**Exercise 4 : Exploring Regression Performance with kernel functions**

**Date: 28/10/2024**

**AIM:**

To explore regression performance with different kernel functions and analyze the impact of kernel matrix operations on dataset patterns and relationships.

**CODE:**

from sklearn.datasets import load\_breast\_cancer

import matplotlib.pyplot as plt

from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.svm import SVC,SVR

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

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

cancer=load\_breast\_cancer()

cancer.keys()

dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename', 'data\_module'])

X=cancer.data[:,:2]

y=cancer.target

svm = SVC(kernel="rbf", gamma=0.5, C=1.0)

svm.fit(X, y)

DecisionBoundaryDisplay.from\_estimator(svm,X,response\_method="predict",cmap=plt.cm.Spectral,alpha=0.8,xlabel=cancer.feature\_names[0],ylabel=cancer.feature\_names[1],)

plt.scatter(X[:,0], X[:,1], c=y,s=20, edgecolors="k")

plt.show()

**OUTPUT:**

A blue and yellow dotted diagram

Description automatically generated

X=np.sort(5\*np.random.rand(40,1),axis=0)

y=np.sin(X).ravel()

y[::5]+=3\*(0.5-np.random.rand(8))

svr=SVR(kernel='poly')

svr.fit(X,y)

y\_pred = svr.predict(X)

plt.scatter(X,y,color='darkorange',label='data')

plt.plot(X,y\_pred,color='cornflowerblue',label='prediction')

plt.legend()

plt.show()

**OUTPUT:**

A line graph with orange dots

Description automatically generated

**CODE:**

data=pd.read\_csv("/content/data.csv")

data=data.drop(columns=['id'],axis=1)

data.info()

A screenshot of a computer screen

Description automatically generated

**CODE:**

data['diagnosis']=data['diagnosis'].map({'M':1,'B':0})

X=data.drop('diagnosis',axis=1)

y=data['diagnosis']

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=42)

scaler=StandardScaler()

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

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

rbf\_model=SVC(kernel='rbf')

rbf\_model.fit(X\_train, y\_train)

y\_pred=rbf\_model.predict(X\_test)

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

print(classification\_report(y\_test, y\_pred))

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

print(f'Accuracy: {accuracy:.4f}')

**OUTPUT:**

A screenshot of a computer screen

Description automatically generated

**CODE:**

linear\_model=SVC(kernel='linear')

linear\_model.fit(X\_train, y\_train)

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

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

print(classification\_report(y\_test, y\_pred))

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

print(f'Accuracy: {accuracy:.4f}')

**OUTPUT:**

A screenshot of a computer screen

Description automatically generated

**CODE:**

poly\_model=SVC(kernel='poly')

poly\_model.fit(X\_train, y\_train)

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

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

print(classification\_report(y\_test, y\_pred))

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

print(f'Accuracy: {accuracy:.4f}')

**OUTPUT:**

A screenshot of a computer screen

Description automatically generated

**CODE:**

sigmoid\_model=SVC(kernel='sigmoid')

sigmoid\_model.fit(X\_train, y\_train)

y\_pred=sigmoid\_model.predict(X\_test)

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

print(classification\_report(y\_test, y\_pred))

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

print(f'Accuracy: {accuracy:.4f}')

**OUTPUT:**

A screenshot of a graph

Description automatically generated

**CODE:**

def gaussian\_kernel\_matrix(X, sigma):

distances=np.sum((X[:,np.newaxis]-X)\*\*2,axis=-1)

kernel\_matrix=np.exp(-distances/(2 \*sigma\*\*2))

return kernel\_matrix

sigma=1.00

**CODE:**

X=data.values

kernel\_matrix=gaussian\_kernel\_matrix(X,sigma)

print(x.shape,kernel\_matrix.shape)

**OUTPUT:**



**CODE:**

print("Kernel matrix:")

print(kernel\_matrix)

**OUTPUT:**

A grid of numbers and letters

Description automatically generated

**RESULT:**

Regression analysis revealed varying performance across linear, polynomial, and RBF kernels, with kernel matrix operations highlighting distinct relationships within the data, aiding in understanding each kernel’s effectiveness.

**Exercise 5 : Exploring Dimensionality Reduction Techniques**

**Date : 9/10/2024**

**AIM:**

To implement the Page Rank algorithm to determine the importance of web pages based on the quantity and quality of links

**CODE:**

import pandas as pd

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_boston

from sklearn.linear\_model import LinearRegression

from sklearn.feature\_selection import RFE

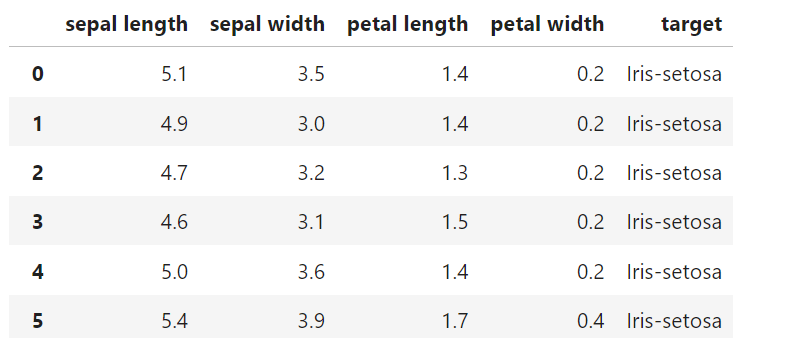
# loading dataset into Pandas DataFrame

iris\_url = "http4s://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

df\_iris = pd.read\_csv(iris\_url, names=['sepal length','sepal width','petal length','petal width','target'])

