Q1.import csv

a = []

with open('enjoysport.csv', 'r') as csvfile:

for row in csv.reader(csvfile):

a.append(row)

print(a)

print("\n The total number of training instances are : ",len(a))

num\_attribute = len(a[0])-1

print("\n The initial hypothesis is : ")

hypothesis = ['0']\*num\_attribute

print(hypothesis)

for i in range(0, len(a)):

if a[i][num\_attribute] == 'yes':

for j in range(0, num\_attribute):

if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:

hypothesis[j] = a[i][j]

else:

hypothesis[j] = '?'

print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

q2. import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read\_csv('enjoysport.csv'))

concepts = np.array(data.iloc[:,0:-1])

print(concepts)

target = np.array(data.iloc[:,-1])

print(target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("initialization of specific\_h and general\_h")

print(specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in

range(len(specific\_h))]

print(general\_h)

for i, h in enumerate(concepts):

if target[i] == "yes":

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

specific\_h[x] ='?'

general\_h[x][x] ='?'

print(specific\_h)

print(specific\_h)

if target[i] == "no":

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

general\_h[x][x] = specific\_h[x]

else:

general\_h[x][x] = '?'

print(" steps of Candidate Elimination Algorithm",i+1)

print(specific\_h)

print(general\_h)

indices = [i for i, val in enumerate(general\_h) if val ==['?', '?', '?', '?', '?', '?']]

for i in indices:

general\_h.remove(['?', '?', '?', '?', '?', '?'])

return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h:", s\_final, sep="\n")

print("Final General\_h:", g\_final, sep="\n")

q3.

import pandas as pd

import numpy as np

import math

# Define a class for the decision tree node

class DecisionTreeNode:

def \_\_init\_\_(self, attribute=None, label=None, branches={}):

self.attribute = attribute # the attribute used to split the data

self.label = label # the label assigned to this node

self.branches = branches # the branches of the decision tree

# Define a function to calculate the entropy of a dataset

def entropy(data):

target = data['target']

n = len(target)

unique, counts = np.unique(target, return\_counts=True)

entropy = 0

for i in range(len(unique)):

p = counts[i] / n

entropy -= p \* math.log2(p)

return entropy

# Define a function to calculate the information gain of an attribute

def information\_gain(data, attribute):

n = len(data)

values = data[attribute].unique()

entropy\_s = entropy(data)

entropy\_attr = 0

for value in values:

subset = data[data[attribute] == value]

subset\_n = len(subset)

subset\_entropy = entropy(subset)

entropy\_attr += subset\_n / n \* subset\_entropy

return entropy\_s - entropy\_attr

# Define the ID3 algorithm

def id3(data, attributes):

target = data['target']

# If all the examples have the same target value, return a leaf node with that value

if len(target.unique()) == 1:

return DecisionTreeNode(label=target.iloc[0])

# If there are no attributes left to split on, return a leaf node with the most common target value

if len(attributes) == 0:

return DecisionTreeNode(label=target.value\_counts().idxmax())

# Otherwise, select the attribute with the highest information gain

gains = {attr: information\_gain(data, attr) for attr in attributes}

best\_attribute = max(gains, key=gains.get)

# Create a new decision tree node with the selected attribute

node = DecisionTreeNode(attribute=best\_attribute)

# Split the data based on the selected attribute and recursively build the tree

for value in data[best\_attribute].unique():

subset = data[data[best\_attribute] == value].drop(best\_attribute, axis=1)

if len(subset) == 0:

node.branches[value] = DecisionTreeNode(label=target.value\_counts().idxmax())

else:

new\_attributes = attributes.copy()

new\_attributes.remove(best\_attribute)

node.branches[value] = id3(subset, new\_attributes)

return node

# Load the dataset

data = pd.read\_csv('play\_tennis.csv')

# Split the dataset into attributes and target variable

attributes = data.columns[:-1].tolist()

# Build the decision tree using ID3 algorithm

root = id3(data, attributes)

