Experiment-1:

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

Program:

import pandas as pd

import numpy as np

dataframe=pd.read\_csv("weather.csv")

print(dataframe)

features=np.array(dataframe)[:,:-1]

print("the attributes are:\n",features)

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

print(target)

for i, val in enumerate(target):

if val=="Yes":

hypothesis = features[i].copy()

break

for i, val in enumerate(features):

if target[i]=="Yes":

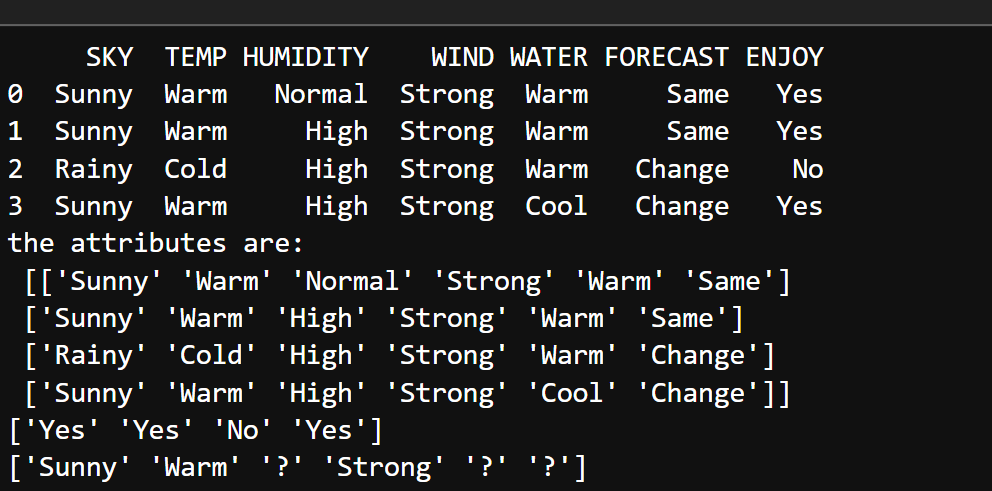
for x in range(len(hypothesis)):

if val[x]!=hypothesis[x]:

hypothesis[x]='?'

print(hypothesis)

Output:



Experiment-2:

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

Program:

import pandas as pd

import numpy as np

dataframe=pd.read\_csv("weather.csv")

#print(dataframe,"n")

features=np.array(dataframe)[:,:-1]

#print("n the attributes are:",features)

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

#print(target)

def learn(features,target):

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

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

print(specific\_h)

print(general\_h)

for i,h in enumerate(features):

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[x])

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]='?'

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

for i in indices:

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

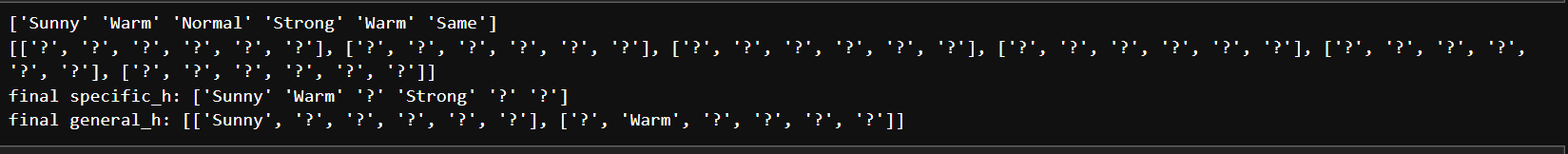
return specific\_h,general\_h

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

print("final specific\_h:",s\_final)

print("final general\_h:",g\_final)

Output:



Experiment-3:

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

Program:

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.datasets import load\_iris

import pandas as pd

import numpy as np

from sklearn import tree

data=load\_iris()

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

df['target']=data.target

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df[data.feature\_names],df['target'])

clf=DecisionTreeClassifier(criterion="entropy",max\_depth=5)

