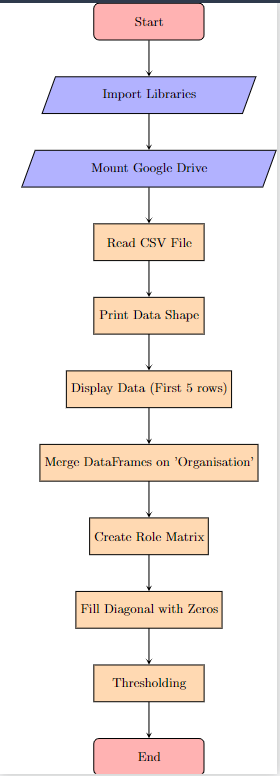
**Experiment 1**

**Aim:** Write a program to construct a social network object using Sociomatrix.

**Flowchart:**

****

**Code:**

#Sociomatrix- It is a square matrix where rows and columns represent the nodes (individuals or entities) in the network

#and the entries in the matrix represent the presence or absence of relationships between nodes.

#social network object

#a "social network object" refers to a data structure that represents a social network or a network of relationships between individuals or entities

import pandas as pd #Pandas is a Python library used for working with data sets. #It has functions for analyzing, cleaning, exploring, and manipulating data.

import numpy as np #. NumPy is used for working with arrays.

import networkx as nx #for working with graphs and networks

import matplotlib.pyplot as plt #for plotting graphs

%matplotlib inline

from operator import itemgetter #If multiple items are specified, returns a tuple of lookup values. eg: itemgetter(1, 3, 5)('ABCDEFG') ->output:('B', 'D', 'F')

from google.colab import drive

drive.mount('/content/drive/')

df = pd.read\_csv ("/content/drive/MyDrive/organization.csv")

print(df.shape)

df.head(5)

df\_merge = df.merge(df,on="Organisation")

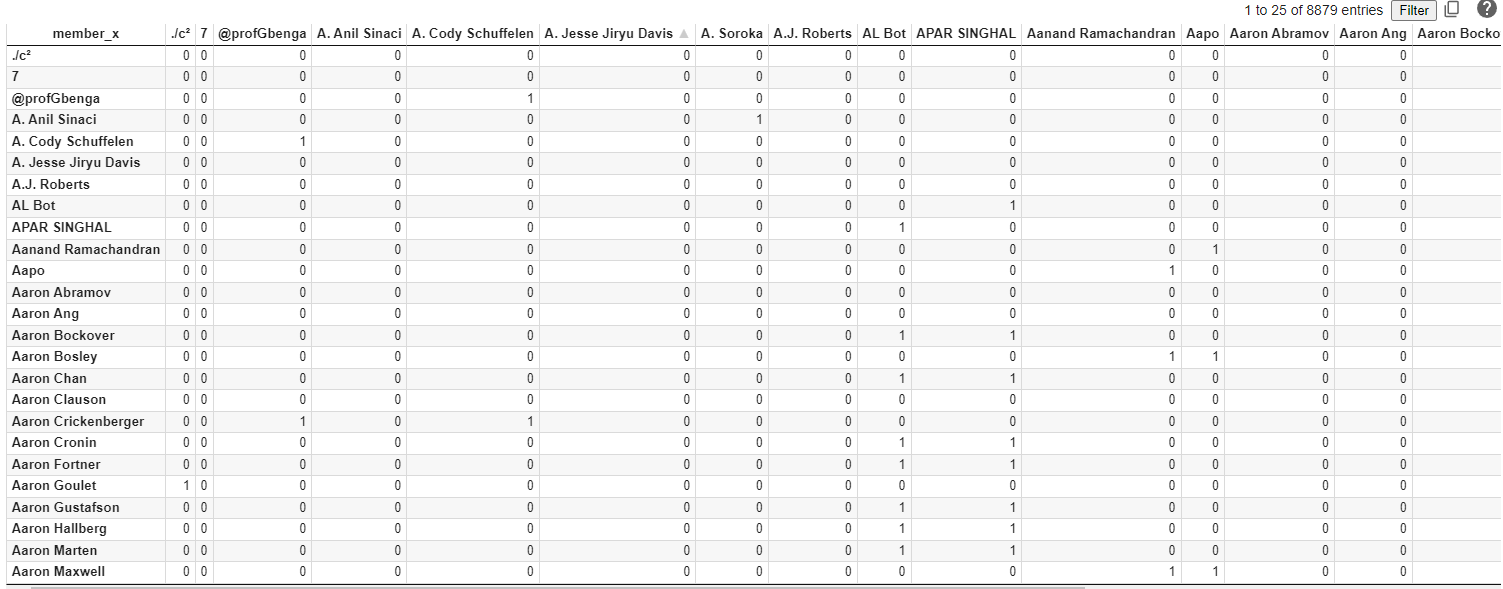
role\_mat = pd.crosstab(df\_merge.member\_x,df\_merge.member\_y) #This function allows you to compute a frequency table of two or more variables # get counts of categories

np.fill\_diagonal(role\_mat.values, 0) # Return the filled value in the diagonal of an array.

role\_mat[role\_mat >= 1] = 1

role\_mat

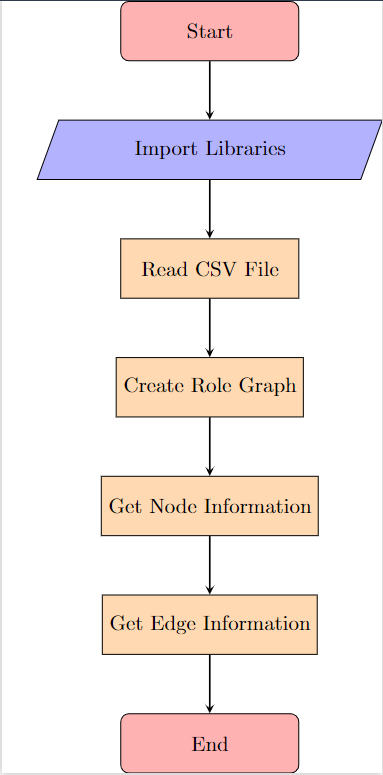
**Output:**



**Experiment 2**

**Aim:** Write a program to create node and edge lists network objects.

**Flowchart:**



**Code:**

import pandas as pd

import numpy as np

import networkx as nx #NetworkX provides classes for graphs which allow multiple edges between any pair of nodes.

