1. **Introduction to Python libraries for Data Mining: NumPy, SciPy, Pandas, Matplotlib, Scikit-Learn**

Write a Python program to do the following operations:

Library: NumPy

a) Create multi-dimensional arrays and find its shape and dimension

b) Create a matrix full of zeros and ones

c) Reshape and flatten data in the array

d) Append data vertically and horizontally

e) Apply indexing and slicing on array

f) Use statistical functions on array - Min, Max, Mean, Median and Standard Deviation

**a) Create multi-dimensional arrays and find its shape and dimension**

import numpy as np #importing NumPy with alias np

a=np.array([[1,2,3],[2,3,4],[3,4,5]]) #creation of multi-dimensional array with array() method

#shape

b=a.shape # b=np.shape(a), here shape is a method which returns number of dimensions and number of values each dimension is holding

print("shape:",b) #print("shape:",a.shape)

#dimension

c=a.ndim #ndim() is a method which returns the dimension of the given Array

print("dimensions:",a.ndim) #print("dimensions:",c)

**b) Create a matrix full of zeros and ones**

#matrix full of zeros

z=np.zeros((2,2)) #zeros() is a method which creates zeros with specified array shape

print("zeros:",z)

#matrix full of ones

o=np.ones((2,2)) #ones() is a method which creates ones with specified array shape

print("ones:",o)

**c) Reshape and flatten data in the array**

#matrix reshape

a=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]])

b=a.reshape(4,2,2) #get a new shape to an array without changing its data

print("Original:",a)

print("Reshape:",b)

#matrix flatten

c=a.flatten() #doesn’t change contents of an existing array. Instead, it returns a new flat array.

print("flatten:",c)

**d) Append data vertically and horizontally**

#Appending data vertically

x=np.array([[10,20],[80,90]])

y=np.array([[30,40],[60,70]])

v=np.vstack((x,y))# Using the vstack() function, items of arrays are arranged vertically

print("vertically:",v)

#Appending data horizontally

h=np.hstack((x,y))# Using the hstack() function, items of arrays are arranged horizontally

print("horizontally:",h)

**e) Apply indexing and slicing on array**

#indexing

a=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]])

temp = a[[0, 1, 2, 3], [1, 1, 1, 1]]

print("indexing",temp)

#slicing

i=a[:4,::2]

print("slicing",i)

**f) Use statistical functions on array - Min, Max, Mean, Median and Standard Deviation**

a=np.array([[1,3,-1,4],[3,-2,1,4]])

b=a.min()

print("minimum:",b)

#max for finding maximum of an array

c=a.max()

print("maximum",c)

#mean

a=np.array([1,2,3,4,5])

d=a.mean()

print("mean:",d)

#median

e=np.median(a)

print("median:",e)

#standard deviation

f=a.std()

print("standard deviation",f)

Key points:

* The word NumPy stands for Numerical Python
* Python lists are a substitute for arrays, but they fail to deliver the performance required while computing large sets of numerical data.
* Numpy arrays are faster, more efficient, and require less syntax than standard python sequences.
* In “*import numpy as np*” *np* is a commonly used alias to NumPy
* Using the *vstack()* function, items of arrays are arranged vertically. Suppose array 1 has elements [22, 32, 43] and array 2 has elements [546, 55, 556]. The two arrays can be arranged vertically using the function *vstack(( arr1 , arr2 ) )* where arr1 and arr2 are array 1 and array 2 respectively.

1. **Demonstrate the following data pre-processing tasks using python libraries.**

a) Loading the dataset

b) Identifying the dependent and independent variables

c) Dealing with missing data

**a) Loading the dataset**

import pandas as pd # importing pandas with alias pd.

import io # io is a standard library no need to install it separately using pip

from google.colab import files# files is an object obtaining from google.colab module

uploaded = files.upload()# show the form to upload a file

dataset = pd.read\_excel(io.BytesIO(uploaded['Salary.xlsx'])) # performing input operation by accessing BytesIO module which is in io library.

