Program 01

```
import requests
from bs4 import BeautifulSoup
import csv
import re

def crawl_and_scrape(url):
    # Send a GET request to the web page
    response = requests.get(url)

    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find all email addresses using regular expressions
    email_pattern = re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-
Z|a-z]{2,}\b')
    email_addresses = re.findall(email_pattern, str(soup))

    return email_addresses

def save_to_csv(email_addresses, filename):
    # Open the CSV file in write mode
    with open(filename, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)

        # Write the email addresses to the CSV file
        for email in email_addresses:
            writer.writerow([email])

# URL of the social media web page to crawl
url = 'https://www.symmetrix.in/contact.html'

# Crawl and scrape email addresses
email_addresses = crawl_and_scrape(url)

# Save the email addresses to a CSV file
save_to_csv(email_addresses, 'email_addresses.csv')

print("Email addresses scraped and saved to 'email_addresses.csv'")
```

Program 02

```
import requests
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from bs4 import BeautifulSoup
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.probability import FreqDist
from nltk.tokenize import sent_tokenize
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
from sumy.summarizers.lsa import LsaSummarizer

# Function to scrape a web page
def scrape_web_page(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    text = soup.get_text()
    return text

# Function to tokenize the text and perform word count
def tokenize_and_count(text):
    tokens = word_tokenize(text.lower())
    # Remove stopwords and punctuations
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token.isalpha() and token
not in stop_words]
    # Perform stemming
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
    # Count word frequencies
    fdist = FreqDist(stemmed_tokens)
    return fdist

# Function to perform text summarization
def summarize_text(text, sentences_count=3):
    parser = PlaintextParser.from_string(text, Tokenizer("english"))
    summarizer = LsaSummarizer()
    summary = summarizer(parser.document, sentences_count)
    summarized_text = " ".join([str(sentence) for sentence in summary])
    return summarized_text

# URL of the web page to scrape
url = "https://www.geeksforgeeks.org/"

# Scrape the web page
```

```
web_page_text = scrape_web_page(url)

# Tokenization and word count
word_frequencies = tokenize_and_count(web_page_text)
print("Word count:")
print(word_frequencies)

# Text summarization
summary = summarize_text(web_page_text)
print("\nSummary:")
print(summary)
```

Program 03

```
import pandas as pd
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
nltk.download('stopwords')
# Load the IMDb movie review dataset from Kaggle
data = pd.read_csv('IMDB Dataset.csv')


# Preprocess the text
stopwords = nltk.corpus.stopwords.words('english')
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
lemmatizer = nltk.stem.WordNetLemmatizer()


def preprocess_text(text):
    tokens = tokenizer.tokenize(text.lower())
    tokens = [lemmatizer.lemmatize(token) for token in tokens if token
not in stopwords]
    return ' '.join(tokens)


data['processed_text'] = data['review'].apply(preprocess_text)
# Split the dataset into training and testing sets
X = data['processed_text']
y = data['sentiment']


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Convert the text into numerical features using TF-IDF vectorization
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
# Train a sentiment analysis model
model = LogisticRegression()
model.fit(X_train, y_train)


# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Program 04

```
import networkx as nx
import matplotlib.pyplot as plt
import urllib.request


# URL of the dataset
url = "https://raw.githubusercontent.com/miladfa7/Social-Network-
Analysis-in-Python/master/facebook_combined.txt"

# Download the dataset
urllib.request.urlretrieve(url, "facebook_combined.txt")


# Create a graph
G1 =nx.read_edgelist("facebook_combined.txt", create_using =
nx.Graph(), nodetype=int)
# Calculate betweenness centrality
betweenness_centrality = nx.betweenness_centrality(G1)

# Calculate Degree Centrality
degree_centrality = nx.degree_centrality(G1)
closeness_centrality = nx.closeness_centrality(G1)
eigenvector_centrality = nx.eigenvector_centrality(G1)

nx.draw(G1, with_labels=True, node_color='skyblue', node_size=100)
plt.show()


