**PRACTICAL 1**

**Aim:** Study Various

* Social Media platforms (Facebook, twitter, YouTube etc)
* Social Media analytics tools (Facebook insights, google analytics netlytic etc)
* Social Media Analytics techniques and engagement metrics (page level, post level, member level) using Gephi Tool

**Theory:**

*Social Media Platforms*

Social media platforms like Facebook, Twitter, and YouTube each offer unique features and cater to different user behaviors and demographics.

* **Facebook** serves as a comprehensive platform for personal networking, brand promotion, and community building, leveraging tools such as News Feed, Pages, Groups, Events, Marketplace, and Facebook Watch. It has a broad user base, predominantly among older users (30+), and facilitates extensive user interaction through likes, shares, and comments.
* **Twitter** is designed for real-time communication and news dissemination, with features like Tweets, Retweets, Hashtags, Trends, and Moments. It is popular among younger audiences (18-29) and is extensively used for short-form updates, hashtag campaigns, and customer service interactions, allowing for quick information spread and engagement.
* **YouTube** focuses on video content sharing and consumption, utilizing features such as Video Uploads, Channels, Playlists, Subscriptions, and YouTube Live. It attracts a wide demographic, especially younger users, and serves as a major platform for vlogging, tutorials, entertainment, and marketing, supporting both long-form and short-form video content.

*Social Media Analytics Tools*

To effectively measure and analyze social media performance, various analytics tools are employed, each providing unique insights and capabilities.

* **Facebook Insights** offers in-depth analytics for Facebook Pages, including metrics such as Page Views, Post Reach, Engagement, Page Likes, and Demographic information. This tool helps in understanding the overall page performance and audience interaction.
* **Google Analytics** is a robust tool for tracking website traffic and analyzing the impact of social media campaigns on web visits. Key metrics include User Sessions, Bounce Rate, Conversion Rate, Traffic Sources, and User Behavior, providing a comprehensive view of how social media drives website engagement and conversions.
* **Netlytic** is a cloud-based tool for text and social network analysis, capable of summarizing and discovering patterns in large text and network data sets. It offers metrics for Text Analysis, Social Network Analysis, Community Detection, and Sentiment Analysis, aiding in the understanding of communication patterns and community structures within social media.

*Social Media Analytics Techniques and Engagement Metrics*

Analyzing social media involves various techniques and engagement metrics at different levels: page level, post level, and member level.

* **Page Level Analytics** focuses on the overall performance of a social media page. Important metrics include Total Followers, Page Views, Impressions, and Engagement Rate, which provide insights into the page’s reach and overall engagement with its audience.
* **Post Level Analytics** examines the performance of individual posts. Key metrics include Reach, Engagement (likes, comments, shares), Click-Through Rate (CTR), and Virality Rate, helping to understand how specific content resonates with the audience.
* **Member Level Analytics** looks at the behavior and engagement of individual users. Metrics such as Active Users, Engagement Frequency, and Influencer Identification are crucial for identifying key users and understanding their interaction patterns.

*Using Gephi for Social Media Network Analysis*

Gephi, an open-source network analysis and visualization tool, is instrumental in exploring and understanding complex social media networks.

* **Network Visualization** with Gephi allows for the representation of users, posts, and pages as nodes, and their relationships or interactions as edges. This visualization helps identify clusters, key influencers, and interaction patterns within the network.
* **Community Detection** uses algorithms like Louvain or Girvan-Newman to identify clusters or groups within the network, providing insights into how different user communities are formed and interact.
* **Centrality Measures** in Gephi, such as Degree Centrality, Betweenness Centrality, and Closeness Centrality, help identify key influencers and central nodes. These measures are critical for understanding the network’s structure and optimizing content dissemination and engagement strategies.
* **Temporal Analysis** in dynamic networks tracks how network structures and interactions evolve over time, offering insights into the growth of user engagement, the spread of viral content, and the impact of specific campaigns.

In conclusion, integrating the features and uses of major social media platforms, the functionalities of prominent analytics tools, and the techniques and metrics essential for comprehensive analysis enables organizations to gain deeper insights into their social media dynamics. Using tools like Gephi for network analysis further enhances the understanding of user interactions and community structures, leading to more effective content strategies and engagement optimization

**PRACTICAL 2**

**Aim**: Scrape an online Social Media Site for Data. Use python to scrape information from twitter. Exploratory Data Analysis and visualization of Social Media Data

**Theory:**   
Social media has become a crucial source of data for businesses to understand their customers and their preferences. With millions of users sharing their thoughts, opinions, and experiences on social media platforms like Twitter, Facebook, LinkedIn, YouTube, and web blogs, businesses can gain valuable insights into customer behavior, market trends, and brand perception. In this article, we will discuss data collection methods for social media platforms, including scraping, crawling, and parsing.