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

clf.predict(x\_test)

tree.plot\_tree(clf);

fn=['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']

cn=['setosa','versicolor','virginica']

fig,axes=plt.subplots(nrows=1,ncols=1,figsize=(10,10),dpi=300)

tree.plot\_tree(clf,

feature\_names=fn,

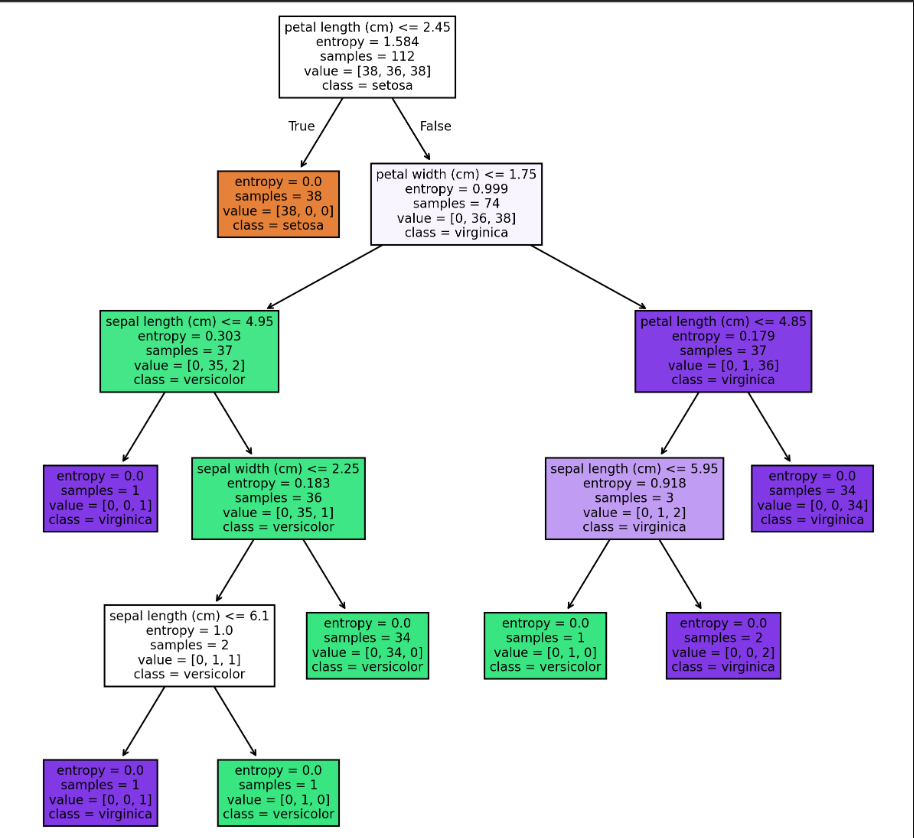
class\_names=cn,

filled=True);

fig.savefig('imagename.png')

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

Output:



Experiment-4: Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

1. Linear Regression

Program:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

from sklearn.metrics import mean\_squared\_error,r2\_score

data=pd.read\_csv('sales\_data.csv')

x=data[['week']]

y=data['sales']

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

model=LinearRegression()

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

user\_week=int(input("Enter the week number you want to predict sales for: "))

predicted\_sales=model.predict([[user\_week]])

print(f"Predicted sales for week {user\_week}:{predicted\_sales[0]}")

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

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

r2=r2\_score(y\_test,y\_pred)

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

print(f"R-squared:{r2}")

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

plt.plot(x,model.predict(x),color='red')

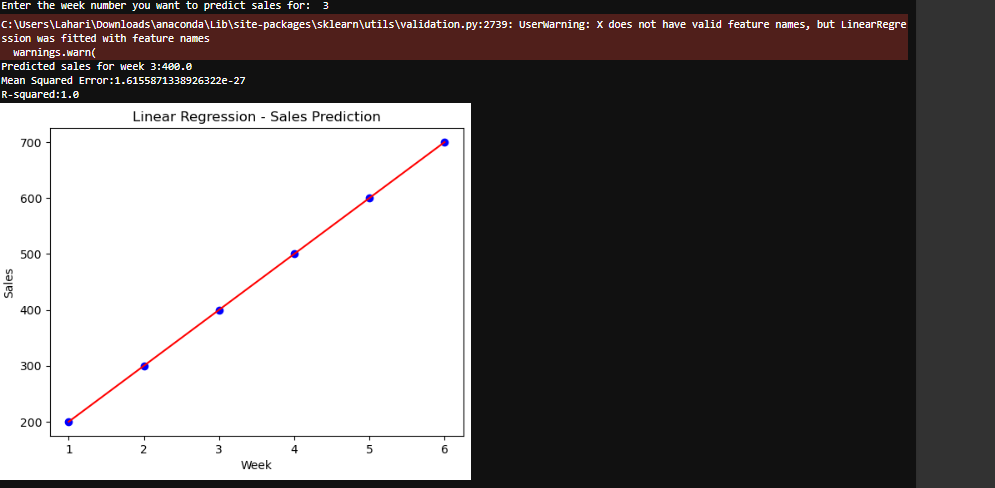
plt.xlabel('Week')

