance:.2f}")
        print(f"Standard Deviation: {standard_deviation:.2f}")

    except ValueError:
        print("Invalid input! Please enter valid numeric values.")
    main()
```

OUTPUT:

```
        Enter the number of elements: 5
Enter 5 numbers:
Number 1: 1
Number 2: 2
Number 3: 3
Number 4: 4
Number 5: 5

Mean: 3.00
Variance: 2.00
Standard Deviation: 1.41
```

4. Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with suitable message.

```
  from collections import Counter
def calculate_digit_frequency(number_str):
    return Counter(number_str)
def main():
    number_str = input("Enter a multi-digit number: ")
    if not number_str.isdigit():
```

```
        print("Invalid input! Please enter a valid multi-digit number containing only digits.")
    frequency = calculate_digit_frequency(number_str)
    print("\nFrequency of each digit:")
    for digit, count in sorted(frequency.items()):
        print(f"Digit {digit}: {count} times")
    main()
```

OUTPUT:

```
 Enter a multi-digit number: 112233
Frequency of each digit:
Digit 1: 2 times
Digit 2: 2 times
Digit 3: 2 times
```

5. Develop a program to print 10 most frequently appearing words in a text file. [Hint: Use dictionary with distinct words and their frequency of occurrences. Sort the dictionary in the reverse order of frequency and display dictionary slice of first 10 items]

```
import os
import re
from collections import Counter
def create_sample_file(filename):
    sample_text = """Hello world! Welcome to the world of programming. Programming is fun. Enjoy programming.
                This file is a sample file to demonstrate the word frequency counting program.
                Programming is everywhere. Enjoy learning Python programming."""
    if not os.path.exists(filename):
        with open(filename, 'w') as file:
            file.write(sample_text)
        print(f"File '{filename}' created and sample text written.")
    else:
        print(f"File '{filename}' already exists.")
def get_most_frequent_words(filename):
    try:
```

```
        with open(filename, 'r') as file:
            text = file.read().lower()  # Convert to lowercase to avoid case sensitivity
            words = re.findall(r'\b\w+\b', text)  # This matches words consisting of alphanumeric
characters
            word_count = Counter(words)
            most_common_words = word_count.most_common(10)
            return most_common_words
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
        return []
def main():
    filename = "sample_text.txt"
    create_sample_file(filename)
    top_words = get_most_frequent_words(filename)
    if top_words:
        print("\nTop 10 most frequently appearing words:")
        for word, freq in top_words:
            print(f"'{word}': {freq} times")
    else:
        print("No words to display”)
    main()
```

OUTPUT:

```
File 'sample_text.txt' created and sample text written.


Top 10 most frequently appearing words:
'programming': 4 times
'is': 3 times
'file': 2 times
'hello': 1 times
'world': 1 times
'welcome': 1 times
```

'to': 1 times

'of': 1 times

'enjoy': 1 times

'learning': 1 times

6. Develop a program to sort the contents of a text file and write the sorted contents into a separate With effect from the academic year 2024-25 text file. [Hint: Use string methods strip(), len(), list methods sort(), append(), and file methods open(), readlines(), and write()].

```
  import os
def create_sample_input_file(input_filename):
   sample_text = """This is the first line.
Apple is a fruit.
Zebra is at the end.
Banana is yellow."""
   if not os.path.exists(input_filename):
      with open(input_filename, 'w') as file:
         file.write(sample_text)
      print(f"File '{input_filename}' created and sample text written.")
   else:
      print(f"File '{input_filename}' already exists.")
def sort_file_contents(input_filename, output_filename):
   try:
      with open(input_filename, 'r') as input_file:
         lines = input_file.readlines()
         lines.sort()
      with open(output_filename, 'w') as output_file:
         for line in lines:
            output_file.write(line)
      print(f"The contents have been sorted and written to '{output_filename}'.")
   except FileNotFoundError:
      print(f"Error: The file '{input_filename}' was not found.")
   except Exception as e:
```

```
        print(f"An error occurred: {e}")
def main():
    input_filename = "input.txt"
    output_filename = "sorted_output.txt"
    create_sample_input_file(input_filename)
    sort_file_contents(input_filename, output_filename)
    main()
```

Output:

```
        Apple is a fruit.
Banana is yellow.
This is the first line.
Zebra is at the end.
```

7. Develop a program to backing Up a given Folder (Folder in a current working directory) into a ZIP File by using relevant modules and suitable methods.