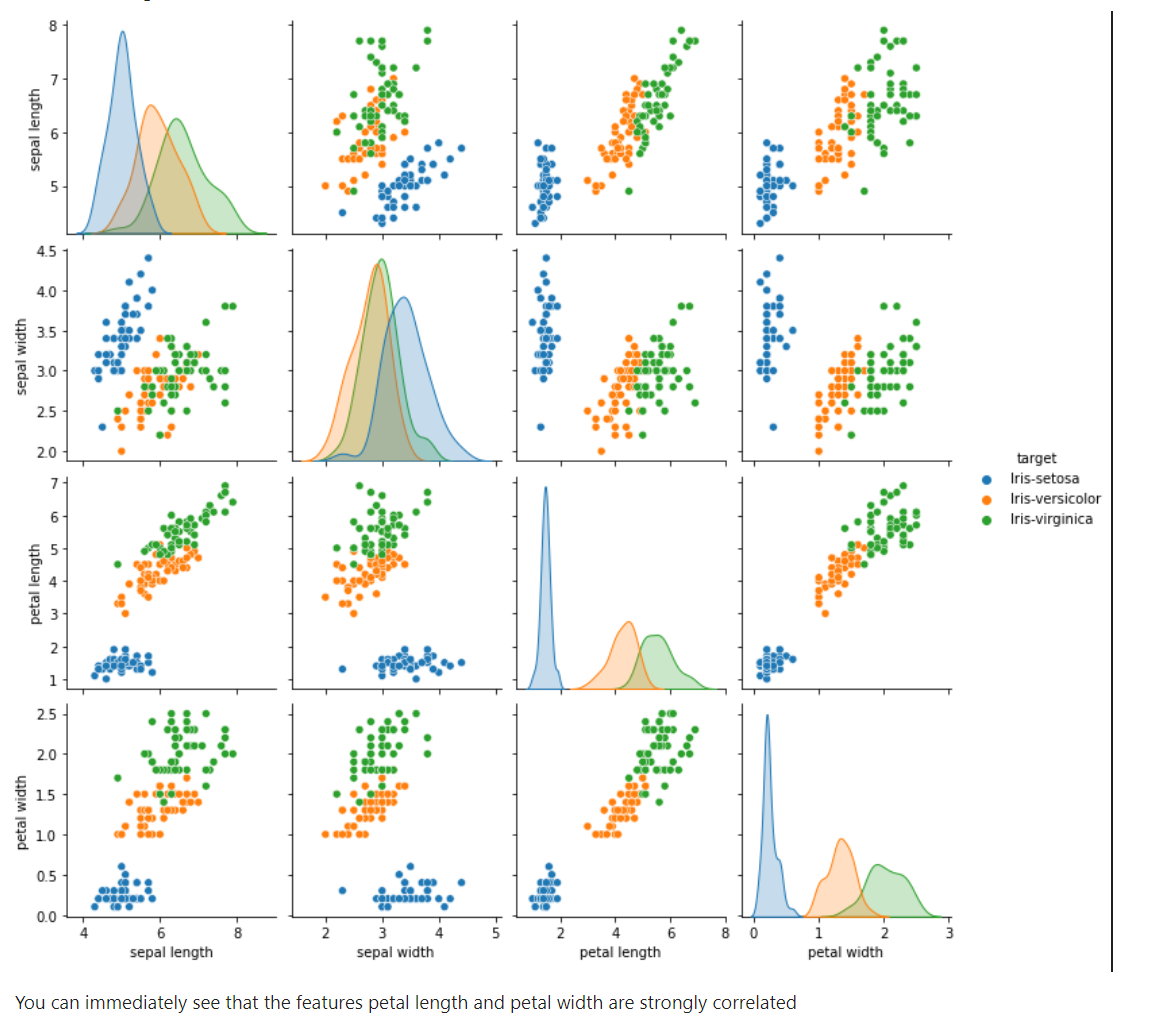
df\_iris.head(15)

**OUTPUT:**



**CODE:**

sns.pairplot(df\_iris, hue='target')



**CODE:**

features\_iris = ['sepal length', 'sepal width', 'petal length', 'petal width']

x\_iris = df\_iris.loc[:, features\_iris].values

y\_iris = df\_iris.loc[:, ['target']].values

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

df\_iris\_standarize = pd.DataFrame(data = x\_iris, columns = features\_iris)

df\_iris\_standarize['target'] = df\_iris['target']

df\_iris\_standarize.head(15)

**PCA Projection to 2D:**

pca\_iris = PCA(n\_components=2)

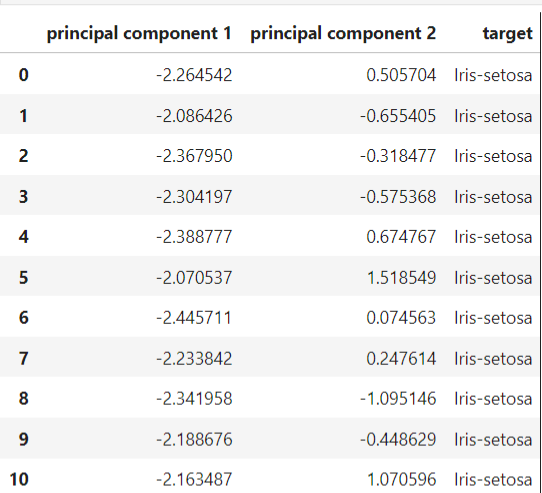
principalComponents\_iris = pca\_iris.fit\_transform(x\_iris)

principalDf\_iris = pd.DataFrame(data = principalComponents\_iris ,columns = ['principal component 1', 'principal component 2'])

finalDf\_iris = pd.concat([principalDf\_iris, df\_iris[['target']]], axis = 1)

finalDf\_iris.head(15)

**OUTPUT:**



**CODE:**

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Principal Component 1', fontsize = 15)

ax.set\_ylabel('Principal Component 2', fontsize = 15)

ax.set\_title('2 Component PCA', fontsize = 20)

iris\_targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

colors = ['r', 'g', 'b']

for target, color in zip(iris\_targets,colors):

indicesToKeep = finalDf\_iris['target'] == target

ax.scatter(finalDf\_iris.loc[indicesToKeep, 'principal component 1']

, finalDf\_iris.loc[indicesToKeep, 'principal component 2']

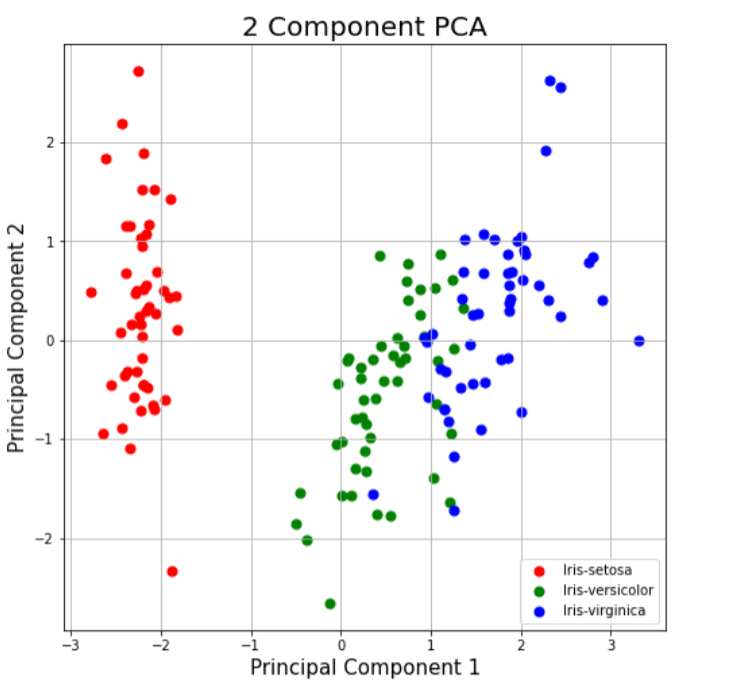
, c = color

, s = 50)

ax.legend(iris\_targets)

ax.grid()

**OUTPUT:**



**CODE:**

pca\_iris.explained\_variance\_ratio\_

from sklearn.datasets import load\_breast\_cancer

breast\_cancer = load\_breast\_cancer()

print(f"Breast cancer dataset type: {type(breast\_cancer)}")

# sklearn.utils.Bunch -> pandas.DataFrame

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

df\_bc['target'] = breast\_cancer.target

df\_bc['target'] = df\_bc['target'].apply(lambda x: 'benign' if x == 0 else 'malignant')

df\_bc.head()