#for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

import matplotlib.pyplot as plt

%matplotlib inline

from operator import itemgetter #If multiple items are specified, returns a tuple of lookup values. eg: itemgetter(1, 3, 5)('ABCDEFG') ->output:('B', 'D', 'F')

df = pd.read\_csv ("/content/drive/MyDrive/organization.csv")

#Returns a graph from Pandas DataFrame containing an edge list

role\_graph= nx.from\_pandas\_edgelist(df,source="member",target="Organisation")

type(role\_graph)

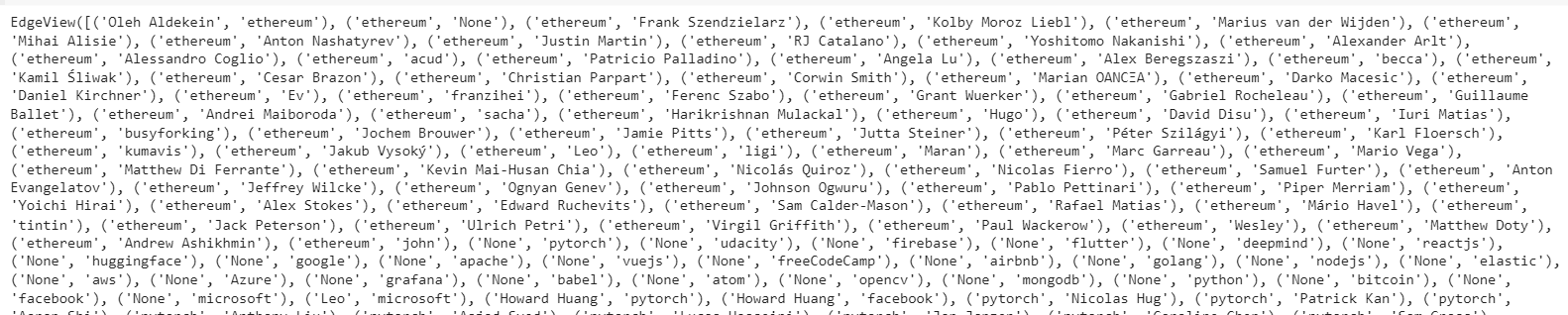
role\_graph.nodes()



len(role\_graph.nodes())

8920

role\_graph.edges()



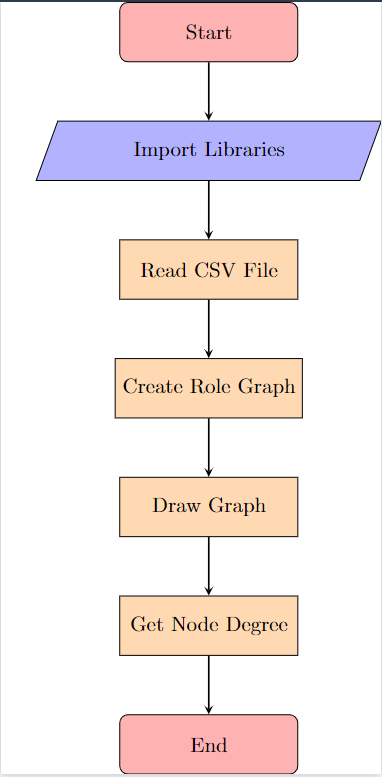
len(role\_graph.edges())

9656

**Experiment 3**

**Aim:** Write a program to visualize a social network with matplotlib library.

**Flowchart:**



**Code:**

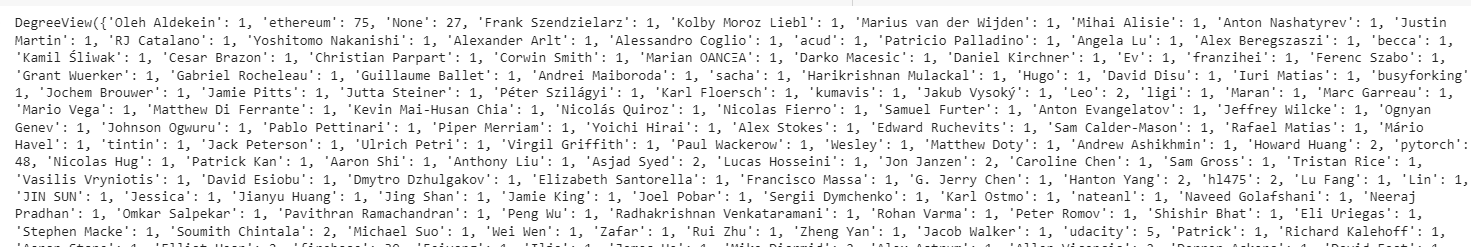
df = pd.read\_csv("/content/drive/MyDrive/organization.csv")

nx.draw(role\_graph)

#Draw the graph as a simple representation with no node labels or edge labels and using the full Matplotlib figure area and no axis labels by default

nx.degree(role\_graph) #The node degree is the number of edges adjacent to the node