***Data Collection Methods:***

There are various data collection methods for social media platforms, and each method has its own advantages and disadvantages. Some of the popular data collection methods are scraping, crawling, and parsing.

1. **Scraping:** Scraping is the process of extracting data from websites or social media platforms. Scraping is an effective way to collect data from social media platforms like Twitter, Facebook, and LinkedIn. The data collected through scraping can be used for various purposes, such as sentiment analysis, trend analysis, and competitor analysis.
2. **Crawling:** Crawling is the process of discovering and indexing web pages through search engines. It involves using automated tools or software to scan websites and social media platforms for new content. Crawling is an effective way to collect data from web blogs and news websites. The data collected through crawling can be used for various purposes, such as content analysis, trend analysis, and competitor analysis.
3. **Parsing:** Parsing is the process of extracting data from structured or unstructured data sources. It involves using automated tools or software to analyze data and extract relevant information. Parsing is an effective way to collect data from social media platforms like Twitter and Facebook. The data collected through parsing can be used for various purposes, such as sentiment analysis, trend analysis, and competitor analysis.

***Execution Steps:***

1. *Identify the social media platform*(s) of interest: Choose the social media platform(s) that you want to collect data from, based on your business needs and objectives.
2. *Set up a data collection tool*: Choose a data collection tool that suits your needs and set it up. There are various data collection tools available, such as Octoparse, ParseHub, and *Beautiful Soup.*
3. *Define* the data fields: Define the data fields that you want to collect from the social media platform. For example, if you are collecting data from Twitter, you may want to collect data on tweets, hashtags, user profiles, and location.
4. *Configure* the data collection tool: Configure the data collection tool to collect the data fields that you have defined.
5. *Run the data collection* tool: Run the data collection tool to collect the data from the social media platform.
6. *Store and analyze* the data: Store the data in a database or spreadsheet and analyze it using data analysis tools like Excel or Python.

***Code:***

import pandas as pd

import requests

from textblob import TextBlob

import json

video\_id = "Q33TkQKlIMg"

max\_result = 100

api\_key = "AIzaSyC\_4xZTiNuz1O-Qu5kYnlg82riP30KRIxY"

video\_info\_url = f"https://www.googleapis.com/youtube/v3/videos?part=id%2C+snippet&id={video\_id}&key={api\_key}"

video\_info\_response = requests.get(video\_info\_url)

video\_info\_data = video\_info\_response.json()

print(json.dumps(video\_info\_data, indent=4))

comments\_url = f"https://www.googleapis.com/youtube/v3/commentThreads?key={api\_key}&video\_id={video\_id}&part=snippet&max\_results={max\_result}"

comments\_response = requests.get(comments\_url)

comments\_data = comments\_response.json()

# print(json.dumps(comments\_data, indent=4))

df = pd.DataFrame(comments\_data['items'])

df1 = pd.DataFrame(df['snippet'])

txt = ""

comments = []

for i in range(0, max\_result):

df2 = pd.DataFrame(df['snippet'][i])

txt = df2['topLevelComment']['snippet']['textOriginal']

comments.append(txt)

def get\_comment\_sentiment(comment):

analysis = TextBlob(comment)

if analysis.sentiment.polarity > 0:

return "Positive"

elif analysis.sentiment.polarity == 0:

return "Neutral"

else:

return "Negative"

comment\_list = []

sentiment\_list = []

for comment in comments:

sentiment = get\_comment\_sentiment(comment)

comment\_list.append(comment)

sentiment\_list.append(sentiment)

sentiment\_df = pd.DataFrame({"Comments": comment\_list, "Sentiment": sentiment\_list})

sentiment\_df

sentiment\_df.to\_csv("Youtube\_Comments\_Sentiment\_Analysis.csv")

***Output:***

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**PRACTICAL 3**

**Aim**: Create sociograms for the persons-by-persons network and the community by community network for a given relevant problem. Create a one-mode network and two node networks for the same. Datasets: les-Misérables, Airlines, Internet Core Routers.

**Theory:**

The "Les Misérables" dataset is available in NetworkX and represents the co-occurrences of characters in Victor Hugo's novel "Les Misérables".

