VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANA SANGAMA, BELAGAVI – 590 018

Assignment Report
on

Data Visualization

Submitted By

Yashas R Yadav - 1RN21CD061

Under the Guidance of
Ms. Vinutha S
Asst. Professor
Department of CSE (Data Science)

RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

2024-2025

Table of Contents

1. Introduction

Data visualization is a crucial aspect of data science that allows researchers and analysts to interpret complex data sets effectively. In this report, we explore various techniques for visualizing data using Python, focusing on the implementation of Kernel Density Estimation, bivariate distributions, geospatial data visualization, and network connections. The primary objective of this assignment is to develop a series of codes that demonstrate these visualization techniques, utilizing libraries such as Bokeh and Pandas. By leveraging these tools, we aim to create interactive and informative visualizations that enhance our understanding of the underlying data patterns and relationships.

2. Question 1: Kernel Density Estimation

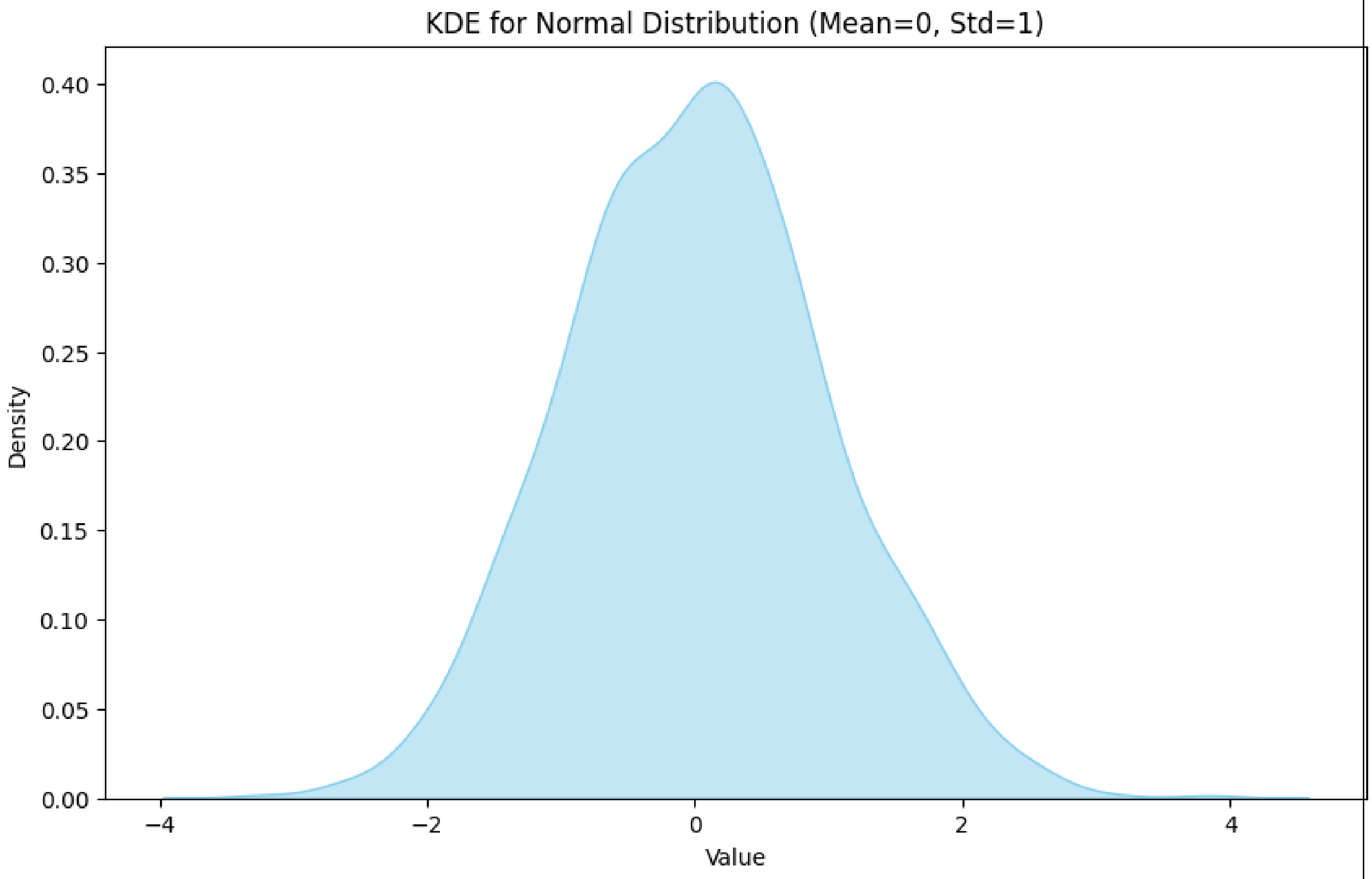
Objective

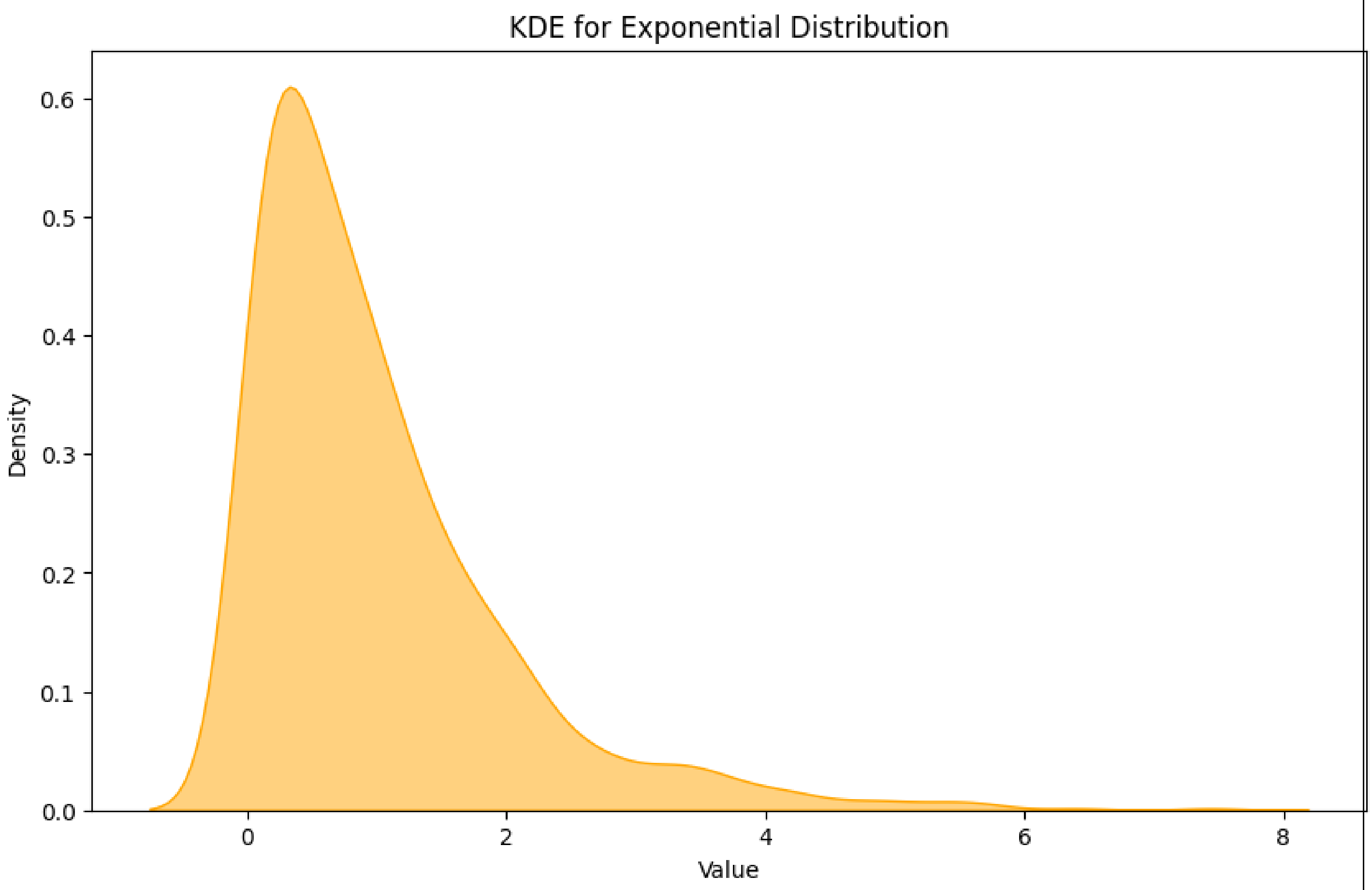
Develop a code to demonstrate Kernel Density Estimation

