VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



Machine Learning (23CS6PCMAL)

Submitted by

Rani Aishwarya H S (1BM22CS217)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Rani Aishwarya H S (1BM22CS217),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge	
Name: Ms. Saritha A N	Dr. Kavitha Sooda
Assistant Professor	Professor & HOD
Department of CSE, BMSCE	Department of CSE, BMSCE

Index

Sl. No	Date	Experiment Title	Pa ge
•			No.
1	21-2-202 5	Write a python program to import and export data using Pandas library functions	
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	
3	10-3-202	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	
4	17-3-202 5	Build Logistic Regression Model for a given dataset	
5	24-3-202 5	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	
6	7-4-2025	Build KNN Classification model for a given dataset	
7	21-4-202 5	Build Support vector machine model for a given dataset	
8	5-5-2025	Implement Random forest ensemble method on a given dataset	
9	5-5-2025	Implement Boosting ensemble method on a given dataset	
10	12-5-202 5	Build k-Means algorithm to cluster a set of data stored in a .CSV file	
11	12-5-202 5	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	

Github Link:

https://github.com/RaniAishwarya/Machine-learning/tree/main

Program 1

Write a python program to import and export data using Pandas library functions
Observation:

7	
	ML-LA13-01
7	ML-LAB-01 10/03/25 + 10 using . (SV
7	Housing CSV
1 10	Astronomy St. aparte visito 18 mm
	impost punclas as pd
7	as programme and all the second and a second a second and
	Jan 1 A A A A A A A A A A A A A A A A A A
	howing - dt = pd. read_csv ("/contend/howing &csv")
	CAPTELL MARKET TORAGE
	print ("Information of all columns; ")
(11)m	print ("Information of all columns:") print (howing - of info())
	CALL Challed CALL CALL
	print ("In Statistical information of numerical column") print (housing - of. clescaine())
	print (nousing - of. clescribe())
The hugh Blanch	
1 (little)	wint ("In Count of unique takets tox Ocean Proximity"
	print ("In Count of unique tabels for 'Ocean Proximity"
el alada	print (housing-df (ocean-proximity), value gise: print (f''(olumn' ocean proximity' no found in the dataframe. Available columns' knowing
	1 Ocan pronumity 10 com to 17 1
The state of	2 m no (Mousing at Colean pronuming), value
X20033303	else: (mints())
	print (+ Column ocean proximity no found
	the dataFrame. Dailable columns I housing
E	It columns to 015 tft 4"9
Allowed .	orint (" In Columns with missing values:")
440	on Mi In Cours of isnull () sum () =
(12)	missing value to housing of isnull (), sum ()
	print(missing values timissing values 707)
The second secon	County of the control
8	The state of the s
	The state of the succession of
Final . bas	The model to monthly as thought the
7	

> Handling Missing data Diabele and Adult income datasets. 23 impost panders as pd import numpy asnjo from skleaen impute import simple Imputer from stlearn preprocessing import MinMaxs cales Standardsealer from supy import steets diahety-file path = ' (confint pataset of Diahetg. C) v' diabetes of = pd read_csv(diabetes file path) numeric-cols = cliabeta-clf. select-dhypes (include:

np. number) columni

(alegorical-col) = difa helis-df. select-dhypes exclude

= np. number). columny numeric-cols = cliabeta-clf. select-d types (include= = np. number). column imputer_numeric = simple Imputer (statetagy = Imoan) diahetes-df[numeric_cols] = imputer-numeric_fit_
to ansporm (diahete-d) [numeric_cols]) imputer- calogorical = Simple Imputer (stater gy = most forward! diaheter df [categorical ds] - imputer ategorical iol)

T	
/	/
bi the Diabety & All Missing technique	ey
Actual Into mo dalaret	Ţ
1. which columns in the Data sol 1.1	
1. which columns in the Data set had missing u	alle
I richitate wen!	
Diabetes dataret	
= 10 there ill is to	
TO COMMON WILLIAM ON VIOLO CO TILLIA	2
Chol, TG, HOL, LDL, VLDL, BMJ	
Salar 19 10 10 20 Property of the salar sa	
we applied many imputation of missing dated	lues.
we appued mean imputation. The missing is	ellies
we applied mean imputation. The missing was replaced with the man of each respect	ré
handled aging most frequent value (mo	were
in talle any most frequent value (mo	de)
1 VII July 1011.	
1 A Clark Destroy	
Adult income Dabaret:	
- tolumes: In this clother set, musing va	lug -
whe depresented by? In the og calases	
to make he mining values clentifiable	UN -
to make the many values clentifiable	1
Thee this, we applied mode imput	atin !
be cate gorical columns to replace musis a	aley -
OTHER SECTIONS	
Marches & we the Marrion to	
Male II a spill of the spill of the spill of the	
	1

7.	which categorical & columns did you identify in the dataset. Non did you encode prem?
C Carllen	Diabeter: gender, Races, chas
	not encocline or latel encoding could be applied
	Adub income Patacet:
Name of the last o	occupation, relationship, taco, ser, native-county
	-> one-hat encoding was applied to lower
(0350 A	converted into use multiple sinary column converted into use multiple sinary column
gallar.	removing the first column from each categoria
- 0	variable set.
3. 1	and stadard isation? when would you use one over the other.
->	Min-More Sculing:
	data to a specific Irange usually 5/w 02

	- wing the primity
_	Xscaled - X-Xmin
	X-Xmin Xmax-Xmin
	when to use: Min-Max scaling is prefered when the dostared contains features with varying unit @ scales & you want to scale dos
	unit @ scales & you want
	unit @ scales & you want to scale them -
	Stander dizertion:
-	Stander dizertion: (D) z-Scorp normalization franctiony data to ha a mean of 0 & a standard deviation of 1. It is alculated using the formula:
1	data to ha a mean of o & a stemdard
	deviation of 1. 86 i) alculated using the
-	X scaled = X - h
e	
	M-man A 5-stundard de viation.
	standadisation is typically used whom
N'U	the data is assumed to follow a gaussian
+	Standardisation is hypically used whom the data is assumed to follow a gaussian distribution, especially whon working with algorithms that assume normally distributed algorithms hast assume normally distributed algorithms hast assume normally distributed
an	algorithms had arumo region.
	data segrenion, logismina,
	0 1 28 125
-	Section 13 125
-	

```
Code:
1) <u>Data Cleaning:</u>
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
# --- Diabetes Dataset ---
# Load the diabetes dataset
diabetes file path = '/content/Dataset of Diabetes .csv' # Replace with your actual file path
diabetes df = pd.read csv(diabetes file path)
# 1. Handling Missing Values:
# a. Identify numeric and categorical columns
numeric cols = diabetes df.select dtypes(include=np.number).columns
categorical cols = diabetes df.select dtypes(exclude=np.number).columns
# b. Impute missing values using the mean for numeric columns only
imputer numeric = SimpleImputer(strategy='mean')
diabetes df[numeric cols] = imputer numeric.fit transform(diabetes df[numeric cols])
# c. Impute missing values using the most frequent value for categorical columns
```