plt.ylabel('Sales')

plt.title('Linear Regression - Sales Prediction')

plt.show()

Output:



c) Binary Classifier

Program:

from numpy import where

from collections import Counter

from sklearn.datasets import make\_blobs

from matplotlib import pyplot

print(make\_blobs)

x,y=make\_blobs(n\_samples=1000,centers=2,random\_state=1)

print(x.shape,y.shape)

counter=Counter(y)

print(c ounter)

for i in range(10):

print(x[i],y[i])

for label,\_ in counter.items():

row\_ix=where(y ==label)[0]

if label == 0:

pyplot.scatter(x[row\_ix, 0],x[row\_ix, 1], label='Class 0', color='blue')

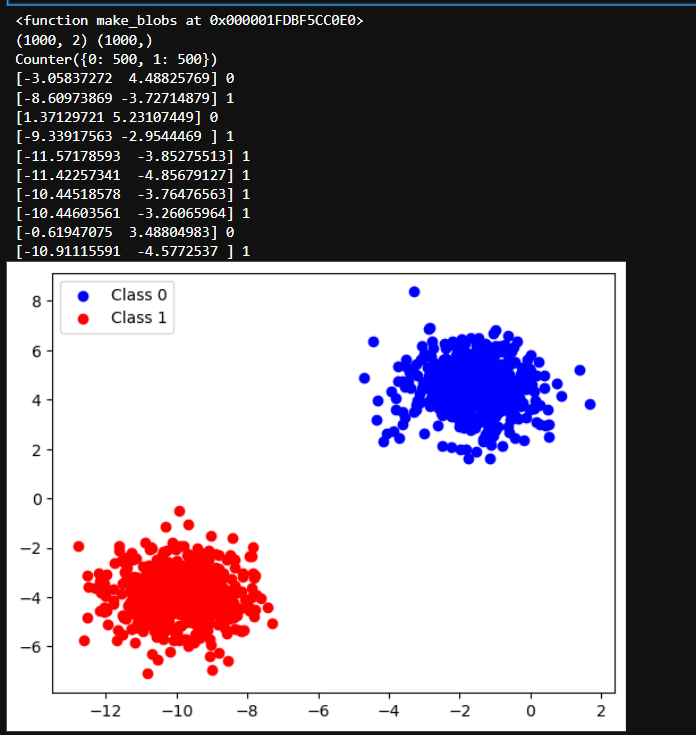
else:

pyplot.scatter(x[row\_ix, 0],x[row\_ix, 1], label='Class 1', color='red')

pyplot.legend()

pyplot.show()

Output:



Experiment-5: Develop a program for Bias, Variance, Remove duplicates , Cross Validation

Program:

Bias-Variance:

from mlxtend.evaluate import bias\_variance\_decomp

from sklearn.tree import DecisionTreeClassifier

from mlxtend.data import iris\_data

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

import numpy as np

# Load data

X, y = iris\_data()

# Split data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, shuffle=True)

# Convert to NumPy arrays

X\_train = np.array(X\_train)

X\_test = np.array(X\_test)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

# Define the classifier

tree = DecisionTreeClassifier()

# Calculate bias, variance, and expected loss

avg\_expected\_loss, avg\_bias, avg\_var = bias\_variance\_decomp(

tree, X\_train, y\_train, X\_test, y\_test, loss='0-1\_loss', num\_rounds=100

)