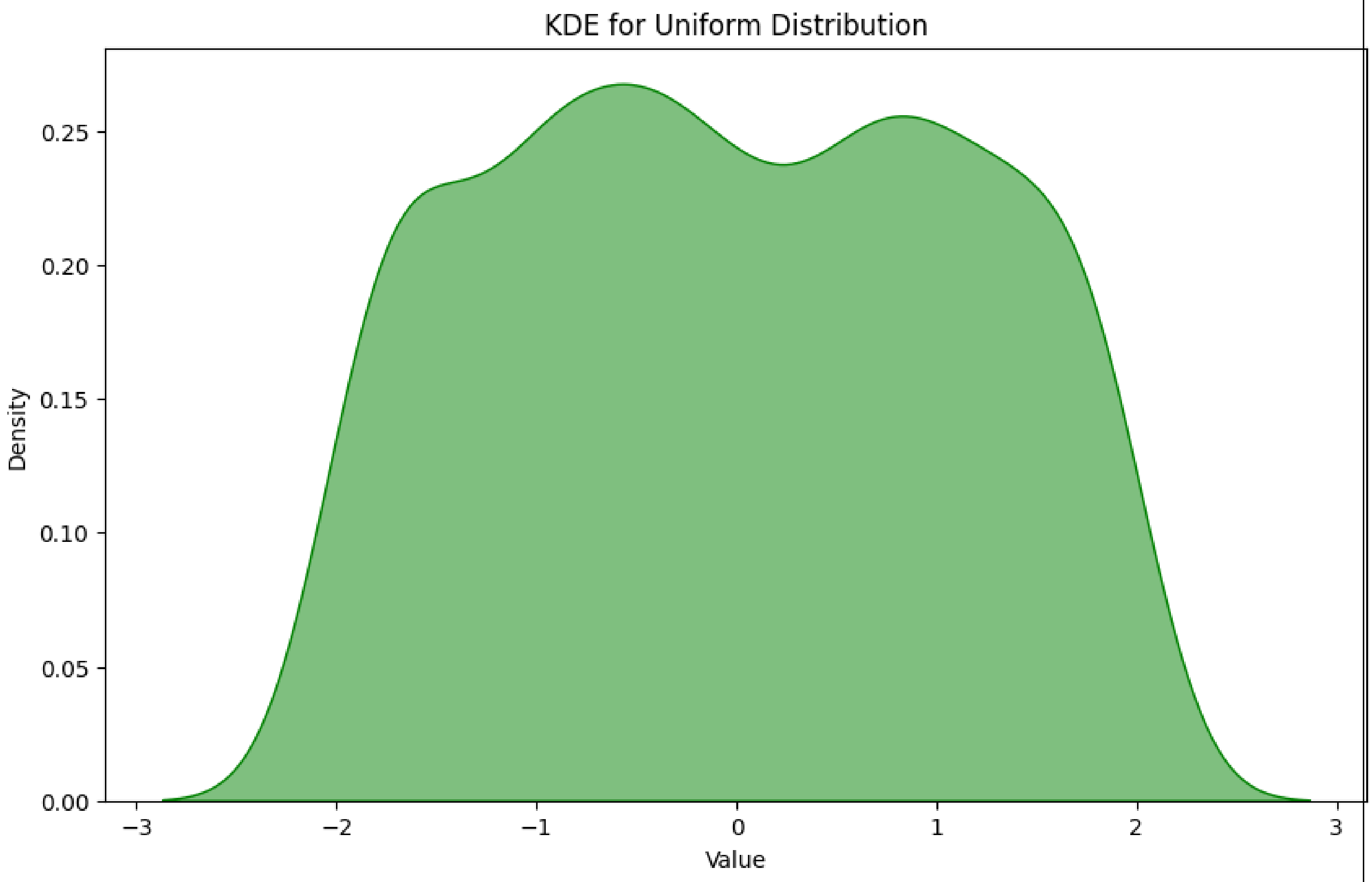
Code Snippet:

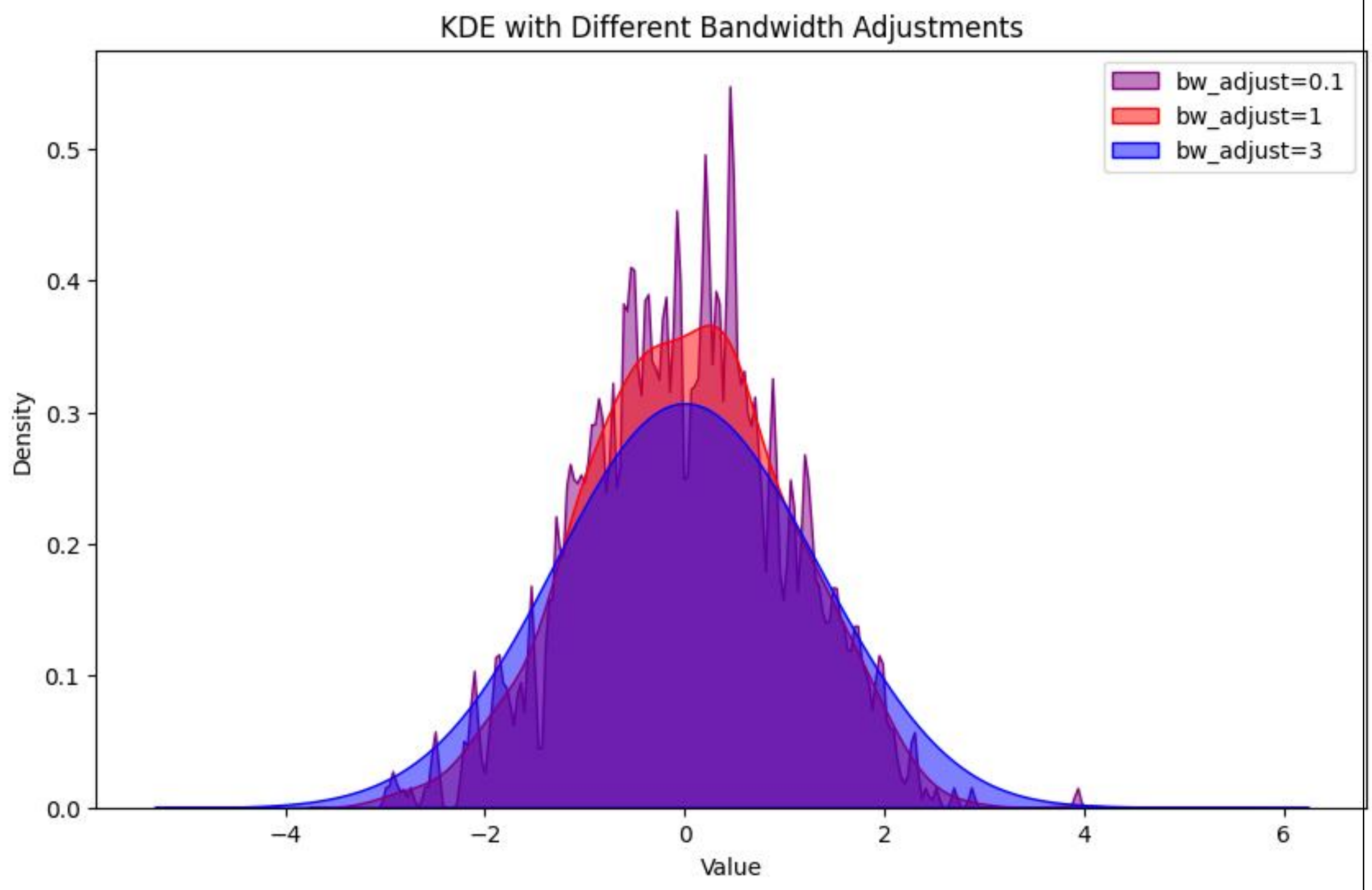
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set a random seed for reproducibility
np.random.seed(42)