# Define a function to classify a new sample using the decision tree

def classify(sample, tree):

if tree.label is not None:

return tree.label

attribute = tree.attribute

value = sample[attribute]

if value not in tree.branches:

return tree.branches[max(tree.branches.keys(), key=int)]

subtree = tree

q4. import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

#Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr # dotproduct of nextlayererror and currentlayerop

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

q5

from math import sqrt

from statistics import mode

l=[[33.6,50,1],[26.6,30,0],[23.4,40,0],[43.1,67,0],[35.3,23,1],[35.9,67,1],[36.7,45,1],[25.7,46,0],[23.3,29,0],[31,56,1]]

n=[43.6,40]

k=3

m=[]

x=[]

for i in l:

a=0

for j in range(len(n)-1):

a+= (i[j]-n[j])\*(i[j]-n[j])

m.append(sqrt(a))

a=sorted(m)

for i in range(k):

x.append(m.index(a[i]))

y=[]

for i in x:

print(l[i])

y.append(l[i][-1])

print()

print("result -->",mode(y))

q6

# import required libraries

from sklearn.datasets import load\_iris

from sklearn.naive\_bayes import GaussianNB

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

from sklearn.metrics import confusion\_matrix, accuracy\_score

# load iris dataset

iris = load\_iris()

# split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=0)

# create Naive Bayes classifier

classifier = GaussianNB()

# train the classifier using the training data

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

# predict the target values for the testing data

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

q7.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Generate sample data

np.random.seed(0)

X = np.linspace(0, 10, 100).reshape(-1, 1)

y = 2 \* X + 1 + np.random.randn(100, 1)

# Create linear regression object

lr\_model = LinearRegression()

# Train the model using the training sets

lr\_model.fit(X, y)

# Print the coefficients

print('Coefficients: ', lr\_model.coef\_)

print('Intercept: ', lr\_model.intercept\_)

# Plot the data and the linear regression line

plt.scatter(X, y, color='blue')

plt.plot(X, lr\_model.predict(X), color='red', linewidth=3)

plt.title('Linear Regression')

plt.xlabel('X')

plt.ylabel('y')

plt.show()

q8.

# Step 1: Import the required modules

from sklearn.datasets import make\_classification

from matplotlib import pyplot as plt

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import confusion\_matrix

import pandas as pd

# Step 2: Generate the dataset

x, y = make\_classification(

n\_samples=100,

n\_features=1,

n\_classes=2,

n\_clusters\_per\_class=1,

flip\_y=0.03,

n\_informative=1,

n\_redundant=0,

n\_repeated=0

)

print(y)

# Step 3: visualize the data

plt.scatter(x, y, c=y, cmap='rainbow')

plt.title('Scatter Plot of Logistic Regression')

plt.show()

# Step 4: Split the dataset

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

x\_train.shape

# Step 4: Perform Logistic Regression

log\_reg = LogisticRegression()

log\_reg.fit(x\_train, y\_train)

# Step 5: Make prediction using the model

y\_pred = log\_reg.predict(x\_test)

# Step 6: Display the Confusion Matrix

confusion\_matrix(y\_test, y\_pred)

q9.

import numpy as np

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Create some sample data

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.array([2, 4, 5, 4, 5]).reshape(-1, 1)

# Create a linear regression object and fit the data

reg = LinearRegression().fit(X, y)

# Predict new values

X\_new = np.array([6]).reshape(-1, 1)

y\_pred = reg.predict(X\_new)

# Plot the data and the linear regression line

plt.scatter(X, y)

plt.plot(X, reg.predict(X), color='red')

plt.show()

import numpy as np

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

import matplotlib.pyplot as plt

# Create some sample data

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.array([2, 4, 5, 4, 5]).reshape(-1, 1)

# Transform the data to include another axis

poly = PolynomialFeatures(degree=2)

X\_poly = poly.fit\_transform(X)

# Create a polynomial regression object and fit the data

reg = LinearRegression().fit(X\_poly, y)

# Predict new values

X\_new = np.array([6]).reshape(-1, 1)

X\_new\_poly = poly.transform(X\_new)

y\_pred = reg.predict(X\_new\_poly)