imputer categorical = SimpleImputer(strategy='most frequent')

```
diabetes df[categorical cols] = imputer categorical.fit transform(diabetes df[categorical cols])
# --- Adult Income Dataset ---
# Load the adult income dataset
adult file path = '/content/adult.csv' # Replace with your actual file path
adult df = pd.read csv(adult file path)
# 1. Handling Missing Values: Replace '?' with NaN and then impute
adult df.replace('?', np.nan, inplace=True)
imputer = SimpleImputer(strategy='most frequent') # Use most frequent for categorical features
# Fit and transform, but keep column names using columns=adult df.columns
adult df imputed = pd.DataFrame(imputer.fit transform(adult df), columns=adult df.columns)
# 2. Handling Categorical Data: One-hot encoding for adult income dataset
# Ensure categorical cols are present in adult df imputed
categorical cols = adult df imputed.select dtypes(include=['object']).columns.tolist()
adult df encoded = pd.get dummies(adult df imputed, columns=categorical cols, drop first=True)
# --- Handling Outliers (for both datasets) ---
```

```
# (Example using Z-score - adjust as needed)
# from scipy import stats
\# z = np.abs(stats.zscore(diabetes df imputed)) \# For diabetes dataset
# diabetes df no outliers = diabetes df imputed[(z < 3).all(axis=1)]
# z = np.abs(stats.zscore(adult df encoded.select dtypes(include=np.number))) # For adult dataset
# adult df no outliers = adult df encoded[(z < 3).all(axis=1)]
# Print the preprocessed dataframes (example)
print("Diabetes data with imputed missing values:\n", diabetes df.head())
print("\nAdult income data with imputed values and encoding:\n", adult df encoded.head())
2) Data Transformations:
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
# --- Diabetes Dataset ---
# Load the diabetes dataset
diabetes file path = '/content/Dataset of Diabetes .csv'
diabetes df = pd.read csv(diabetes file path)
# 1. Handling Missing Values:
```

```
# a. Identify numeric and categorical columns
numeric cols = diabetes df.select dtypes(include=np.number).columns
categorical cols = diabetes df.select dtypes(exclude=np.number).columns
# b. Impute missing values using the mean for numeric columns only
imputer numeric = SimpleImputer(strategy='mean')
diabetes df[numeric cols] = imputer numeric.fit transform(diabetes df[numeric cols])
# c. Impute missing values using the most frequent value for categorical columns
imputer categorical = SimpleImputer(strategy='most frequent')
diabetes df[categorical cols] = imputer categorical.fit transform(diabetes df[categorical cols])
# 2. Data Transformations:
# a. Min-Max Scaling
scaler minmax = MinMaxScaler()
diabetes df[numeric cols] = scaler minmax.fit transform(diabetes df[numeric cols])
# b. Standard Scaling (create a separate copy)
diabetes df std = diabetes df.copy()
scaler std = StandardScaler()
diabetes df std[numeric cols] = scaler std.fit transform(diabetes df std[numeric cols])
```

```
# --- Adult Income Dataset ---
# Load the adult income dataset
adult file path = '/content/adult.csv'
adult df = pd.read csv(adult file path)
#1. Handling Missing Values: Replace '?' with NaN and then impute
adult df.replace('?', np.nan, inplace=True)
imputer = SimpleImputer(strategy='most frequent')
adult df imputed = pd.DataFrame(imputer.fit transform(adult df), columns=adult df.columns)
# 2. Handling Categorical Data: One-hot encoding
categorical cols = adult df imputed.select dtypes(include=['object']).columns.tolist()
adult df encoded = pd.get dummies(adult df imputed, columns=categorical cols, drop first=True)
# ... (previous code)
# 3. Data Transformations:
# Get numeric columns after encoding.
# Check if any numeric columns exist to avoid error
numeric cols adult = adult df encoded.select dtypes(include=np.number).columns
```

```
# Initialize adult df encoded std before the if statement
# to an empty DataFrame if no numeric columns are found
adult df encoded std = pd.DataFrame()
if len(numeric cols adult) > 0:
  # a. Min-Max Scaling
  # ... (code for Min-Max scaling remains the same) ...
  # b. Standard Scaling (create a separate copy)
  adult df encoded std = adult df encoded.copy()
  scaler_std_adult = StandardScaler()
  adult df encoded std[numeric cols adult] =
scaler std adult.fit transform(adult df encoded std[numeric cols adult])
else:
  print("No numeric columns found for scaling in adult income dataset.")
# ... (rest of the code)
# Print the preprocessed dataframes (examples)
print("Diabetes data with Min-Max scaling:\n", diabetes df.head())
print("\nDiabetes data with Standard scaling:\n", diabetes df std.head())
print("\nAdult income data with Min-Max scaling:\n", adult df encoded.head())
print("\nAdult income data with Standard scaling:\n", adult df encoded std.head())
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Observation:

(AB-0)
import numer as noticed the
import matelotlib, pupilot as alt
import pandas as od
import numey as no import matphollib pyphol as plt import pandas as pd import math
import copy
-> dataset = pd. sead_csv(' content weather.csv')
x = dataset . iloc I:,:] - values
(==: 11)
output: 1=(a) oto b) m) = 15;
array ([[Sunny', Hot', 'High', 'weak', 'No'],
['Sunny' 'Hot', High', Stong!, No'],
a may ([['Sunny', Hot', 'High', 'weak', 'No'], ['Sunny', 'Hot', 'High', 'Shong', 'No'], ['Overcust', 'Hot', 'High', 'weak', 'Yes'],
[Rain', Mild, fligh, Weak, yes? [Rain', 'Cool', Normal', Weak', yes] ['Rain', 'Cool'; Normal', Strong', No'],
['Rain', 'Cool', Noimal', 'weak', yes]
['Rain', 'Gool', Normal', Strong', No'],
['Overcast', 'cool', Normal', Shong', 'Nes']
['Sunny', 'Mild', High', weak!, 'No),
T' Sunny, '(oo!, 'Normal', Weak', 'Yes'),
['Rain', Mild', Normal, weak! Vest
['Overcast, 'Coo', Normal, Strong, 'Nes'], ['Sunny', 'Mild', 'High', 'Weak!, 'Ves'], ['Rain', 'Mild', 'Normal', 'Weak!, 'Ves'], ['Sunny', 'Mild', 'Normal', 'Weak!, 'Ves'], ['Sunny', 'Mild', 'Normal', 'Weat Strong', 'Ye'], ['Overcast! Wild', 'Kigh', 45 horg', 'Ye'], ['Overcast!, 'Hol', 'Normal', 'Weak!, 'Yej'],
(10 verasti Mila, High, 48 horge, Yel),
1:00 except, 'Hod', 'No ima', weap, yes,
('Rain', 'Mild', 1 High', 'Strong', 'No'], chype=object)
_ attribute= ['Owlook,' Temp?, 'Heimidity; 'Wird']
- autinute Court, Temp, Francury, vone
solution enders and

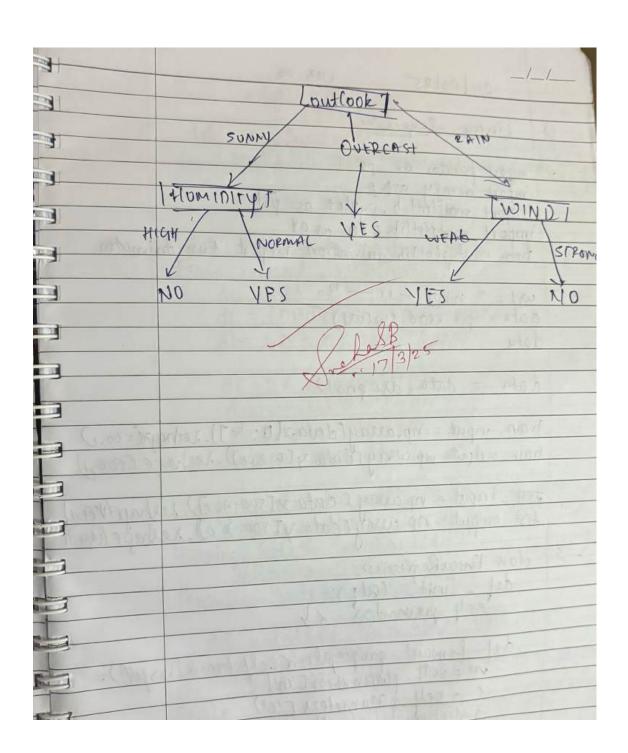
/		-
	//_	-
	alone 10 1 colind):	8
->	class Node (object):	60
	det_init (self): self:value = None	-
	sey de de diam han	1
	Self devision= None Self-child = None.	6
	Seq. Criticy prove.	-07
-5	def find Entropy (data, rows): yes=0 no=0 ans=-1	19
	105 = Q 11 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1
	no =0	
	ans=-1	100
	idx = Cen(data[a)/-1	
	to to was in nows:	2
	[Sun 1 101 1 100] (Que 1 100]	6
	Collow anistall between	-
	if data[i][ida]== Voi?:	8
	Yes=verting and	
	0100	-
	1 odd odno znoti loo' isomo' I	2
	12 yes (westing)	
	1 = yes (yestno) 1 local manus.	1
	201/4-lino(Cylestno), blink	0
1001	ip x1 = 0 and 41 =00	
1.5	entropy 1* (x* moth, tog 2 (s) + yx moth	8
	5/1140 W11 () A 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	6
=9000	y = lno(y lestho) y = 0 and y' = 0 entrop $y = -1 \times C \times \text{modh. log } 2(x) + y \times \text{modh.}$ (0 if x = 0)	1
4	ans	8
60/2-	101 - il 11.6 - = 13 lonlen 12 Bud atta	1
SON E	- a the public condition of the -	8
	ahs=0	-
	return entropy, ans	-

-		
	-) del finda r	-/_/_
	-) del findMarGain(data, , marGain = 0	Dus Columbia
-	margain = 0	- (Bammy)
7		
	entropy, ans = find	Entropy (date
	if Enhaly co:	(ASS) (- 40 WS) -
	entropy, ans = find if entropy o: return margo for j in olumns:	in refidy and
	for j in clums:	The tops
100	my did - 11	DELLE MELLER
	i = 100	
	to i in now	
	key- da	tucillidad
	Durate Millian IX. vous column	of in mydio:
	my dist	[Keyl-h
	0/98:	tkoy)= HOR
	midia	[Key] = mydict [tay
	gain = entr	not be seen
	gain entre	100 11
=	to tey in	mydict:
	1,01=0	9212
	yes=0 no=0	Dor -
	1	INITION!
=_	haise of the same	atalication == key: datalication == key: yes = yes -1 se:
	Not leave of the	doutal BIE-17- = yes!
		Aren = Ires +1
	Dell 21 57 6	10
	11013113	no= ka+1
3	. Dispers of box	1001 1600
		(100)
3	y= ho!	(yes-110)
	1 = 3 [126 3 1 0]	
	7)	

i x1=0 and y1=0.
gain += Inydid [key) + (x+ may) log 20x7+ y* mooh. shog 2 (x) Dhy
(og 20x7+ yr mboh. bog 2 (2) Dyy
Lear stebles distill care parties
i gain; morgain:
max Gain = gaix
Sel Colored and
Refian max Gain, retidx, ans
de druid Treo (daly, rows, column)
maxGain de au - Gidharles cus
root = No del) . find Mar Gain (x, vows, aglimmy)
anotichilds = [1]
anotichilds=[190]
ij margain = 0:
if ans == 1. your value = (yes)
'you value = 'yes'
voot value = 'No'
gettum root
good value = attentate[idr]
mydict = 1
tot i'in rows:
koy=data TiJ sior)
if they not in my did.
else: mydritteyj=j
else.
imydiatkoy Jt=1
V

newcolumns = copy decopapy columns) for key in my did ne w sons = [July fue temp-buildtree data, newrows, newcolung temp. clusion = tely soft childs appenditency traverse (root): and (rook devision) mint (noot value) The contract of third of this contract of the n- Len (200+ childy).

	10
-> calculate()	
2 Million San considerate	91
outlook	-
outlook?	2
funny Humidi hy High	100
Humitch hy	
High	1
Land Do so to land a const	-
Te berned	
Yas	1
1 Over east	
	1
Rain (hot) 12 min to 12	-
wind sinch done of	
Rain (toos) second file with the work with the control to the cont	15
Mal	6
	15
Strong	
Poly	日
-2 from a contract of a	100
from Trach 102 paport Digraph	132
from trython. display involor I mago	E
dot= nigrouphy	EDE
from graphinz import Digraph from Ipython. Clippay import I mage dot = higraph 1) dot no de l'image, label = 'image='(content) tonnin Ditinal (1). jpg.)	100
tennintthal (12. ipa)	
	-5
dos 10 d = (000)	
and formation of the state of t	E
Lender ('mage graph' siew = False)	-
dof. format = 'png' dof. sendu ('image graph', siew = False) diglay [Imagel' image graph graph graph)	Bibs
	-
	-