```
import os
import shutil
def backup_folder_to_zip(folder_name, zip_name):
    try:
        if os.path.exists(folder_name) and os.path.isdir(folder_name):
            # Create a ZIP file from the folder
            shutil.make_archive(zip_name, 'zip', folder_name)
            print(f"Backup successful! Folder '{folder_name}' has been backed up as '{zip_name}.zip'.")
        else:
            print(f"Error: The folder '{folder_name}' does not exist or is not a valid directory.")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    folder_name = input("Enter the name of the folder to back up: ")
    zip_name = input("Enter the desired name for the ZIP file (without extension): ")
```

```
    backup_folder_to_zip(folder_name, zip_name)
    main()
```

Output:

Enter the name of the folder to back up: test_folder

Enter the desired name for the ZIP file (without extension): backup_test_folder

Backup successful! Folder 'test_folder' has been backed up as 'backup_test_folder.zip'.

8. Write a function named DivExp which takes TWO parameters a, b and returns a value c (c=a/b). Write suitable assertion for a>0 in function DivExp and raise an exception for when b=0. Develop a suitable program which reads two values from the console and calls a function DivExp.

```
def DivExp(a, b):
    assert a > 0, "Error: 'a' must be greater than 0”
    if b == 0:
        raise ZeroDivisionError("Error: Division by zero is not allowed.")
    c = a / b
    return c
def main():
    try:
        a = float(input("Enter the value of a (must be greater than 0): "))
        b = float(input("Enter the value of b (must not be zero): "))
        result = DivExp(a, b)
        print(f"The result of {a} / {b} is: {result}")

    except AssertionError as e:
        print(e)
    except ZeroDivisionError as e:
        print(e)
    except ValueError:
        print("Error: Please enter valid numeric values.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

```
    main()
```

OUTPUT:

Enter the value of a (must be greater than 0): 10

Enter the value of b (must not be zero): 2

The result of 10.0 / 2.0 is: 5.0

9. Define a function which takes TWO objects representing complex numbers and returns new complex number with a addition of two complex numbers. Define a suitable class 'Complex' to represent the complex number. Develop a program to read N (N >=2) complex numbers and to compute the addition of N complex numbers.

```
class Complex:
    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
    def __add__(self, other):
        # Adding the real parts and imaginary parts separately
        real_part = self.real + other.real
        imaginary_part = self.imaginary + other.imaginary
        return Complex(real_part, imaginary_part)
    def display(self):
        return f"{self.real} + {self.imaginary}i" if self.imaginary >= 0 else f"{self.real} - {-self.imaginary}i"
def add_complex_numbers(n):
    total_sum = Complex(0, 0)  # Start with a complex number 0 + 0i
    for i in range(n):
        real = float(input(f"Enter the real part of complex number {i+1}: "))
        imaginary = float(input(f"Enter the imaginary part of complex number {i+1}: "))
        complex_number = Complex(real, imaginary)
        total_sum = total_sum + complex_number  # Add the current complex number to the sum
    return total_sum
def main():
    try:
        N = int(input("Enter the number of complex numbers (N >= 2): "))
```

```
        if N < 2:
            print("Error: N must be greater than or equal to 2.")
        result =add_complex_numbers(N)
        print(f"The sum of the {N} complex numbers is: {result.display()}")
    except ValueError:
        print("Error: Please enter valid numeric values.")
    main()
```

OUTPUT:

```
Enter the number of complex numbers (N >= 2): 3
Enter the real part of complex number 1: 3
Enter the imaginary part of complex number 1: 2
Enter the real part of complex number 2: 1
Enter the imaginary part of complex number 2: 4
Enter the real part of complex number 3: 5
Enter the imaginary part of complex number 3: -3
The sum of the 3 complex numbers is: 9 + 3i
```

10. Develop a program that uses class Student which prompts the user to enter marks in three subjects and calculates total marks, percentage and displays the score card details. [Hint: Use list to store the marks in three subjects and total marks. Use __init__() method to initialize name, USN and the lists to store marks and total, Use getMarks() method to read marks into the list, and display() method to display the score card details.

```
class Student:
    def __init__(self, name, usn):
        self.name = name
        self.usn = usn
        self.marks = []  # List to store marks in three subjects
        self.total = 0   # Total marks

    def getMarks(self):
        try:
            # Input marks for three subjects
```

```
            for i in range(3):
                mark = float(input(f"Enter marks for subject {i+1}: "))
                self.marks.append(mark)
            self.total = sum(self.marks)
        except ValueError:
            print("Invalid input. Please enter numeric values for marks.")
            exit()
    def display(self):
        percentage = (self.total / 300) * 100 if self.total <= 300 else 0
        print("\n----- Score Card -----")
        print(f"Name: {self.name}")
        print(f"USN: {self.usn}")
        print(f"Marks in 3 subjects: {self.marks}")
        print(f"Total Marks: {self.total}/300")
        print(f"Percentage: {percentage:.2f}%")
def main():
    name = input("Enter the student's name: ")
    usn = input("Enter the student's USN: ")

    student = Student(name, usn)
    student.getMarks()
    student.display()
    main()
```