# Plot the data and the polynomial regression curve

plt.scatter(X, y)

plt.plot(X, reg.predict(X\_poly), color='red')

plt.show()

q10. import numpy as np

from scipy.stats import norm

# Define the data

data = np.array([1.2, 2.3, 0.7, 1.6, 1.1, 1.8, 0.9, 2.2])

# Initialize the parameters

mu1 = 0

mu2 = 1

sigma1 = 1

sigma2 = 1

p1 = 0.5

p2 = 0.5

# Run the EM algorithm

for i in range(10):

# E-step

likelihood1 = norm.pdf(data, mu1, sigma1)

likelihood2 = norm.pdf(data, mu2, sigma2)

weight1 = p1 \* likelihood1 / (p1 \* likelihood1 + p2 \* likelihood2)

weight2 = p2 \* likelihood2 / (p1 \* likelihood1 + p2 \* likelihood2)

# M-step

mu1 = np.sum(weight1 \* data) / np.sum(weight1)

mu2 = np.sum(weight2 \* data) / np.sum(weight2)

sigma1 = np.sqrt(np.sum(weight1 \* (data - mu1)\*\*2) / np.sum(weight1))

sigma2 = np.sqrt(np.sum(weight2 \* (data - mu2)\*\*2) / np.sum(weight2))

p1 = np.mean(weight1)

p2 = np.mean(weight2)

# Print the final estimates of the parameters

print("mu1:", mu1)

print("mu2:", mu2)

print("sigma1:", sigma1)

print("sigma2:", sigma2)

print("p1:", p1)

print("p2:", p2)

q11.

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

import plotly.io as pio

pio.templates.default = "plotly\_white"

data = pd.read\_csv("CREDITSCORE.csv")

print(data.head())

print(data.info())

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

x = np.array(data[["Annual\_Income", "Monthly\_Inhand\_Salary",

"Num\_Bank\_Accounts", "Num\_Credit\_Card",

"Interest\_Rate", "Num\_of\_Loan",

"Delay\_from\_due\_date", "Num\_of\_Delayed\_Payment",

"Credit\_Mix", "Outstanding\_Debt",

"Credit\_History\_Age", "Monthly\_Balance"]])

y = np.array(data[["Credit\_Score"]])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

test\_size=0.33,

random\_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(xtrain, ytrain)

print("Credit Score Prediction : ")

a = float(input("Annual Income: "))

b = float(input("Monthly Inhand Salary: "))

c = float(input("Number of Bank Accounts: "))

d = float(input("Number of Credit cards: "))

e = float(input("Interest rate: "))

f = float(input("Number of Loans: "))

g = float(input("Average number of days delayed by the person: "))

h = float(input("Number of delayed payments: "))

i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")

j = float(input("Outstanding Debt: "))

k = float(input("Credit History Age: "))

l = float(input("Monthly Balance: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l]])

print("Predicted Credit Score = ", model.predict(features))

q12.

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

iris = pd.read\_csv("IRIS.csv")

print(iris.head())

print()

print(iris.describe())

print("Target Labels", iris["species"].unique())

import plotly.io as io

import plotly.express as px

fig = px.scatter(iris, x="sepal\_width", y="sepal\_length", color="species")

fig.show()

x = iris.drop("species", axis=1)

y = iris["species"]

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y,test\_size=0.2,random\_state=0)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(x\_train, y\_train)

x\_new = np.array([[6, 2.9, 1, 0.2]])

prediction = knn.predict(x\_new)

print("Prediction: {}".format(prediction))

q13.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.tree import DecisionTreeRegressor

#Importing the dataset

data = pd.read\_csv("CarPrice.csv")

#Data Exploration

data.head()

data.shape

data.isnull().sum() #Checking if the dataset has NULL Values

data.info()

data.describe()

data.CarName.unique()

#Analysing correlations & using heatmap

print(data.corr())

plt.figure(figsize=(20, 15))

correlations = data.corr()

sns.heatmap(correlations, cmap="coolwarm", annot=True)

plt.show()