# Print results

print(f'Average Expected Loss: {round(avg\_expected\_loss, 4)}')

print(f'Average Bias: {round(avg\_bias, 4)}')

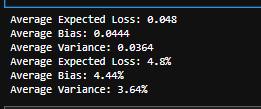
print(f'Average Variance: {round(avg\_var, 4)}')

print(f'Average Expected Loss: {round(avg\_expected\_loss \* 100, 2)}%')

print(f'Average Bias: {round(avg\_bias \* 100, 2)}%')

print(f'Average Variance: {round(avg\_var \* 100, 2)}%')

Output:



Remove Duplicates:

#FOR LIST

import pandas as pd

data={

"A":["TeamA","TeamB","TeamB","TeamC","TeamA"],

"B":[50,40,40,30,50],

"C":[True,False,False,False,True]

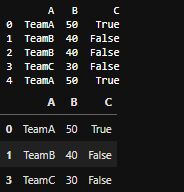
}

df=pd.DataFrame(data)

print(df)

display(df.drop\_duplicates())

Output:



#FOR DATASET

import pandas as pd

from mlxtend.data import iris\_data

x,y=iris\_data()

df\_iris=pd.DataFrame(x,columns=["Sepal\_Length","Sepal\_Width","Petal\_Length","Petal\_Width"])

df\_iris["Species"]=y

print("Original Iris DataFrame:")

print(df\_iris)

print(f"\n Total records in original dataframe:{len(df\_iris)}")

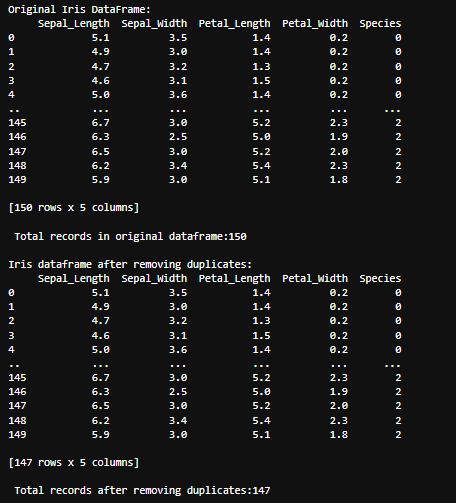
df\_iris\_no\_duplicates=df\_iris.drop\_duplicates()

print("\nIris dataframe after removing duplicates:")

print(df\_iris\_no\_duplicates)

print(f"\n Total records after removing duplicates:{len(df\_iris\_no\_duplicates)}")

Output:



Cross Validation:

from mlxtend.data import iris\_data

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import KFold,cross\_val\_score

x,y=iris\_data()

clf=DecisionTreeClassifier(random\_state=1)

k\_folds=KFold(n\_splits=6)

scores=cross\_val\_score(clf,x,y,cv=k\_folds)

print("Cross Validation:",scores)

print("Average CV Score:",scores.mean())

print("Number of CV Scores used in Average:",len(scores))

Output:



Experiment-6: Write a program to implement Categorical Encoding, One-hot Encoding

Program:

from numpy import asarray

from sklearn.preprocessing import OneHotEncoder

data=asarray([['red'],['green'],['blue']])

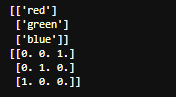
print(data)

encoder=OneHotEncoder(sparse\_output=False)

onehot=encoder.fit\_transform(data)

print(onehot)

Output:



Experiment-8:

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Program:

#import necessary modules

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

#load dataset

irisData=load\_iris()

#print(irisData)

#create feature and target arrays

x=irisData.data

y=irisData.target

#split into training and test set

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

knn=KNeighborsClassifier(n\_neighbors=7)

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

#predict on dataset which model has not seen before

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

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

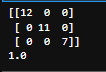
cm=confusion\_matrix(y\_test,y\_pred)

print(cm)

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

print(a)

Output:



Experiment-10:

Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to

perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Program:

import java.util.HashMap;

import java.util.Scanner;

public class SimpleNaiveBayes {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Training data: word frequencies in positive and negative texts

HashMap<String, Integer> positiveWords = new HashMap<>();