```
Code:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy
dataset = pd.read csv('D:\Datasets\Tennis.csv')
X = dataset.iloc[:,:].values
X
attribute = ['Outlook', 'Temp', 'Humidity', 'Wind']
def findEntropy(data, rows):
    yes=0
    no=0
    ans=-1
    idx=len(data[0])-1
    entropy=0
    for i in rows:
        if data[i][idx]=='Yes':
            yes=yes+1
        else:
            no=no+1
    x=yes/(yes+no)
    y=no/(yes+no)
    if x!=0 and y!=0:
        entropy= -1*(x*math.log2(x)+y*math.log2(y))
    if x==1:
        ans = 1
    if y==1:
        ans = 0
    return entropy, ans
def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
```

```
if entropy == 0:
        """if ans == 1:
            print("Yes")
        else:
            print("No")"""
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx = j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
                mydict[key] = 1
            else:
                mydict[key] = mydict[key] + 1
        gain = entropy
        # print(mydict)
        for key in mydict:
            yes = 0
            no = 0
            for k in rows:
                if data[k][j] == key:
                     if data[k][-1] == 'Yes':
                         yes = yes + 1
                    else:
                         no = no + 1
            # print(yes, no)
            x = yes/(yes+no)
            y = no/(yes+no)
            # print(x, y)
            if x != 0 and y != 0:
              gain += (mydict[key] * (x*math.log2(x) +
y*math.log2(y)))/14
        # print(gain)
        if gain > maxGain:
            # print("hello")
            maxGain = gain
            retidx = j
    return maxGain, retidx, ans
```

```
def buildTree(data, rows, columns):
    maxGain, idx, ans = findMaxGain(X, rows, columns)
    root = Node()
    root.childs = []
    # print(maxGain)
    if maxGain == 0:
        if ans == 1:
            root.value = 'Yes'
        else:
            root.value = 'No'
        return root
    root.value = attribute[idx]
    mydict = {}
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1
        else:
            mydict[key] += 1
    newcolumns = copy.deepcopy(columns)
    newcolumns.remove(idx)
    for key in mydict:
        newrows = []
        for i in rows:
            if data[i][idx] == key:
                newrows.append(i)
        # print(newrows)
        temp = buildTree(data, newrows, newcolumns)
        temp.decision = key
        root.childs.append(temp)
    return root
def traverse(root):
    print(root.decision)
    print(root.value)
    n = len(root.childs)
```

```
if n > 0:
        for i in range(0, n):
            traverse(root.childs[i])
def calculate():
    rows = [i for i in range(0, 14)]
    columns = [i for i in range(0, 4)]
    root = buildTree(X, rows, columns)
    root.decision = 'Start'
    traverse(root)
calculate()
from graphviz import Digraph
from IPython.display import Image
dot = Digraph()
dot.node('image', label='', image='/content/TennisDTFinal
(1).jpg')
# Replace /path/to/your/image.jpg with the actual path to your
image
dot.format = 'png'
dot.render('image graph', view=False)
# Render to a file named 'image graph.png' and display the image
display(Image('image graph.png'))
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset Observation:

24/03/25 (ab-05	
24 03125	
U linear Regression:	
10000	
import pandar as po	
Import memby ash p Import madplotlib. pyplot as plt	
1)))DOIT mathematically all all	
from malphotis onimation import tun (Animation	
-7 w1 = " ntps - 11 - 11	
data = pd. read-csv(ur)	1
data	
data - data de a a - 1)	1
data = data. deo gnal)	k 1
train_input = np. array(data x(0; 57). reshape(x60,2)	1
train_input = np. array(data. y[0:500]). reshaye(500,) train_ordput= np. array(data. y[0:500]). reshaye (500))	
	2
test_input = np. away (dota. x [500; 70 a]), reshape (199),	
test output = np. cusay (data. y [500; 70 d)]. reshape (igg)	
	2
class imeaikegresits;	
ae - ini+ - (sel):	2
-> clas linearRegresion: delinit(self): self.pexametos - Ly	1
del formard propogationeself, train in put):	E
m = self = pasametras timis	
predictions = np. multiply(m, toin-input) +(E
gradictions = no multiply me to in just) +(

	//_
	return prodictions.
I.	
	det cost function (self, prediction, train output) cost = np. man(train-output - prediction)
1	and function sey prediction, train output
1	cosi - np. mani cadin-output - prediction
100	KA) X
W 3	-xerian cost
L.	setuen cost
	del backwood-Propohaballall has
	train putout productions
Y	det backwood-propobation/ self, rocin-input, train output, predictions):
7	dl = (predict)
11-	Alexantions - Frain - adjud)
Value of the same	om = 3 + np. meaninp. multiply (buin or house
200	daivative) = 1 1 dj = (predictions - frain-adout) dm = 3 x np. meaninp. multiply (pain-output) d(= 2 x np. meaninp. multiply (pain-output)
	a(=5# np.meanl.dl)
0	d(= 5 * np.meanl.df) desivatives ['dm'] = dm
	derivatives l'dr'] = dc
	Seturn derivatives
	PAR GUENNICE STATE OF THE STATE
	Apr color - co dos Codo dos to
	ately and moreus key, overview es, basning ale
	lenger parameters ('m') - sett parameters ('m') -
-	couning - rate & desiratives ['dm']
	self personaters T'c' = pell might efent c' -
	des update gasa motors rely desiratives, (coaring rate) self parameters ['m'] = celf parameters ['m'] - learning - rate + classratives ['d'] reliparameters ['c'] = pelf parameters ['c'] - learning - rate + clasivatives ('d')
	The same statement and
	All suscell less than the
	rainisty, pain-infry, train-outpid,
	clarking rate, iteus!
	learning rate, iters: Self parameters timi)- mg. random uni form
	(01)4-1
	(0,1) + - 1
	Selfparameters ('c') Ap. random. untology

interval - 200, blit - True) g, update, frames = itel, ani. sque s'linear_regression. A git Plt. xlabel ('Inpud')
plt. ylabel ('Output')
plt. title ('Linear Regression) 51+ legend 1) pt. show O Return self parameters, Self-loss output. fteration Loss = 1223 85901767012 fleration 2 Fleralin 2 Loss = 10 40095 2170 38 78 51)
Fleralin 3, Loss - 8, 132 134 34 19 50 354