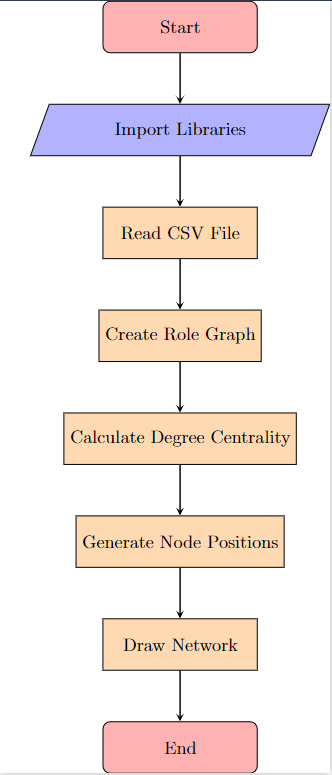
**Output:**



**Experiment 4**

**Aim:** Write a program to calculate the node-level degree centrality measures for social network and plot the graph.

**Flowchart:**



**Code:**

import pandas as pd

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

%matplotlib inline

from operator import itemgetter #suite of standard Python operators

df=pd.read\_csv("/content/drive/MyDrive/organization.csv")

nx.degree\_centrality(role\_graph) #The degree centrality for a node v is the fraction of nodes it is connected to.

#G = nx.Graph([(0, 1), (0, 2), (0, 3), (1, 2), (1, 3)])

#nx.degree\_centrality(G)

#{0: 1.0, 1: 1.0, 2: 0.6666666666666666, 3: 0.6666666666666666} 0 is connected with all 3 nodes so its degree centrality is 1 while 2 is connected with 2 out of 3 so 66%

pos = nx.spring\_layout(role\_graph) #Position nodes using Fruchterman-Reingold force-directed algorithm.

#The algorithm simulates a force-directed representation of the network treating edges as springs holding nodes close

#while treating nodes as repelling objects, sometimes called an anti-gravity force. Simulation continues until the positions are close to an equilibrium.

degCent = nx.degree\_centrality(role\_graph)

node\_color = [20000.0 \* role\_graph.degree(v) for v in role\_graph]

node\_size = [v \* 10000 for v in degCent.values()]

plt.figure(figsize=(15,15))

nx.draw\_networkx(role\_graph, with\_labels=False,

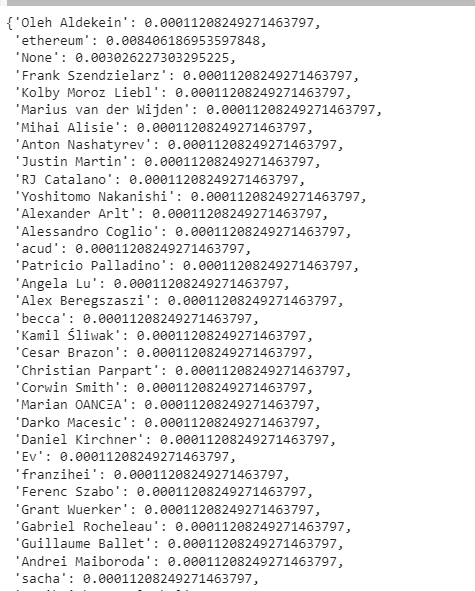
node\_color=node\_color,

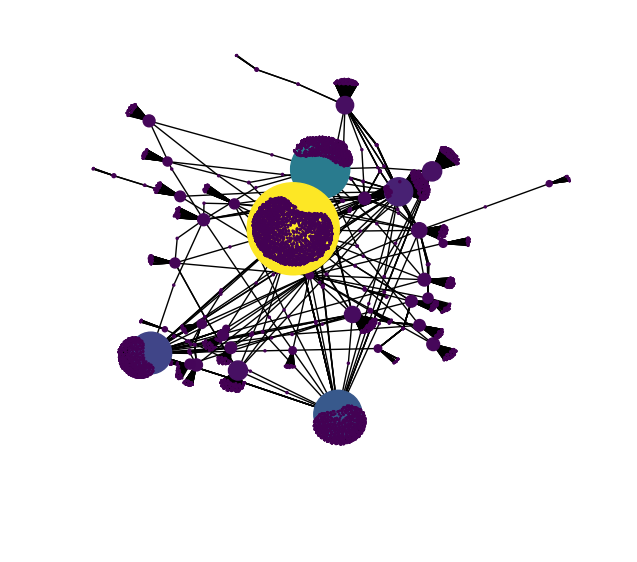
node\_size=node\_size )

plt.axis('off')

sorted(degCent, key=degCent.get, reverse=True)[:5]

**Output:**

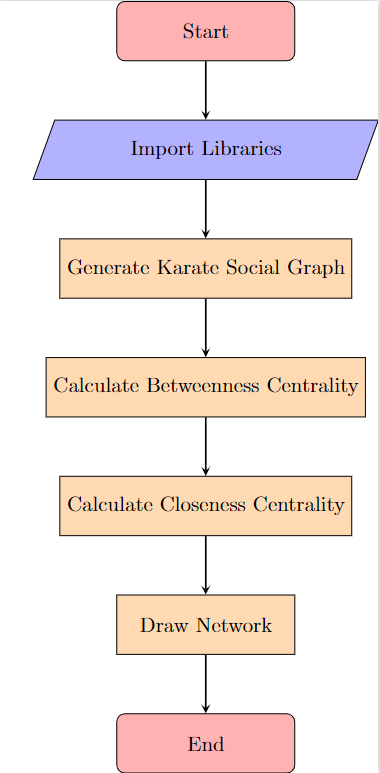




**Experiment 5**

**Aim:** Write a program to calculate the node-level betweenness centrality and closeness centrality measures for social network and plot the graph.

