```
#1a)
import pandas as pd
url='C:/Users/MRCET1/Desktop/train.csv'
df=pd.read_csv(url)
df.head(5)

#1b)
import pandas as pd
marks_data=pd.DataFrame({'ID':{0:23,1:43,2:12,3:13,4:67,5:89},'NAME':{0:'Ram',1:'Deep',2:'Yash',3:'Arjun',4:'Adit
ya',5:'Divya'},'Marks':{0:89,1:92,2:45,3:78,4:56,5:76},'Grade':{0:'b',1:'a',2:'f',3:' c',4:'e',5:'c'}})
filename='C:/Users/MRCET1/Desktop/Marksdata.xlsx'
marks_data.to_excel(filename)
print('Data frame written to Excel')

#2a)
#K-MEANS CLUSTERING ALGORITHM#
# ----------installations---------------
# 1.pip install scikit-learn
# 2.pip install matplotlib
# 3.pip install k-means-constrained
# 4.pip install pandas
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from k_means_constrained import KMeansConstrained
import pandas as pd
df = pd.read_csv('student_clustering.csv')
X = df.iloc[:, :].values
km = KMeansConstrained(n_clusters=4, max_iter=500)
y_means = km.fit_predict(X)
plt.scatter(X[y_means == 0, 0], X[y_means == 0, 1], color='red')
plt.scatter(X[y_means == 1, 0], X[y_means == 1, 1], color='blue')
plt.scatter(X[y_means == 2, 0], X[y_means == 2, 1], color='green')
plt.scatter(X[y_means == 3, 0], X[y_means == 3, 1], color='yellow')
plt.show()

#2b)
#Logistic Regression#
import matplotlib.pyplot as plt
```

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
x
y
model = LogisticRegression(solver='liblinear', random_state=0)
model.fit(x, y)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, l1_ratio=None,
max_iter=100,multi_class='warn', n_jobs=None, penalty='l2',random_state=0, solver='liblinear', tol=0.0001,
verbose=0,warm_start=False)
model = LogisticRegression(solver='liblinear', random_state=0).fit(x, y)
model.classes_
model.intercept_
model.coef_
model.predict_proba(x)

#3a)
import pandas,numpy
from sklearn.preprocessing import MinMaxScaler
df=pandas.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv',s
ep=';')
array=df.values
x=array[:,0:8]
y=array[:,8]
scaler=MinMaxScaler(feature_range=(0,1))
rescaledX=scaler.fit_transform(x)
numpy.set_printoptions(precision=2)
print("OUTPUT")
rescaledX[0:2,:]

#3b)Standardizing Data
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(x)
rescaledX=scaler.transform(x)
print("OUTPUT")
rescaledX[0:2,:]
```

```
#3)c.Normalizing Data
from sklearn.preprocessing import Normalizer
scaler=Normalizer().fit(x)
normalized=scaler.transform(x)
print("OUTPUT")
normalized[0:2,:]

#9)a.Binarizing Data
from sklearn.preprocessing import Binarizer
binarizer=Binarizer().fit(x)
binarized=binarizer.transform(x)
print("OUTPUT")
binarized[0:2,:]

#9)b.Mean Removal
from sklearn.preprocessing import scale
data_standardized=scale(df)
data_standardized.mean(axis=0)
print('OUTPUT')
data_standardized.std(axis=0)



#4)
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Function importing Dataset
def importdata():
  balance_data =
pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-'+'databases/balance-scale/balance-scale.data',sep=
',', header = None)
  # Printing the dataswet shape
```

```
  print ("Dataset Length: ", len(balance_data))
  print ("Dataset Shape: ", balance_data.shape)
  # Printing the dataset obseravtions
  print ("Dataset: ",balance_data.head())
  return balance_data
# Function to split the dataset
def splitdataset(balance_data):
  # Separating the target variable
  X = balance_data.values[:, 1:5]
  Y = balance_data.values[:, 0]
  # Splitting the dataset into train and test
  X_train, X_test, y_train, y_test = train_test_split(
  X, Y, test_size = 0.3, random_state = 100)
  return X, Y, X_train, X_test, y_train, y_test
# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
  # Creating the classifier object
  clf_gini = DecisionTreeClassifier(criterion = "gini",
  random_state = 100,max_depth=3, min_samples_leaf=5)
  # Performing training
  clf_gini.fit(X_train, y_train)
  return clf_gini
# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):
  # Decision tree with entropy
  clf_entropy = DecisionTreeClassifier(
  criterion = "entropy", random_state = 100,
  max_depth = 3, min_samples_leaf = 5)
  # Performing training
  clf_entropy.fit(X_train, y_train)
  return clf_entropy
# Function to make predictions
def prediction(X_test, clf_object):
  # Predicton on test with giniIndex
  y_pred = clf_object.predict(X_test)
  print("Predicted values:")
  print(y_pred)
  return y_pred
```

```
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
  print("Confusion Matrix: ",
  confusion_matrix(y_test, y_pred))
  print ("Accuracy : ",
  accuracy_score(y_test,y_pred)*100)
  print("Report : ",
  classification_report(y_test, y_pred))
# Driver code
def main():
# Building Phase
  data = importdata()
  X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
  clf_gini = train_using_gini(X_train, X_test, y_train)
  clf_entropy = tarin_using_entropy(X_train, X_test, y_train)
  # Operational Phase
  print("Results Using Gini Index:")
  # Prediction using gini
  y_pred_gini = prediction(X_test, clf_gini)
  cal_accuracy(y_test, y_pred_gini)
  print("Results Using Entropy:")
  # Prediction using entropy
  y_pred_entropy = prediction(X_test, clf_entropy)
  cal_accuracy(y_test, y_pred_entropy)
# Calling main function
if __name__=="__main__":
  main()