	//_
OU, I	Multiple linear Regression:
	import numby as no
d mps	import metplothin pyplot as plt.
the state of the s	mapor maporas, paper os pa
	dala = L
J. 3	"Feature": [1,2,3,4,5,6,7,6,9,10]
	1 - lature . [2,3,5,7,11,13,14,14,23,29]
	Featurs": [3,6,9, 12, 15, 18, 21, 24, 27, 30], "Tougo": [5,9,15,21,31,41,53,60,80,96]
13	7
	df = pd. Data Frame (data)
-	
	Y = of deop(columns=t"target")). value Y = of t " target") value, response(-1, I)
9	Y- aft" Target" J. values. xeghaje(-1, I)
- III	X = NO. 1. 202 22 1
	x=np. hstack (1 np.onuf(x. shape [6], 1), x))
6 3	5 (S18) com (1) 13 (6)
	bela = np. linalg colve(x.t ax +0.01* np. iclansity (X. shape[1]), x. tay)
3-1	(X. share(1)) x. taxe)
	a analy of the party and and
	Y-Pred = 1(2) beta
	(a) all (a) Margain + of
	mse-nponean(1y-y-pred) **2)
	total-variance = np. sum (y- pp. mean(y)) + 2)
	explained - vaniance = m smally - prod-np monty
	(X)

```
print(" Model Cofficients", befali:]. flatten())

print("Mean Swared tror:", mse)

print("R - S viace of sace:", 1)

pld. Satters ( y y - psed, color="blue")

pld. pld ( y y , color="red", linestyle="2")

plt-x label ("tretual values")

plt. fite ("Alheel us predicted values")

plt. 8 how ()
```

Code:

```
#multiple linear regression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Sample dataset
data = {
    "Feature1": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Feature2": [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
    "Feature3": [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
    "Target": [5, 9, 15, 22, 31, 41, 53, 66, 80, 96]
}
df = pd.DataFrame(data)
# Split dataset into features (X) and target variable (y)
X = df.drop(columns=["Target"]).values
y = df["Target"].values.reshape(-1, 1)
# Add intercept column (bias term)
X = np.hstack((np.ones((X.shape[0], 1)), X))
# Compute the coefficients using the Normal Equation
beta = np.linalg.solve(X.T @ X + 0.01 * np.identity(X.shape[1]),
X.T @ y)
```

```
# Make predictions
y pred = X @ beta
# Evaluate the model
mse = np.mean((y - y pred) ** 2)
total variance = np.sum((y - np.mean(y)) ** 2)
explained variance = np.sum((y pred - np.mean(y)) ** 2)
r2 = explained variance / total variance
# Display results
print("Model Coefficients:", beta[1:].flatten())
print("Intercept:", beta[0][0])
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
# Plot actual vs predicted values
plt.scatter(y, y_pred, color='blue')
plt.plot(y, y, color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.show()
  Linear Regression:
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.axes as ax
from matplotlib.animation import FuncAnimation
url = 'https://media.geeksforgeeks.org/wp-content/uploads/
20240320114716/data for lr.csv'
data = pd.read csv(url)
data
# Drop the missing values
```

```
data = data.dropna()
# training dataset and labels
train input = np.array(data.x[0:500]).reshape(500, 1)
train output = np.array(data.y[0:500]).reshape(500, 1)
# valid dataset and labels
test input = np.array(data.x[500:700]).reshape(199, 1)
test output = np.array(data.y[500:700]).reshape(199, 1)
class LinearRegression:
    def __init__(self):
        self.parameters = {}
   def forward propagation(self, train input):
        m = self.parameters['m']
        c = self.parameters['c']
        predictions = np.multiply(m, train input) + c
        return predictions
   def cost function(self, predictions, train output):
        cost = np.mean((train output - predictions) ** 2)
        return cost
   def backward propagation(self, train input, train output,
predictions):
        derivatives = {}
        df = (predictions-train output)
        # dm= 2/n * mean of (predictions-actual) * input
        dm = 2 * np.mean(np.multiply(train input, df))
        \# dc = 2/n * mean of (predictions-actual)
        dc = 2 * np.mean(df)
        derivatives['dm'] = dm
        derivatives['dc'] = dc
        return derivatives
    def update_parameters(self, derivatives, learning rate):
        self.parameters['m'] = self.parameters['m'] -
learning rate * derivatives['dm']
```

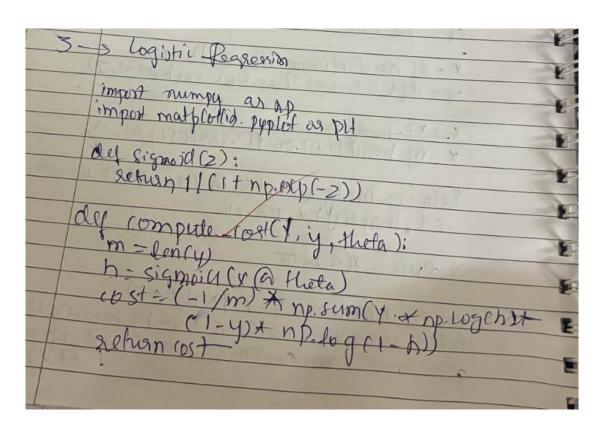
```
self.parameters['c'] = self.parameters['c'] -
learning rate * derivatives['dc']
   def train(self, train input, train output, learning rate,
iters):
        # Initialize random parameters
        self.parameters['m'] = np.random.uniform(0, 1) * -1
        self.parameters['c'] = np.random.uniform(0, 1) * -1
        # Initialize loss
        self.loss = []
        # Initialize figure and axis for animation
        fig, ax = plt.subplots()
        x vals = np.linspace(min(train_input), max(train_input),
100)
        line, = ax.plot(x vals, self.parameters['m'] * x vals +
                        self.parameters['c'], color='red',
label='Regression Line')
        ax.scatter(train input, train output, marker='o',
                color='green', label='Training Data')
        # Set y-axis limits to exclude negative values
        ax.set ylim(0, max(train output) + 1)
        def update(frame):
            # Forward propagation
            predictions = self.forward propagation(train input)
            # Cost function
            cost = self.cost function(predictions, train output)
            # Back propagation
            derivatives = self.backward propagation(
                train input, train output, predictions)
            # Update parameters
            self.update parameters(derivatives, learning rate)
            # Update the regression line
            line.set ydata(self.parameters['m']
```

```
* x vals + self.parameters['c'])
            # Append loss and print
            self.loss.append(cost)
            print("Iteration = {}, Loss = {}".format(frame + 1,
cost))
            return line,
        # Create animation
        ani = FuncAnimation(fig, update, frames=iters,
interval=200, blit=True)
        # Save the animation as a video file (e.g., MP4)
        ani.save('linear regression A.gif', writer='ffmpeg')
        plt.xlabel('Input')
        plt.ylabel('Output')
        plt.title('Linear Regression')
        plt.legend()
       plt.show()
       return self.parameters, self.loss
#Example usage
from matplotlib.animation import FuncAnimation #import
FuncAnimation
linear reg = LinearRegression()
parameters, loss = linear reg.train(train input, train output,
0.0001, 20)
```

Program 4

Build Logistic Regression Model for a given dataset

Observation:



des gradient dexont (x, y, theta, alpha, m - Cency cost history = [7 for in sange (iteration):

gradient = (1/m) + XI (a (signor) (X

Thera -= cipha + gradient

cost history cippend (rought - cost (V),

y theran) Return theta, cost history predict (X1 theta): setien (sigmoid (V @ thota) >= 05). astype (no) ng random. seed 42) x= np. remolom. rand(100,1) +10 y= (x>5). astype (fr.t). ravel() X-b= hp.c- [np.ones(1), shape(1)) & There = hp.zeros(x-b_shape(1)) & alpha = 0.) ifea 400 = 1000 theta, west history gradient descent (y.b, V) 4 prod=predictxb, theto