**Flowchart:**



**Code:**

import random

import networkx as nx

# G is the Karate Social Graph

G = nx.karate\_club\_graph()

# betweenness\_centrality

b=nx.betweenness\_centrality(G)

print(b)

# closeness\_centrality

c = nx.closeness\_centrality(G)

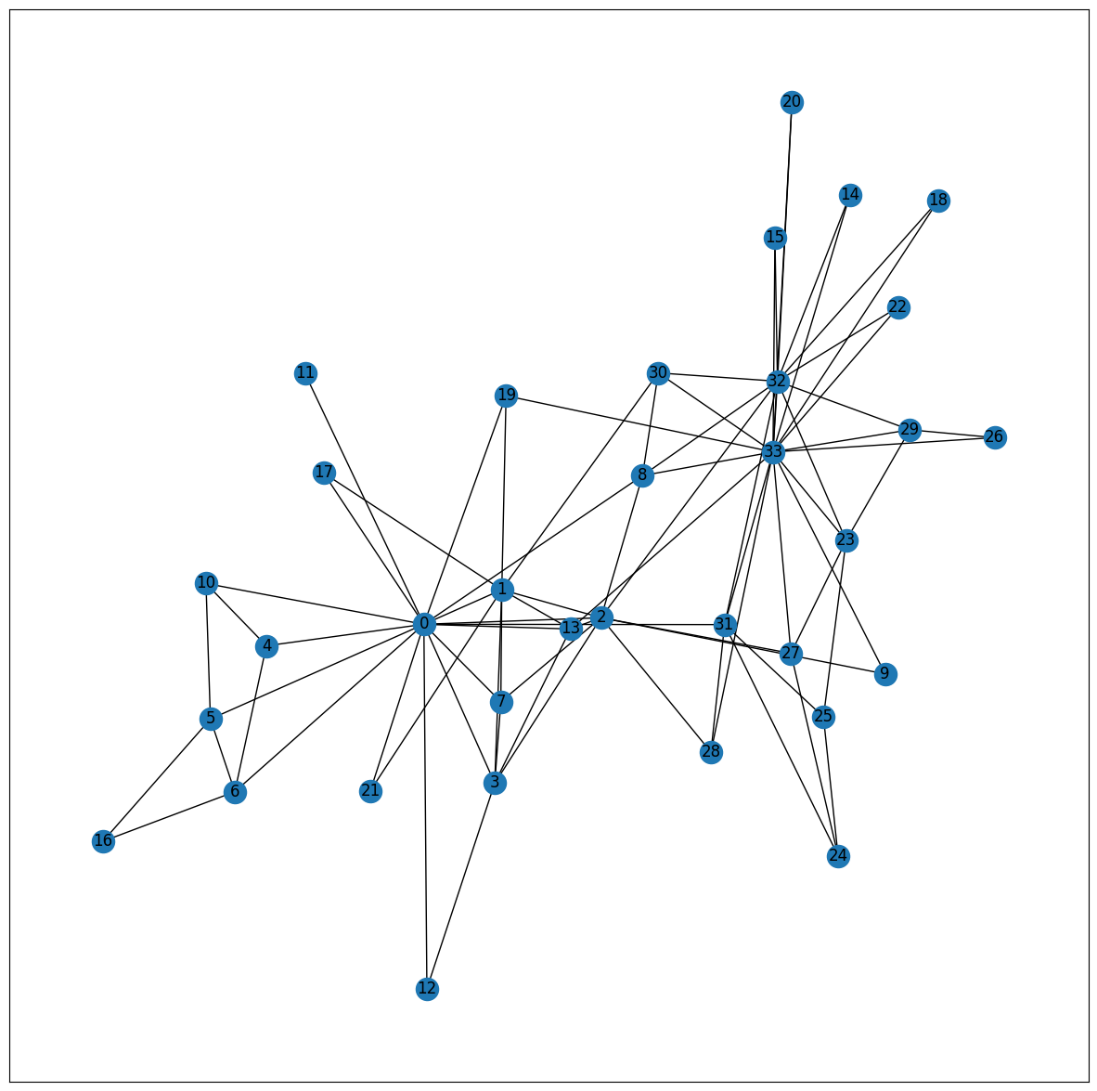
print(c)

plt.figure(figsize =(15, 15))

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

**Output:**

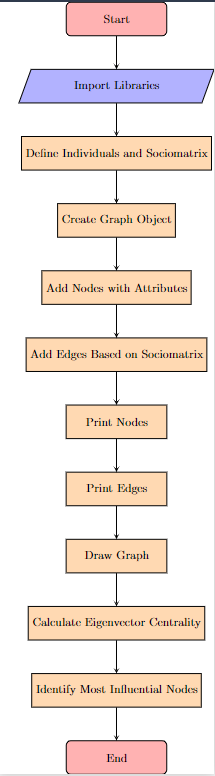
****



**Experiment 6**

**Aim:** Write a program to create network-level measures of centrality for social network.

**Flowchart:**

****

**Code:**

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

# Define the names of individuals

names = ['Node\_A', 'Node\_B', 'Node\_C']

# Define the sociomatrix

sociomatrix = np.array([

# A , B , C

[0, 1, 0 ], # Node\_A

[1, 0, 1 ], # Node\_B

[0, 1, 0 ] # Node\_C

])

# Creating a Graph Object

G = nx.Graph()

# Adding Nodes with Attributes

for i, name in enumerate(names):

G.add\_node(name)

# Adding Edges Based on Sociomatrix

for i in range(len(names)):

for j in range(i + 1, len(names)):

if sociomatrix[i][j] == 1:

G.add\_edge(names[i], names[j])

# Print the nodes of the graph

print("Nodes of the social network:")

print(names)

# Print the edges of the graph

print("Edges of the social network:")

print(list(G.edges()))

# Drawing the Graph

pos = nx.spring\_layout(G) # Positions for all nodes

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

nx.draw(G, pos, with\_labels=True, node\_size=500, node\_color='skyblue',

font\_size=10, font\_weight='bold', edge\_color='gray')

# Displaying the Graph

plt.title('Social Network Visualization')

plt.show()

