1)Implement and demonstrate the FIND-S algorithm for finding the most specific

hypothesis based on a given set of training data samples.

PROGRAM :-

data = [['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]

num\_atts = len(data[0]) - 1

hypothesis = data[0][0:num\_atts]

for i in range(1, len(data)):

if data[i][num\_atts] == 'Yes':

hypothesis = [data[i][j] if hypothesis[j] == data[i][j] else '?' for j in range(num\_atts)]

print("Maximally specific hypothesis:")

print(hypothesis)

2) For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python to output a

description of the set of all hypotheses consistent with the training examples

PROGRAM :-

import pandas as pd

from sklearn.datasets import load\_iris

iris = load\_iris()

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

concepts = data.values.tolist()

target = iris.target.tolist()

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

num\_atts = len(concepts[0])

general\_h = [["?" for \_ in range(num\_atts)] for \_ in range(num\_atts)]

for i, h in enumerate(concepts):

if target[i] == 0:

for x in range(num\_atts):

specific\_h[x] = '?'

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

print("Specific hypothesis:", specific\_h)

print("General hypothesis:", general\_h)

3) Demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate

data set for building the decision tree and apply this knowledge to classify a new sample.

PROGRAM :-

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

# 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:

# If the value is not present in branches, return the label of the majority branch

majority\_branch = max(tree.branches, key=lambda k: len(tree.branches[k].branches))

return tree.branches[majority\_branch].label

subtree = tree.branches[value]

return classify(sample, subtree) # Recursively classify using the subtree

# Example usage

new\_sample = {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'high', 'windy': 'false'}

predicted\_label = classify(new\_sample, root)

print("Predicted label:", predicted\_label)

4) Build an Artificial Neural Network by implementing the

Backpropagation algorithm and test the same using appropriate data sets.

PROGRAM :-

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)

5) Write a program for Implementation of K-Nearest Neighbours (K-NN) in Python

PROGRAM :-

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))

6) Write a program to implement Naive Bayes algorithm in python and to display the results using confusion matrix and

accuracy.

PROGRAM :-

# 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)

# display confusion matrix and accuracy score

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

print("Confusion Matrix:")

print(cm)

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

print("Accuracy Score:", acc)

7) Write a program to implement Logistic Regression (LR) algorithm in python

PROGRAM :-

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()

8) Write a program to implement Linear Regression (LR) algorithm in python

PROGRAM :-

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

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)

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

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

plt.show()

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

x\_train.shape

log\_reg = LogisticRegression()

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

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

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

9) Compare Linear and Polynomial Regression using Python

PROGRAM :-

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make\_pipeline

# Sample data

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

y = np.array([2, 4, 5, 4, 6]) # Dependent variable

# Create a linear regression model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

linear\_pred = linear\_model.predict(X)

# Create a polynomial regression model (degree 2)

degree = 2

poly\_model = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

poly\_model.fit(X, y)

poly\_pred = poly\_model.predict(X)

# Plot the data, linear regression line, and polynomial regression curve

plt.scatter(X, y, label='Actual Data')

plt.plot(X, linear\_pred, color='blue', label='Linear Regression')

plt.plot(X, poly\_pred, color='red', label=f'Polynomial Regression (degree {degree})')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.title('Linear vs Polynomial Regression')

plt.show()

10) Write a Python Program to Implement Expectation & Maximization Algorithm

PROGRAM :- import numpy as np

from scipy.stats import multivariate\_normal

np.random.seed(0)

true\_mu1 = np.array([2, 2])

true\_cov1 = np.array([[1, 0.5], [0.5, 1]])

true\_mu2 = np.array([7, 7])

true\_cov2 = np.array([[1, -0.5], [-0.5, 1]])

true\_weights = [0.4, 0.6]

n\_samples = 300

n\_features = 2

X = np.concatenate([

np.random.multivariate\_normal(true\_mu1, true\_cov1, int(n\_samples \* true\_weights[0])),

np.random.multivariate\_normal(true\_mu2, true\_cov2, int(n\_samples \* true\_weights[1]))], axis=0)

n\_clusters = 2

def e\_step(X, mus, sigmas, weights):

responsibilities = []

for i in range(n\_clusters):

pdf = multivariate\_normal.pdf(X, mean=mus[i], cov=sigmas[i])

responsibilities.append(weights[i] \* pdf)

responsibilities = np.array(responsibilities)

responsibilities /= np.sum(responsibilities, axis=0)

return responsibilities

def m\_step(X, responsibilities):

n\_samples, \_ = X.shape

mus = []

sigmas = []

weights = []

for i in range(n\_clusters):

r\_sum = np.sum(responsibilities[i])

weight = r\_sum / n\_samples

mu = np.sum(responsibilities[i][:, np.newaxis] \* X, axis=0) / r\_sum

sigma = (X - mu).T.dot((X - mu) \* responsibilities[i][:, np.newaxis]) / r\_sum

mus.append(mu)

sigmas.append(sigma)

weights.append(weight)

return np.array(mus), np.array(sigmas), np.array(weights)

initial\_weights = np.ones(n\_clusters) / n\_clusters

initial\_mus = np.random.rand(n\_clusters, n\_features) \* np.max(X, axis=0)

initial\_sigmas = np.array([np.eye(n\_features)] \* n\_clusters)

mus = initial\_mus

sigmas = initial\_sigmas

weights = initial\_weights

max\_iters = 100

tolerance = 1e-6

for i in range(max\_iters):

prev\_mus = mus.copy()

responsibilities = e\_step(X, mus, sigmas, weights)

mus, sigmas, weights = m\_step(X, responsibilities)

if np.allclose(prev\_mus, mus, atol=tolerance):

print(f"Converged after {i + 1} iterations.")