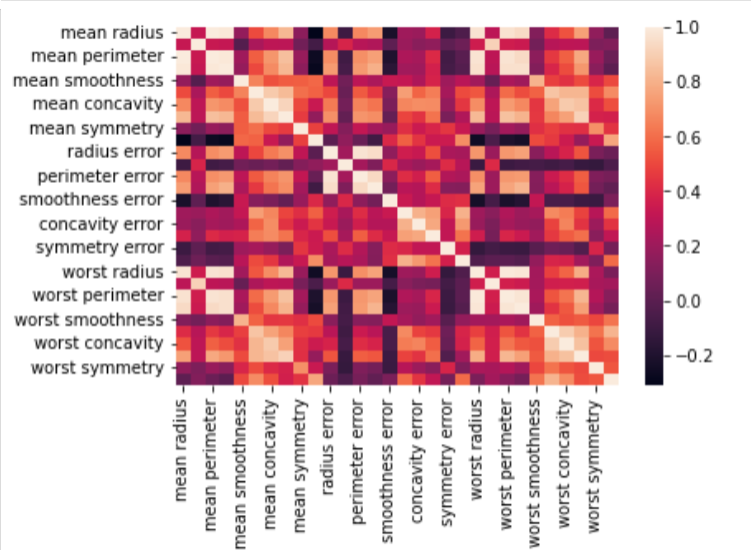
**OUTPUT:**

A table with numbers and text

Description automatically generated

**CODE:**

dataplot = sns.heatmap(df\_bc.corr())



**CODE:**

# pandas.core.indexes.base.Index -> List

features\_breast\_cancer = df\_bc.columns.tolist()

features\_breast\_cancer = features\_breast\_cancer[:-1] # Skip target

x\_breast\_cancer = df\_bc.loc[:, features\_breast\_cancer].values

y\_breast\_cancer = df\_bc.loc[:, ['target']].values

# Standarization

print(type(x\_breast\_cancer))

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

# Werifization of standarization - standard deviation == 1 and mean == 0.

x\_breast\_cancer.std(axis=0)

df\_bc\_standarize = pd.DataFrame(data=x\_breast\_cancer, columns=features\_breast\_cancer)

df\_bc\_standarize['target'] = df\_bc['target']

**Perform PCA to reduce dimensionality of breast cancer dataset:**

**CODE:**

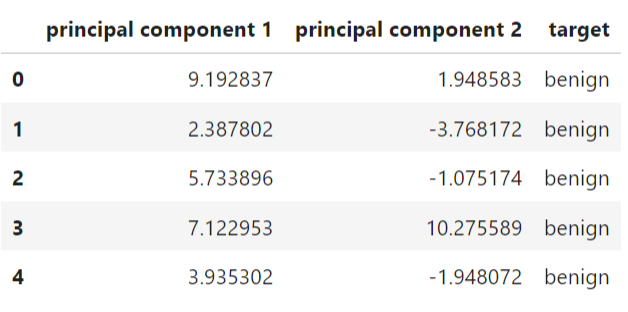
principalComponents\_breast\_cancer= pca\_breast\_cancer.fit\_transform(x\_breast\_cancer)

principalDf\_breast\_cancer\_columns = ['principal component 1', 'principal component 2']

principalDf\_breast\_cancer = pd.DataFrame(data=principalComponents\_breast\_cancer, columns=principalDf\_breast\_cancer\_columns)

finalDf\_breast\_cancer = pd.concat([principalDf\_breast\_cancer, df\_bc[['target']]], axis=1)

finalDf\_breast\_cancer.head()



**CODE:**

variance\_ratio = pca\_breast\_cancer.explained\_variance\_ratio\_

plt.title("Projection by variance")

plt.plot(variance\_ratio, 'ro')

plt.xlabel('Number of components')

plt.ylabel('Cumulative explained variance')

plt.show()

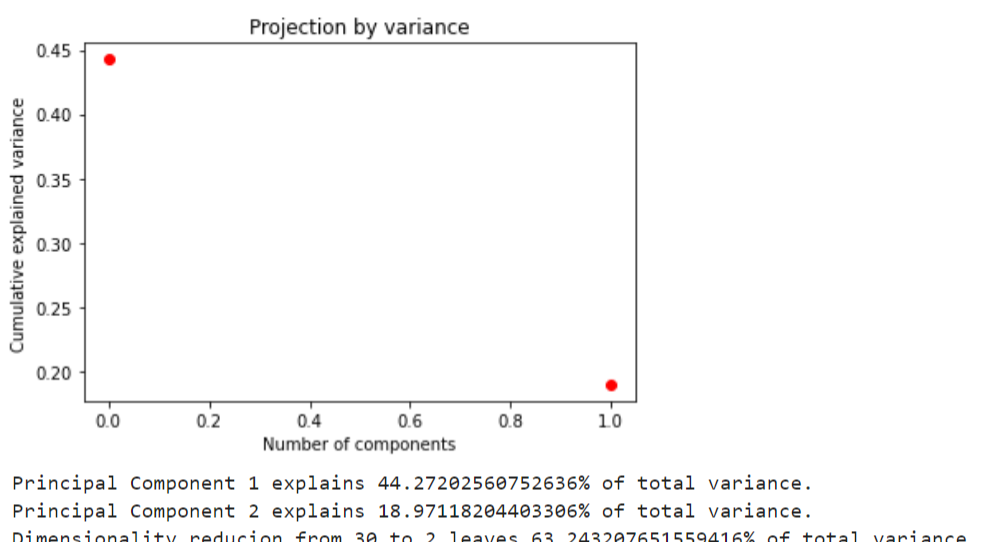
print(f"Principal Component 1 explains {variance\_ratio[0] \* 100}% of total variance.")

print(f"Principal Component 2 explains {variance\_ratio[1] \* 100}% of total variance.")

print(f"Dimensionality reducion from {len(df\_bc.columns)-1} to {len(variance\_ratio)} "

f"leaves {sum(variance\_ratio) \* 100}% of total variance.")

**OUTPUT:**



**CODE:**

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Principal Component 1', fontsize = 15)

ax.set\_ylabel('Principal Component 2', fontsize = 15)

ax.set\_title('2 Component PCA', fontsize = 20)

breast\_cancer\_targets = ['benign', 'malignant']

colors = ['r', 'g']

for target, color in zip(breast\_cancer\_targets, colors):