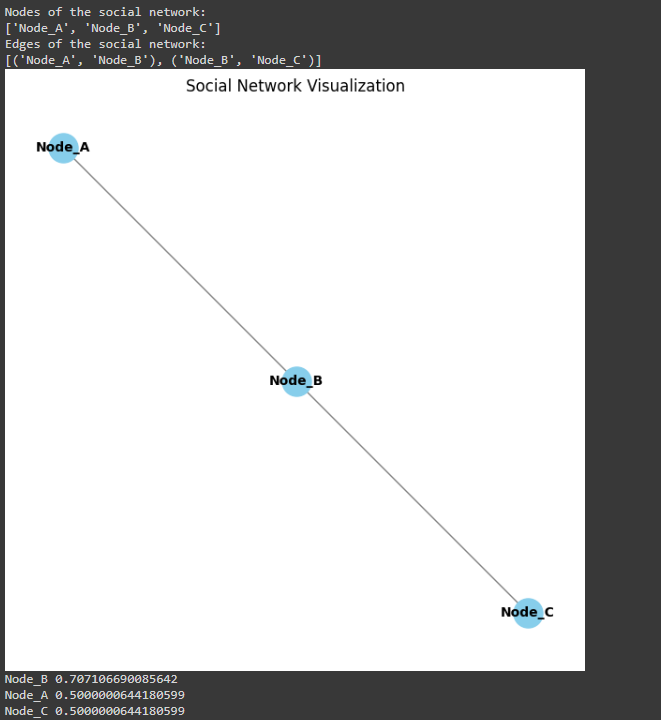
eigenvector\_centrality = nx.eigenvector\_centrality(G)

#Sort for identifying most inflential nodes using eigenvector centrality

for node in sorted(eigenvector\_centrality, key=eigenvector\_centrality.get, reverse=True):

print(node, eigenvector\_centrality[node])

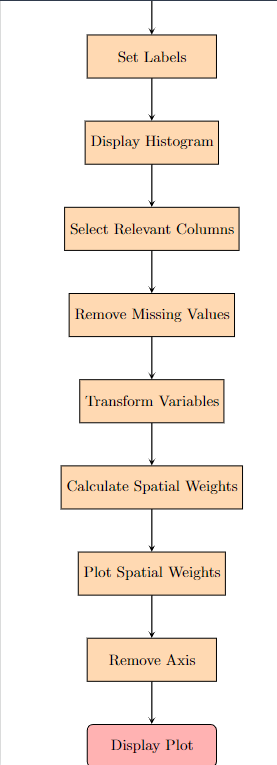
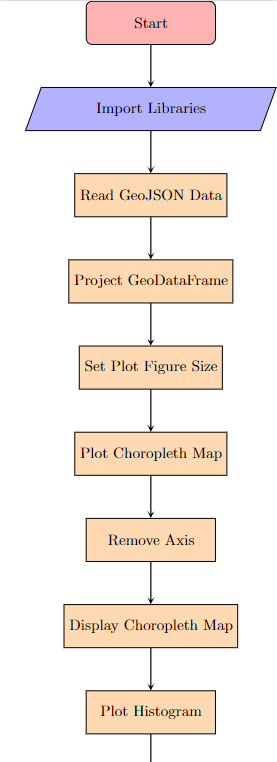
**Output:**

****

**Experiment 7**

**Aim:** Write a program to incorporate and plot network connectivity in a spatial regression model.

**Flowchart:**



**Code:**

# Importing necessary libraries

import geopandas

import matplotlib.pyplot as plt

import numpy

import pysal.lib

# Reading GeoJSON data containing world energy indicators into a GeoDataFrame

countries = geopandas.read\_file("https://michaelminn.net/tutorials/data/2019-world-energy-indicators.geojson")

# Projecting the GeoDataFrame to Web Mercator (EPSG:3857) projection

countries = countries.to\_crs("EPSG:3857")

# Setting the size of the plot figure

plt.rcParams['figure.figsize'] = [9, 6]

# Plotting the choropleth map of MM\_BTU\_per\_Capita using GeoDataFrame plot function

axis = countries.plot("MM\_BTU\_per\_Capita", cmap="coolwarm", legend=True, scheme="quantiles")

# Removing axis

axis.set\_axis\_off()

# Displaying the plot

plt.show()

# Plotting a histogram of MM\_BTU\_per\_Capita

axis = plt.hist(countries["MM\_BTU\_per\_Capita"])

# Setting labels for x-axis and y-axis

plt.xlabel("MM BTU per Capita")

plt.ylabel("Number of Countries")

# Displaying the histogram plot

plt.show()

# Defining variables for modeling

dependent\_name = ["Democracy\_Index"]

independent\_names = ["GDP\_per\_Capita\_PPP\_Dollars", "Military\_Percent\_GDP", "Resource\_Rent\_Percent\_GDP", "Industry\_Percent\_GDP"]

# Selecting relevant columns from GeoDataFrame and removing rows with missing values

model\_data = countries[dependent\_name + independent\_names + ["geometry", "Latitude", "Longitude"]]

model\_data = model\_data.dropna()

# Transforming certain variables by taking their natural logarithm plus one

transform\_vars = ["GDP\_per\_Capita\_PPP\_Dollars", "Military\_Percent\_GDP", "Resource\_Rent\_Percent\_GDP"]

model\_data[transform\_vars] = numpy.log(model\_data[transform\_vars] + 1)

# Calculating spatial weights using K-nearest neighbors (KNN) method

weights = pysal.lib.weights.KNN.from\_dataframe(model\_data, k=4)

# Plotting spatial weights on a map

axis = model\_data.plot(edgecolor="lightgray", facecolor="none")

model\_data["index"] = model\_data.index

weights.plot(gdf=model\_data, indexed\_on="index", ax=axis)