# ----------------------------
# 1. Normal Distribution (Gaussian)
# ----------------------------
data_normal = np.random.normal(loc=0, scale=1, size=1000)

# Plot KDE for Normal distribution
plt.figure(figsize=(10, 6))
sns.kdeplot(data_normal, fill=True, color="skyblue", alpha=0.5)
plt.title("KDE for Normal Distribution (Mean=0, Std=1)")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()

# ----------------------------
# 2. Exponential Distribution
# ----------------------------
data_exponential = np.random.exponential(scale=1, size=1000)

# Plot KDE for Exponential distribution
plt.figure(figsize=(10, 6))
sns.kdeplot(data_exponential, fill=True, color="orange", alpha=0.5)
plt.title("KDE for Exponential Distribution")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()

# ----------------------------
# 3. Uniform Distribution
# ----------------------------
data_uniform = np.random.uniform(low=-2, high=2, size=1000)

# Plot KDE for Uniform distribution
plt.figure(figsize=(10, 6))
sns.kdeplot(data_uniform, fill=True, color="green", alpha=0.5)
plt.title("KDE for Uniform Distribution")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()

# ----------------------------
# 4. KDE with Different Bandwidth
# ----------------------------
# Create another normal distribution dataset
data_normal2 = np.random.normal(loc=0, scale=1, size=1000)

# Plot KDE with different bandwidths
plt.figure(figsize=(10, 6))
sns.kdeplot(data_normal2, fill=True, color="purple", alpha=0.5, bw_adjust=0.1)  # Lower bandwidth (sharper)
sns.kdeplot(data_normal2, fill=True, color="red", alpha=0.5, bw_adjust=1)    # Default bandwidth
sns.kdeplot(data_normal2, fill=True, color="blue", alpha=0.5, bw_adjust=3)    # Higher bandwidth (smoother)

plt.title("KDE with Different Bandwidth Adjustments")
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend(["bw_adjust=0.1", "bw_adjust=1", "bw_adjust=3"])
plt.show()

# ----------------------------
# 5. KDE with Different Kernels (Gaussian and Tophat)
# ----------------------------
data_normal3 = np.random.normal(loc=0, scale=1, size=1000)

# Plot KDE with different kernels
plt.figure(figsize=(10, 6))
sns.kdeplot(data_normal3, fill=True, color="purple", alpha=0.5, kernel='gau')  # Gaussian kernel
sns.kdeplot(data_normal3, fill=True, color="orange", alpha=0.5, kernel='top') # Tophat kernel

plt.title("KDE with Different Kernels")
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend(["Gaussian Kernel", "Tophat Kernel"])
plt.show()

# ----------------------------
# 6. KDE with Log Transformation
# ----------------------------
data_skewed = np.random.lognormal(mean=0, sigma=1, size=1000)

# Plot KDE for Log-normal distribution (skewed data)
plt.figure(figsize=(10, 6))
sns.kdeplot(data_skewed, fill=True, color="brown", alpha=0.5)
plt.title("KDE for Log-normal Distribution (Skewed Data)")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()

# ----------------------------
# 7. KDE with Multiple Distributions Overlayed
# ----------------------------
# Generate data from multiple distributions
data_mixed = np.concatenate([data_normal, data_exponential, data_uniform])

# Plot KDE for combined dataset
plt.figure(figsize=(10, 6))
sns.kdeplot(data_normal, fill=True, color="blue", alpha=0.5, label="Normal")
sns.kdeplot(data_exponential, fill=True, color="orange", alpha=0.5, label="Exponential")
sns.kdeplot(data_uniform, fill=True, color="green", alpha=0.5, label="Uniform")
sns.kdeplot(data_mixed, fill=True, color="purple", alpha=0.5, label="Mixed")