1_1_ print (Au Accusator): 2 accusacy: 26(0) pt. scatter(x, y, plt-scates (x, y, good, color='xed', moustas='x'
leasel=1 predicted labels') pl. xladel (* cature v")

pl. yladel (* tours 10 or 1)"

pl. logerd ()

pl. show() Accusacy: 0.97 4

Code:

```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
            return 1 / (1 + np.exp(-z))
def compute cost(X, y, theta):
            m = len(y)
            h = sigmoid(X @ theta)
            cost = (-1/m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - y) + np.log(1 - 
h))
          return cost
def gradient descent(X, y, theta, alpha, iterations):
            m = len(y)
 cost history = []
          for in range(iterations):
                          gradient = (1/m) * X.T @ (sigmoid(X @ theta) - y)
                          theta -= alpha * gradient
                          cost history.append(compute cost(X, y, theta))
return theta, cost history
def predict(X, theta):
            return (sigmoid(X @ theta) \geq 0.5).astype(int)
# Generate synthetic binary classification data
np.random.seed(42)
X = np.random.rand(100, 1) * 10 # Feature values between 0 and
10
y = (X > 5).astype(int).ravel() # Label: 1 if X > 5, else 0
# Add intercept term
X b = np.c [np.ones((X.shape[0], 1)), X]
# Initialize parameters
theta = np.zeros(X b.shape[1])
alpha = 0.1
```

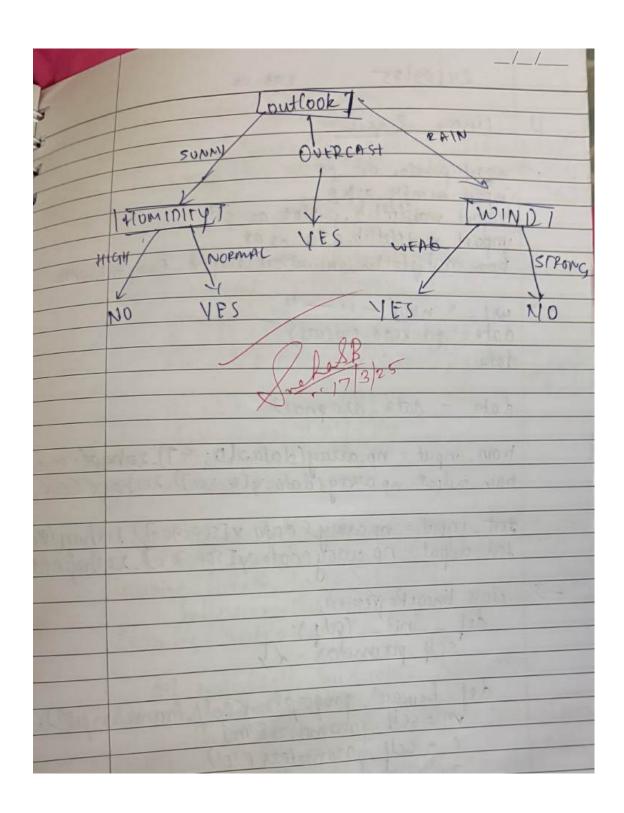
```
iterations = 1000
# Train logistic regression using gradient descent
theta, cost history = gradient descent(X b, y, theta, alpha,
iterations)
# Make predictions
y pred = predict(X b, theta)
# Compute accuracy
accuracy = np.mean(y pred == y)
print(f"Accuracy: {accuracy:.2f}")
# Plot the decision boundary
plt.scatter(X, y, color='blue', label='Actual Data')
plt.scatter(X, y_pred, color='red', marker='x', label='Predicted
Labels')
plt.xlabel("Feature X")
plt.ylabel("Class (0 or 1)")
plt.legend()
plt.title("Logistic Regression Model (Without Scikit-learn)")
plt.show()
```

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

gain + invided (key) + Cxt morth gain + invided (key) + Cxt morth (og 24x74 yr morth bog 2 ry) Dyly	
i v!= 0 and totkey) + (x morth	
gain to now moon sog 2 (y) Why	
(log Mill	-3
Carried Carried Control of the Contr	
nar Gain: gain: mar Gain: gain get idx: J getidx ans	
max yain = gain	jell- S
get 101 = 1 so Lide and	
getien max yain, rething	-
and the state of t	
des build Treo (dals, sors, column) maxquin, Mr, ans = find Maxquin (X, vows, wolumn root = No del)	NI D
maxGain, idr, ans = tind MaxGain (x, votos, Goum	my -
root = Poder)	
200+ childs = []	
The state of the s	
if margain == 0: if ans == 1: your value = 'yes' else'.	
il and == 1:	2.0
to al malua = (Max)	
100°	in later to
voot value = 'No'	
100. terre - 100	Part Part
gettun root	
good value = attentiatel idn?	No. of Lot
setum root soot value = attentiate[idn] mydict = L!	-
tot ich rows:	Part III
toy-data Tillion	Education of the last of the l
I pay not in number	
mardick Charge	-
Class the read of	
else: mydict [kay] t= 1	
maid Leay It=1	
	N.
	1

menicolumns = copy deco propri reglimno
new columns - copy dece papy columns)
iax ley in my all
me w 80 ms = Ly
for i in rows: for i in rows: dataci)[idx] = tey
if balactiff 10x1 = tey
rewrons append & temp-buildtreet data, newrow, reweden
temp-buildTree (data, newrow, newrow, newsolu
nost childs, appenditense)
seturn soot (hilds, appenditence)
toles traverse (root).
punt (root clevision)
print (root. cleviston)
12/
n= Cen (200+ childy).
Jor i in range (0,n): fourisse (root, childs [i]) - cles (alculate():
lar i in range (0,n)
for (13 co (root, childs [i])
The state of the s
ala colo letaco: Addoscaid = tan
columns = [i for i in range (0,1m)] columns = [i for i in range (0, 1m)] root = puilotre (x, rows, Lolumn) root -cleusion = Steep) traverse (root)
now for for in vanage (0,4)7
column = (1 tro, (x vous 10 lumn)
root = pulling
7004-Cleusson
traverse from
Man Anaga to and I Deport to solide

W STAND OF THE STA	1
-> calculate()	
output: Start	1
Suny hy	113
High	
High Normal	
Normal.	13
(d)	
- Everiar	
Pain (1004) wasnot 12/2	4
Ram	
wind and to dead the	AND
Werken to and triber	
(a)	-13
Shong	73
201	
The state of the s	
-> from graph vi2 Import Digraph	
from Trython display import I mage	3
dot= nigraph 1)	
plot no del'image, label = 1 image= (content)	
from graph wiz Import Digraph from Ipython display import I mage dot = higraph 1) lot no de'l' image! label = '' image= '(content) tennint that [12. jpg')	-5
100 t - por d. 10 10 10 10 10 10 10 10 10 10 10 10 10	
dot format = (png)	
dot sender l'image a appl views Fallon)	
dislay [Inasel 1 image and a see	
dot sender l'image graph, view = False) diglay (Irasel'imano graph graph graph)	
	No.
	L



```
Code:
def buildTree(data, rows, columns):
  maxGain, idx, ans = findMaxGain(X, rows, columns)
  root = Node()
  root.childs = []
  # print(maxGain)
  if maxGain == 0:
    if ans == 1:
       root.value = 'Yes'
    else:
       root.value = 'No'
    return root
  root.value = attribute[idx]
  mydict = \{\}
  for i in rows:
    key = data[i][idx]
    if key not in mydict:
       mydict[key] = 1
    else:
       mydict[key] += 1
```

```
newcolumns = copy.deepcopy(columns)
  newcolumns.remove(idx)
  for key in mydict:
    newrows = []
    for i in rows:
       if data[i][idx] == key:
         newrows.append(i)
    # print(newrows)
    temp = buildTree(data, newrows, newcolumns)
    temp.decision = key
    root.childs.append(temp)
  return root
def traverse(root):
  print(root.decision)
  print(root.value)
  n = len(root.childs)
  if n > 0:
    for i in range(0, n):
       traverse(root.childs[I])
```

```
def calculate():
  rows = [i \text{ for } i \text{ in range}(0, 14)]
  columns = [i \text{ for } i \text{ in range}(0, 4)]
  root = buildTree(X, rows, columns)
  root.decision = 'Start'
  traverse(root)
calculate()
from graphviz import Digraph
from IPython.display import Image
dot = Digraph()
dot.node('image', label=", image='/content/TennisDTFinal (1).jpg')
# Replace /path/to/your/image.jpg with the actual path to your image
dot.format = 'png'
dot.render('image graph', view=False)
# Render to a file named 'image_graph.png' and display the image
display(Image('image_graph.png'))
```