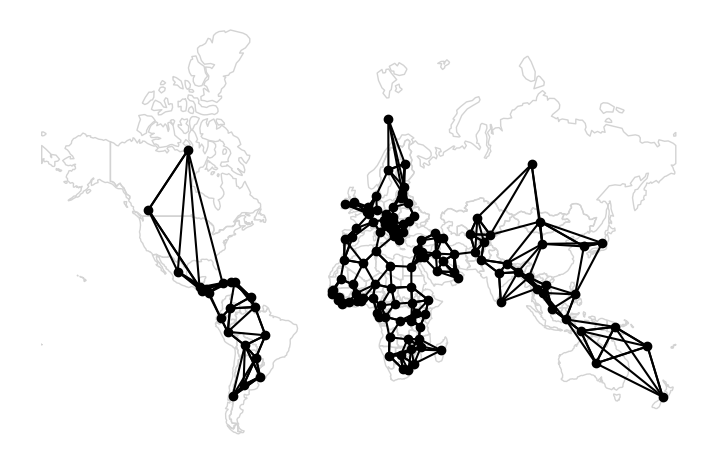
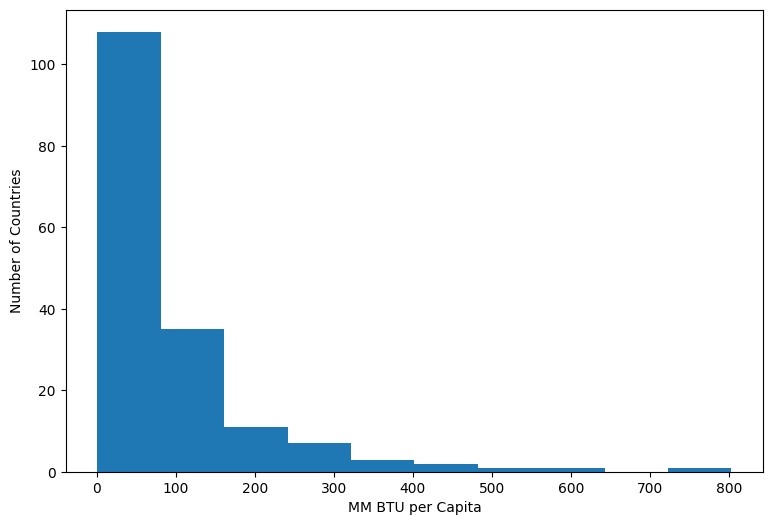
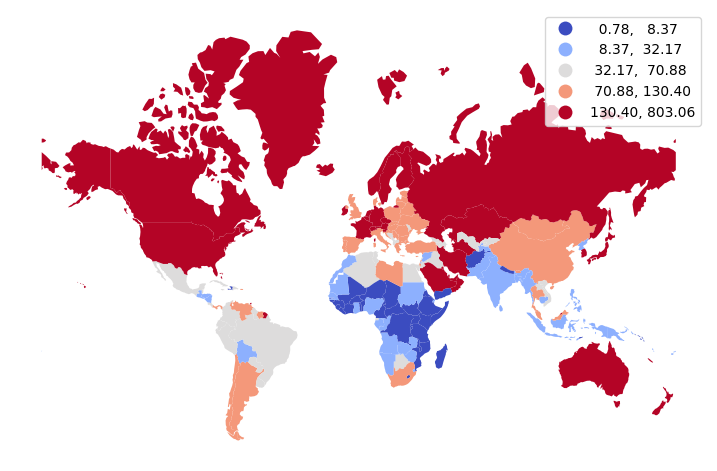
# Removing axis

axis.set\_axis\_off()

# Displaying the plot

plt.show()

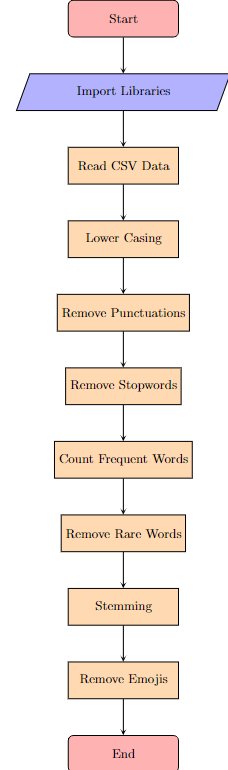
**Output:**



**Experiment 8**

**Aim:** Write a program for preprocessing of social media text and detect sentiment from it.

**Flowchart:**

****

**Code:**

#TEXT PREPROCESSING

import numpy as np

import pandas as pd

import re

import nltk

import spacy

import string

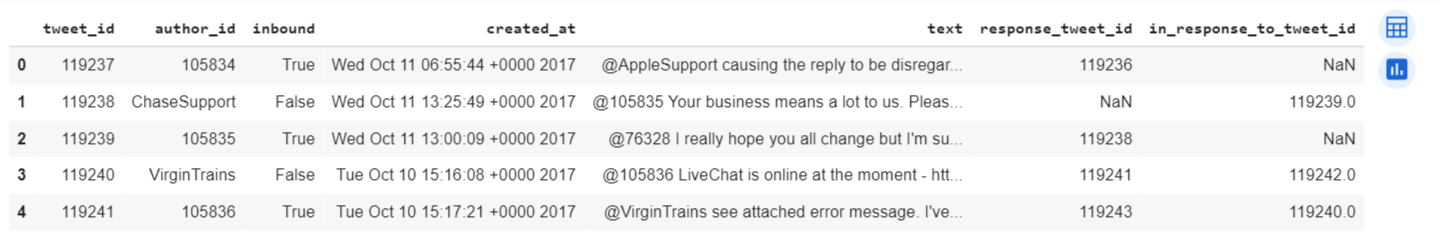
pd.options.mode.chained\_assignment = None

full\_df = pd.read\_csv("/content/drive/MyDrive/sample.csv")

df = full\_df[["text"]]

df["text"] = df["text"].astype(str)