#dataset = pd.read\_csv('Salary.csv') # to import the dataset into a variable

#dataset = pd.read\_excel('Salary.xlsx') # run *pip install openpyxl* to load excel file

**b) Identifying the dependent and independent variables**

X = dataset.iloc[:, :-1].values # attributes to determine independent variable / Class

Y = dataset.iloc[:, -1].values # dependent variable / Class

**c) Dealing with missing data**

dataset = pd.read\_excel('Salary.xlsx')

num\_var = dataset[['Age', 'Income']] #select numerical variables from dataset

print(num\_var)

# impute the missing fields in a dataset using scikit-learn library’s SimpleImputer Class

from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing\_values=np.nan, strategy="mean")

X\_mean = imp.fit\_transform(num\_var)# fit\_transform method do calculating the mean of columns and then replacing the missing values according to it

print(X\_mean)

1. **Write a python program to impute missing values with various techniques on given dataset.**

a) Remove rows/ attributes

b) Replace with mean or mode

c) Write a python program to perform transformation of data using Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.

**a) Remove rows/ attributes**

dataset = pd.read\_excel('Salary.xlsx')

dataset.drop(["Region", "Online Shopper"], axis = 1, inplace = True) # .drop() method for deleting and filtering data frame

dataset # display

**b) Replace with mean or mode**

import pandas as pd

import io

from google.colab import files

uploaded = files.upload()

dataset = pd.read\_excel(io.BytesIO(uploaded['Salary.xlsx']))

# Replacing column with mean of that column

dataset['Age'] = dataset['Age'].fillna(dataset['Age'].mean())#fillna () method replaces the NULL values with mean

dataset['Income'] = dataset['Income'].fillna(dataset['Income'].mean())

# Replacing column with mode (number that occurs the most in a given list of numbers) of that column

import statistics

dataset['Age']=dataset['Age'].fillna(statistics.mode(dataset['Age']))

dataset['Income']=dataset['Income'].fillna(statistics.mode(dataset['Income']))

**c) Write a python program to perform** **transformation of data using** **Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.**

**transformation of data using Discretization (Binning)**

import numpy as np

import math

from sklearn.datasets import load\_iris # loading iris dataset from sklearn datasets

from sklearn import datasets, linear\_model, metrics

# load iris data set

dataset = load\_iris()

a = dataset.data

b = np.zeros(150)

# take 1st column among 4 column of data set

for i in range (150):

b[i]=a[i,1]

b=np.sort(b) #sort the array

# create bins

bin1=np.zeros((30,5))

bin2=np.zeros((30,5))

bin3=np.zeros((30,5))

# Bin mean

for i in range (0,150,5): #the sequence of the values from 0 to 149 gets created. But between each element will have a gap of ‘5’.

k=int(i/5) # int () function converts the specified value into an integer number

mean=(b[i] + b[i+1] + b[i+2] + b[i+3] + b[i+4])/5

for j in range(5):

bin1[k,j]=mean

print("Bin Mean: \n",bin1)

# Bin boundaries

for i in range (0,150,5):

k=int(i/5)

for j in range (5):

if (b[i+j]-b[i]) < (b[i+4]-b[i+j]):

bin2[k,j]=b[i]

else:

bin2[k,j]=b[i+4]

print("Bin Boundaries: \n",bin2)

# Bin median

for i in range (0,150,5):

k=int(i/5)

for j in range (5):

bin3[k,j]=b[i+2]

print("Bin Median: \n",bin3)

**Transformation of data using normalization (MinMaxScaler or MaxAbsScaler)**

import numpy as np

Data = np.array([[300,24],[200,21],[126,18],[567,27],[420,19],[189,30]])

print (Data)

from sklearn.preprocessing import MaxAbsScaler #Transform features by scaling each feature to a given range.

from sklearn.preprocessing import MinMaxScaler

MMS\_scaler\_model = MinMaxScaler()

MMS\_scaler\_model.fit(Data)

MAS\_scaler\_model = MaxAbsScaler()

MAS\_scaler\_model.fit(Data)

MMS\_scaled\_data = MMS\_scaler\_model.transform(Data)

MAS\_scaled\_data = MAS\_scaler\_model.transform(Data)

print(MMS\_scaled\_data)

print(MAS\_scaled\_data)

1. **Demonstrate the following data preprocessing tasks using python libraries.**

a) Dealing with categorical data

b) Scaling the features

c) Splitting dataset into Training and Testing Sets

1. **Dealing with categorical data**

import pandas as pd

import io

from google.colab import files

uploaded = files.upload()

dataset = pd.read\_excel(io.BytesIO(uploaded['Salary.xlsx']))