Build KNN Classification model for a given dataset

THE THE PARTY OF T	
	07/04/25
	07/04/25 LAB-04 -/-/
	K Nearest Number
	inner to 101
-	import rumpy as no
	emport maplotlib. Pyplot as plt
	import numpy as no import matphothis pyphot as plt from collections import counter
=	def endidean-distance (21, 12):
=	Return np. syrt (np. sum((x1-x2) **2))
	class knn:
	definit-(self) = 3):
	Self. E=t
	CAMPAGE MARKET AND
	def fit(self, x,y): relf. x-train=np.array(x) relf. y-train=np.array(y)
	self. X-train= nparay(x)
	relf. y-train = np. array (y)
213 243	Des Colonians visit de la colonia de la colo
3 10 10 10 10 10 10 10 10 10 10 10 10 10	def predict (self, x): return t self-predict (x) for x in X?
Blowd Basis	return [self. predict (x) for x in X ?
3	Sales (1) (1) (1) (1) (1)
	del-predict (self, w):
- BULL NO	distances = Teachiclean_distance(x,x,train) for
	K-indicus=np. argsort (distances) [: Self. K]
1 1838 1	K-noarest-lately- [self y fainti) for in Endice
	most common - (or west) in Mode
	most common = (ounter of nearest (asels).
	return most common Co ICo]

	//_	E
	F-0-567	THE
	dy score (self, x, y). predictions = self. predict(x) Return np. mean(predictions == y)	
	predictions = self. predictions == V)	
	Return np. mean (predictions ==)	
	$x_{\text{fain}} = \text{np.areay}[[t_{1,2}], [2,3], [3,1], [6,5], [7,7], [8,6]])$	
	X-fain = np. array (1 1,23, 12,3)	
	[[[[[[[[[[[[[[[[[[[
	1 frain = 117. aux (1507575)	-
	X fest = nv. assum (co	
	1000 = \$NN(E=3)	
	ton=tNN(==3) Fnn-fit (X-toqin, y-train) prediction = Enn. predic(X-test)	
	prediction = tnn. predictx-test)	
	plt. figure (figsig=(8)6)	Ē
	for in range (len(X-train)): plf. scatter (X-trainti) [] (1], X-train [i] [1], color='red' is y-trainti] == 0 else 'di	
	plf. scatter (X - trainti] [0], X train[i][1], color='red' if y trainti] == oelse 'de [abcl = f"class of trainti] !" if fulle Ly trainti] !" not in plt gra(). get leagued have lately ()[1] else"!)	w I
	label = foclass of focintis) "; I fulle	01
	(mintil) not input gra (). get leagued har	olter
	inteln ()[1] else"")	9
	Autor ()	FI
37 2	put scatter (x-test to][0], x text [0][1], color tyre, s= 200, marker = 'X', Label = 'Text point')	an'
and de	S= 200, marker = 1x label = Test point	=
T 34. 1	Lead of a contract of the color	7 3
ibit-1	plt. xatler ex-tost (0) (0) X tost (8) (5) (000	mil
16960	plt xatler (x-test(0)[0] x-fest (0)[v], color 1 gran 1, s-200, marker "x", label 2 Test p plt xlatel ("te autire 1") plt . Y label ("Tecarthre 2")	01
	p/L.xlatel ("te autile 1")	Ü
	by. Y labell recipine 9"	
4714		

E	
3	_/_/_
	plt. legend() plt. grid (True) plt. show()
3	plt. show()
9 40	

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
# Euclidean distance function
def euclidean distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))
# KNN Class
class KNN:
    def init (self, k=3):
        self.k = k
    def fit(self, X, y):
        self.X train = np.array(X)
        self.y train = np.array(y)
    def predict(self, X):
        return [self. predict(x) for x in X]
    def predict(self, x):
        distances = [euclidean distance(x, x train) for x train
in self.X train]
        k indices = np.argsort(distances)[:self.k]
        k nearest labels = [self.y train[i] for i in k indices]
        most common = Counter(k nearest labels).most common(1)
        return most common[0][0]
    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)
# Sample data
X_{train} = np.array([[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 1])
611)
y_{train} = np.array([0, 0, 0, 1, 1, 1])
X \text{ test} = \text{np.array}([[5, 5]])
# Instantiate and fit KNN
```

```
knn = KNN(k=3)
knn.fit(X_train, y train)
prediction = knn.predict(X test)
# Plotting
plt.figure(figsize=(8, 6))
# Training data
for i in range(len(X train)):
    plt.scatter(X train[i][0], X train[i][1],
                color='red' if y_train[i] == 0 else 'blue',
                label=f"Class {y train[i]}" if f"Class
{y_train[i]}" not in plt.gca().get_legend handles labels()[1]
else "")
# Test point
plt.scatter(X_test[0][0], X_test[0][1], color='green', s=200,
marker='X', label='Test Point')
plt.title(f"KNN Classification (Predicted Class:
{prediction[0]})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

Build Support vector machine model for a given dataset

	12 y - Clark Clark
>	SVM
	5 (10)
	THE VILLEY TO THE
	import number 2
	impost malay with
	import numpy as no import matp'lokih. pyplot as plt
	Class SYM:
	del : + Coll !
	def_init_ (self, learning_rate=0.001,
	lambda param = 0.01, n-itel = 1000):
	self 1x = learning - rate
712	self. lambdaparam = labnbda-param
In the Times	neg anglagy ram > campla - parym
	self. n_i ters - n_i ters
	self. w = None
100	rell b - None
in a	der fit(set(x,y):
	de finsey, x, y/:
	y- np. whole (y <=0-1,1)
	n-samples, n-features = X. shape
TI .	1) - Samp -) respectively - respectively
100 11	self. w -np. zeros (n- lanuls)
	self. is -np zeros (n-features)
10	for - in range self. n. ites):
	Por - Intravious suf- notatory).
	for idx, x-i in enumerate (x):
	for idx, x-i in enumerate(x): condition = y(idx)*(np.dot(x-i self.w)+xelf.b) >= 1
1	self. W)+8011, b) >= 1
	JOI
	V

if undition: param * self w) olse: lambda param * self. w self . b += Self . Iv * ylidx def predict (relf, X): approx = np. dot (X, relf. w+ relf. b old kun np. signcapprox def visualize (self, X, Y, new-point = None, prediction=flone?: def get hyperplane (x, w, b, offset):

itturn (-wto] * x1 bt offset) (wti) --fig = pt. figuret) ax = fig .add_sub plot (1,1,1) for i, sample in thumerate (x): if yei) == 1: plt.scattersample [0] sample [1] marker = 10' 100(or = 1 blue) label = 16 tags + 12 17 i== 0 else " " maka='x', colo1='ved', label='class-1'1F

x0 = np. lingual (np. min(xt:, 0)) $np. max(Xi:, p])+1, 100)$ $x1 = qet. hyperplane(xo, self. w, soldon)$ $x1.m=qet. hyperplane(xo, self. w, soldon)$ $x1.p=qet. hyperplane(xo, self. w, soldon)$	_/_/)-1,
x0 = np. linspuce (np. min(xt:, 0)) $np. max(XI:, p])+1, 100)$ $x1 = qet - hyperplane (xo, self. w, self. w,$)-1,
np. max(XI:, p])+1,100) x1 = get - hyperplane (x0, self. w, s x1-m-get - hyperplane (x0, self. w, s)-1,
np. max(X[:, p])+1,100) x1 = get_hyperplane (x0, self. w, s x1-m-get_hyperplane (x0, self. w, s)-1,
x = get hyperplane (xo, self w, s x1 = get hyperplane (xo, self w, s	
XI = get hyperplane (XO, self. W,S	
*1-m-get-hyperplano (xo, self. w,s	01, 1
to style with the sty	44.20
YID - VALUE ON PARTY OF THE	elf-b,-
n'-p-get-nyperparazo, say.w,	self.b,
- Carlo	
ax.plot(x0, x1; k-), label = Decision ax.plot(x0, x1-m, 'k', label = 11 ax.plot(x0, x1-p, 'k')	Bounda
ax plot (x0, X1-m, 'k', lahel = 11	10×9m
ax.dlot(x0, X1-p, k1)	U
(127) Harris, do a dicise miss	
il new-point is not None:	
if now-point is not None: color= green if prediction == les	lse
'orange'	
'orange' label = f'New Point: Classk"," if pre == 1 else" o"'s'	ediction
==1 else"0";	
plt satter (new point to), new point	nt(i),
c=color, 9=100, edgecolors= black, label=	=
== 1 else"o"; plt satter (new point to), new point c=color, 9=100, edgecolors= black, label= (ahel, marker='**)	- i
	-
ar logend()	
plt. * Label ("Feature 1") plt. * Y label ("Feature 2") plt. * He ("SVM with New Point Prediction")	
plt y label ("Feature 2")	
plt. If the ("SVM withe New York	
Prediction")	
pht-gride true	
Prediction") Prediction" Pt grid (True) Pt show(). ifname == "main": X = np.array([
i - ame == "main":	j.
X = np.array (The	