#Training a Car Price Prediction Model

predict = "price"

data = data[["symboling", "wheelbase", "carlength",

"carwidth", "carheight", "curbweight",

"enginesize", "boreratio", "stroke",

"compressionratio", "horsepower", "peakrpm",

"citympg", "highwaympg", "price"]]

x = np.array(data.drop([predict], 1))

y = np.array(data[predict])

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

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2)

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()

model.fit(xtrain, ytrain)

predictions = model.predict(xtest)

from sklearn.metrics import mean\_absolute\_error

model.score(xtest, predictions)

q14.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

#Importing Dataset

dataset = pd.read\_csv("HousePricePrediction.csv")

#Exploring dataset

print(dataset.head(5))

dataset.shape

obj = (dataset.dtypes == 'object')

object\_cols = list(obj[obj].index)

print("Categorical variables:",len(object\_cols))

int\_ = (dataset.dtypes == 'int')

num\_cols = list(int\_[int\_].index)

print("Integer variables:",len(num\_cols))

fl = (dataset.dtypes == 'float')

fl\_cols = list(fl[fl].index)

print("Float variables:",len(fl\_cols))

plt.figure(figsize=(12, 6))

sns.heatmap(dataset.corr(),

cmap = 'BrBG',

fmt = '.2f',

linewidths = 2,

annot = True)

unique\_values = []

for col in object\_cols:

unique\_values.append(dataset[col].unique().size)

plt.figure(figsize=(10,6))

plt.title('No. Unique values of Categorical Features')

plt.xticks(rotation=90)

sns.barplot(x=object\_cols,y=unique\_values)

plt.figure(figsize=(18, 36))

plt.title('Categorical Features: Distribution')

plt.xticks(rotation=90)

index = 1

for col in object\_cols:

y = dataset[col].value\_counts()

plt.subplot(11, 4, index)

plt.xticks(rotation=90)

sns.barplot(x=list(y.index), y=y)

index += 1

dataset.drop(['Id'],axis=1,inplace=True)

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

new\_dataset = dataset.dropna()

new\_dataset.isnull().sum()

from sklearn.preprocessing import OneHotEncoder

s = (new\_dataset.dtypes == 'object')

object\_cols = list(s[s].index)

print("Categorical variables:")

print(object\_cols)

print('No. of. categorical features: ',len(object\_cols))

OH\_encoder = OneHotEncoder(sparse=False)

OH\_cols = pd.DataFrame(OH\_encoder.fit\_transform(new\_dataset[object\_cols]))

OH\_cols.index = new\_dataset.index

OH\_cols.columns = OH\_encoder.get\_feature\_names()

df\_final = new\_dataset.drop(object\_cols, axis=1)

df\_final = pd.concat([df\_final, OH\_cols], axis=1)

from sklearn.metrics import mean\_absolute\_error

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

X = df\_final.drop(['SalePrice'], axis=1)

Y = df\_final['SalePrice']

X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(X, Y, train\_size=0.8, test\_size=0.2, random\_state=0)

from sklearn import svm

from sklearn.svm import SVC

from sklearn.metrics import mean\_absolute\_percentage\_error

model\_SVR = svm.SVR()

model\_SVR.fit(X\_train,Y\_train)

Y\_pred = model\_SVR.predict(X\_valid)

print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred))

#LinearRegression

from sklearn.linear\_model import LinearRegression

model\_LR = LinearRegression()

model\_LR.fit(X\_train, Y\_train)

Y\_pred = model\_LR.predict(X\_valid)

print(mean\_absolute\_percentage\_error(Y\_valid, Y\_pred))

q16.

import numpy

import pandas as pd

import numpy as np

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import BernoulliNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import PassiveAggressiveClassifier

from sklearn.metrics import classification\_report

iris= pd.read\_csv("D:/GEO/BE COURSES/LAB/DATASET/IRIS.csv")

print(iris.head())

x = iris.drop("species", axis=1)

y = iris["species"]

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y,test\_size=0,random\_state=42)

#x = np.array(data[["Age", "EstimatedSalary"]])

