ML PGMS

1. Simulating a simple calculator

```
print("Operation: +, -, *, /")
select = input("Select operations: ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if select == "+":
   print(num1, "+", num2, "=", num1+num2)
elif select == "-":
   print(num1, "-", num2, "=", num1-num2)
elif select == "*":
   print(num1, "*", num2, "=", num1*num2)
elif select == "-":
   print(num1, "/", num2, "=", num1/num2)
else:
   print ("Invalid input")
```

2. Armstrong Series

```
lower = int(input("Enter the lower range : "))
upper = int(input("Enter the upper range : "))
for num in range(lower, upper + 1):
   order = len(str(num))
   sum = 0
   temp = num
   while temp > 0:
      digit = temp%10
      sum+=digit**order
      temp//=10
   if num == sum:
      print (num)
```

3. Fibonacci Series

```
nterms = int(input("How many terms? "))
n1, n2 = 0, 1
count = 0
if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence:")
   while count < nterms:
      print(n1)
      nth = n1 + n2
      n1 = n2 n2 = nth count += 1
```

4. Modules and Functions

```
def summation(a,b):
   return a+b
def multiplication(a,b):
   return a*b
def divide(a,b):
   return a/b
a = int(input("Enter the first number:"))
b = int(input("Enter the second number:"))
print("Sum = ",summation(a,b))
print("Product = ",multiplication(a,b))
print("Divisor = ",divide(a,b))
```

5. Working with Strings

a) From the string input count the special characters, alphabets, digits, lowercase and uppercase characters.

```
def Count(string):
   alpha = upper = lower = number = special = 0
   for char in string:
      if char.isalpha():
         alpha += 1
         if char.isupper():
            upper += 1
         else:
            lower += 1
      elif char.isdigit():
         number += 1
      elif char != " ":
         special += 1
   print('Digits:', number)
   print('Alphabets:', alpha)
   print('Special characters:', special)
   print('Lowercase:', lower)
   print('Uppercase:', upper)
string = input("Enter a string: ")
Count(string)
```

b) Print the String "Welcome". Matrix size must be N X M. (N is an odd natural number, and M is 3 times N.). The design should have 'WELCOME' written in the centre. The design pattern should only use |, .and - characters.

```
N, M = map(int, input("Enter N and M: ").split())
for i in range(1, N, 2):
   print(('.|.' * i).center(M, '-'))
print('WELCOME'.center(M, '-'))
for i in range(N-2, 0, -2):
   print(('.|.' * i).center(M, '-'))
```

6. Data Pre-processing: Building Good Training sets

```
import pandas as pd
from sklearn.model_selection import train_test_split
df = pd.read_csv("Heart.csv")
print(df.describe(), df.info())
df.replace(0, pd.NA, inplace=True)
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
object_cols = df.select_dtypes(include=['object']).columns
df[object_cols] = df[object_cols].fillna(df[object_cols].mode().iloc[0])
df = df.infer_objects()
x, y = df.iloc[:, :-1].values, df.iloc[:, -1].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

7. Manipulating the twitter Dataset

```
import pandas as pd, re
from nltk.stem import PorterStemmer
data = pd.read_csv("tweets1.csv")
stemmer = PorterStemmer()
data['new'] = data['text'].str.replace(r"@[\w]*", "", regex=True)
data['new'] = data['new'].str.replace("[^a-zA-Z#]", " ", regex=True)
data['new'] = data['new'].apply(lambda x: ' '.join(w for w in x.split() if len(w) > 3))
data['new'] = data['new'].apply(lambda x: [stemmer.stem(w) for w in x.split()])
print(data['new'].head())
```

8. Evaluating the Results of Machine Learning

```
y = ['0','1','0','1','1','1','0','1','0','1','0','0','0','1','1','1','0','1','1','0']
y_pred = ['0','0','0','0','1','0','1','1','1','1','0','0','0','0','0','1','0','1','1','0']
TP = TN = FP = FN = 0
for actual, pred in zip(y, y_pred):
    if actual == '1' and pred == '1': TP += 1
    elif actual == '0' and pred == '0': TN += 1
    elif actual == '1' and pred == '0': FN += 1
    elif actual == '0' and pred == '1': FP += 1
confusion_matrix = [TP, TN, FP, FN]
print("Confusion Matrix:", confusion_matrix)
ACC = (TP + TN) / (TP + FP + TN + FN) if (TP + FP + TN + FN) else 0
PREC = TP / (TP + FP) if (TP + FP) else 0
REC = TP / (TP + FN) if (TP + FN) else 0
SN = REC
SP = TN / (TN + FP) if (TN + FP) else 0
MCE = 1 - ACC
print("ACCURACY:", ACC)
```

```
print("PRECISION:", PREC)
print("RECALL:", REC)
print("SENSITIVITY:", SN)
print("SPECIFICITY:", SP)
print("MISCLASSIFICATION ERROR:", MCE)
```

9. Implementing Linear Regression

```
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv("Salary_Data.csv")
x, y = df["YearsExperience"], df["Salary"]
def LinearRegressor(x, y):
    b1 = np.cov(x, y, bias=True)[0, 1] / np.var(x)
    return np.mean(y) - b1 * np.mean(x), b1
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5, shuffle=True)
b0, b1 = LinearRegressor(x_train, y_train)
y_pred = b0 + b1 * x_test
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R²:", r2_score(y_test, y_pred))
plt.scatter(x_test, y_test, color="k")
plt.plot(x_test, y_pred, color="b")
plt.show()
```

10. Implementing Classification Algorithm

```
import pandas as pd, seaborn as sn, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
X, y = pd.read_csv("Iris.csv").iloc[:, 1:5].values, pd.read_csv("Iris.csv").iloc[:, 5].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
y_pred = GaussianNB().fit(X_train, y_train).predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred) * 100)
sn.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d")
plt.xlabel("Predicted"), plt.ylabel("Actual"), plt.show()
```

11. Implementing Clustering using K-Means clustering algorithm

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from sklearn.cluster import KMeans
df = pd.read_csv("Mall_Customers.csv")
X = df.iloc[:, [3, 4]].values
kmeans = KMeans(n_clusters=5, init="k-means++", random_state=0).fit(X)
df["cluster"] = kmeans.labels_
for i, c in enumerate(["red", "blue", "green", "cyan", "magenta"]):
    plt.scatter(*X[df["cluster"] == i].T, s=100, c=c, label=f"Cluster {i+1}")
plt.scatter(*kmeans.cluster_centers_.T, s=300, c="yellow", label="Centroids")
plt.title("Customer Clusters"), plt.xlabel("Annual Income (k$)"), plt.ylabel("Spending Score"),
plt.legend() plt.show()
```