indicesToKeep = finalDf\_breast\_cancer['target'] == target

ax.scatter(finalDf\_breast\_cancer.loc[indicesToKeep, 'principal component 1'],

finalDf\_breast\_cancer.loc[indicesToKeep, 'principal component 2'],

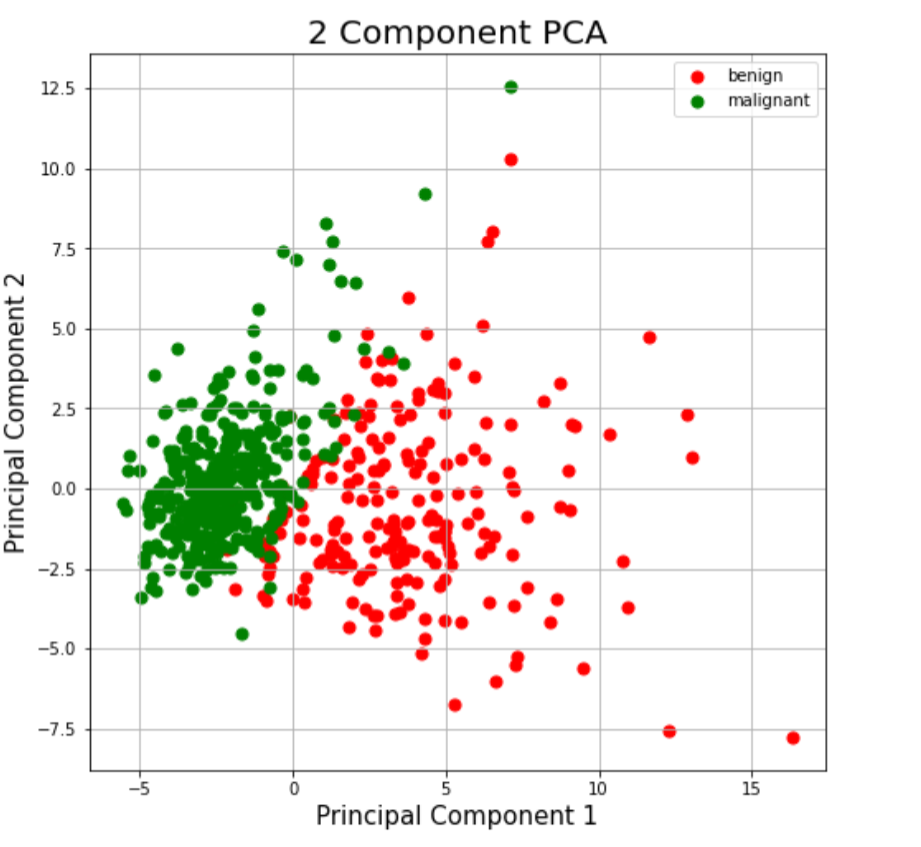
c = color,

s = 50)

ax.legend(breast\_cancer\_targets)

ax.grid()

**OUTPUT:**



**Kernel PCA:**

import matplotlib.pyplot as plt

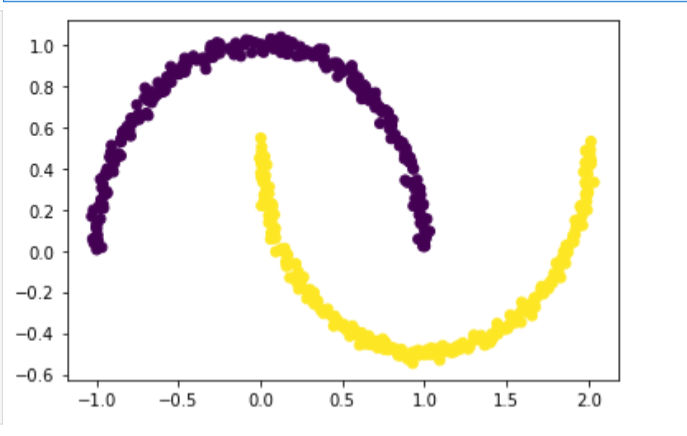
from sklearn.datasets import make\_moons

X, y = make\_moons(n\_samples = 500, noise = 0.02, random\_state = 417)

plt.scatter(X[:, 0], X[:, 1], c = y)

plt.show()

**OUTPUT:**



**CODE:**

pca = PCA(n\_components = 2)

X\_pca = pca.fit\_transform(X)

plt.title("PCA")

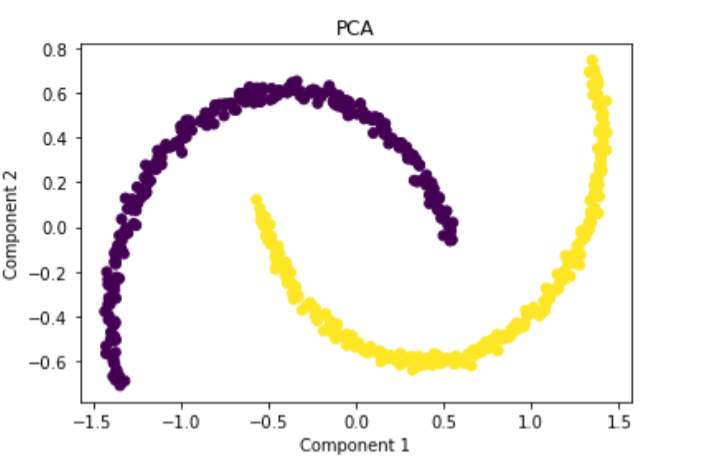
plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c = y)

plt.xlabel("Component 1")

plt.ylabel("Component 2")

plt.show()

**OUTPUT:**



from sklearn.decomposition import KernelPCA

kpca = KernelPCA(kernel ='rbf', gamma = 15)

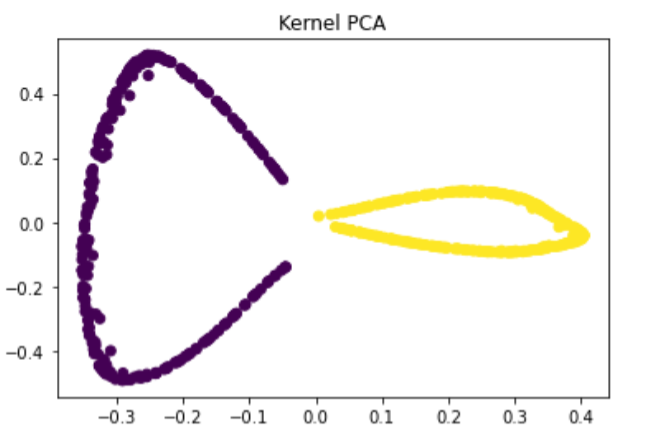
X\_kpca = kpca.fit\_transform(X)

plt.title("Kernel PCA")

plt.scatter(X\_kpca[:, 0], X\_kpca[:, 1], c = y)

plt.show()

**OUTPUT:**



**CODE:**

from sklearn.decomposition import KernelPCA

kpca = KernelPCA(kernel ='rbf', gamma = 5)

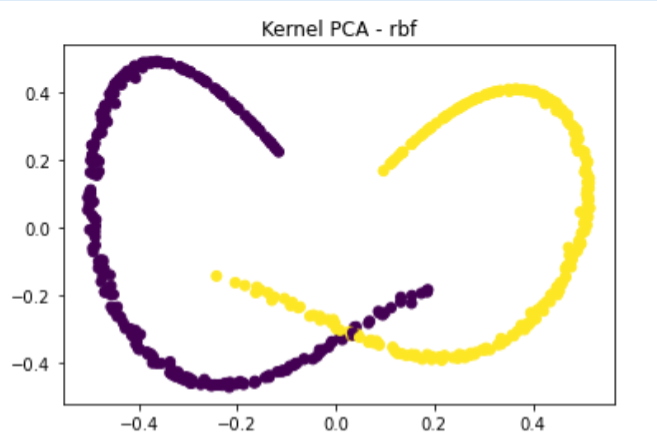
X\_kpca = kpca.fit\_transform(X)

plt.title("Kernel PCA - rbf")

plt.scatter(X\_kpca[:, 0], X\_kpca[:, 1], c = y)

plt.show()

**OUTPUT:**



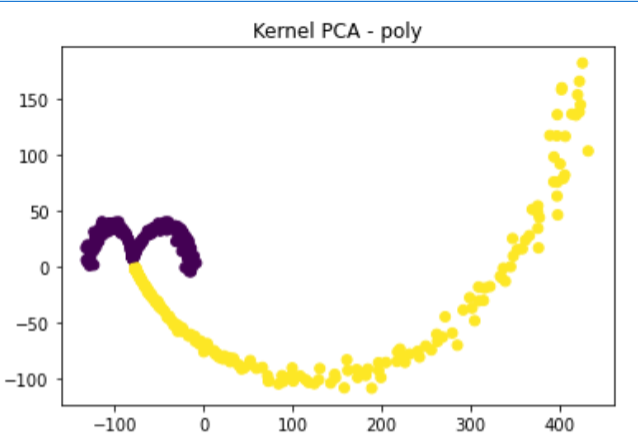
kpca = KernelPCA(kernel ='poly', gamma=15)

X\_kpca = kpca.fit\_transform(X)

plt.title("Kernel PCA - poly")

plt.scatter(X\_kpca[:, 0], X\_kpca[:, 1], c = y)

plt.show()



**RESULT:**

The Page Rank algorithm successfully assigned importance scores to each web page based on link structure.