break

print("Estimated means:")

print(mus)

print("Estimated covariances:")

print(sigmas)

print("Estimated weights:")

print(weights)

11) Write a program for the task of Credit Score Classification.

PROGRAM :-

import numpy as np

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report, accuracy\_score

X = np.array([[25, 50000], [30, 80000], [35, 75000], [22, 30000], [40, 100000], [28, 60000]])

y = np.array([0, 1, 1, 0, 1, 0])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier()

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

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

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

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=['Bad Credit', 'Good Credit'])

print("Accuracy:", accuracy)

print("Classification Report:")

print(classification\_rep)

12) Implement Iris Flower Classification using KNN.

PROGRAM :- import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, accuracy\_score

iris = load\_iris()

X, y = iris.data, iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler().fit(X\_train)

X\_train, X\_test = scaler.transform(X\_train), scaler.transform(X\_test)

k = 3

clf = KNeighborsClassifier(n\_neighbors=k).fit(X\_train, y\_train)

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

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

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

print("Accuracy:", accuracy, "\nClassification Report:\n", classification\_rep)

13) Implement the Car Price Prediction Model using Python

PROGRAM :- import numpy as np

import pandas as pd

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

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_absolute\_error

from sklearn.datasets import load\_diabetes

diabetes = load\_diabetes()

X, y = diabetes.data, diabetes.target

data = pd.DataFrame(data=np.c\_[X, y], columns=np.append(diabetes.feature\_names, "target"))

data = data.drop(columns=['sex'])

X, y = data.drop(columns=["target"]), data["target"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = DecisionTreeRegressor(random\_state=42)

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

predictions = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, predictions)

print("Mean Absolute Error:", mae)

new\_data = pd.DataFrame({'age': [0.05], 'bmi': [0.1], 'bp': [0.2], 's1': [0.3], 's2': [0.4], 's3': [0.5], 's4': [0.6], 's5': [0.7], 's6': [0.8]})

new\_data\_aligned = new\_data.reindex(columns=X.columns, fill\_value=0)

predicted\_target = model.predict(new\_data\_aligned)

print("Predicted Target for the new data:", predicted\_target[0])

14) Implement House price Prediction using appropriate machine learning algorithm

PROGRAM :- import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

from sklearn.datasets import fetch\_california\_housing

X, y = fetch\_california\_housing(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression().fit(X\_train, y\_train)

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

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.2f}")

15) Implement Iris Flower Classification using Naive Bayes classifier

PROGRAM :- import numpy as np, pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.datasets import load\_iris

X, y = load\_iris(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = GaussianNB().fit(X\_train, y\_train)

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

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

print(f"Accuracy: {accuracy:.2f}\nClassification Report:\n{classification\_report(y\_test, y\_pred, target\_names=load\_iris().target\_names)}")16) Compare different types Classification Algorithms and evaluate their performance.

PROGRAM :-

import numpy as np, pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.datasets import load\_iris

iris = load\_iris()

X, y = iris.data, iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

classifiers = {

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'SVM': SVC(random\_state=42),

'Random Forest': RandomForestClassifier(random\_state=42)}

for name, clf in classifiers.items():

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

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

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

report = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

print(f"{name} Classifier:\nAccuracy: {accuracy:.2f}\nClassification Report:\n{report}\n{'='\*40}")

17) Implement Mobile Price Prediction using appropriate machine learning algorithm

PROGRAM :-

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

from sklearn.datasets import load\_iris

iris = load\_iris()

X, y = iris.data, iris.target

iris\_df = pd.DataFrame(np.c\_[X, y], columns=iris.feature\_names + ['target'])

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

sns.heatmap(iris\_df.corr(), annot=True, cmap="coolwarm", linecolor='white', lw=1)

plt.show()

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

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

logreg = LogisticRegression().fit(X\_train, y\_train)

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

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

print("Accuracy:", accuracy)

print("Predicted labels:", y\_pred)

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

class\_counts = np.c\_[unique, counts]

print("Predicted class counts:", class\_counts)

18) Implement Perceptron based IRIS classification

PROGRAM :-

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))

19) Implementation of Naive Bayes classification for Bank Loan prediction

PROGRAM :-

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)

20) Implement Future Sales Prediction using a suitable machine learning algorithm

PROGRAM :-

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))