OUTPUT:

```
        Enter the student's name: John Doe
Enter the student's USN: 1A18CS101
Enter marks for subject 1: 85
Enter marks for subject 2: 90
Enter marks for subject 3: 80
----- Score Card -----
Name: John Doe
```

USN: 1A18CS101

Marks in 3 subjects: [85.0, 90.0, 80.0]

Total Marks: 255.0/300

Percentage: 85.00%

11) simple linear regression : predict the sepal length of the irirs

```
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
iris = datasets.load_iris()
X = iris.data[:, 2].reshape(-1, 1)  # Petal Length (cm)
y = iris.data[:, 0]  # Sepal Length (cm)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f'Intercept: {model.intercept_}')
print(f'Coefficient: {model.coef_}')

plt.scatter(X_test, y_test, color='blue', label='Actual data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Sepal Length (cm)')
plt.legend()
plt.show()
for true, pred in zip(y_test, y_pred):
    print(f'True: {true:.2f}, Predicted: {pred:.2f}')
```

12) implementation of k nearest neighbour algorithm

```
 import numpy as np
```

```
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
iris = datasets.load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target (species label)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
X_train_2D = X_train[:, :2]  # Using only sepal length and sepal width
X_test_2D = X_test[:, :2]  # Using only sepal length and sepal width
knn.fit(X_train_2D, y_train)
x_min, x_max = X_train_2D[:, 0].min() - 1, X_train_2D[:, 0].max() + 1
y_min, y_max = X_train_2D[:, 1].min() - 1, X_train_2D[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_train_2D[:, 0], X_train_2D[:, 1], c=y_train, marker='o', label='Training points')
plt.scatter(X_test_2D[:, 0], X_test_2D[:, 1], c=y_test, marker='x', label='Testing points')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title(f'KNN Decision Boundary (k=3)')
plt.legend()
plt.show()
```

13. Implementation of SVM Classification using Binary class

```
   import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
iris = datasets.load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target (species label)
binary_classes = [0, 1]
X_binary = X[np.isin(y, binary_classes)]
y_binary = y[np.isin(y, binary_classes)]
X_train, X_test, y_train, y_test = train_test_split(X_binary, y_binary, test_size=0.3, random_state=42)
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
X_train_2D = X_train[:, :2]  # Use only sepal length and sepal width
X_test_2D = X_test[:, :2]
svm_model.fit(X_train_2D, y_train)
x_min, x_max = X_train_2D[:, 0].min() - 1, X_train_2D[:, 0].max() + 1
y_min, y_max = X_train_2D[:, 1].min() - 1, X_train_2D[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))
Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.coolwarm)
```

```
plt.scatter(X_train_2D[:, 0], X_train_2D[:, 1], c=y_train, marker='o', label='Training points',
cmap=plt.cm.coolwarm)

plt.scatter(X_test_2D[:, 0], X_test_2D[:, 1], c=y_test, marker='x', label='Testing points',
cmap=plt.cm.coolwarm)

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.title('SVM Decision Boundary (Binary Class: Setosa vs Versicolor)')

plt.legend()

plt.show()
```

14. Implementation of Decision tree

```
    import numpy as np

import pandas as pd

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

iris = datasets.load_iris()

X = iris.data  # Features (sepal length, sepal width, petal length, petal width)

y = iris.target  # Target (species label)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

dt_model = DecisionTreeClassifier(random_state=42)

dt_model.fit(X_train, y_train)

y_pred = dt_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%')

plt.figure(figsize=(12, 8))

plot_tree(dt_model, filled=True, feature_names=iris.feature_names, class_names=iris.target_names,

rounded=True, fontsize=12)

plt.title("Decision Tree for Iris Classification")

plt.show()
```

15. Implementation of K Means

```
 import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
iris = datasets.load_iris()
X = iris.data  # Features (sepal length, sepal width, petal length, petal width)
y = iris.target  # Target labels (species), which we won't use for clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_
pca = PCA(n_components=2)
X_2D = pca.fit_transform(X)
plt.figure(figsize=(8, 6))
plt.scatter(X_2D[:, 0], X_2D[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='k', s=100)
centers_2D = pca.transform(kmeans.cluster_centers_)
plt.scatter(centers_2D[:, 0], centers_2D[:, 1], c='red', marker='X', s=200, label='Centroids')
plt.title('K-Means Clustering (Iris Dataset)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
from sklearn.metrics import adjusted_rand_score
print(f"Adjusted Rand Index: {adjusted_rand_score(y, labels)}")
```