*Step 1: Load the Dataset* First, we need to load the "Les Misérables" dataset. We'll use NetworkX's built-in function to load it.

*Step 2: Create One-Mode and Two-Mode Networks*

* One-mode network: This is a graph where the nodes represent entities of a single type (e.g., characters), and edges represent relationships between them.
* Two-mode network: This is a bipartite graph where nodes represent two different types of entities (e.g., characters and scenes), and edges represent relationships between different types of nodes.

*Step 3: Generate Sociograms*

Visualizing the networks using Matplotlib and NetworkX's drawing functions.

1. Loading the dataset: We load the "Les Misérables" dataset using nx.les\_miserables\_graph().
2. Creating one-mode network: We directly use the dataset as the one-mode network.
3. Creating two-mode network: We create a bipartite graph where one set of nodes represents characters and the other set represents communities (groups based on centrality in this example).
4. Generating sociograms: We use Matplotlib to visualize the one-mode and two-mode networks. You can follow a similar approach for the "Airlines" and "Internet Core Routers" datasets by loading the respective datasets and visualizing them.

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

from networkx.algorithms import bipartite

# Step 1: Load the Les Misérables dataset

G = nx.les\_miserables\_graph()

# Step 2: Create one-mode and two-mode networks

# One-mode network (characters only)

one\_mode\_graph = G

# Create a bipartite two-mode network (characters and their communities)

# In this example, we'll assume "community" as groups of characters based on their centrality (for simplicity)

centrality = nx.degree\_centrality(G)

communities = {}

for node, cent in centrality.items():

group = int(cent \* 10) # Group based on centrality value

if group not in communities:

communities[group] = []

communities[group].append(node)

# Creating a bipartite graph

B = nx.Graph()

character\_nodes = list(G.nodes())

community\_nodes = list(communities.keys())

B.add\_nodes\_from(character\_nodes, bipartite=0)

B.add\_nodes\_from(community\_nodes, bipartite=1)

for group, chars in communities.items():

for char in chars:

B.add\_edge(char, group)

# Step 3: Generate sociograms

# One-mode network sociogram

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

pos = nx.spring\_layout(one\_mode\_graph)

nx.draw(one\_mode\_graph, pos, with\_labels=True, node\_color="skyblue", edge\_color="gray",

node\_size=500, font\_size=10)

plt.title("One-mode Network: Les Misérables Characters")

plt.show()

# Two-mode network sociogram

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

pos = nx.spring\_layout(B)

node\_colors = ["skyblue" if node in character\_nodes else "lightgreen" for node in B.nodes()]

nx.draw(B, pos, with\_labels=True, node\_color=node\_colors, edge\_color="gray", node\_size=500, font\_size=10)

plt.title("Two-mode Network: Les Misérables Characters and Communities")

plt.show()

**Output:**

**A network of blue dots and lines

Description automatically generated**

**A circle of blue dots and green dots

Description automatically generatedPRACTICAL 4**

**Aim**: Develop Content (text, emoticons, image, audio, video) based social media analytics model for business. (e.g., Content Based Analysis: Topic, Issue, Trend, sentiment/opinion analysis, audio, video, image analytics)**.**

**Theory:**

Social media analytics involves extracting useful insights from the vast amounts of data generated on platforms like Twitter, Facebook, Instagram, and YouTube. This includes analyzing text, images, audio, and video content to understand trends, sentiments, and user behavior. A comprehensive analytics model integrates multiple content types to provide a holistic view of social media dynamics.

*Topic Modeling*

Topic modeling is a method for uncovering hidden thematic structures in a collection of documents. It helps in understanding the main themes and subjects discussed across social media posts. The Latent Dirichlet Allocation (LDA) algorithm is a popular method for topic modeling.

* ***Latent Dirichlet Allocation (LDA):*** LDA is a generative probabilistic model that assumes each document is a mixture of topics and each topic is a mixture of words. By iteratively updating the assignments of words to topics, LDA can discover the underlying topic structure.

*Trend analysis*

Trend analysis involves identifying the frequency and evolution of specific topics or keywords over time. *Term Frequency-Inverse Document Frequency (TF-IDF)* is often used to identify the importance of words in a document relative to a collection of documents.

* ***TF-IDF:*** TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection. It increases proportionally to the number of times a word appears in a document and is offset by the number of documents that contain the word.