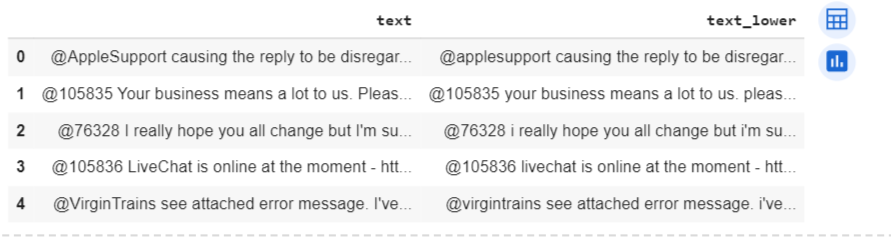
full\_df.head()



#Lower Casing

df["text\_lower"] = df["text"].str.lower()

df.head()



#Removal of Punctuations

#drop the new column created in last cell

df.drop(["text\_lower"], axis=1, inplace=True)

PUNCT\_TO\_REMOVE = string.punctuation

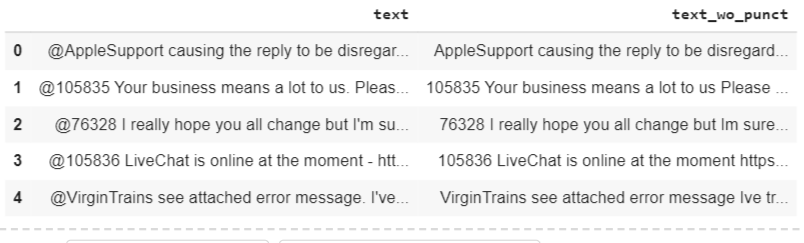
def remove\_punctuation(text):

"""custom function to remove the punctuation"""

return text.translate(str.maketrans('', '', PUNCT\_TO\_REMOVE))

df["text\_wo\_punct"] = df["text"].apply(lambda text: remove\_punctuation(text))

df.head()



#Removal of stopwords

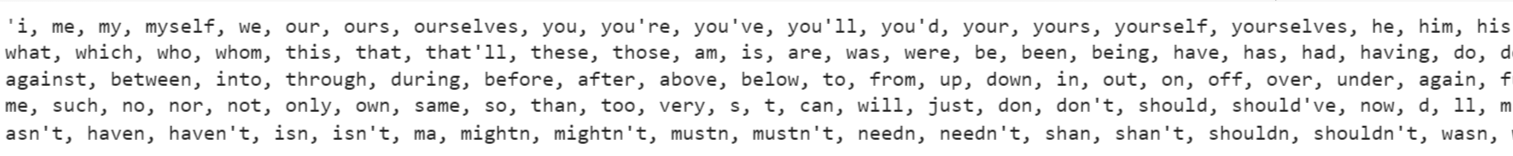
#Stopwords are commonly occuring words in a language like 'the', 'a' and so on. They can be removed from the text most of the times.

#import nltk

#nltk.download('stopwords')

from nltk.corpus import stopwords

", ".join(stopwords.words('english'))



STOPWORDS = set(stopwords.words('english'))

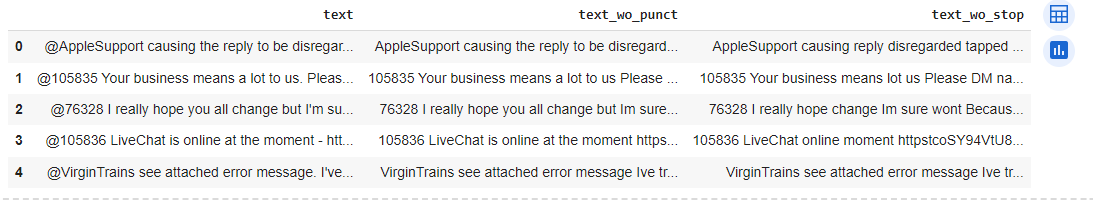
def remove\_stopwords(text):

"""custom function to remove the stopwords"""

return " ".join([word for word in str(text).split() if word not in STOPWORDS])

df["text\_wo\_stop"] = df["text\_wo\_punct"].apply(lambda text: remove\_stopwords(text))

df.head()



#Removal of Frequent words

from collections import Counter

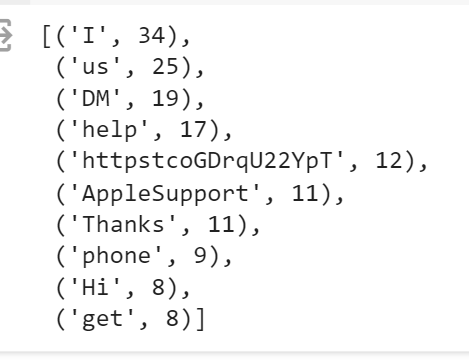
cnt = Counter()

for text in df["text\_wo\_stop"].values:

for word in text.split():

cnt[word] += 1

cnt.most\_common(10)



#Removal of Rare words

# Drop the two columns which are no more needed

df.drop(["text\_wo\_punct", "text\_wo\_stop"], axis=1, inplace=True)

n\_rare\_words = 10

RAREWORDS = set([w for (w, wc) in cnt.most\_common()[:-n\_rare\_words-1:-1]])