**Exercise 7:Analyzing Graph Data and Identifying Communities**

**Date: 28/09/2024**

**AIM:**

To analyze graph data by calculating centrality measures, identifying key nodes, and detecting communities, helping to understand the graph’s structure and influential nodes.

**Code:**

import pandas as pd

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

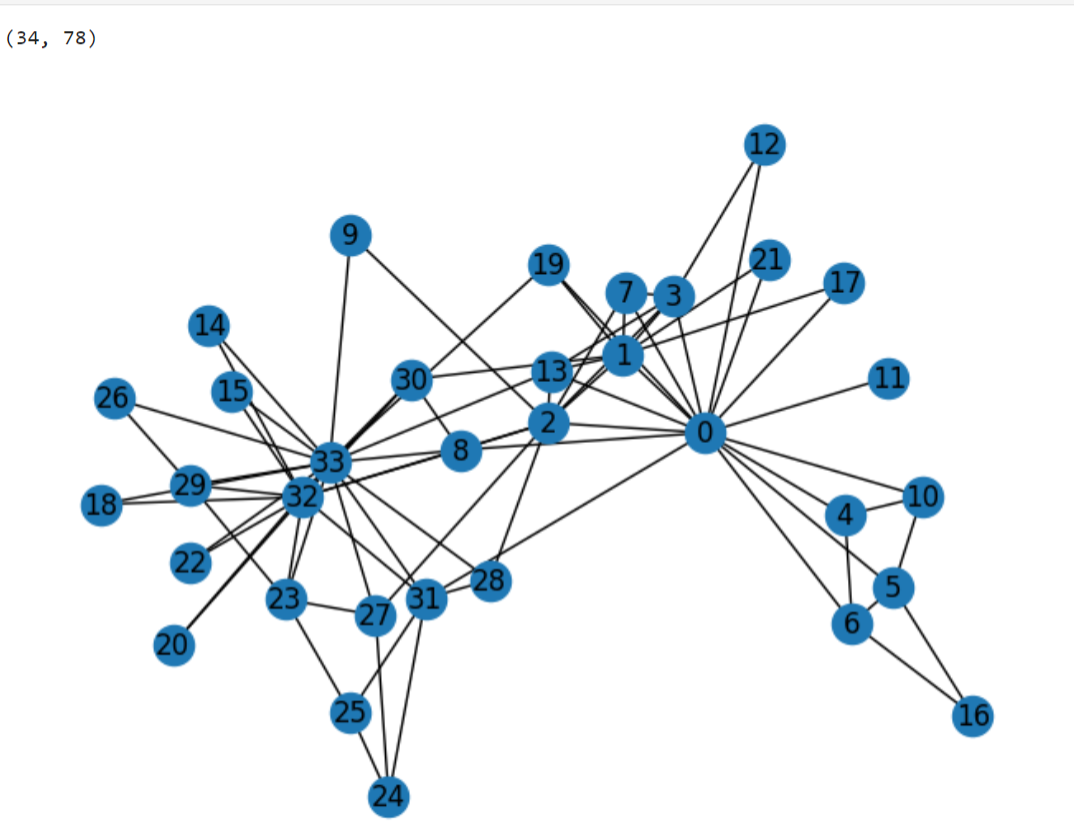
import random

G=nx.karate\_club\_graph()

nx.draw(G,with\_labels=True)

len(G.nodes),len(G.edges)

**OUTPUT:**



**COMMUNITY DETECTION USING GIRVAN-NEWMAN ARCHITECTURE:**

def edge\_to\_remove(graph):

edge\_betweenness = nx.edge\_betweenness\_centrality(graph)

return max(edge\_betweenness, key=edge\_betweenness.get)

def girvan\_newman(graph):

sg\_count = nx.number\_connected\_components(graph)

while sg\_count == 1:

edge = edge\_to\_remove(graph)

graph.remove\_edge(\*edge)

sg\_count = nx.number\_connected\_components(graph)

return nx.connected\_components(graph)

c=girvan\_newman(G.copy())

node\_groups = []

for i in c:

node\_groups.append(list(i))

node\_groups

color\_map = []

for node in G:

if node in node\_groups[0]:

color\_map.append('red')

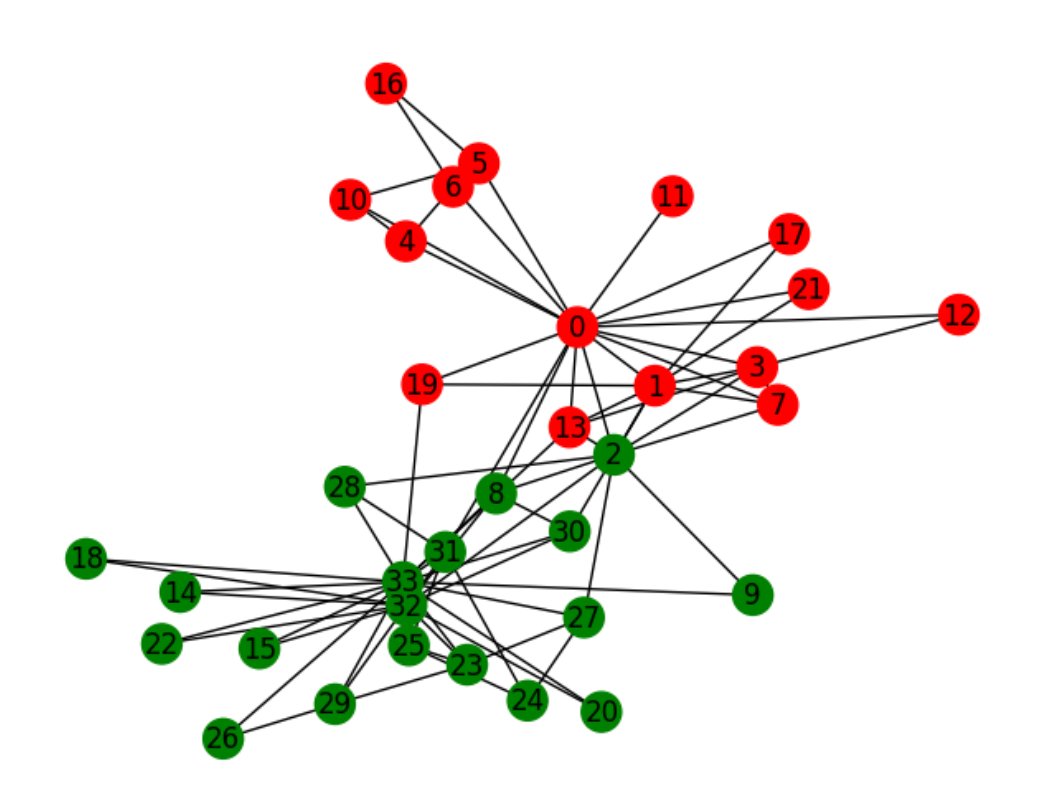
else:

color\_map.append('green')

nx.draw(G, node\_color=color\_map, with\_labels=True)

plt.show()