plt.title("KDE for Multiple Distributions")
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend()
plt.show()
```

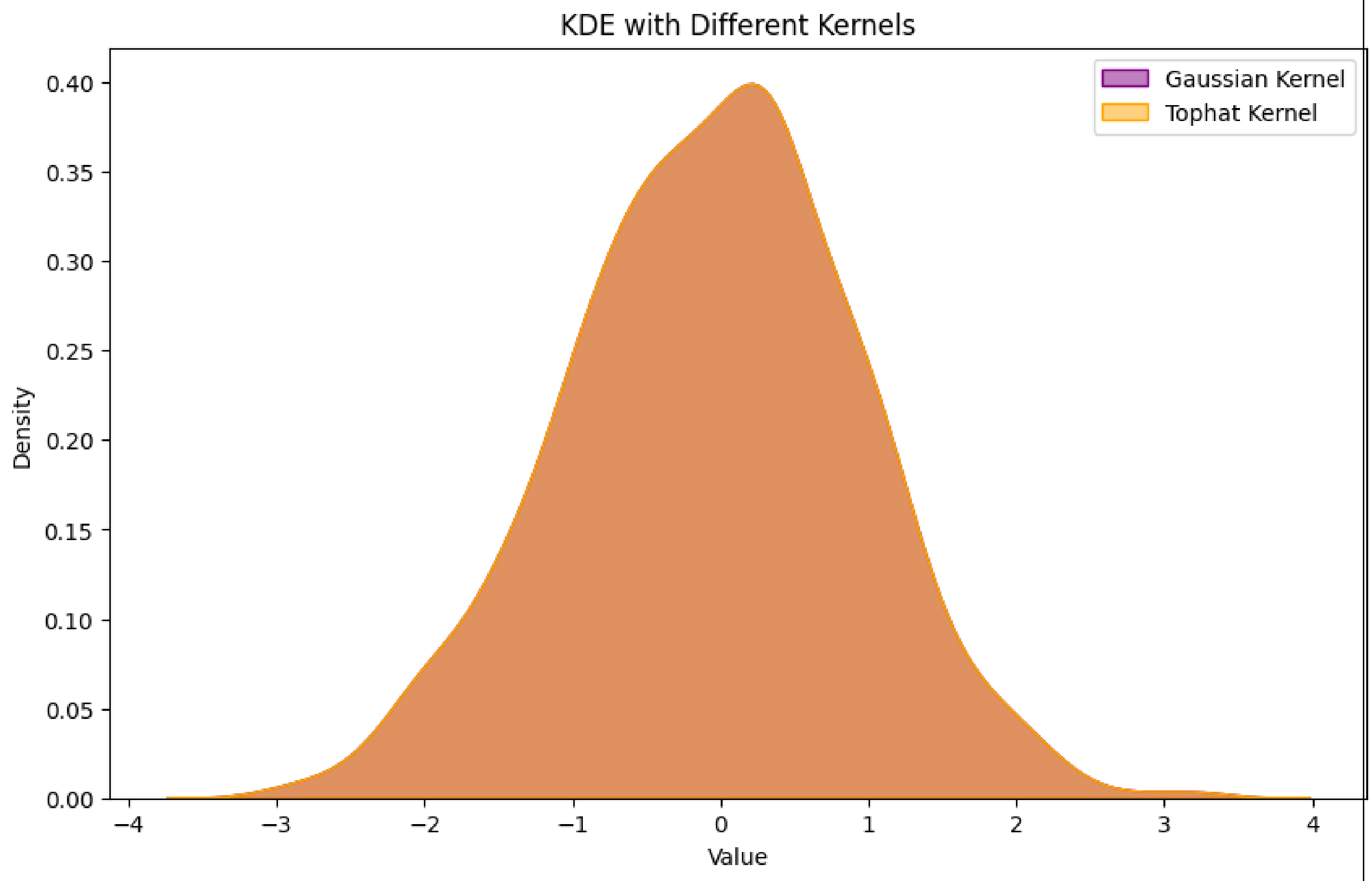
Output:

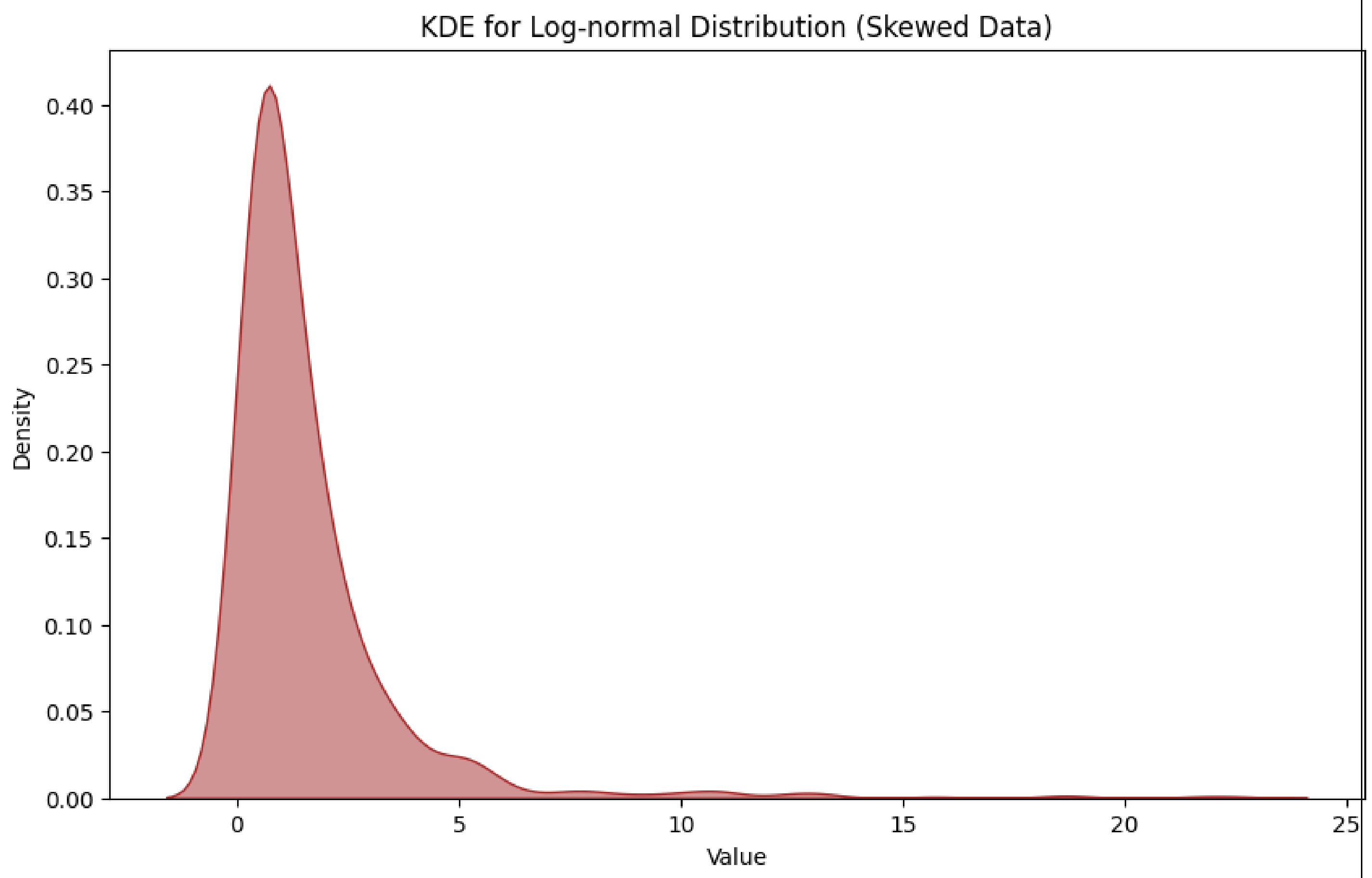


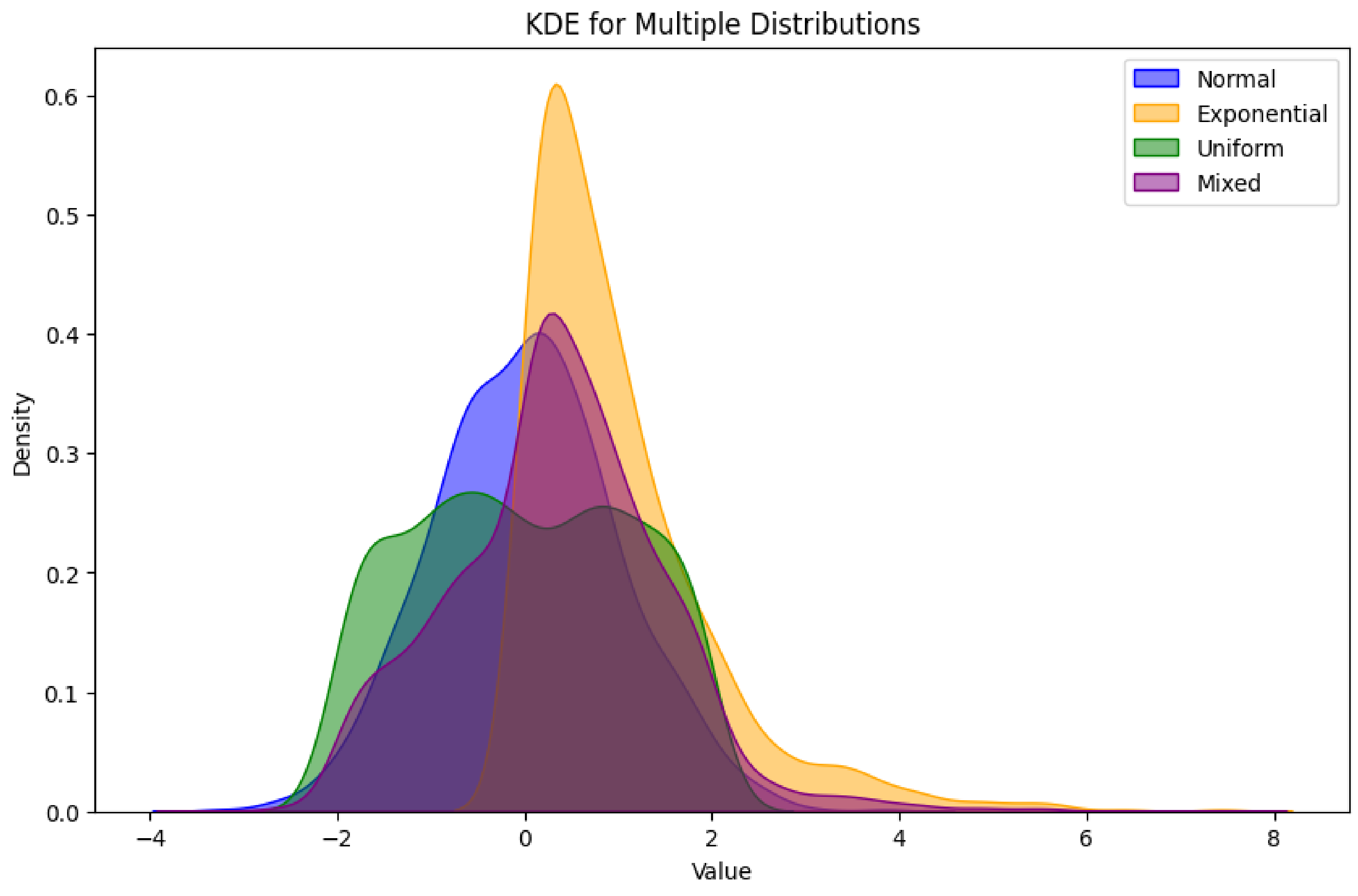

KDE for Exponential Distribution
Density
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
Value


KDE for Uniform Distribution
Density
0.25
0.20
0.15
0.10
0.05
0.00
−3
−2
−1
0
1
2
3
Value


KDE with Different Bandwidth Adjustments
bw_adjust=0.1
bw_adjust=1
bw_adjust=3
Density
0.0
0.1
0.2
0.3
0.4
0.5
−4
−2
0
2
4
6
Value


KDE with Different Kernels
Gaussian Kernel
Tophat Kernel
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
−4
−3
−2
−1
0
1
2
3
4
Value


KDE for Log-normal Distribution (Skewed Data)
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
Value


KDE for Multiple Distributions
Normal
Exponential
Uniform
Mixed
Density
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4