*Sentiment Analysis*

Sentiment analysis assesses the emotional tone of a text. It helps in understanding public opinion and emotional responses. *VADER (Valence Aware Dictionary and Sentiment Reasoner)* is a commonly used tool for sentiment analysis, particularly suited for social media text.

* **VADER**: VADER is a rule-based model for general sentiment analysis that is attuned to sentiments expressed in social media. It uses a combination of qualitative analysis and lexical heuristics to assign sentiment scores.

A comprehensive social media analytics model requires integrating multiple analysis techniques tailored to different types of content. By combining text, image, audio, and video analysis, businesses can gain deep insights into social media dynamics, helping them make informed decisions and tailor their strategies effectively.

**Code:**

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.decomposition import LatentDirichletAllocation

import matplotlib.pyplot as plt

# Sample Data

texts = ["This is the first document.", "The Document is pathetic.", "This document is really nice", "Is this the first document?"]

# Vectorize the text

vectorizer = CountVectorizer(stop\_words='english')

X = vectorizer.fit\_transform(texts)

# Topic Modeling with LDA

lda = LatentDirichletAllocation(n\_components=2, random\_state=42)

lda.fit(X)

# Display topics

def display\_topics(model, feature\_names, no\_top\_words):

for topic\_idx, topic in enumerate(model.components\_):

print(f"Topic {topic\_idx}:")

print(" ".join([feature\_names[i] for i in topic.argsort()[:-no\_top\_words - 1:-1]]))

display\_topics(lda, vectorizer.get\_feature\_names\_out(), 3)

# Trend Analysis with TF-IDF

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english')

tfidf = tfidf\_vectorizer.fit\_transform(texts)

df = pd.DataFrame(tfidf.toarray(), columns=tfidf\_vectorizer.get\_feature\_names\_out())

print(df)

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()

# Analyze sentiments

sentiments = [analyzer.polarity\_scores(text) for text in texts]

sentiment\_df = pd.DataFrame(sentiments)

print(sentiment\_df)

**Output:**

**A computer code with numbers and letters

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated**

**PRACTICAL 5**

**Aim**: Develop Structure based social media analytics model for any business. (e.g., Structure Based Models, community detection, influence analysis)

**Theory:**

To develop a structure-based social media analytics model for a business, we can focus on two key areas: community detection and influence analysis. These methods are crucial for understanding how information spreads within a social network and identifying key influencers who can drive engagement and growth.

Let's outline a practical approach using Python:

1. **Data Collection**: Gather social media data, such as tweets, Facebook posts, or any relevant interaction data. This step involves using APIs (e.g., Twitter API, Facebook Graph API) to collect data.
2. **Network Construction**: Construct a social network graph where nodes represent users and edges represent interactions (e.g., likes, retweets, comments).
3. **Community Detection**: Identify communities within the social network using algorithms like Louvain or Girvan-Newman.
4. **Influence Analysis**: Determine the influence of each node using centrality measures such as PageRank, betweenness centrality, or eigenvector centrality.
5. **Visualization**: Visualize the network and the detected communities to gain insights.

Let's implement this step-by-step in Python:

**Step 1: Data Collection**

For simplicity, let's assume we have a pre-collected dataset of social interactions in CSV format.

**Step 2: Network Construction**

We'll use the **networkx** library to construct the social network graph.

**Step 3: Community Detection**

We'll use the Louvain method for community detection, which is available in the **community** package.

**Step 4: Influence Analysis**

We'll compute PageRank to measure the influence of each node.

**Step 5: Visualization**

We'll use **matplotlib** and **networkx** for visualization. First, ensure you have the necessary libraries installed:

**Explanation:**

1. **Data Collection**: We create a synthetic dataset representing interactions between users.
2. **Network Construction**: We construct a graph using the **networkx** library where each edge represents an interaction.
3. **Community Detection**: We apply the Louvain method to detect communities within the network. The detected community for each node is added as an attribute.
4. **Influence Analysis**: We compute the PageRank for each node to determine its influence and add it as an attribute.
5. **Visualization**: We visualize the network, coloring nodes by their community and displaying their PageRank values.

This model provides a structured approach to analyzing social media networks, allowing businesses to identify key communities and influential users within their networks.