#y = np.array(data[["Purchased"]])

#xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.10, random\_state=42)

decisiontree = DecisionTreeClassifier()

logisticregression = LogisticRegression()

knearestclassifier = KNeighborsClassifier()

#svm\_classifier = SVC()

bernoulli\_naiveBayes = BernoulliNB()

passiveAggressive = PassiveAggressiveClassifier()

knearestclassifier.fit(x\_train, y\_train)

decisiontree.fit(x\_train, y\_train)

logisticregression.fit(x\_train, y\_train)

passiveAggressive.fit(x\_train, y\_train)

data1 = {"Classification Algorithms": ["KNN Classifier", "Decision Tree Classifier",

"Logistic Regression", "Passive Aggressive Classifier"],

"Score": [knearestclassifier.score(x,y), decisiontree.score(x, y),

logisticregression.score(x, y), passiveAggressive.score(x,y) ]}

score = pd.DataFrame(data1)

score

q17.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

#importing dataset

data = pd.read\_csv("mobile\_prices.csv")

print(data.head())

plt.figure(figsize=(12, 10))

sns.heatmap(data.corr(), annot=True, cmap="coolwarm", linecolor='white', linewidths=1)

#data preparation

x = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

x = StandardScaler().fit\_transform(x)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=0)

# Logistic Regression algorithm provided by Scikit-learn:

from sklearn.linear\_model import LogisticRegression

lreg = LogisticRegression()

lreg.fit(x\_train, y\_train)

y\_pred = lreg.predict(x\_test)

#accuracy of the model:

accuracy = accuracy\_score(y\_test, y\_pred) \* 100

print("Accuracy of the Logistic Regression Model: ",accuracy)

#predictions made by the model:

print(y\_pred)

(unique, counts) = np.unique(y\_pred, return\_counts=True)

price\_range = np.asarray((unique, counts)).T

print(price\_range)

q18.

from sklearn import datasets

import numpy as np

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

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data[:, [2, 3]]

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=1, stratify=y)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

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

ppn = Perceptron(eta0=0.1, random\_state=1)

ppn.fit(X\_train\_std, y\_train)

y\_pred = ppn.predict(X\_test\_std)

print('Accuracy: %.3f' % accuracy\_score(y\_test, y\_pred))

print('Accuracy: %.3f' % ppn.score(X\_test\_std, y\_test))

from sklearn import datasets

import numpy as np

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

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()

X = iris.data[:, [2, 3]]

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=1, stratify=y)

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

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

ppn = Perceptron(eta0=0.1, random\_state=1)

ppn.fit(X\_train\_std, y\_train)

y\_pred = ppn.predict(X\_test\_std)

print('Accuracy: %.3f' % accuracy\_score(y\_test, y\_pred))

print('Accuracy: %.3f' % ppn.score(X\_test\_std, y\_test))

q19.

import numpy as np

import pandas as pd

dataset = pd.read\_csv("breastcancer.csv")

X = dataset.iloc[:, :-1].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 = 0.25, 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)

GaussianNB(priors=None, var\_smoothing=1e-09)

from sklearn.metrics import confusion\_matrix, accuracy\_score

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

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

q20.

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

import plotly.io as io

io.renderers.default='browser'

data = pd.read\_csv("futuresale prediction.csv")

print(data.head())

print(data.sample(5))

print(data.isnull().sum())

import plotly.express as px

import plotly.graph\_objects as go

figure = px.scatter(data\_frame = data, x="Sales",

y="TV", size="TV", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Newspaper", size="Newspaper", trendline="ols")

figure.show()

figure = px.scatter(data\_frame = data, x="Sales",

y="Radio", size="Radio", trendline="ols")

figure.show()

correlation = data.corr()

print(correlation["Sales"].sort\_values(ascending=False))

x = np.array(data.drop(["Sales"], 1))

y = np.array(data["Sales"])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(xtrain, ytrain)

print(model.score(xtest, ytest))

features = [[TV, Radio, Newspaper]]

features = np.array([[230.1, 37.8, 69.2]])

print(model.predict(features))