6
8
Value

3. Question 2: Bivariate Distribution

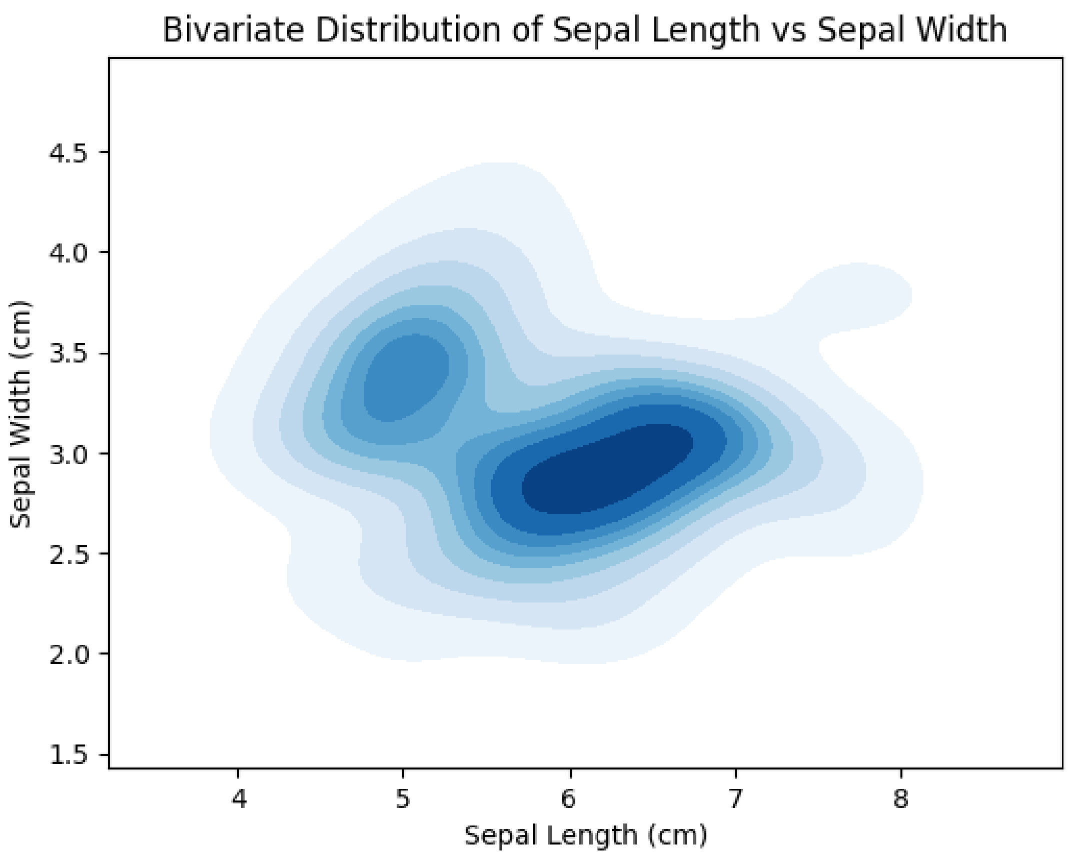
Objective

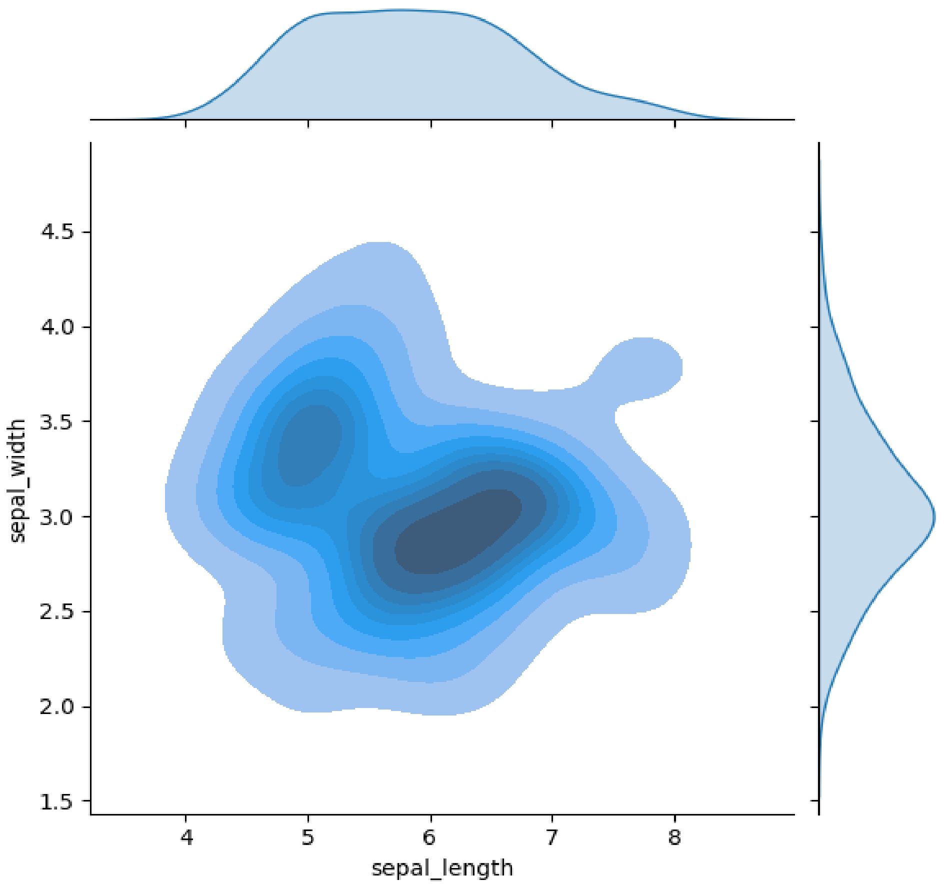
Develop a code to plot bivariate distribution considering a suitable data set available on the open source

Code Snippet:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load the Iris dataset from seaborn
iris = sns.load_dataset('iris')

# Select the two variables: Sepal Length and Sepal Width
x = iris['sepal_length']
y = iris['sepal_width']

# Create a bivariate distribution plot using KDE (Kernel Density Estimate)
sns.kdeplot(x=x, y=y, cmap="Blues", fill=True)

# Set the plot labels and title
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Bivariate Distribution of Sepal Length vs Sepal Width')

# Display the plot
plt.show()
# Alternatively, use sns.jointplot for a joint distribution plot
sns.jointplot(x='sepal_length', y='sepal_width', data=iris, kind='kde', fill=True)
plt.show()
```