**Code:**

import pandas as pd

import networkx as nx

import community as community\_louvain

import matplotlib.pyplot as plt

# install python\_louvain

# Step 1: Data Collection (Load the data)

# For demonstration, let's create a synthetic dataset

data = {

'source': ['A', 'A', 'B', 'C', 'D', 'E', 'E', 'F', 'F', 'G'],

'target': ['B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K']

}

df = pd.DataFrame(data)

# Step 2: Network Construction

G = nx.from\_pandas\_edgelist(df, 'source', 'target')

# Step 3: Community Detection

partition = community\_louvain.best\_partition(G)

# Add community information to nodes

for node, community in partition.items():

G.nodes[node]['community'] = community

# Step 4: Influence Analysis

pagerank = nx.pagerank(G)

nx.set\_node\_attributes(G, pagerank, 'pagerank')

# Step 5: Visualization

pos = nx.spring\_layout(G) # Layout for visualization

plt.figure(figsize=(12, 8))

# Draw nodes with community colors

colors = [partition[node] for node in G.nodes()]

nx.draw\_networkx\_nodes(G, pos, node\_size=500, node\_color=colors, cmap=plt.cm.jet)

nx.draw\_networkx\_edges(G, pos, alpha=0.5)

nx.draw\_networkx\_labels(G, pos, font\_size=12)

# Draw node labels for pagerank

pagerank\_labels = {node: f'{round(rank, 2)}' for node, rank in pagerank.items()}

nx.draw\_networkx\_labels(G, pos, labels=pagerank\_labels, font\_color='red')

plt.savefig('socialnet.png')

plt.title('Social Network with Community Detection and Influence Analysis')

plt.show()

**Output:**

**A diagram of a constellation

Description automatically generated**

**PRACTICAL 6**

**Aim**: Develop a dashboard and reporting tool based on real time social media data Using Power BI

**Theory:**

Developing a dashboard and reporting tool based on real-time social media data is a critical step in monitoring the performance of a business's social media strategy. In this article, we will discuss the theory behind social media dashboards, the execution steps to develop the dashboard and reporting tool, and the benefits it can offer for businesses.

A social media dashboard is a tool that provides real-time monitoring of social media activities, including engagement, reach, and impressions. It helps businesses to track the performance of their social media strategy and identify areas for improvement. The dashboard can provide data visualization, including charts and graphs, to enable users to interpret the data easily.

Reporting tools are used to create regular reports based on the data gathered by the dashboard. These reports can provide insights into the performance of the social media strategy, identify trends, and suggest areas for improvement.

**Execution Steps:**

1. *Identify Key Performance Indicators (KPIs):* Identify the KPIs that are important for the business, such as engagement rate, reach, and impressions.
2. *Data Collection:* Collect data from social media platforms, including user profiles, followers, and interactions between users.
3. *Data Processing:* Pre-process the data to remove any irrelevant data and prepare it for analysis.
4. *Data Visualization:* Use data visualization tools such as charts and graphs to represent the data in an easily understandable format.
5. *Dashboard Development:* Develop a dashboard using a tool such as Tableau, Power BI, or Google Data Studio.
6. *Reporting Tool Development:* Develop a reporting tool that generates regular reports based on the data collected by the dashboard.
7. *Implementation:* Implement the dashboard and reporting tool to provide real-time monitoring of social media activities and generate regular reports for the business.

**Code:**

import dash

from dash import html, dcc

import pandas as pd

from ntscraper import Nitter

# Initialize Nitter scraper

scraper = Nitter(0)

# Function to fetch tweets

def get\_tweets(name, modes, no):

tweets = scraper.get\_tweets(name, mode=modes, number=no)

final\_tweets = []

for x in tweets["tweets"]:

data = [

x["link"],

x["text"],

x["date"],

x["stats"]["likes"],

x["stats"]["comments"],

]

final\_tweets.append(data)

dat = pd.DataFrame(

final\_tweets, columns=["twitter\_link", "text", "date", "likes", "comments"]

)

return dat

# Fetch data

data = get\_tweets("World cup 2023", "term", 8)

# Initialize Dash app

app = dash.Dash(\_\_name\_\_)

# Define layout

app.layout = html.Div(

[

html.H1("Twitter Dashboard"),

dcc.Graph(

id="tweets-graph",

figure={

"data": [

{"x": data["date"], "y": data["likes"], "type": "bar", "name": "Likes"},

{"x": data["date"], "y": data["comments"], "type": "bar", "name": "Comments"},

],

"layout": {"title": "Likes and Comments Over Time"},

},

),

html.Table(

[

html.Thead(

html.Tr(

[html.Th(col) for col in data.columns])),

html.Tbody(

[

html.Tr([html.Td(data.iloc[i][col]) for col in data.columns])

for i in range(len(data))