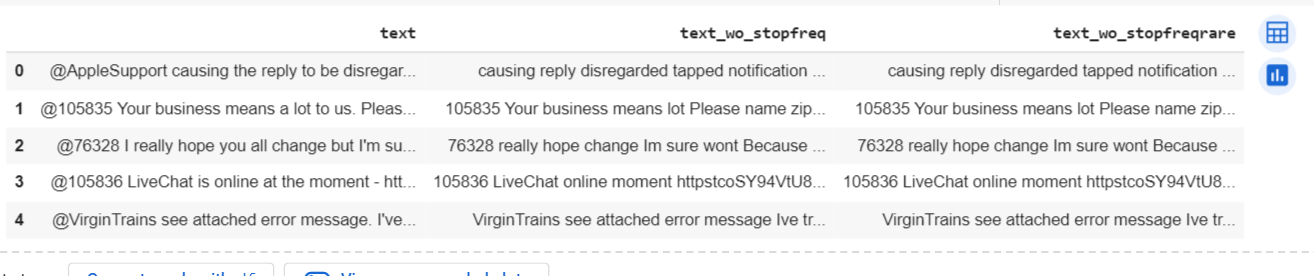
def remove\_rarewords(text):

"""custom function to remove the rare words"""

return " ".join([word for word in str(text).split() if word not in RAREWORDS])

df["text\_wo\_stopfreqrare"] = df["text\_wo\_stopfreq"].apply(lambda text: remove\_rarewords(text))

df.head()



#Stemming

#Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form

#for example, if there are two words in the corpus walks and walking, then stemming will stem the suffix to make them walk

from nltk.stem.porter import PorterStemmer

# Drop the two columns

df.drop(["text\_wo\_stopfreq", "text\_wo\_stopfreqrare"], axis=1, inplace=True)

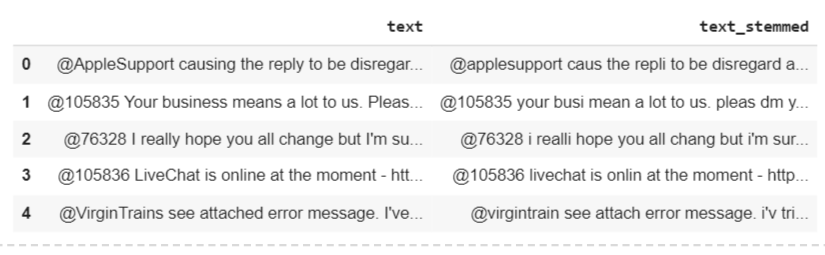
stemmer = PorterStemmer()

def stem\_words(text):

return " ".join([stemmer.stem(word) for word in text.split()])

df["text\_stemmed"] = df["text"].apply(lambda text: stem\_words(text))

df.head()



#Removal of Emojis

def remove\_emoji(string):

emoji\_pattern = re.compile("["

u"\U0001F600-\U0001F64F" # emoticons

u"\U0001F300-\U0001F5FF" # symbols & pictographs

u"\U0001F680-\U0001F6FF" # transport & map symbols

u"\U0001F1E0-\U0001F1FF" # flags (iOS)

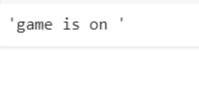
u"\U00002702-\U000027B0"

u"\U000024C2-\U0001F251"

"]+", flags=re.UNICODE)

return emoji\_pattern.sub(r'', string)

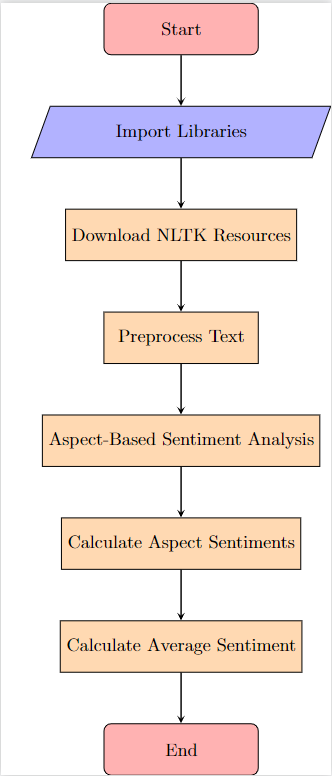
remove\_emoji("game is on 🔥🔥")



**Experiment 9**

**Aim:** Write a program for Aspect-based sentiment analysis of social media text.

**Flowchart:**

****

**Code:**

import nltk

from textblob import TextBlob

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.corpus import stopwords

from collections import defaultdict

# Download necessary NLTK resources

nltk.download('punkt')

nltk.download('stopwords')

def preprocess\_text(text):

# Tokenize text into sentences

sentences = sent\_tokenize(text)

# Tokenize each sentence into words, remove stopwords and punctuation

stop\_words = set(stopwords.words('english'))

preprocessed\_sentences = []

for sentence in sentences:

words = word\_tokenize(sentence)

words = [word.lower() for word in words if word.isalnum() and word.lower() not in stop\_words]

preprocessed\_sentences.append(words)

return preprocessed\_sentences

def aspect\_based\_sentiment\_analysis(text):

preprocessed\_text = preprocess\_text(text)

print(preprocessed\_text)

aspect\_sentiments = defaultdict(list)

for sentence in preprocessed\_text:

# Convert list of words back to sentence

sentence = " ".join(sentence)

blob = TextBlob(sentence)

# Extract noun phrases as aspects

noun\_phrases = blob.noun\_phrases

# Analyze sentiment for each aspect

for aspect in noun\_phrases:

aspect\_sentiments[aspect].append(blob.sentiment.polarity)

# Calculate average sentiment for each aspect

aspect\_average\_sentiments = {}

for aspect, sentiments in aspect\_sentiments.items():

aspect\_average\_sentiments[aspect] = sum(sentiments) / len(sentiments)

return aspect\_average\_sentiments

# Example usage

social\_media\_text = """

The camera quality of this phone is amazing. But battery life is poor.

I love the design of the new laptop. The performance is also great!

"""

aspect\_sentiments = aspect\_based\_sentiment\_analysis(social\_media\_text)

for aspect, sentiment in aspect\_sentiments.items():

print(f"Aspect: {aspect}, Sentiment: {sentiment}")

**Output:**