plt.bar(betweenness_centrality.keys(), betweenness_centrality.values())
plt.title("Betweenness Centrality")
plt.xlabel("Nodes")
plt.ylabel("Centrality")
plt.show()
plt.bar(degree_centrality.keys(), degree_centrality.values())
plt.title("Degree Centrality")
plt.xlabel("Nodes")
plt.ylabel("Centrality")
plt.show()
plt.bar(closeness_centrality.keys(), closeness_centrality.values())
plt.title("Closeness Centrality")
plt.xlabel("Nodes")
plt.ylabel("Centrality")
plt.show()
plt.bar(eigenvector_centrality.keys(), eigenvector_centrality.values())
plt.title("Eigenvector Centrality")
plt.xlabel("Nodes")
plt.ylabel("Centrality")
plt.show()
```

Program 05

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import SpectralClustering

# Load the Karate club network data
G = nx.karate_club_graph()

# i. Visualize the network using matplotlib
pos = nx.spring_layout(G)  # Position nodes using Fruchterman-Reingold
force-directed algorithm
nx.draw(G, pos, with_labels=True, node_color='lightblue',
edge_color='gray')
plt.title("Zachary's Karate Club Network")
plt.show()

# ii. Visualizing Important Nodes in the Graph
# Calculate degree centrality for each node
degree_centrality = nx.degree_centrality(G)

# Sort nodes by degree centrality in descending order
sorted_nodes = sorted(degree_centrality, key=degree_centrality.get,
reverse=True)

# Take the top 5 nodes with highest degree centrality
important_nodes = sorted_nodes[:5]

# Create a subgraph with the important nodes and their neighbors
subgraph = G.subgraph(important_nodes +
list(G.neighbors(important_nodes[0])))

# Visualize the subgraph
pos = nx.spring_layout(subgraph)
nx.draw(subgraph, pos, with_labels=True, node_color='lightblue',
edge_color='gray')
plt.title("Important Nodes and Their Neighbors")
plt.show()

# iii. Perform spectral clustering
# Pre-processing: Constructing the Laplacian Matrix
adjacency_matrix = nx.to_numpy_array(G)
n = len(G.nodes)
degree_matrix = np.diag(np.sum(adjacency_matrix, axis=1))
laplacian_matrix = degree_matrix - adjacency_matrix

# Decomposition: Compute eigenvalues and eigenvectors of the Laplacian
Matrix
```

```
eigenvalues, eigenvectors = np.linalg.eig(laplacian_matrix)
embedding = eigenvectors[:, 1:3]  # Map each point to a 2D
representation

# K Means Algorithm: Create groups of clusters
k = 2  # Number of clusters
spectral_clustering = SpectralClustering(n_clusters=k,
affinity='nearest_neighbors', random_state=42)
labels = spectral_clustering.fit_predict(embedding)

# Visualize the clusters
pos = nx.spring_layout(G)
plt.scatter(embedding[:, 0], embedding[:, 1], c=labels, cmap='viridis')
nx.draw_networkx_edges(G, pos, alpha=0.5)
plt.title("Spectral Clustering")
plt.show()
```

Program 06

```
import networkx as nx
import pandas as pd
import numpy as np
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
%matplotlib inline


def plot_graph(G):
    plt.figure(figsize=(8, 6))
    nx.draw(G, with_labels=True)
    plt.show()
def answer_one():
    edges = []
    with
open("/content/drive/MyDrive/SMA/Program5/Employee_Movie_Choices.txt",
"r") as file:
        for line in file:
            edge = line.strip().split("\t")
            edges.append(edge)
    G = nx.Graph()
    G.add_edges_from(edges)
    return G
answer_one()
plot_graph(answer_one())
employees = set()
movies = set()
with
open("/content/drive/MyDrive/SMA/Program5/Employee_Movie_Choices.txt",
"r") as file:
    next(file)
    for line in file:
        employees.add(line.strip().split("\t")[0])
        movies.add(line.strip().split("\t")[1])
def answer_two():

    G = answer_one()
    for node in G.nodes():
        if node in employees:
            G.add_node(node, type="employee")
        else:
            G.add_node(node, type="movie")
    return G
answer_two()
plot_graph(answer_two())
def answer_three():
    B = answer_two()
```

```
    weighted_projection = bipartite.weighted_projected_graph(B,
employees)
    return weighted_projection
plot_graph(answer_three())


def answer_four():

    Rel =
nx.read_edgelist('/content/drive/MyDrive/SMA/Program5/Employee_Relation
ships.txt', data=[('relationship_score', int)])
    Rel_df = pd.DataFrame(Rel.edges(data=True), columns=['From', 'To',
'relationship_score'])
    G = answer_three()
    G_df = pd.DataFrame(G.edges(data=True), columns=['From', 'To',
'movies_score'])
    G_copy_df = G_df.copy()
    G_copy_df.rename(columns={"From":"From_", "To":"From"},
inplace=True)
    G_copy_df.rename(columns={"From_":"To"}, inplace=True)
    G_final_df = pd.concat([G_df, G_copy_df])
    final_df = pd.merge(G_final_df, Rel_df, on = ['From', 'To'],
how='right')
    final_df['movies_score'] = final_df['movies_score'].map(lambda x:
x['weight'] if type(x)==dict else None)
    final_df['relationship_score'] =
final_df['relationship_score'].map(lambda x: x['relationship_score'])
    final_df['movies_score'].fillna(value=0, inplace=True)
    return
final_df['movies_score'].corr(final_df['relationship_score'])
answer_four()
```

Program 07

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a random connected graph with 20 nodes
graph = nx.connected_watts_strogatz_graph(20, 4, 0.3)

# Calculate centrality measures
degree_centrality = nx.degree_centrality(graph)
betweenness_centrality = nx.betweenness_centrality(graph)
eigenvector_centrality = nx.eigenvector_centrality(graph)

# Visualize the graph
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(graph, seed=42)
nx.draw(graph, pos, with_labels=True, node_color='lightblue',
node_size=200, alpha=0.8, font_size=8)

# Calculate degree distribution and plot histogram
degrees = [degree for node, degree in graph.degree()]
plt.figure(figsize=(8, 6))
plt.hist(degrees, bins=range(max(degrees) + 2), align='left',
color='skyblue', edgecolor='gray')
plt.xlabel('Degree')
plt.ylabel('Frequency')
plt.title('Degree Distribution')
plt.xticks(range(max(degrees) + 1))
plt.grid(True)
plt.show()

# Print centrality measures for each node
print("Node\tDegree Centrality\tBetweenness Centrality\tEigenvector
Centrality")
for node in graph.nodes:
    print(f"{node}\t{degree_centrality[node]}\t\t\t{betweenness_central
ity[node]}\t\t\t{eigenvector_centrality[node]}")
```