Experiment No. 7

Title: Visualizing Hierarchical Data Structures using Dendrograms, Treemaps, and Sunburst Charts

Aim: To visualize hierarchical data structures using various techniques such as dendrograms, treemaps, and sunburst charts to understand the relationships and hierarchical organization within the data.

Theory:
Introduction:
Hierarchical data structures represent data organized in a tree-like structure, where each node is a subset of its parent node. Visualizing such structures helps in understanding the relationships and organization within the data. Common techniques for visualizing hierarchical data include dendrograms, treemaps, and sunburst charts.

Techniques

- **Dendrogram:** A tree diagram used to illustrate the arrangement of clusters produced by hierarchical clustering.
- **Treemap:** A method for displaying hierarchical data using nested rectangles, where each rectangle represents a category, and its size is proportional to a specific measure.
- **Sunburst Chart:** A radial chart that represents hierarchical data using concentric circles, where each level of the hierarchy is represented by a ring.

Steps: -
Step 1: Import Necessary Libraries
First, import the libraries required for data manipulation and visualization.

Code: -

```
# Install the squarify module
!pip install squarify

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import dendrogram, linkage
import plotly.express as px
import squarify
```

Step 2: Create Sample Hierarchical Data
Generate sample hierarchical data points for visualization.

Code: -

```
# Sample hierarchical data
data = {
    'category': ['A', 'B', 'C', 'D', 'E', 'F'],
```

```
    'value': [10, 20, 30, 40, 50, 60],
    'subcategory': ['A1', 'B1', 'C1', 'D1', 'E1', 'F1']
}

df = pd.DataFrame(data)
```

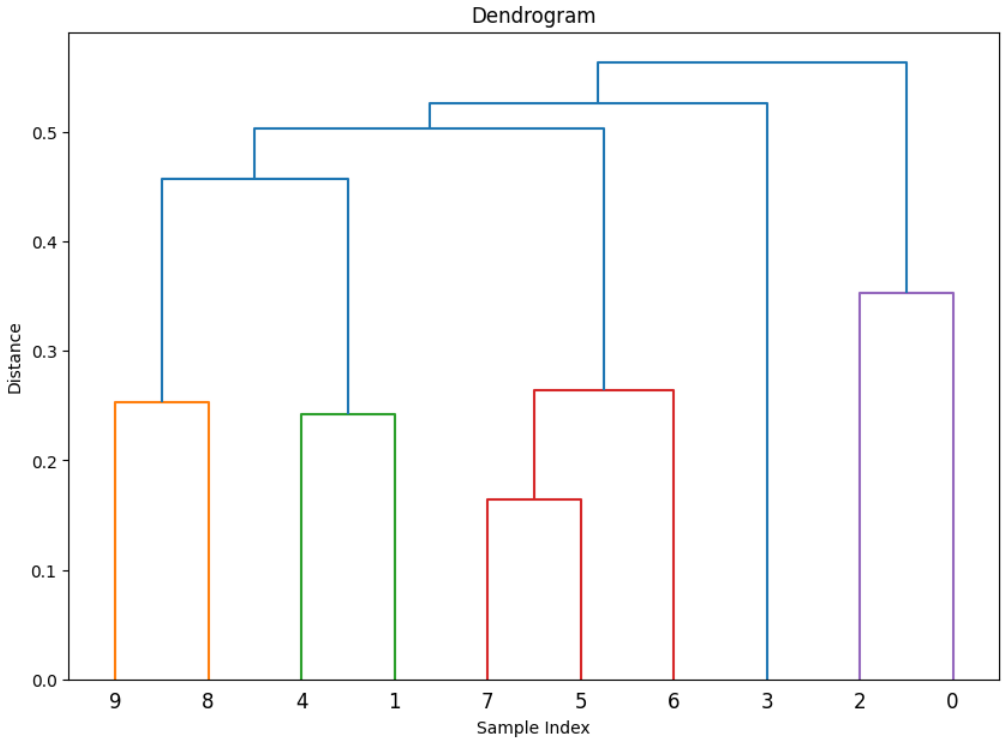
Step 3: Dendrogram
Create a dendrogram to visualize hierarchical clustering.