**OUTPUT:**



**MODULARITY OF COMMUNITY DETECTION**

from networkx.algorithms.community import modularity

c = girvan\_newman(G.copy())

node\_groups = []

for community in c:

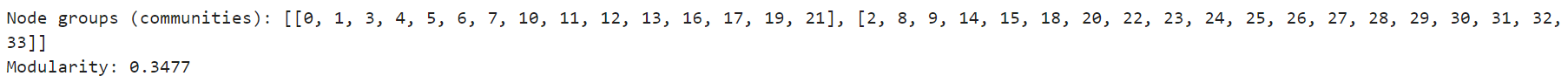
node\_groups.append(list(community))

mod\_value = modularity(G, node\_groups)

print(f"Node groups (communities): {node\_groups}")

print(f"Modularity: {mod\_value:.4f}")

**OUTPUT:**



**CODE:**

color\_map = []

for node in G:

if node in node\_groups[0]:

color\_map.append('red')

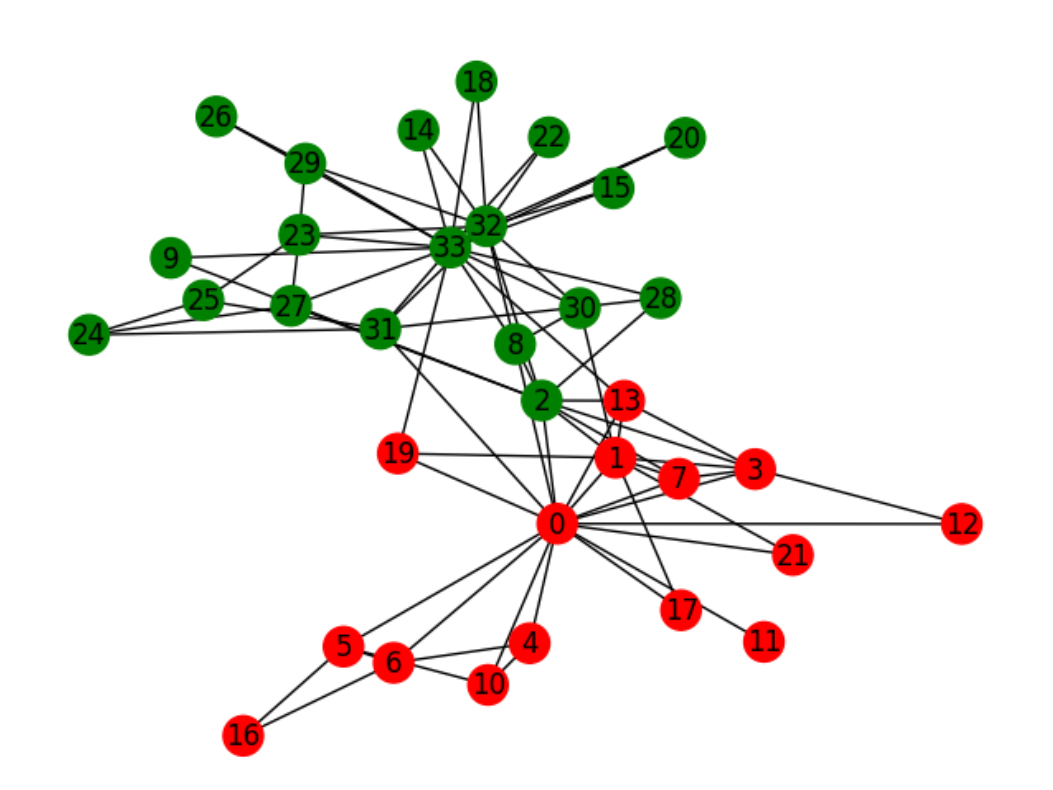
else:

color\_map.append('green')

nx.draw(G, node\_color=color\_map, with\_labels=True)

plt.show()

**OUTPUT:**



**CALCULATION OF DEGREE CENTRALITY**

degree\_centrality=nx.degree\_centrality(G)

ranked\_nodes=sorted(degree\_centrality.items(),key=lambda x:x[1],reverse=True)

print("Nodes ranked by degree centrality:")

for node,centrality in ranked\_nodes:

print(f"Node {node}: {centrality:.4f}")

highest\_centrality\_node=ranked\_nodes[0]

print(f"\nNode with the highest degree centrality: Node {highest\_centrality\_node[0]} with centrality {highest\_centrality\_node[1]:.4f}")

print("\nThe node with the highest centrality might play a central role in connecting many other nodes. It could indicate that this node is a hub or leader within the network, influencing communication or interactions between others.")

**OUTPUT:**

Nodes ranked by degree centrality:

Node 33: 0.5152

Node 0: 0.4848

Node 32: 0.3636

Node 2: 0.3030

Node 1: 0.2727

Node 3: 0.1818

Node 31: 0.1818

Node 8: 0.1515

Node 13: 0.1515

Node 23: 0.1515

Node 5: 0.1212

Node 6: 0.1212

Node 7: 0.1212

Node 27: 0.1212

Node 29: 0.1212

Node 30: 0.1212

Node 4: 0.0909

Node 10: 0.0909

Node 19: 0.0909

Node 24: 0.0909

Node 25: 0.0909

Node 28: 0.0909

Node 9: 0.0606

Node 12: 0.0606

Node 14: 0.0606

Node 15: 0.0606

Node 16: 0.0606

Node 17: 0.0606

Node 18: 0.0606

Node 20: 0.0606

Node 21: 0.0606

Node 22: 0.0606

Node 26: 0.0606

Node 11: 0.0303

Node with the highest degree centrality: Node 33 with centrality 0.5152

The node with the highest centrality might play a central role in connecting many other nodes. It could indicate that this node is a hub or leader within the network, influencing communication or interactions between others.

**CALCULATION OF BETWEENNESS CENTRALITY:**

betweenness\_centrality=nx.betweenness\_centrality(G)

ranked\_nodes=sorted(betweenness\_centrality.items(),key=lambda x: x[1],reverse=True)

print("Top 5 nodes ranked by betweenness centrality:")

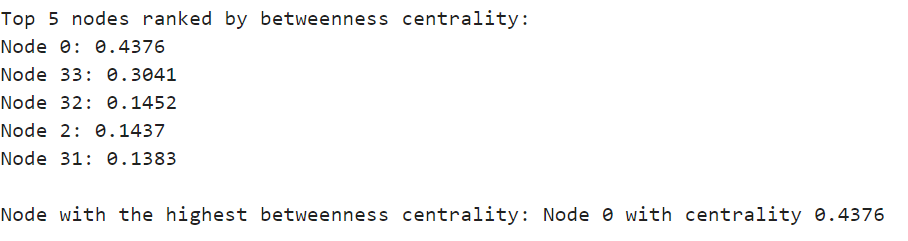
for i in range(5):

node,centrality=ranked\_nodes[i]

print(f"Node {node}: {centrality:.4f}")

highest\_betweenness\_node=ranked\_nodes[0]

print(f"\nNode with the highest betweenness centrality: Node {highest\_betweenness\_node[0]} with centrality {highest\_betweenness\_node[1]:.4f}")



**RESULT:**

Centrality analysis identified key nodes with high influence, while community detection revealed distinct groups within the network

**Exercise No 8:Implementation of Page Rank Algorithm**

**Date:28/09/2024**

**AIM:**

To implement the Page Rank algorithm to rank web pages by analyzing link structure and incorporating teleportation to simulate user behavior.