	100
[1,7]	
[2,8]	
[3,8]	
[8,1],	
[9,2]	
[10,2]	
and the state of t	
Y = np. assay ([0,0,0,1,1,])	-4
	-
new point = np. array [[ts,5]])	1
sum = SVMC)	-
ovm.fit(x,y)	
sum = SVMC) sum fit (X, y) prediction = sum predict (new-point) [6]	
sum vicualizaty	1
prediction=prediction), y, new point = new pointo	
int Pullo	
as: L' Class 1' if préliction == 1 elge (Class 0')	spedi
-1 else (lass o')	1
The sentence of the sentence o	
THE PARTY OF THE P	
THE RESERVE THE PARTY OF THE PA	

```
Code:
import numpy as np
import matplotlib.pyplot as plt
class SVM:
    def init (self, learning rate=0.001, lambda param=0.01,
n iters=1000):
        self.lr = learning rate
        self.lambda param = lambda param
        self.n iters = n iters
        self.w = None
        self.b = None
   def fit(self, X, y):
        y = np.where(y \le 0, -1, 1) # Convert labels to -1 and
1
        n samples, n features = X.shape
        self.w = np.zeros(n features)
        self.b = 0
        for in range(self.n iters):
            for idx, x i in enumerate(X):
                condition = y[idx] * (np.dot(x_i, self.w) +
self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda param *
self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda param *
self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]
    def predict(self, X):
        approx = np.dot(X, self.w) + self.b
        return np.sign(approx)
   def visualize(self, X, y, new point=None, prediction=None):
        def get hyperplane(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]
       fig = plt.figure()
```

```
ax = fig.add subplot(1, 1, 1)
        # Plot existing data points
        for i, sample in enumerate(X):
            if y[i] == 1:
                plt.scatter(sample[0], sample[1], marker='o',
color='blue', label='Class +1' if i == 0 else "")
            else:
                plt.scatter(sample[0], sample[1], marker='x',
color='red', label='Class -1' if i == 0 else "")
        # Plot decision boundary
        x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1,
100)
        x1 = get hyperplane(x0, self.w, self.b, 0)
        x1 m = get hyperplane(x0, self.w, self.b, -1)
        x1 p = get hyperplane(x0, self.w, self.b, 1)
        ax.plot(x0, x1, 'k-', label='Decision Boundary')
        ax.plot(x0, x1 m, 'k--', label='Margins')
        ax.plot(x0, x1 p, 'k--')
        # Plot the new point
        if new point is not None:
            color = 'green' if prediction == 1 else 'orange'
            label = f'New Point: Class {"1" if prediction == 1
else "0"}'
            plt.scatter(new point[0], new point[1], c=color,
s=100, edgecolors='black', label=label, marker='*')
        ax.legend()
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
        plt.show()
# 🚀 Example usage
if name == " main ":
   # Training data
```

```
X = np.array([
        [1, 7],
        [2, 8],
        [3, 8],
        [8, 1],
        [9, 2],
       [10, 2]
    ])
   y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1
   # New point to classify
   new point = np.array([[5, 5]])
   # Train and predict
    svm = SVM()
    svm.fit(X, y)
   prediction = svm.predict(new point)[0]
   # Visualize
    svm.visualize(X, y, new point=new point[0],
prediction=prediction)
    # Print prediction
   print(f"New point {new point[0]} classified as: {'Class 1'
if prediction == 1 else 'Class 0'}")
```

Implement Random forest ensemble method on a given dataset

->	PANDOM FORFST
376	Do so subrung formi
a log la	Anne de legen + del celent institution
	from ste learn model-selection import touin-
	101 30 HT
	from skleagn. ensemple import Randomforest [
	tanil oc
	from stleanmetrics impost accuracy score, classification report
	classification report
	Lorn google colab import files
100	uploaded = files. upload ()
11/190	for filename in uploaded keys1): dk.pd. read-(sv. (filename)
21/19	dk.pd. read-(sv (filename)
JI BO	print (f"pata loadled from: & filename)"
	disp (af had (1)
	-
(DE	X = df.iloc6:,:-17
	y = d[.; loc[:, -1]
	x-to ain, x-test, VI-train, x-text = train_
	tostsplit (x, y, tost size =0.2, rundoon)
	x-to ain, x test, y-train, y-text = train_ +cet split (x, y, text size =0.2, rundoon_
- 12	D2

```
Code.
# STEP 1: Import required libraries
import pandas as pd
from sklearn.model selection import train test split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy score,
classification report
from google.colab import files
# STEP 2: Upload your dataset
uploaded = files.upload()
# STEP 3: Load the dataset (assuming it's a CSV)
for filename in uploaded.keys():
    df = pd.read csv(filename)
   print(f"Data loaded from: {filename}")
   display(df.head()) # Display first 5 rows of data
# STEP 4: Preprocessing
# Assume the last column is the target variable (label)
X = df.iloc[:, :-1] # Features (all rows, all columns except
last)
y = df.iloc[:, -1] # Target (last column)
# STEP 5: Split the data into training and testing sets
X train, X test, y train, y test = train test split(X, y,
test size=0.2, random state=42)
# STEP 6: Initialize and train the Random Forest model
rf model = RandomForestClassifier(n estimators=100,
random state=42) # 100 trees in the forest
rf model.fit(X train, y train)
# STEP 7: Make predictions on the test set
y pred = rf model.predict(X test)
# STEP 8: Evaluate the model
accuracy = accuracy score(y test, y pred)
print(f"Accuracy of Random Forest Model: {accuracy * 100:.2f}%")
# STEP 9: Print classification report
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Implement Boosting ensemble method on a given dataset

=> Bob sting	5-
import numpu as ha	2
import numpy as np from sklecum here import pecision Tree (la from sklecum datasets import make class from sklecum metrics import necessary so class AdaBoost:	UNITIFE
from schean metrics import necessary co	suffice than
class AdaBoost:	DV I
estimators (self, n_estimators > 7	50)
self.n. extimators = n-estimators self.aiphas - [] self.models = []	
The state of the s	

Ł	_/_/_
3	
3	det fit (self, x, y):
3	det fit (self, x, y): n-sampler, n-feature = x.shape w = np.ones (n-samples) In-samples
3	
3	for restimator in range (self.n_estimators. model = pecisionTrecclassificy(max_
3	(XODTA - 1)
3	model. fit (x, y, sample_weight=n) x-pred = model predict(x)
	exx = = = = = = = = = = = = = = = = = =
	(np.sum(w) sum(w*(y-pred!=y))
	alpha = 0.5 * np. log(1-411)/err)
	self alphos append (alpha)
	alpha = 0.5 * np.log(1-411)/err) il en <1 else o sell alphos append (alpha) self models append (model)
	w=w*np.exp(-alpho*y*y pred) w=w/np.sum(w)
-1	
	del predict (self,x): fral-pred-np.zoros (X. Shapeto.)
A-	self alphas? self alpha in zip (self modely, self alphas? final - prod += olpha + model . gradition

//_
return np. sign (Final pred)
. /
John 15/25

Code:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make classification
from sklearn.metrics import accuracy score
class AdaBoost:
   def init (self, n estimators=50):
        self.n estimators = n estimators
        self.alphas = [] # Weights of each weak classifier
        self.models = [] # Weak classifiers (e.g., decision
stumps)
   def fit(self, X, y):
        # Initialize weights for each data point
        n samples, n features = X.shape
        w = np.ones(n samples) / n samples # Equal weights
initially
        for estimator in range(self.n estimators):
            # Train weak classifier (decision stump)
            model = DecisionTreeClassifier(max depth=1) #
Decision stump
            model.fit(X, y, sample weight=w)
            y pred = model.predict(X)
            # Calculate error rate
            err = np.sum(w * (y pred != y)) / np.sum(w)
            # Compute alpha (weight for the classifier)
            alpha = 0.5 * np.log((1 - err) / err) if err < 1
else 0
            self.alphas.append(alpha)
            self.models.append(model)
            # Update weights for misclassified samples
            w = w * np.exp(-alpha * y * y pred) # Update
weights based on classifier performance
            w = w / np.sum(w) # Normalize the weights
```

```
def predict(self, X):
        # Initialize the final prediction
        final pred = np.zeros(X.shape[0])
        for model, alpha in zip(self.models, self.alphas):
            final pred += alpha * model.predict(X)
        # Return the sign of the final prediction
        return np.sign(final pred)
   def score(self, X, y):
        # Return accuracy of the model
        return accuracy score(y, self.predict(X))
# Generate a synthetic binary classification dataset
X, y = make classification(n samples=500, n features=20,
n classes=2, random state=42)
# Convert labels to -1 and 1 for AdaBoost
y = 2 * y - 1
# Create and train AdaBoost model
adaboost = AdaBoost(n estimators=50)
adaboost.fit(X, y)
# Evaluate the model
accuracy = adaboost.score(X, y)
print(f"Model accuracy: {accuracy:.4f}")
```