HashMap<String, Integer> negativeWords = new HashMap<>();

// Training: Manually adding some words

train(positiveWords, "good happy awesome nice great love excellent amazing wonderful positive");

train(negativeWords, "bad sad terrible awful hate worse horrible horrible negative ugly");

// Get user input

System.out.print("Enter a sentence to classify: ");

String testText = scanner.nextLine().toLowerCase(); // Convert input to lowercase

// Classify user input

classify(positiveWords, negativeWords, testText);

scanner.close();

}

// Training function: Counts word occurrences

public static void train(HashMap<String, Integer> category, String text) {

for (String word : text.split(" ")) {

category.put(word, category.getOrDefault(word, 0) + 1);

}

}

// Classification function: Predicts if the text is positive or negative

public static void classify(HashMap<String, Integer> pos, HashMap<String, Integer> neg, String text) {

double positiveScore = 1.0, negativeScore = 1.0; // Probability initialization

for (String word : text.split(" ")) {

positiveScore \*= (pos.getOrDefault(word, 1) + 1); // Add-one smoothing

negativeScore \*= (neg.getOrDefault(word, 1) + 1);

}

// Print result

if (positiveScore > negativeScore)

System.out.println("The text is POSITIVE 😊");

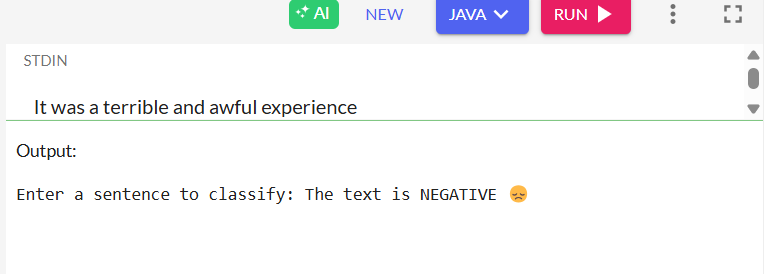
else

System.out.println("The text is NEGATIVE 😞");

}

}

Output:



Experiment-12: Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Program:

import numpy as nm

import seaborn as sns

import pandas as pd

data=pd.read\_csv('sales\_data.csv')

print(data)

print(data.head())

#print(data.tail())

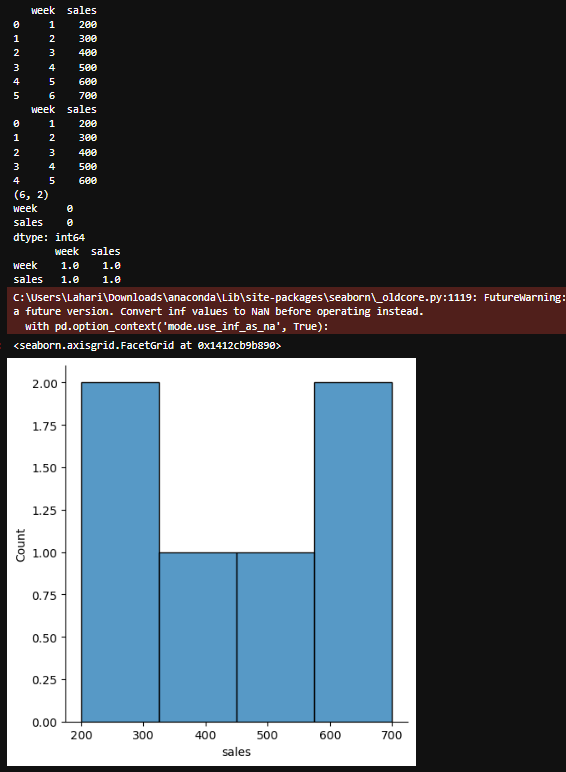
print(data.shape)

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

corelation=print(data.corr())

sns.displot(data['sales'])

Output:



Experiment-14:

Write a program to Implement Support Vector Machines and Principle Component Analysis.