**A screenshot of a math test

Description automatically generated**

**CODE:**

import networkx as nx

import matplotlib.pyplot as plt

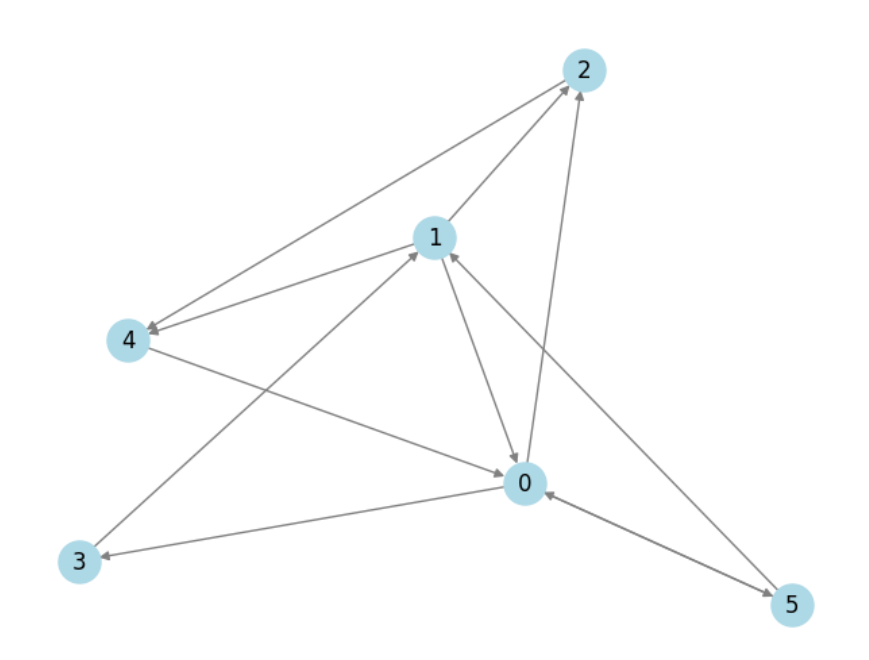
n = 6

p = 0.4

G = nx.erdos\_renyi\_graph(n, p, directed=True)

nx.draw(G, with\_labels=True, node\_color='lightblue', node\_size=500, font\_size=12, edge\_color='gray')

**OUTPUT:**



**CODE:**

adj\_matrix = nx.adjacency\_matrix(G).todense()

print("Adjacency Matrix:\n", adj\_matrix)

A number of numbers in a row

Description automatically generated with medium confidence

**CODE:**

def page\_rank(graph, damping\_factor=0.85, max\_iterations=100, tolerance=1e-6):

num\_nodes = len(graph)

initial\_pr = 1.0 / num\_nodes

page\_rank = {node: initial\_pr for node in graph}

out\_degrees = {node: len(graph[node]) for node in graph}

for i in range(max\_iterations):

# print(f"Iteration : {i} ")

# print(f"Page Rank : {page\_rank}")

prev\_page\_rank = page\_rank.copy()

total\_diff = 0.0

for node in graph:

page\_rank[node] = (1 - damping\_factor) / num\_nodes

for neighbor in graph[node]:

page\_rank[node] += damping\_factor \* prev\_page\_rank[neighbor] / out\_degrees[neighbor]

diff = abs(page\_rank[node] - prev\_page\_rank[node])

total\_diff += diff

if total\_diff < tolerance:

break

return page\_rank

**CODE:**

A close-up of a text

Description automatically generated

def page\_rank(graph, damping\_factor=0.85, max\_iterations=100, tolerance=1e-06):

num\_nodes = len(graph)

page\_rank = {node: 1 / num\_nodes for node in graph}

prev\_page\_rank = page\_rank.copy()

out\_degrees = {node: len(list(graph.successors(node))) for node in graph}

for i in range(max\_iterations):

print(f"Iteration : {i} ")

print(f"Page Rank : {page\_rank}")

for node in graph:

page\_rank[node] = (1 - damping\_factor) / num\_nodes

for neighbor in graph.predecessors(node):

if out\_degrees[neighbor] > 0:

page\_rank[node] += damping\_factor \* prev\_page\_rank[neighbor] / out\_degrees[neighbor]

else:

page\_rank[node] += damping\_factor \* prev\_page\_rank[neighbor] / num\_nodes

diff = sum(abs(page\_rank[node] - prev\_page\_rank[node]) for node in graph)

if diff < tolerance:

break

prev\_page\_rank = page\_rank.copy()

return page\_rank

page\_rank(G)

**OUTPUT:**

Iteration : 0

Page Rank : {0: 0.16666666666666666, 1: 0.16666666666666666, 2: 0.16666666666666666, 3: 0.16666666666666666, 4: 0.16666666666666666, 5: 0.16666666666666666}

Iteration : 1

Page Rank : {0: 0.2847222222222222, 1: 0.2375, 2: 0.11944444444444445, 3: 0.07222222222222223, 4: 0.2138888888888889, 5: 0.07222222222222223}

Iteration : 2

Page Rank : {0: 0.3047916666666667, 1: 0.11708333333333334, 2: 0.17296296296296296, 3: 0.10567129629629629, 4: 0.19381944444444446, 5: 0.10567129629629629}

Iteration : 3

Page Rank : {0: 0.2678304398148148, 1: 0.15973090277777777, 2: 0.14453125, 3: 0.11135763888888889, 4: 0.20519212962962963, 5: 0.11135763888888889}

Iteration : 4

Page Rank : {0: 0.29199739583333334, 1: 0.16698098958333335, 2: 0.14614238040123456, 3: 0.10088529128086421, 4: 0.19310865162037036, 5: 0.10088529128086421}

Iteration : 5

Page Rank : {0: 0.27932988305362655, 1: 0.15362874638310187, 2: 0.15504387586805557, 3: 0.10773259548611111, 4: 0.19653230372299382, 5: 0.10773259548611111}

Iteration : 6

Page Rank : {0: 0.28136695605468753, 1: 0.1623590592447917, 2: 0.1476716116737397, 3: 0.1041434668651942, 4: 0.20031543929639278, 5: 0.1041434668651942}

Iteration : 7

Page Rank : {0: 0.2855308302723324, 1: 0.1577829202531226, 2: 0.1507223710015191, 3: 0.10472063754882814, 4: 0.19652260337536973, 5: 0.10472063754882814}

Iteration : 8

Page Rank : {0: 0.28125564456570096, 1: 0.1585188128747559, 2: 0.15060556264887892, 3: 0.10590040191049419, 4: 0.19781917608967597, 5: 0.10590040191049419}