Build k-Means algorithm to cluster a set of data stored in a .CSV file

7	K-MEAMS:
Cordness	class to Means: definit - Crett, t=2, toleseura = 0.001, mout_ ites = 5007: self max_iterations = max_ites relf tolerance = tolerance
(4)	def eucliclean-distance (self, point 1, points): Return np. linalg. norm (point) - point 2, axis o
Nana	del predict Ceelf clata): distances = Cry. bradg. norm (data - self. centroid) (centroid J) hor controid in Self. controids) gettern classification

```
_//
del main [1:
   K = 3
   Center-1 = np. caray ([1,1])
  center=2 = np. asser ([ 515])
center=3 = np. asser ([ 515])
  cluster 1 = np. random vanda (100,2) + center-
  cluster 3 = np. random ranon (100,2)+ Center 3 cluster 3 = np. tarolom ranon (100,2)- Center 3
   data = np. concatenatell cluster, cluster,
 (lusta-3), 9715=6)
 K-means = K-duering 16
 t means Rit (classa
   colors = 10 *["," "g", "c", "b", "£"]
   for centroid in Emany controids
 plt-slattu (t. means centroids [centroid 7 [0] = means. centroid [ [1], s=130, marker= 2
   by clurky-index in +-mong clares:
indust:
    colors = colors[clusta_index]
(Dlox = color, s=30) features [0], features a)
```

	//_
3	elet
3	def fit (self, data): self controids =
	sell contaids = 44
3	for i in range (self. t.)
3	self. centroids ti] = data[i]
	seq. ceroolus en = acaracis
I potent	for in range (self-max-iterations).
	for i in range (self.max_iterations): nelf.classes = 25
The same of	for j in range (self. t):) ref. classes ti) = []
	refl. classes ti)=[7
The sale	The second of the characters of the second second as
	for point in closus:
100	distances=[] for inclox in relf. controids:
	for index in helf consoids:
	distances append (self, cullidaan.
	distance (point, self. controids [incless])
	clistances)) cluster_inclox = distances in dex (mix
	relf. Classes (chapter = in dos) append (p
	in)
1	prévious = ded (self centroids)
	for cluster index in self classes:
	" self centroids byluster index] =
	ny-average (self: cluses (cluster ixplex), anis=0
F	is Optimal = True
	is Optimal = True break
RE .	

143 Mic	i name== !! main_!!	
	main ()	
	I aman would be to be to be	

```
Code:
class K Means:
    def init (self, k=2, tolerance = 0.001, max iter = 500):
        self.k = k
        self.max iterations = max iter
        self.tolerance = tolerance
    def euclidean distance(self, point1, point2):
        #return math.sqrt((point1[0]-point2[0])**2 + (point1[1]-
point2[1])**2 + (point1[2]-point2[2])**2) #sqrt((x1-x2)^2 +
(y1-y2)^2
        return np.linalg.norm(point1-point2, axis=0)
    def predict(self,data):
        distances = [np.linalg.norm(data-
self.centroids[centroid]) for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification
    def fit(self, data):
        self.centroids = {}
        for i in range(self.k):
            self.centroids[i] = data[i]
        for i in range(self.max iterations):
            self.classes = {}
            for j in range(self.k):
                self.classes[j] = []
            for point in data:
                distances = []
                for index in self.centroids:
distances.append(self.euclidean distance(point, self.centroids[in
```

dex]))

```
cluster index = distances.index(min(distances))
                self.classes[cluster index].append(point)
            previous = dict(self.centroids)
            for cluster index in self.classes:
                self.centroids[cluster index] =
np.average(self.classes[cluster index], axis = 0)
            isOptimal = True
            for centroid in self.centroids:
                original centroid = previous[centroid]
                curr = self.centroids[centroid]
                if np.sum((curr - original centroid)/
original centroid * 100.0) > self.tolerance:
                    isOptimal = False
            if isOptimal:
                break
def main():
    K=3
    center 1 = np.array([1,1])
    center 2 = np.array([5,5])
   center 3 = np.array([8,1])
   # Generate random data and center it to the three centers
    cluster 1 = np.random.randn(100, 2) + center 1
    cluster 2 = np.random.randn(100,2) + center 2
    cluster 3 = np.random.randn(100,2) + center 3
    data = np.concatenate((cluster 1, cluster 2, cluster 3),
axis = 0)
    k means = K Means(K)
    k means.fit(data)
```

```
# Plotting starts here
colors = 10*["r", "g", "c", "b", "k"]

for centroid in k_means.centroids:
    plt.scatter(k_means.centroids[centroid][0],
k_means.centroids[centroid][1], s = 130, marker = "x")

for cluster_index in k_means.classes:
    color = colors[cluster_index]
    for features in k_means.classes[cluster_index]:
        plt.scatter(features[0], features[1], color = color,s = 30)

if __name__ == "__main__":
    main()
```

Implement Dimensionality reduction using Principal Component Analysis (PCA) method Observation:

3	
21 04	25 _/_/_
	PRINCIPAL COMPONENT ANALYSIS
I CAN	ANALYSIS
	imanet - anda
3	import pandas as pd
	import numpy as no
	from stleam decomposition import PCA
	from stelearn preprocessing import Standard Scales
	from 900 gle. was import files
and the	
	uploacled = files upload ()
	for filename in uploaded. Keys(): d = p0. read_csvl'(content/pizza.csv') print(f"uploaded: L'(content/pizza.csv') display(de pard(s))
	d = po. read_csvl'(content/pizza.csv')
F-	Print It Uploa dad: 21/ content pizza.csv)
1	I display (df. heads)
70	numeric-Al = df. select-dhiper (include = 1 22 - 1 100 7)
60	numeric- of = of seled-drypes (include = cnp. number)- print ("Numerical features found: ", list (numeric-clf. column)
2	Selected_features=numeric_df.columns
R	V = numaic de l'alected l'atendi depara ()
4	X = nymoic df [setected features]. droppa () X = scaled = Standood Scalor(). fit - transform ()
1	
	principle-components=pa. fit-toans form (x scaled)
1	
	pa-df=pd. patatrame (data=principle component),
	Columns - C'RI', Pay
	(arr

	//_
	VIt. Liquide (figsing = (86))
	of scattering of C'DC' Don de C'DC' 2 and
	plt. figure (figsing = (8,6)) plt. scatter(pca af ['PCI'], pco-df ['PC2'], culpha=0.7) plt. x latel ('Principal component 1') plt. y label ('principal component 2') plt. bite ('so pa visualisation') plt. grid (True)
	ni + real of C' a real of the
	Dit Like Land panala components
male	1011 and (7 visualisation)
	pdt. grid (True)
	pt. show()
	print ("Explained variona patro:", pra. explained _
	variance ratio)
	Dist [ft Nadal
	print (1" Model accuracy: Laccusacy: -41)
	The state of the s
-	K-MEANS:
	K-MEANS:
John Williams	class K Means:
Ja Bo	def_init(rett, ==2, toleseunce = 0.001, mout_
-	
	self max-iterations = max-iter
	self to lesance = to lesance
(8)	det tudiclean-distance (self, point 1, points):
	def Eucliclean-distance (self. point 1, points): return np. linalg. norm (point 1 - point 2, axis o)
	del predict Ceelf clata): clistanos = Cry. linalq. norm (dato - self. centroid) (centroid]) hor controid in Self. controide) Return classification
Marin	distances = Cry. linala. norm (data - self. Centricl)
	(centroid]) for controid in self. controids?
El sage a	getien classification

```
Code.
# STEP 1: Import packages
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from google.colab import files
# STEP 2: Upload the CSV file
uploaded = files.upload()
# STEP 3: Load the dataset
for filename in uploaded.keys():
   df = pd.read csv('/content/Pizza.csv')
   display(df.head())
# STEP 4: Select numerical columns
numeric df = df.select dtypes(include=[np.number])
print(" Numerical features found: ", list(numeric df.columns))
# OPTIONAL: Manually select columns if needed
# selected features = ['feature1', 'feature2', ...]
selected features = numeric df.columns # use all numeric
features for now
# STEP 5: Standardize data
X = numeric df[selected features].dropna()
X scaled = StandardScaler().fit transform(X)
# STEP 6: Apply PCA
pca = PCA(n components=2)
principal components = pca.fit transform(X scaled)
# STEP 7: Create DataFrame for components
pca df = pd.DataFrame(data=principal components, columns=['PC1',
'PC2'])
# STEP 8: Visualize the first two principal components
```