A) Design a simple machine learning model to train the training instances and test the Same.

Code:

```
from random import randint
TRAIN_SET_LIMIT=1000
TRAIN_SET_COUNT=100
TRAIN_INPUT = list()
TRAIN_OUTPUT = list()
for i in range (TRAIN_SET_COUNT) :
        a = randint(0, TRAIN_SET_LIMIT )
        b = randint(0, TRAIN_SET_LIMIT )
        c = randint(0, TRAIN_SET_LIMIT )
        op = a + (2 * b) + (3 * c)
        TRAIN_INPUT.append([a ,b, c])
        TRAIN_OUTPUT.append(op)
from sklearn.linear_model import LinearRegression
predictor = LinearRegression(n_jobs =-1)
predictor.fit(X = TRAIN_INPUT , y = TRAIN_OUTPUT)
X_Test =[[20 ,40,60]]
outcome = predictor.predict(X = X_Test)
coefficients = predictor.coef_
print('Outcome : {} \nCoefficients : {}'.format( outcome ,coefficients))
```

B) Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

Code :

```
import csv
num_attributes = 6
a =[]
print("\n The Given Training Data Set \n")
with open('C:/Users/dsouz/OneDrive/Desktop/ML prac/enjoysport.csv' ,'r') as csvfile :
        reader = csv.reader(csvfile)
        for row in reader:
        a.append (row)
        print(row)
print("\n the initial value of hypothesis :")
hypothesis = ['0'] * num_attributes
print(hypothesis)
for j in range (0,num_attributes):
        hypothesis[j] = a[0][j]
```

```
print("\n Find S: Finding a Maximally Specific Hypothesis\n")
for i in range( 0, len(a)) :
        if a[i] [num_attributes] =='yes' :
        for j in range (0, num_attributes) :
        if a[i][j]!= hypothesis[j] :
                hypothesis[j] ='?'
        else :
                hypothesis[j] = a[i][j]
print(" for training instance No:{0} the Hypothesis is".format(1) ,hypothesis)
print("\n The Maximally Specific Hypothesis for a given Training Exam ples :\n")
print(hypothesis)
```

Practical 2

A) Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
plt.style.use('ggplot')
# Load the data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Z-score the features
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)# The PCA model
pca = PCA(n_components=2) # estimate only 2 PCs
X_new = pca.fit_transform(X)
fig, axes = plt.subplots(1,2)
axes[0].scatter(X[:,0], X[:,1], c=y)
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')
axes[0].set_title('Before PCA')
axes[1].scatter(X_new[:,0], X_new[:,1], c=y)
axes[1].set_xlabel('PC1')
axes[1].set_ylabel('PC2')
axes[1].set_title('After PCA')
plt.show()
```

B) For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Code :

```
import csv
with open("C:/Users/dsouz/OneDrive/Desktop/ML prac/enjoysport.csv") as f:
	csv_file=csv.reader(f)
	data=list(csv_file)
	s=data[1][:-1]
	g=[['?' for i in range(len(s))] for j in range(len(s))]
for i in data:
	if i[-1]=="Yes":
	for j in range(len(s)):
	if i[j]!=s[j]:
		s[j]='?'
		g[j][j]='?'
	elif i[-1]=="No":
	for j in range(len(s)):
	if i[j]!=s[j]:
		g[j][j]=s[j]
	else:
		g[j][j]="?"
	print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
	print(s)
	print(g)
gh=[]
for i in g:
	for j in i:
	if j!='?':
	gh.append(i)
	break
print("\nFinal specific hypothesis:\n",s)
print("\nFinal general hypothesis:\n",gh)
```

Practical No 3

A) Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Code:

```
import numpy as np
import pandas as pd
from sklearn import datasets
wine= datasets.load_wine()
print(wine)
print("Feature:",wine.feature_names)
print("Labels:",wine.target_names)
X=pd.DataFrame(wine['data'])
print(X.head(0))
print(wine.data.shape)
y=print(wine.target)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(wine.data, wine.target,
test_size=0.30, random_state=109)
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(X_train ,y_train)
y_pred =gnb.predict(X_test)
print(y_pred)
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
from sklearn.metrics import confusion_matrix
cm=np.array(confusion_matrix(y_test ,y_pred))
cm
```

C) Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

Code:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
```

```
# Load a sample dataset (Iris dataset)
data = load_iris()
X = data.data
y = data.target
iris_df = pd.DataFrame(data=np.c_[data['data'], data['target']],
columns=data['feature_names']+['target'])
print(iris_df.head())



# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_state=42)

# Create a RandomForestClassifier with custom parameters
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
dt_y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test,dt_y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

clrf = RandomForestClassifier(n_estimators= 52,random_state=42)

# Train the classifier on the training data
clrf.fit(X_train, y_train)

# Make predictions on the test data
rf_y_pred = clrf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, rf_y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

dt_confusion_matrix=  confusion_matrix(y_test, dt_y_pred)
print("confusion matrix for decision tree:")
print(dt_confusion_matrix)

rf_confusion_matrix=  confusion_matrix(y_test, rf_y_pred)
print("confusion matrix for Random forest:")
print(rf_confusion_matrix)
```

Practical No 4

A) For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data=pd.read_csv("C:/Users/dsouz/OneDrive/Desktop/ML prac/headbrain.csv")
print(data.shape)
print(data.head())
X=data['Head Size(cm^3)'].values
Y=data['Brain Weight(grams)'].values
mean_x=np.mean(X)
mean_y=np.mean(Y)
n=len(X)
numer = 0
denom = 0
for i in range(n):
        numer+= (X[i] - mean_x) * (Y[i]- mean_y)
        denom +=(X[i] - mean_x) ** 2
m = numer/denom
c= mean_y -( m * mean_x)
print("Coefficients")
print(m,c)
max_x = np.max(X) + 100
min_x = np.min(X) - 100
x= np.linspace(min_x ,max_x , 1000)
y=c + m * x
plt.plot(x, y , color='#58b970', label='Regression Line')
plt.scatter(X,Y ,c ='#ef5423',label='Scatter plot')
plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```

B) For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm

Code:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('https://raw.githubusercontent.com/mk-
gurucharan/Classification/master/DMVWrittenTests.csv')
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values
dataset.head(5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
LogisticRegression()
y_pred = classifier.predict(X_test)
y_pred
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
Cm
```

Practical 5

A) Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample

Code :

```
import numpy as np
import math
import csv

def read_data(filename):
        with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
        metadata.append(name)
        for row in datareader:
        traindata.append(row)
```

```
        return (metadata, traindata)

class Node:
        def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

        def __str__(self):
        return self.attribute

def subtables(data, col, delete):
        dict = {}
        items = np.unique(data[:, col])
        count = np.zeros((items.shape[0], 1), dtype=np.int32)

        for x in range(items.shape[0]):
        for y in range(data.shape[0]):
        if data[y, col] == items[x]:
                count[x] += 1

        for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
        if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
        dict[items[x]] = np.delete(dict[items[x]], col, 1)

        return items, dict

def entropy(S):
        items = np.unique(S)

        if items.size == 1:
        return 0

        counts = np.zeros((items.shape[0], 1))
        sums = 0

        for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

        for count in counts:
        sums += -1 * count * math.log(count, 2)
        return sums
```

```
def gain_ratio(data, col):
        items, dict = subtables(data, col, delete=False)

        total_size = data.shape[0]
        entropies = np.zeros((items.shape[0], 1))
        intrinsic = np.zeros((items.shape[0], 1))

        for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

        total_entropy = entropy(data[:, -1])
        iv = -1 * sum(intrinsic)

        for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

        return total_entropy / iv

def create_node(data, metadata):
        if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

        gains = np.zeros((data.shape[1] - 1, 1))

        for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

        split = np.argmax(gains)

        node = Node(metadata[split])
        metadata = np.delete(metadata, split, 0)

        items, dict = subtables(data, split, delete=True)

        for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))

        return node

def empty(size):
        s = ""
        for x in range(size):
```

```
        s += "   "
        return s

def print_tree(node, level):
        if node.answer != "":
        print(empty(level), node.answer)
        return
        print(empty(level), node.attribute)
        for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)

metadata, traindata = read_data("C:/Users/dsouz/OneDrive/Desktop/ML
prac/tennisdata.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

B) Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set

Code :

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

dataset=load_iris()
#print(dataset)
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=
0)

kn=KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train,y_train)

for i in range(len(X_test)):
        x=X_test[i]
        x_new=np.array([x])
        prediction=kn.predict(x_new)
        print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",predictio
n,dataset["target_names"][prediction])
print(kn.score(X_test,y_test))
```

Practical No 6

A) Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix

Code:

```
from math import sqrt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# calculate euclidean distance
def euclidean_distance(a, b):
	return sqrt(sum((e1-e2) ** 2 for e1, e2 in zip(a, b)))
# calculate manhattan distance
def manhattan_distance(a, b):
	return sum(abs(e1-e2) for e1, e2 in zip(a, b))
# calculate minkowski distance
def minkowski_distance(a, b, p):
	return sum(abs(e1-e2)**p for e1, e2 in zip(a, b)) ** (1/p)
# define data
# actual values
actual = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]
# predicted values
predicted = [1, 0, 0, 1, 0, 0, 0, 1, 0, 0]
# calculate distance
dist1 = euclidean_distance(actual, predicted)
dist2 = manhattan_distance(actual, predicted)
# calculate distance (p-1)
dist3 = minkowski_distance(actual, predicted,1)
# calculate distance (p=1)
dist3 = minkowski_distance(actual, predicted,1)
print(dist3)
# calculate distance (p=2)
dist3 = minkowski_distance(actual, predicted, 2)
print(dist3)
# confusion matrix
matrix = confusion_matrix(actual, predicted, labels=[1, 0])
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual, predicted, labels=[1, 0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
matrix = classification_report(actual, predicted, labels=[1, 0])
print('Classification report : \n', matrix)
```

Practical No 7

A) Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

Code :

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('C:/Users/dsouz/OneDrive/Desktop/ML prac/Mall_Customers.csv')
x = dataset.iloc[:, [3, 4]].values
#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x) #visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'purple', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'yellow', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta ', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

Practical No 8

A) Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

Code :

```
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
```

```
from pgmpy.inference import VariableElimination

data = pd.read_csv("C:/Users/dsouz/OneDrive/Desktop/ML prac/ds4.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)

model = BayesianModel([
        ('age', 'Lifestyle'),
        ('Gender', 'Lifestyle'),
        ('Family', 'heartdisease'),
        ('diet', 'cholestrol'),
        ('Lifestyle', 'diet'),
        ('cholestrol', 'heartdisease'),
        ('diet', 'cholestrol')
])

model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

HeartDisease_infer = VariableElimination(model)

print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4')
print('For Gender enter Male:0, Female:1')
print('For Family History enter Yes:1, No:0')
print('For Diet enter High:0, Medium:1')
print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3')
print('for Cholesterol enter High:0, BorderLine:1, Normal:2')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
        'age': int(input('Enter Age: ')),
        'Gender': int(input('Enter Gender: ')),
        'Family': int(input('Enter Family History: ')),
        'diet': int(input('Enter Diet: ')),
        'Lifestyle': int(input('Enter Lifestyle: ')),
        'cholestrol': int(input('Enter Cholestrol: '))
})

print(q)
```

B) mplement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Code :

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
def kernel(point, xmat, k):
        m,n = np.shape(xmat)
        weights = np.mat(np.eye((m)))
        for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
        return weights

def localWeight(point, xmat, ymat, k):
        wei = kernel(point,xmat,k)
        W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
        return W

def localWeightRegression(xmat, ymat, k):
        m,n = np.shape(xmat)
        ypred = np.zeros(m)
        for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
        return ypred

# load data points
data = pd.read_csv('C:/Users/dsouz/OneDrive/Desktop/ML prac/10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Practical No 10

A) Bayesian documents

Code

```
import pandas as pd
msg = pd.read_csv('C:/Users/dsouz/OneDrive/Desktop/ML prac/document.csv',
names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

X = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)

#df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
#print(df[0:5])

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)

for doc, p in zip(Xtrain, pred):
        p = 'pos' if p == 1 else 'neg'
        print("%s -> %s" % (doc, p))

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
print('Recall: ', recall_score(ytest, pred))
print('Precision: ', precision_score(ytest, pred))
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```