Program:

from sklearn.datasets import load\_breast\_cancer

import matplotlib.pyplot as plt

from sklearn.svm import SVC

cancer=load\_breast\_cancer()

x=cancer.data[:, :2]

y=cancer.target

svm=SVC(kernel="linear")

svm.fit(x,y)

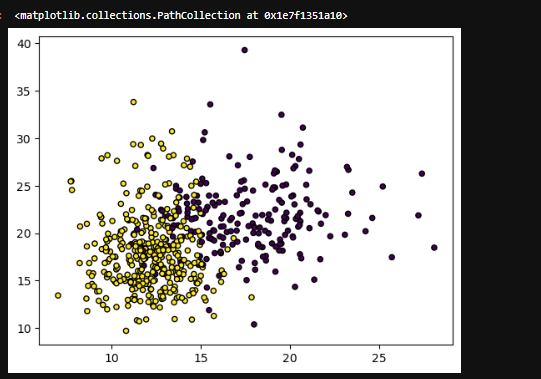
plt.scatter(x[:, 0],x[:, 1],

c=y,

s=20,edgecolors="k")

#plt.show()

Output:



Experiment-15:

Write a program to Implement Principle Component Analysis.

Program:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

from sklearn.datasets import load\_breast\_cancer

cancer=load\_breast\_cancer()

df=pd.DataFrame(cancer['data'],columns=cancer['feature\_names'])

print(cancer)

print(df.head())

print(df.shape)

from sklearn.preprocessing import StandardScaler

scalar=StandardScaler()

scalar.fit(df)

scaled\_data=scalar.transform(df)

from sklearn.decomposition import PCA

pca=PCA(n\_components=2)

pca.fit(scaled\_data)

x\_pca=pca.transform(scaled\_data)

print(x\_pca.shape)

plt.figure(figsize=(6,4))

plt.scatter(x\_pca[:, 0],x\_pca[:, 1],c=cancer['target'],cmap='plasma')

plt.xlabel('First Principal component')

plt.ylabel('Second Principal component')

df\_comp=pd.DataFrame(pca.components\_,columns=cancer['feature\_names'])

plt.figure(figsize=(14,6))

sns.heatmap(df\_comp,cmap='coolwarm')

plt.title('Heatmap of PCA Components')

plt.xlabel('Original Features')

plt.ylabel('Principal components')

plt.show()

Output:

{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,

1.189e-01],

[2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,

8.902e-02],

[1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,

8.758e-02],

...,

[1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,

7.820e-02],

[2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,

1.240e-01],

[7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,

7.039e-02]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,

1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,

1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,

1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,

0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,

1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,

1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,

0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,

1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,

1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,

0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,

0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,

1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,

1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,

1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,

1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,

1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,

1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]), 'frame': None, 'target\_names': array(['malignant', 'benign'], dtype='<U9'), 'DESCR': '.. \_breast\_cancer\_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset\n--------------------------------------------\n\n\*\*Data Set Characteristics:\*\*\n\n:Number of Instances: 569\n\n:Number of Attributes: 30 numeric, predictive attributes and the class\n\n:Attribute Information:\n - radius (mean of distances from center to points on the perimeter)\n - texture (standard deviation of gray-scale values)\n - perimeter\n - area\n - smoothness (local variation in radius lengths)\n - compactness (perimeter^2 / area - 1.0)\n - concavity (severity of concave portions of the contour)\n - concave points (number of concave portions of the contour)\n - symmetry\n - fractal dimension ("coastline approximation" - 1)\n\n The mean, standard error, and "worst" or largest (mean of the three\n worst/largest values) of these features were computed for each image,\n resulting in 30 features. For instance, field 0 is Mean Radius, field\n 10 is Radius SE, field 20 is Worst Radius.\n\n - class:\n - WDBC-Malignant\n - WDBC-Benign\n\n:Summary Statistics:\n\n===================================== ====== ======\n Min Max\n===================================== ====== ======\nradius (mean): 6.981 28.11\ntexture (mean): 9.71 39.28\nperimeter (mean): 43.79 188.5\narea (mean): 143.5 2501.0\nsmoothness (mean): 0.053 0.163\ncompactness (mean): 0.019 0.345\nconcavity (mean): 0.0 0.427\nconcave points (mean): 0.0 0.201\nsymmetry (mean): 0.106 0.304\nfractal dimension (mean): 0.05 0.097\nradius (standard error): 0.112 2.873\ntexture (standard error): 0.36 4.885\nperimeter (standard error): 0.757 21.98\narea (standard error): 6.802 542.2\nsmoothness (standard error): 0.002 0.031\ncompactness (standard error): 0.002 0.135\nconcavity (standard error): 0.0 0.396\nconcave points (standard error): 0.0 0.053\nsymmetry (standard error): 0.008 0.079\nfractal dimension (standard error): 0.001 0.03\nradius (worst): 7.93 36.04\ntexture (worst): 12.02 49.54\nperimeter (worst): 50.41 251.2\narea (worst): 185.2 4254.0\nsmoothness (worst): 0.071 0.223\ncompactness (worst): 0.027 1.058\nconcavity (worst): 0.0 1.252\nconcave points (worst): 0.0 0.291\nsymmetry (worst): 0.156 0.664\nfractal dimension (worst): 0.055 0.208\n===================================== ====== ======\n\n:Missing Attribute Values: None\n\n:Class Distribution: 212 - Malignant, 357 - Benign\n\n:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n:Donor: Nick Street\n\n:Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\[nhttps://goo.gl/U2Uwz2\n\nFeatures](nhttps://goo.gl/U2Uwz2\\n\\nFeatures" \t "http://localhost:8888/notebooks/ML%20LAB/_blank) are computed from a digitized image of a fine needle\naspirate (FNA) of a breast mass. They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a classification method which uses linear\nprogramming to construct a decision tree. Relevant features\nwere selected using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. dropdown:: References\n\n - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction\n for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on\n Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n San Jose, CA, 1993.\n - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and\n prognosis via linear programming. Operations Research, 43(4), pages 570-577,\n July-August 1995.\n - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)\n 163-171.\n', 'feature\_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',

'mean smoothness', 'mean compactness', 'mean concavity',

'mean concave points', 'mean symmetry', 'mean fractal dimension',

'radius error', 'texture error', 'perimeter error', 'area error',

'smoothness error', 'compactness error', 'concavity error',

'concave points error', 'symmetry error',

'fractal dimension error', 'worst radius', 'worst texture',

'worst perimeter', 'worst area', 'worst smoothness',

'worst compactness', 'worst concavity', 'worst concave points',

'worst symmetry', 'worst fractal dimension'], dtype='<U23'), 'filename': 'breast\_cancer.csv', 'data\_module': 'sklearn.datasets.data'}

mean radius mean texture mean perimeter mean area mean smoothness \

0 17.99 10.38 122.80 1001.0 0.11840

1 20.57 17.77 132.90 1326.0 0.08474

2 19.69 21.25 130.00 1203.0 0.10960

3 11.42 20.38 77.58 386.1 0.14250

4 20.29 14.34 135.10 1297.0 0.10030

mean compactness mean concavity mean concave points mean symmetry \

0 0.27760 0.3001 0.14710 0.2419

1 0.07864 0.0869 0.07017 0.1812

2 0.15990 0.1974 0.12790 0.2069

3 0.28390 0.2414 0.10520 0.2597

4 0.13280 0.1980 0.10430 0.1809

mean fractal dimension ... worst radius worst texture worst perimeter \

0 0.07871 ... 25.38 17.33 184.60

1 0.05667 ... 24.99 23.41 158.80

2 0.05999 ... 23.57 25.53 152.50

3 0.09744 ... 14.91 26.50 98.87

4 0.05883 ... 22.54 16.67 152.20

worst area worst smoothness worst compactness worst concavity \

0 2019.0 0.1622 0.6656 0.7119

1 1956.0 0.1238 0.1866 0.2416

2 1709.0 0.1444 0.4245 0.4504

3 567.7 0.2098 0.8663 0.6869

4 1575.0 0.1374 0.2050 0.4000

worst concave points worst symmetry worst fractal dimension

0 0.2654 0.4601 0.11890

1 0.1860 0.2750 0.08902

2 0.2430 0.3613 0.08758

3 0.2575 0.6638 0.17300

4 0.1625 0.2364 0.07678

[5 rows x 30 columns]

(569, 30)

(569, 2)