Iteration : 9

Page Rank : {0: 0.2830676341360321, 1: 0.16002301243588007, 2: 0.14960276294146277, 3: 0.10468909929361528, 4: 0.1979283918993946, 5: 0.10468909929361528}

Iteration : 10

Page Rank : {0: 0.2830718538377713, 1: 0.15847860159935948, 2: 0.15054234986204176, 3: 0.10520249633854242, 4: 0.1975022020237427, 5: 0.10520249633854242}

Iteration : 11

Page Rank : {0: 0.2824902031172136, 1: 0.15913318283164157, 2: 0.1501059623738537, 3: 0.10520369192070186, 4: 0.19786326783588737, 5: 0.10520369192070186}

Iteration : 12

Page Rank : {0: 0.28298308186243437, 1: 0.1591347071988949, 2: 0.1501266260188423, 3: 0.10503889088321053, 4: 0.19767780315340744, 5: 0.10503889088321053}

Iteration : 13

Page Rank : {0: 0.28275582834544766, 1: 0.15892458587609343, 2: 0.15026670690070995, 3: 0.10517853986102307, 4: 0.19769579915570284, 5: 0.10517853986102307}

Iteration : 14

Page Rank : {0: 0.2827709413881754, 1: 0.15910263832280444, 2: 0.15014278402943665, 3: 0.10511415136454351, 4: 0.19775533353049657, 5: 0.10511415136454351}

Iteration : 15

Page Rank : {0: 0.282844628688981, 1: 0.15902054298979298, 2: 0.1501975142514443, 3: 0.10511843339331636, 4: 0.19770044728314906, 5: 0.10511843339331636}

Iteration : 16

Page Rank : {0: 0.2827765348966108, 1: 0.15902600257647836, 2: 0.15019513197565265, 3: 0.10513931146187797, 4: 0.19772370762750233, 5: 0.10513931146187797}

Iteration : 17

Page Rank : {0: 0.28280672625134395, 1: 0.15905262211389443, 2: 0.1501773856173753, 3: 0.10512001822070641, 4: 0.19772322957597363, 5: 0.10512001822070641}

Iteration : 18

Page Rank : {0: 0.28280566248231453, 1: 0.15902802323140067, 2: 0.15019348203681754, 3: 0.10512857243788079, 4: 0.19771568737370576, 5: 0.10512857243788079}

Iteration : 19

Page Rank : {0: 0.28279591746931276, 1: 0.159038929858298, 2: 0.15018621095221932, 3: 0.10512827103665578, 4: 0.19772239964685842, 5: 0.10512827103665578}

Iteration : 20

Page Rank : {0: 0.2828045850169262, 1: 0.15903854557173613, 2: 0.1501865400761564, 3: 0.10512550994963862, 4: 0.1977193094359042, 5: 0.10512550994963862}

Iteration : 21

Page Rank : {0: 0.28280067599444025, 1: 0.15903502518578927, 2: 0.150188887000121, 3: 0.10512796575479576, 4: 0.19771948031005815, 5: 0.10512796575479576}

Iteration : 22

Page Rank : {0: 0.2828008675119779, 1: 0.1590381563373646, 2: 0.15018678200106503, 3: 0.10512685819842474, 4: 0.19772047775274315, 5: 0.10512685819842474}

Iteration : 23

Page Rank : {0: 0.2828021317864155, 1: 0.15903674420299155, 2: 0.1501877234239804, 3: 0.10512691246172709, 4: 0.19771957566315856, 5: 0.10512691246172709}

Iteration : 24

Page Rank : {0: 0.2828009879674331, 1: 0.15903681338870204, 2: 0.150187681530332, 3: 0.10512727067281773, 4: 0.1977199757678976, 5: 0.10512727067281773}

Iteration : 25

Page Rank : {0: 0.2828014998987927, 1: 0.1590372701078426, 2: 0.15018737705090496, 3: 0.10512694659077271, 4: 0.19771995976091444, 5: 0.10512694659077271}

Iteration : 26

Page Rank : {0: 0.2828014779617444, 1: 0.1590368569032352, 2: 0.15018765150188002, 3: 0.10512709163799128, 4: 0.19771983035715796, 5: 0.10512709163799128}

[5]:

{0: 0.28280131253898055,

1: 0.15903704183843886,

2: 0.15018752821174422,

3: 0.10512708542249424,

4: 0.19771994656584801,

5: 0.10512708542249424}

**CODE:**

nx.pagerank(G)

A screenshot of a computer program

Description automatically generated

**GOOGLE WEB GRAPH:**

**A screenshot of a graph

Description automatically generated**

**CODE:**

file\_path = "/content/drive/MyDrive/DATA ANALYTICS LAB/Datasets/web-Google.txt"

edges = pd.read\_csv(file\_path, sep="\t", comment='#', header=None, names=['FromNodeId', 'ToNodeId'])

A screenshot of a cell phone

Description automatically generated

**CODE:**

edges.values

**OUTPUT:**

A screenshot of a computer code

Description automatically generated

**CODE:**

G = nx.DiGraph()

G.add\_edges\_from(edges.values)

num\_nodes = G.number\_of\_nodes()

num\_edges = G.number\_of\_edges()

density = nx.density(G)

avg\_in\_degree = sum(dict(G.in\_degree()).values()) / num\_nodes

avg\_out\_degree = sum(dict(G.out\_degree()).values()) / num\_nodes

print(f"Number of nodes: {num\_nodes}")

print(f"Number of edges: {num\_edges}")

print(f"Density of the graph: {density:.5f}")

print(f"Average in-degree: {avg\_in\_degree:.2f}")

print(f"Average out-degree: {avg\_out\_degree:.2f}")

**OUTPUT:**

A black text on a white background

Description automatically generated

**CODE:**

in\_degrees = dict(G.in\_degree())

out\_degrees = dict(G.out\_degree())

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.hist(in\_degrees.values(), bins=50, color='lightblue', edgecolor='black')

plt.title("In-Degree Distribution")

plt.xlabel("In-Degree")

plt.ylabel("Frequency")

plt.subplot(1, 2, 2)

plt.hist(out\_degrees.values(), bins=50, color='lightgreen', edgecolor='black')

plt.title("Out-Degree Distribution")

plt.xlabel("Out-Degree")

plt.ylabel("Frequency")

plt.tight\_layout()

plt.show()

**OUTPUT:**

A comparison of a graph

Description automatically generated

**CODE:**

sccs = list(nx.strongly\_connected\_components(G))

print(f"Number of strongly connected components: {len(sccs)}")

largest\_scc = max(sccs, key=len)

print(f"Size of the largest strongly connected component: {len(largest\_scc)}")

Number of strongly connected components: 371764

Size of the largest strongly connected component: 434818

pagerank\_scores = nx.pagerank(G, alpha=0.85)

top\_10\_pagerank = sorted(pagerank\_scores.items(), key=lambda item: item[1], reverse=True)[:10]

print("\nTop 10 nodes by PageRank score:")

for node, score in top\_10\_pagerank:

print(f"Node {node}: {score:.6f}")

**OUTPUT:**

A screenshot of a computer code

Description automatically generated

**RESULT:**

The Page Rank algorithm computed importance scores for each web page, successfully converging after several iterations, with more influential pages receiving higher ranks due to inbound link quality and quantity.