Output:


Bivariate Distribution of Sepal Length vs Sepal Width
Sepal Width (cm)
Sepal Length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8

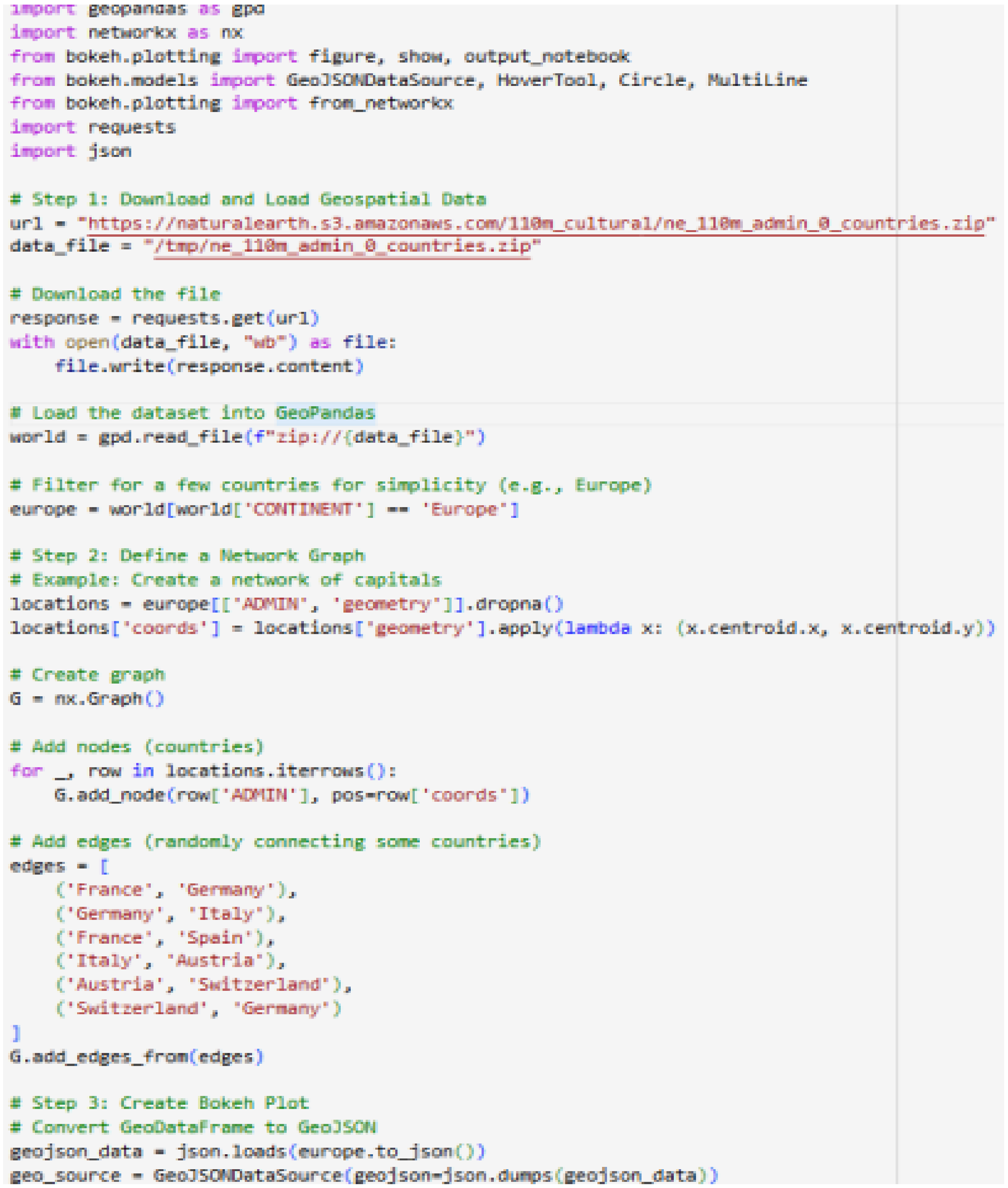

sepal_width
sepal_length
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8

4. Question 3: Geospatial data

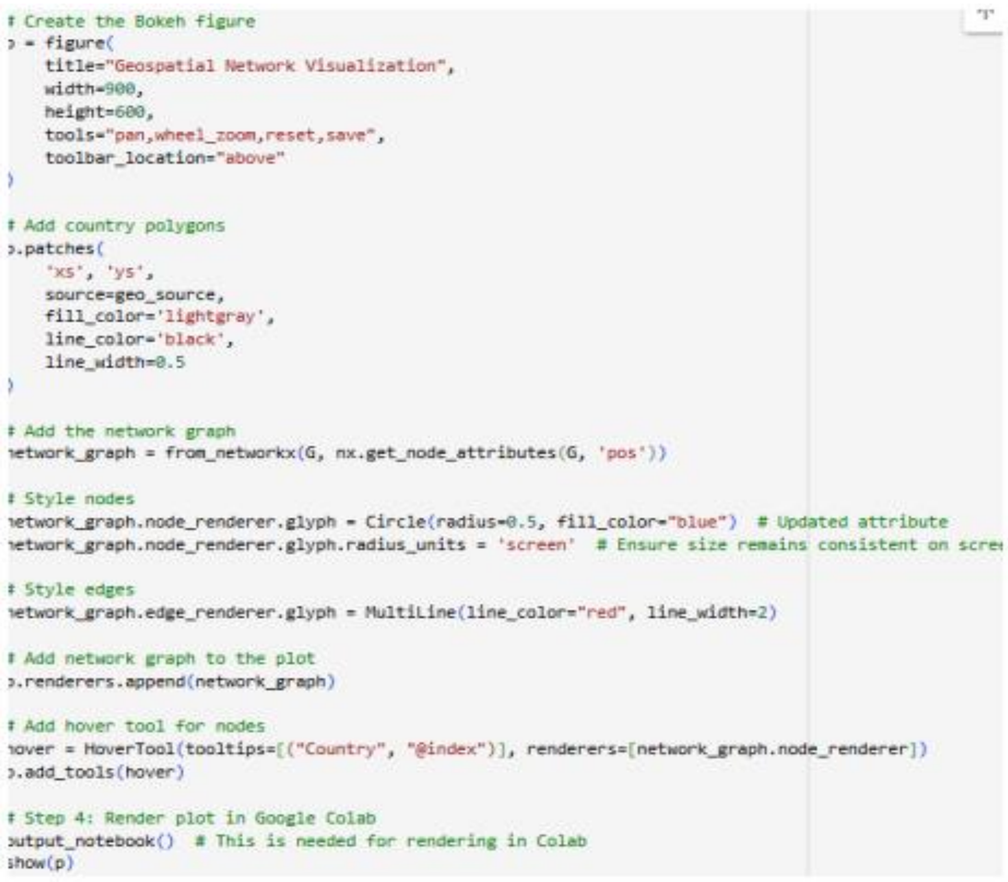
Objective:

Develop a code to showcase various Geospatial data also make use of Bokeh to make it more interactive.