X = dataset.iloc[:,[0]].values #select ‘Region’ column

Y = dataset.iloc[:,[3]].values #select ‘Online shopper’ column

from sklearn.preprocessing import OneHotEncoder # to convert categorical data, or text data, into numbers

onehotencoder = OneHotEncoder()

ohe\_X = onehotencoder.fit\_transform(X)

ohe\_Y = onehotencoder.fit\_transform(Y)

X = ohe\_X.toarray()

Y = ohe\_Y.toarray()

print(X)

print(Y)

**b) Scaling the features**

import numpy as np

from sklearn.preprocessing import StandardScaler#

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

sc\_y = StandardScaler()

Y\_train = Y\_train.values.reshape((len(Y\_train), 1))

Y\_train = sc\_y.fit\_transform(Y\_train)

Y\_train = Y\_train.ravel()

print(X\_train)

print(Y\_train)

print(X\_test)

print(Y\_test)

**c) Splitting dataset into Training and Testing Sets**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

dataset = pd.read\_excel('Pre\_Salary.xlsx')

print(dataset)

#output vector

y = dataset.OnlineShopper

#input vector

x=dataset.drop('OnlineShopper',axis=1)

#split

X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(x,y,test\_size=0.2)

#verify

print("shape of original dataset :", dataset.shape)

print("shape of input - training set", X\_train.shape)

print("shape of output - training set", Y\_train.shape)

print("shape of input - testing set", X\_test.shape)

print("shape of output - testing set", Y\_test.shape)

1. **Demonstrate the following Similarity and Dissimilarity Measures using python**

a) Pearson’s Correlation

b) Cosine Similarity

c) Jaccard Similarity

d) Euclidean Distance

e) Manhattan Distance

**a) Pearson’s Correlation**

import numpy as np

from scipy.stats import pearsonr# used to find Pearson correlation coefficient

import matplotlib.pyplot as plt

np.random.seed(42)# The seed() method initializes the basic random number generator

# prepare data

x = np.random.randn(15)

y = x + np.random.randn(15)

# plot x and y

plt.scatter(x, y)

#polyfit() method fits our data inside a polynomial function

#returns the unique values from the given lists

plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)))

plt.xlabel('x')

plt.ylabel('y')

plt.show()

# calculate Pearson's correlation

corr, \_ = pearsonr(x, y)

print('Pearsons correlation: %.3f' % corr)

**b) Cosine Similarity**

# import required libraries

import numpy as np

from numpy.linalg import norm

# define two lists or array

A = np.array([2,1,2,3,2,9])

B = np.array([3,4,2,4,5,5])

# compute cosine similarity

cosine = np.dot(A,B)/(norm(A)\*norm(B))

print("Cosine Similarity:", cosine)

**c) Jaccard Similarity**

def jaccard\_similarity(list1, list2):

s1 = set(list1)

s2 = set(list2)

return float(len(s1.intersection(s2)) / len(s1.union(s2)))

list1 = ['dog', 'cat', 'cat', 'rat']

list2 = ['dog', 'cat', 'mouse']

jaccard\_similarity(list1, list2)

**d) Euclidean Distance**

import numpy as np

# initializing points in

point1 = np.array((1, 2, 3))

point2 = np.array((1, 1, 1))

#finding sum of squares

sum\_sq = np.sum(np.square(point1 - point2))

#Doing squareroot and

#printing Euclidean distance

print(np.sqrt(sum\_sq))

**e) Manhattan Distance**

from scipy.spatial import distance

a = (1, 0, 2, 3)

b = (4, 4, 3, 1)

# mahattan distance b/w a and b

d = distance.cityblock(a, b)

print(d)

1. **Build a model using linear regression algorithm on any dataset.**

**I. Simple linear regression**

# Importing the necessary libraries

import pandas as pd

from sklearn.datasets import load\_boston

# loading the data

boston = load\_boston()

df = pd.DataFrame(boston.data,columns=boston.feature\_names)

# print the columns present in the dataset

print(df.columns)

# print the top 5 rows in the dataset

print(df.head())

df['MEDV']=boston.target # target variabl(dependent)