]

),

]

),

]

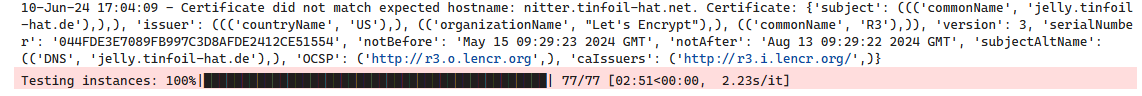
)

# Run the app

if \_\_name\_\_ == "\_\_main\_\_":

app.run\_server(debug=True)

**Output:**

****

**A graph with blue rectangles

Description automatically generated**

**PRACTICAL 7**

**Aim**: Use Google Visualization Charts to analyze social media data

**Theory:**

*Google Visualization Charts:*

Google Visualization Charts is a robust library for creating interactive charts and data tools that can be embedded into web pages. It allows for the visualization of complex data in various forms, such as line charts, pie charts, bar charts, and tables. The library's integration with JavaScript enables dynamic and responsive data representation, enhancing user experience and data comprehension. The charts are not only visually appealing but also interactive, allowing users to engage with the data, such as by hovering over elements to see detailed information. This interaction facilitates a deeper understanding of the data trends and patterns. Additionally, Google Charts is designed to be highly customizable, enabling developers to tailor the appearance and behavior of the charts to meet specific requirements.

*Ntscraper:*

ntscraper is a specialized library used for web scraping, particularly designed to interface with Nitter, an alternative front-end for Twitter. Nitter provides a more accessible and less rate-limited way to access Twitter data compared to the official Twitter API. ntscraper leverages this capability to extract tweets and associated metadata efficiently. The library simplifies the process of retrieving social media data by abstracting the complexities involved in making direct HTTP requests and parsing HTML content. By specifying search terms, modes, and the number of tweets to fetch, ntscraper provides a streamlined approach to gather relevant data for further analysis. This capability is crucial for collecting real-time or historical social media data to analyze trends, sentiments, and user engagement.

*Jinja2:*

Jinja2 is a powerful templating engine for Python that facilitates the generation of dynamic HTML content. It enables the creation of HTML templates with placeholders that can be populated with data at runtime. This dynamic insertion of data into templates allows for the creation of personalized and data-driven web pages. Jinja2 is particularly useful for web applications that require the display of data that changes over time or based on user input. By separating the HTML structure from the data logic, Jinja2 promotes a clean and maintainable codebase. The templating engine supports various control structures like loops and conditionals, making it flexible and capable of handling complex data rendering scenarios.

**Code:**

# Function to get tweets

def get\_tweets(name, modes, no):

tweets = scraper.get\_tweets(name, mode=modes, number=no)

final\_tweets = []

for x in tweets['tweets']:

data = [x['link'], x['text'], x['date'], x['stats']['likes'], x['stats']['comments']]

final\_tweets.append(data)

dat = pd.DataFrame(final\_tweets, columns=['twitter\_link', 'text', 'date', 'likes', 'comments'])

return dat

# Fetch tweets with term 'World cup 2023'

data = get\_tweets('World cup 2023', 'term', 10)

# HTML template for Google Visualization Charts

html\_template = """

<!DOCTYPE html>

<html>

<head>

<title>Social Media Data Visualization</title>

<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>

<script type="text/javascript">

google.charts.load('current', {'packages':['corechart', 'table']});

google.charts.setOnLoadCallback(drawCharts);

function drawCharts() {

var lineData = new google.visualization.DataTable();

lineData.addColumn('string', 'Date');

lineData.addColumn('number', 'Likes');

lineData.addRows([

{% for row in data.itertuples() %}

['{{ row.date }}', {{ row.likes }}],

{% endfor %}

]);

var lineOptions = {

title: 'Likes Over Time',

curveType: 'function',

legend: { position: 'bottom' }

};

var lineChart =

new google.visualization.LineChart(document.getElementById('line\_chart'));

lineChart.draw(lineData, lineOptions);

var pieData = new google.visualization.DataTable();

pieData.addColumn('string', 'Tweet');

pieData.addColumn('number', 'Comments');

pieData.addRows([

{% for row in data.itertuples() %}

['{{ row.text|truncate(50) }}', {{ row.comments }}],

{% endfor %}

]);

var pieOptions = {

title: 'Comments Distribution',

is3D: true,

};

var pieChart = new google.visualization.PieChart(document.getElementById('pie\_chart'));

pieChart.draw(pieData, pieOptions);

var barData = new google.visualization.DataTable();

barData.addColumn('string', 'Tweet');

barData.addColumn('number', 'Likes');

barData.addRows([

{% for row in data.itertuples() %}

['{{ row.text|truncate(50) }}', {{ row.likes }}],

{% endfor %}

]);

var barOptions = {

title: 'Likes Distribution',

chartArea: {width: '50%'},

hAxis: {

title: 'Likes',

minValue: 0

},

vAxis: {

title: 'Tweet'