Code Snippet:

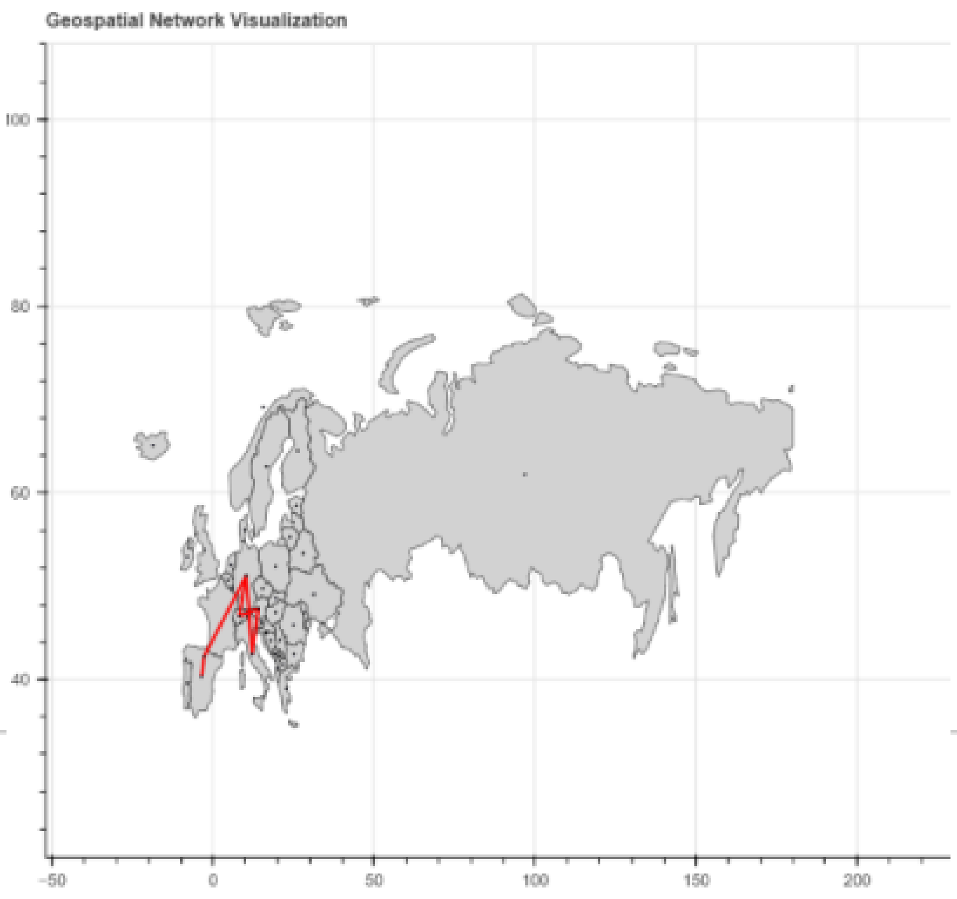


```python
import geopandas as gpd
import networkx as nx
from bokeh.plotting import figure, show, output_notebook
from bokeh.models import GeoJSONDataSource, HoverTool, Circle, MultiLine
from bokeh.plotting import from_networkx
import requests
import json

# Step 1: Download and Load Geospatial Data
url = "https://naturalearth.s3.amazonaws.com/110m_cultural/ne_110m_admin_0_countries.zip"
data_file = "/tmp/ne_110m_admin_0_countries.zip"

# Download the file
response = requests.get(url)
with open(data_file, "wb") as file:
    file.write(response.content)

# Load the dataset into GeoPandas
world = gpd.read_file(f"zip://{data_file}")

# Filter for a few countries for simplicity (e.g., Europe)
europe = world[world['CONTINENT'] == 'Europe']

# Step 2: Define a Network Graph
# Example: Create a network of capitals
locations = europe[['ADMIN', 'geometry']].dropna()
locations['coords'] = locations['geometry'].apply(lambda x: (x.centroid.x, x.centroid.y))

# Create graph
G = nx.Graph()

# Add nodes (countries)
for _, row in locations.iterrows():
    G.add_node(row['ADMIN'], pos=row['coords'])

# Add edges (randomly connecting some countries)
edges = [
    ('France', 'Germany'),
    ('Germany', 'Italy'),
    ('France', 'Spain'),
    ('Italy', 'Austria'),
    ('Austria', 'Switzerland'),
    ('Switzerland', 'Germany')
]
G.add_edges_from(edges)

# Step 3: Create Bokeh Plot
# Convert GeoDataFrame to GeoJSON
geojson_data = json.loads(europe.to_json())
geo_source = GeoJSONDataSource(geojson=json.dumps(geojson_data))
```



```
# Create the Bokeh figure
p = figure(
    title="Geospatial Network Visualization",
    width=900,
    height=600,
    tools="pan,wheel_zoom,reset,save",
    toolbar_location="above"
)

# Add country polygons
p.patches(
    'xs', 'ys',
    source=geo_source,
    fill_color='lightgray',
    line_color='black',
    line_width=0.5
)

# Add the network graph
network_graph = from_networkx(G, nx.get_node_attributes(G, 'pos'))

# Style nodes
network_graph.node_renderer.glyph = Circle(radius=0.5, fill_color="blue")  # Updated attribute
network_graph.node_renderer.glyph.radius_units = 'screen'  # Ensure size remains consistent on scre

# Style edges
network_graph.edge_renderer.glyph = MultiLine(line_color="red", line_width=2)

# Add network graph to the plot
p.renderers.append(network_graph)

# Add hover tool for nodes
hover = HoverTool(tooltips=[("Country", "@index")], renderers=[network_graph.node_renderer])
p.add_tools(hover)

# Step 4: Render plot in Google Colab
output_notebook()  # This is needed for rendering in Colab
show(p)
```

Output:



5. Question 4 Interconnection Using Geospatial Data

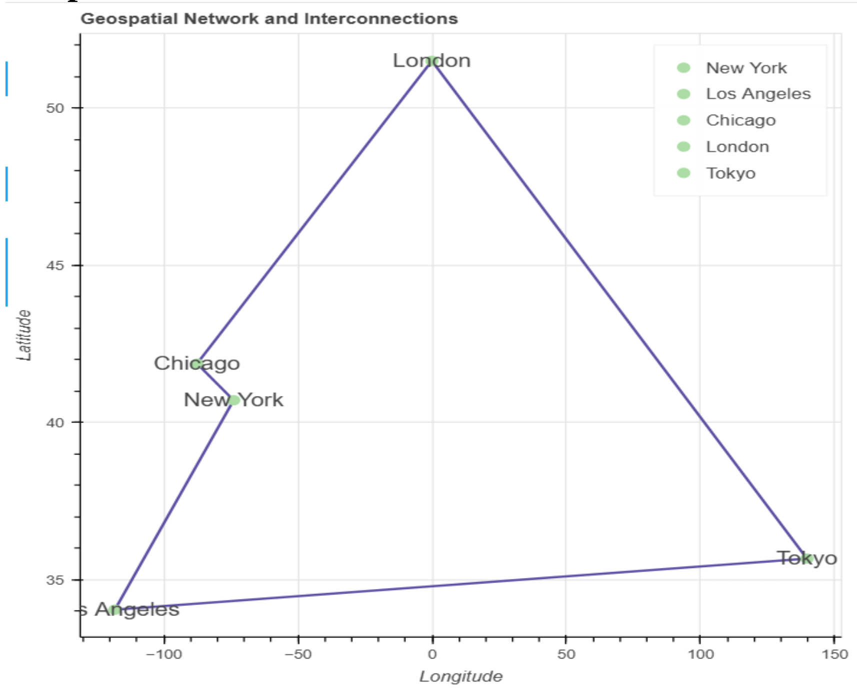
Objective:

Develop a code to plot network and interconnection using geospatial data

Code Snippet:

```
import pandas as pd
from bokeh.plotting import figure, show, output_file
from bokeh.models import HoverTool, ColumnDataSource, LabelSet
from bokeh.palettes import Spectral11

# Real city data: Names, latitude, and longitude
city_data = {
    'City': ['New York', 'Los Angeles', 'Chicago', 'London', 'Tokyo'],
    'Latitude': [40.7128, 34.0522, 41.8781, 51.5074, 35.6762],  # Latitudes
    'Longitude': [-74.0060, -118.2437, -87.6298, -0.1278, 139.6503]  # Longitudes
}

df = pd.DataFrame(city_data)

# Define connections (edges) between cities
edges = [
    ('New York', 'Los Angeles'),
    ('New York', 'Chicago'),
    ('Chicago', 'London'),
    ('London', 'Tokyo'),
    ('Tokyo', 'Los Angeles'),
]

# Step 1: Create Graph using NetworkX
import networkx as nx

G = nx.Graph()

# Add nodes (cities)
for idx, row in df.iterrows():
    G.add_node(row['City'], pos=(row['Longitude'], row['Latitude']))

# Add edges (connections)
G.add_edges_from(edges)

# Step 2: Prepare Bokeh Data Sources
# Extract node positions (longitude, latitude)
node_positions = {node: data['pos'] for node, data in G.nodes(data=True)}
node_df = pd.DataFrame.from_dict(node_positions, orient='index', columns=['Longitude', 'Latitude'])
node_df['City'] = node_df.index

# Reorder columns (important for Bokeh)
node_df = node_df[['City', 'Longitude', 'Latitude']]

# Extract edge coordinates (from and to cities)
edge_x = []
edge_y = []
for edge in G.edges():
    x0, y0 = node_positions[edge[0]]
    x1, y1 = node_positions[edge[1]]
    edge_x.append([x0, x1])
    edge_y.append([y0, y1])

# Create Bokeh ColumnDataSource for edges and nodes
edge_source = ColumnDataSource(data={'x': edge_x, 'y': edge_y})
node_source = ColumnDataSource(node_df)

# Step 3: Set up Bokeh Plot
output_file("geospatial_network.html")  # Save the plot to an HTML file
p = figure(title="Geospatial Network and Interconnections", toolbar_location="left",
           tools="pan, wheel_zoom, box_zoom, reset, hover, tap",
           x_axis_label='Longitude', y_axis_label='Latitude')

# Add edges (connections between nodes)
for x, y in zip(edge_x, edge_y):
    p.line(x, y, line_width=2, color=Spectral11[0])

# Add nodes (locations)
p.scatter(x='Longitude', y='Latitude', source=node_source, size=8, color=Spectral11[3], legend_field='City')

# Add hover tool to show city name
hover = HoverTool()
hover.tooltips = [("City", "@City")]
p.add_tools(hover)

# Add labels for each city
labels = LabelSet(x='Longitude', y='Latitude', text='City', source=node_source,
                  text_align='center', text_baseline='middle')

# Add the labels as renderers in the plot
p.add_layout(labels)

# Show the plot
show(p)
```

Output:



6. **Question 5 Networked Program**

Objective

Develop a code showcase networked program including retrieving image over HTTP, parsing HTML and scraping the web

Code Snippet:

```
import requests
from bs4 import BeautifulSoup
from PIL import Image
from io import BytesIO
import os

# Function to download an image from a URL and save it locally
def download_image(image_url, save_path):
    try:
        # Send HTTP GET request to retrieve the image
        response = requests.get(image_url)
        response.raise_for_status()  # Check for any errors (e.g., 404)

        # Open the image from the response content
        img = Image.open(BytesIO(response.content))
        img.save(save_path)  # Save the image locally
        print(f”Image saved as {save_path}”)
    except Exception as e:
        print(f”Error downloading image: {e}”)

# Function to scrape an HTML page and retrieve image URLs
def scrape_and_download_images(webpage_url, download_folder=’downloaded_images’):
    try:
        # Send HTTP GET request to the webpage
        response = requests.get(webpage_url)
        response.raise_for_status()

        # Parse the HTML content of the page
        soup = BeautifulSoup(response.text, ‘html.parser’)

        # Create folder to save images if it doesn’t exist
        if not os.path.exists(download_folder):
            os.makedirs(download_folder)

        # Find all image tags (<img>) in the HTML content
        images = soup.find_all(‘img’)

        # Download each image
        for index, img_tag in enumerate(images):
            img_url = img_tag.get(‘src’)
            if img_url:
                # Handle relative URLs (convert to absolute URL if necessary)
                if not img_url.startswith(‘http’):
                    img_url = requests.compat.urljoin(webpage_url, img_url)

                # Generate a save path for the image
                image_name = f”image_{index + 1}.jpg”
                save_path = os.path.join(download_folder, image_name)
            # Download and save the image
            download_image(img_url, save_path)

    except Exception as e:
        print(f"Error scraping webpage: {e}")

# Example usage:
if __name__ == "__main__":
    webpage_url = 'https://www.amazon.in/'  #
    scrape_and_download_images(webpage_url)
```

Output:

```
Image saved as downloaded_images/image_3.jpg
Image saved as downloaded_images/image_4.jpg
Image saved as downloaded_images/image_5.jpg
Image saved as downloaded_images/image_6.jpg
Image saved as downloaded_images/image_7.jpg
Image saved as downloaded_images/image_8.jpg
Image saved as downloaded_images/image_9.jpg
Image saved as downloaded_images/image_10.jpg
Image saved as downloaded_images/image_11.jpg
Image saved as downloaded_images/image_12.jpg
Image saved as downloaded_images/image_13.jpg
Image saved as downloaded_images/image_14.jpg
Image saved as downloaded_images/image_15.jpg
Image saved as downloaded_images/image_16.jpg
Image saved as downloaded_images/image_17.jpg
```

7. Question 6: Web Services

Objective

Develop a code to showcase the web services including eXtensible Markup Language

Code Snippet:

```
import requests
import xml.etree.ElementTree as ET

# Define the URL of the web service (OpenWeatherMap API)
API_KEY = 'a5d34c2ec7fe589ef12b7b104c44a0fa'  # Replace with your OpenWeatherMap
API key
CITY_NAME = 'London'
URL =
f'http://api.openweathermap.org/data/2.5/weather?q={CITY_NAME}&mode=xml&appid={
API_KEY}'

# Function to fetch and parse XML data from the web service
def fetch_weather_data(url):
    try:
        # Send GET request to the API
        response = requests.get(url)
        response.raise_for_status()  # Check for errors

        # Parse the XML data from the response
        tree = ET.ElementTree(ET.fromstring(response.content))
        root = tree.getroot()

        # Extract relevant data from the XML
        city = root.find('city').get('name')
        temperature = root.find('temperature').get('value')
        weather = root.find('weather').get('value')
        humidity = root.find('humidity').get('value')
        pressure = root.find('pressure').get('value')

        # Print the extracted data
        print(f"Weather data for {city}:")
        print(f"Temperature: {temperature}°C")
        print(f"Weather: {weather}")
        print(f"Humidity: {humidity}%")
        print(f"Pressure: {pressure} hPa")

    except requests.exceptions.RequestException as e:
        print(f"Error fetching data: {e}")
    except ET.ParseError as e:
        print(f"Error parsing XML: {e}")

# Call the function to fetch and display weather data
fetch_weather_data(URL)
```

Output:

```
Weather data for London:
Temperature: 279.42°C
Weather: overcast clouds
Humidity: 87%
Pressure: 1032 hPa
```

10. Conclusion

In conclusion, this report has successfully demonstrated the application of various data visualization techniques using Python. Through the development of code for Kernel Density Estimation, bivariate distributions, and geospatial data visualization, we have illustrated the power of visual tools in interpreting complex data sets. The integration of real-world map backgrounds in our geospatial visualizations further enhances the interactivity and contextual understanding of the data. These techniques not only facilitate better data analysis but also improve communication of insights to stakeholders. As data continues to grow in complexity, the importance of effective visualization will only increase, making these skills essential for data scientists and analysts.

11. References

- Bokeh Documentation
- Pandas Documentation
- Seaborn: statistical data visualization. Journal of Open Source Software
- Data Analysis in Python with Pandas
- CartoDB

GitHub Repo Link: https://github.com/Yashas711/DV-ASSIGNMENT-2