#5a)
#NAVIE BAYES CLASSIFICATION#
import pandas as pd
dataset = pd.read_csv('C:/Users/MRCET1/Desktop/train.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred)

#5b)KNN CLASSIFICATION MODEL#
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('C:/Users/MRCET1/Desktop/train.csv')
y = df['diagnosis']
X = df.drop('diagnosis', axis=1)
X = X.drop('Unnamed: 32', axis=1)
X = X.drop('id', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
K = []
training = []
test = []
scores = {}
for k in range(2, 21):
 clf = KNeighborsClassifier(n_neighbors=k)
 clf.fit(X_train, y_train)
 training_score = clf.score(X_train, y_train)
 test_score = clf.score(X_test, y_test)
 K.append(k)
 training.append(training_score)
 test.append(test_score)
 scores[k] = [training_score, test_score]
ax = sns.stripplot(training)
ax.set(xlabel='values of k', ylabel='Training Score')
```

```
plt.show()
ax = sns.stripplot(test)
ax.set(xlabel='values of k', ylabel='Test Score')
plt.show()
plt.scatter(K, training, color='k')
plt.scatter(K, test, color='g')
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
import seaborn as sns
csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
col=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width','Class']
iris=pd.read_csv(csv_url,names=col)
iris.head()

#6b)
a=iris['Class'].value_counts()
species=a.index
count=a.values
plt.bar(species,count,color='lightblue')
plt.xlabel('species')
plt.ylabel('count')
plt.title('Bar Graph')
plt.show()

#8b)
import numpy as np
data_ = np.random.randn(1000)
plt.hist(data_,bins = 40,color='gold')
plt.grid(True)
plt.xlabel('points')
plt.title('Histogram')
plt.show()

# Import libraries
from matplotlib import pyplot as plt
```

```
import numpy as np


# Creating dataset
cars = ['AUDI', 'BMW', 'FORD',
                'TESLA', 'JAGUAR', 'MERCEDES']

data = [23, 17, 35, 29, 12, 41]

# Creating plot
fig = plt.figure(figsize =(10, 7))
plt.pie(data, labels = cars)

# show plot
plt.show()

#8a)LINE PLOT
import numpy as np
x=np.linspace(0,20,30)
y=x**2
plt.plot(x,y)
plt.xlabel('x-values')
plt.ylabel('x^2-values')
plt.title('line plot')
plt.grid(True)
plt.show()

#7a)1)#SIMPLE LINEAR REGRESSION#
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
  n = np.size(x)
  m_x = np.mean(x)
  m_y = np.mean(y)
  SS_xy = np.sum(y*x) - n*m_y*m_x
  SS_xx = np.sum(x*x) - n*m_x*m_x
  b_1 = SS_xy / SS_xx
  b_0 = m_y - b_1*m_x
```

```
  return (b_0, b_1)
def plot_regression_line(x, y, b):
  plt.scatter(x, y, color = "m",marker = "o", s = 30)
  y_pred = b[0] + b[1]*x
  plt.plot(x, y_pred, color = "g")
  plt.xlabel('x')
  plt.ylabel('y')
  plt.show()
def main():
  x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
  y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
  b = estimate_coef(x, y)
  print("Estimated coefficients:\nb_0 = {} \\nb_1 = {}".format(b[0], b[1]))
  plot_regression_line(x, y, b)
if __name__ == "__main__":
  main()

#7b)#MULTIPLE LINEAR REGRESSION#
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
  n = np.size(x)
  m_x = np.mean(x)
  m_y = np.mean(y)
  SS_xy = np.sum(y*x) - n*m_y*m_x

  SS_xx = np.sum(x*x) - n*m_x*m_x
  b_1 = SS_xy / SS_xx
  b_0 = m_y - b_1*m_x
  return (b_0, b_1)
def plot_regression_line(x, y, b):
  plt.scatter(x, y, color = "m",
  marker = "o", s = 30)
  y_pred = b[0] + b[1]*x
  plt.plot(x, y_pred, color = "g")
  plt.xlabel('x')
  plt.ylabel('y')
  plt.show()
```

```
def main():
  x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
  y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
  b = estimate_coef(x, y)
  print("Estimated coefficients:\nb_0 = {} \\nb_1 = {}".format(b[0], b[1]))
  plot_regression_line(x, y, b)
if __name__ == "__main__":
  main()

#6a)
length_width = iris[['Petal_Length','Petal_Width','Sepal_Length','Sepal_Width']]
length_width.boxplot()
plt.xlabel('Flower measurements')
plt.ylabel('values')
plt.title("Iris dataset analysis")

#6c)SCATTER PLOT
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
df= pd.DataFrame(data= np.c_[iris['data'], iris['target']],columns= iris['feature_names'] + ['target'])
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', 0, 1)
X = df.iloc[0:100, [0, 2]].values
plt.scatter(X[:50, 0], X[:50, 1],color='blue', marker='o', label='Setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],color='green', marker='s', label='Versicolor')
plt.xlabel('Sepal length [cm]')
plt.ylabel('Petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```