}

};

var barChart = new google.visualization.BarChart(document.getElementById('bar\_chart'));

barChart.draw(barData, barOptions);

var tableData = new google.visualization.DataTable();

tableData.addColumn('string', 'Tweet');

tableData.addColumn('string', 'Date');

tableData.addColumn('number', 'Likes');

tableData.addColumn('number', 'Comments');

tableData.addRows([

{% for row in data.itertuples() %}

['<a href="{{ row.twitter\_link }}" target="\_blank">{{ row.text|truncate(50) }}</a>', '{{ row.date }}', {{ row.likes }}, {{ row.comments }}],

{% endfor %}

]);

var table = new google.visualization.Table(document.getElementById('table\_div'));

table.draw(tableData, {showRowNumber: true, width: '100%', height: '100%'});

}

</script>

</head>

<body>

<h1>Social Media Data Visualization</h1>

<div id="line\_chart" style="width: 900px; height: 500px;"></div>

<div id="pie\_chart" style="width: 900px; height: 500px;"></div>

<div id="bar\_chart" style="width: 900px; height: 500px;"></div>

<div id="table\_div" style="width: 900px; height: 500px;"></div>

</body>

</html>

"""

# Render the template with the data

template = Template(html\_template)

html\_content = template.render(data=data)

# Save the rendered HTML to a file with UTF-8 encoding

with open('visualization.html', 'w', encoding='utf-8') as file:

file.write(html\_content)

print("HTML file created successfully.")

**Output:**

**A graph showing a line

Description automatically generated with medium confidence**

**A pie chart with a green and red circle with Crust in the background

Description automatically generated**

**A graph with blue lines and numbers

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**PRACTICAL 8**

**Aim**: Use Graph Neural Networks on the datasets (Planetoid Cora Dataset)/ Jazz Musicians Network.

**Theory:**

Graph Neural Networks (GNNs) are powerful tools for analyzing graph-structured data. We'll use the popular Planetoid Cora dataset, which is widely used for benchmarking GNNs in node classification tasks.

To implement this, we'll use the **PyTorch** and **PyTorch Geometric** libraries. PyTorch Geometric is a library built on top of PyTorch to facilitate the implementation of GNNs.

**Steps to Implement GNN on the Cora Dataset**

*Step 1: Install Required Libraries*

*Step 2: Load the Cora Dataset*

PyTorch Geometric provides convenient access to various datasets, including Cora.

*Step 3: Define the GNN Model*

We'll define a simple GCN (Graph Convolutional Network) model.

*Step 4: Train the Model*

Now, we train the model using the Adam optimizer.

*Step 5: Evaluate the Model*

After training, we evaluate the model's performance

1. **Loading the Cora Dataset**: We use **Planetoid** from **torch\_geometric.datasets** to load the Cora dataset, normalizing its features.
2. **Defining the GNN Model**: We define a simple two-layer GCN model using **GCNConv** layers.
3. **Training the Model**: The model is trained using the Adam optimizer and negative log-likelihood loss.
4. **Evaluating the Model**: We compute accuracy on the training, validation, and test sets to monitor performance.

**Code:**

import torch

import torch.nn.functional as F

from torch\_geometric.datasets import Planetoid

import torch\_geometric.transforms as T

from torch\_geometric.nn import GCNConv

# Load the Cora dataset

dataset = Planetoid(root='/tmp/Cora', name='Cora', transform=T.NormalizeFeatures())

data = dataset[0]

print(f'Dataset: {dataset}:')

print('======================')

print(f'Number of graphs: {len(dataset)}')

print(f'Number of features: {dataset.num\_features}')

print(f'Number of classes: {dataset.num\_classes}')

class GCN(torch.nn.Module):

def \_\_init\_\_(self):

super(GCN, self).\_\_init\_\_()

self.conv1 = GCNConv(dataset.num\_features, 16)

self.conv2 = GCNConv(16, dataset.num\_classes)

def forward(self, data):

x, edge\_index = data.x, data.edge\_index

x = self.conv1(x, edge\_index)

x = F.relu(x)

x = F.dropout(x, training=self.training)

x = self.conv2(x, edge\_index)

return F.log\_softmax(x, dim=1)

model = GCN()

print(model)