X = df[['RM']]

y = df[['MEDV']]

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 10)

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

regressor.score(X\_test, y\_test)

# predict the y values

y\_pred=regressor.predict(X\_test)

# a data frame with actual and predicted values of y

evaluate = pd.DataFrame({'Actual': y\_test.values.flatten(), 'Predicted': y\_pred.flatten()})

evaluate.head(10)

evaluate.head(10).plot(kind = 'bar')

**II. Multiple linear regression**

X = df[['LSTAT','INDUS','CRIM','NOX','TAX','PTRATIO','CHAS','ZN','DIS']]

y = df[['MEDV']]

# Splitting the dataset into train and test sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 10)

# Fitting the training data to our model

regressor.fit(X\_train, y\_train)

#score of this model

regressor.score(X\_test, y\_test)

# predict the y values

y\_pred=regressor.predict(X\_test)

# a data frame with actual and predicted values of y

evaluate = pd.DataFrame({'Actual': y\_test.values.flatten(), 'Predicted': y\_pred.flatten()})

evaluate.head(10)

1. **Build a classification model using Decision Tree algorithm on iris dataset**

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

from sklearn.tree import export\_graphviz

#from sklearn.externals.six import StringIO

from IPython.display import Image

from pydot import graph\_from\_dot\_data

import pandas as pd

import numpy as np

from io import StringIO

import six

import sys

sys.modules['sklearn.externals.six'] = six

iris = load\_iris()

X = pd.DataFrame(iris.data, columns=iris.feature\_names)

y = pd.Categorical.from\_codes(iris.target, iris.target\_names)

X.head()

y = pd.get\_dummies(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=1)

dt = DecisionTreeClassifier()

dt.fit(X\_train, y\_train)

dot\_data = StringIO()

export\_graphviz(dt, out\_file=dot\_data, feature\_names=iris.feature\_names)

(graph, ) = graph\_from\_dot\_data(dot\_data.getvalue())

Image(graph.create\_png())

y\_pred = dt.predict(X\_test)

species = np.array(y\_test).argmax(axis=1)

predictions = np.array(y\_pred).argmax(axis=1)

confusion\_matrix(species, predictions)

1. **Apply Naïve Bayes Classification algorithm on any dataset.**

# Assigning features and label variables

wheather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',

'Rainy','Sunny','Overcast','Overcast','Rainy']

temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

# Import LabelEncoder

from sklearn import preprocessing

#creating labelEncoder

le = preprocessing.LabelEncoder()

# Converting string labels into numbers.

wheather\_encoded=le.fit\_transform(wheather)

print(wheather\_encoded)

# Converting string labels into numbers

temp\_encoded=le.fit\_transform(temp)

label=le.fit\_transform(play)

print("Temp:",temp\_encoded)

print("Play:",label)

#Combinig weather and temp into single listof tuples

features=list(zip(wheather\_encoded,temp\_encoded))

print(features)

#Import Gaussian Naive Bayes model

from sklearn.naive\_bayes import GaussianNB

#Create a Gaussian Classifier

model = GaussianNB()

# Train the model using the training sets

model.fit(features,label)

#Predict Output

predicted= model.predict([[0, 2]]) # 0:Overcast, 2:Mild

print ("Predicted Value:", predicted)

#Import scikit-learn dataset library

from sklearn import datasets

#Load dataset

wine = datasets.load\_wine()

# print the names of the 13 features

print("Features: ", wine.feature\_names)

# print the label type of wine(class\_0, class\_1, class\_2)

print("Labels: ", wine.target\_names)

# print data(feature)shape

wine.data.shape

# print the wine data features (top 5 records)

print(wine.data[0:5])

# print the wine labels (0:Class\_0, 1:class\_2, 2:class\_2)

print(wine.target)

# Import train\_test\_split function

from sklearn.model\_selection import train\_test\_split

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(wine.data, wine.target, test\_size=0.3,random\_state=109) # 70% training and 30% test

#Import Gaussian Naive Bayes model

from sklearn.naive\_bayes import GaussianNB

#Create a Gaussian Classifier

gnb = GaussianNB()

#Train the model using the training sets

gnb.fit(X\_train, y\_train)

#Predict the response for test dataset

y\_pred = gnb.predict(X\_test)

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))