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model = model.to(device)

data = data.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight\_decay=5e-4)

def train():

model.train()

optimizer.zero\_grad()

out = model(data)

loss = F.nll\_loss(out[data.train\_mask], data.y[data.train\_mask])

loss.backward()

optimizer.step()

return loss.item()

def test():

model.eval()

logits, accs = model(data), []

for mask in [data.train\_mask, data.val\_mask, data.test\_mask]:

pred = logits[mask].max(1)[1]

acc = pred.eq(data.y[mask]).sum().item() / mask.sum().item()

accs.append(acc)

return accs

for epoch in range(200):

loss = train()

train\_acc, val\_acc, test\_acc = test()

print(f'Epoch: {epoch:03d}, Loss: {loss:.4f}, Train Acc: {train\_acc:.4f}, Val Acc: {val\_acc:.4f}, Test Acc: {test\_acc:.4f}')

model.eval()

logits = model(data)

pred = logits[data.test\_mask].max(1)[1]

test\_acc = pred.eq(data.y[data.test\_mask]).sum().item() / data.test\_mask.sum().item()

print(f'Test Accuracy: {test\_acc:.4f}')

**Output:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**PRACTICAL 9**

**Aim**: Analyze Twitter conversations to identify the most active and influential users using Machine Learning Algorithms with Gephi Tool.

**Theory:**

The practical aims to analyze Twitter conversations to identify the most active and influential users using network analysis and machine learning techniques. The integration of tools like networkx for network analysis, matplotlib for visualization, and Gephi for advanced network analysis, along with machine learning algorithms, provides a robust framework for social network analysis. The key metrics such as degree centrality and betweenness centrality help in understanding the influence and connectivity of users in the Twitter network.

*Network Analysis:* Network analysis involves the study of complex networks represented as graphs, where nodes represent entities (in this case, Twitter users) and edges represent interactions between them (such as retweets, mentions, or replies). Directed graphs (DiGraphs) are used to capture the directionality of interactions, which is essential in understanding the flow of information and influence in the network.

*NetworkX Library:* networkx is a Python library used for the creation, manipulation, and study of complex networks. It provides tools for constructing graphs, analyzing their structure, and visualizing them. In this practical, we use networkx to create a directed graph of Twitter interactions and calculate centrality measures to identify influential users.

*Visualization with Matplotlib:* matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used here to visualize the network of Twitter users, providing a graphical representation of the interactions and connections between them.

*Centrality Measures:* Centrality measures are used to identify the most important nodes within a graph. In this practical, we focus on two key centrality measures:

* *Degree Centrality:* This measures the number of direct connections a node has. In the context of Twitter, a user with high degree centrality has many direct interactions with other users, indicating high activity.
* *Betweenness Centrality*: This measures the extent to which a node lies on the shortest paths between other nodes. A user with high betweenness centrality acts as a bridge in the network, playing a critical role in the flow of information.

*Gephi Tool:* Gephi is an open-source network analysis and visualization software. It provides advanced features for visualizing large networks and running various graph algorithms. By importing the interaction data into Gephi, we can perform a more detailed and visually intuitive analysis of the Twitter network. Gephi’s visualization capabilities help in identifying clusters, key influencers, and the overall structure of the network.

*Machine Learning for Influential User Identification***:** Machine learning algorithms can be used to further analyze the activity patterns of users and classify them based on their influence and activity levels. Clustering algorithms like K-Means can group users into clusters with similar interaction patterns, helping to identify the most active and influential users.

**Code:**

import networkx as nx

import matplotlib.pyplot as plt

# Sample data (replace with your own data)

edges = [("user1", "user2"), ("user1", "user3"), ("user2", "user3"), ("user2", "user4")]

# Create a directed graph

G = nx.DiGraph()

# Add edges to the graph

G.add\_edges\_from(edges)

# Visualize the network

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

plt.show()

# Calculate degree centrality

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

print("Degree Centrality:", degree\_centrality)

# Calculate betweenness centrality

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

print("Betweenness Centrality:", betweenness\_centrality)

**Output:**