Code: -

```
# Generate sample data for clustering
np.random.seed(42)
sample_data = np.random.rand(10, 3)

# Perform hierarchical clustering
linked = linkage(sample_data, 'single')

# Create dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

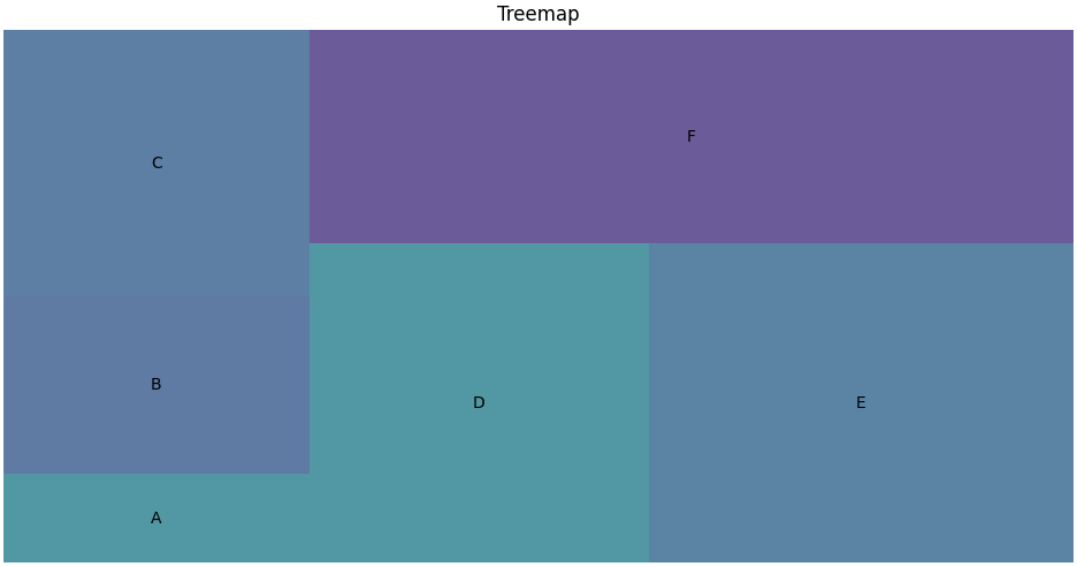


Hierarchical clustering is performed on sample data, and a dendrogram is created to visualize the resulting cluster arrangement.

Step 4: Treemap
Create a treemap to visualize hierarchical data.

Code: -

```
# Create a treemap
plt.figure(figsize=(12, 6))
squarify.plot(sizes=df['value'], label=df['category'], alpha=0.8)
plt.axis('off')
plt.title('Treemap')
plt.show()
```



A treemap is created using the squarify library, displaying hierarchical data with rectangles proportional to the values.

Step 5: Sunburst Chart
Create a sunburst chart to visualize hierarchical data.

Code: -

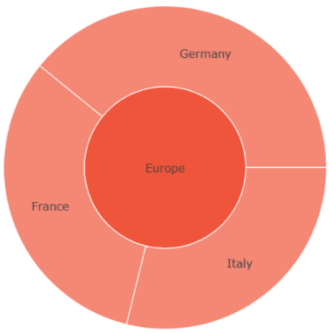
```
# Create a sample hierarchical dataset for sunburst chart
sunburst_data = pd.DataFrame({
    'region': ["Asia", "Asia", "Asia", "Europe", "Europe", "Europe"],
    'country': ["China", "Japan", "India", "Germany", "France", "Italy"],
    'population': [1409517397, 127484450, 126576461, 82114224, 67106161, 60483973]
})

# Create sunburst chart
fig = px.sunburst(sunburst_data, path=['region', 'country'], values='population', title='Sunburst
Chart')
fig.show()
```

Sunburst Chart

Sunburst Chart

Sunburst Chart



Conclusion:

In this experiment, we visualized hierarchical data structures using dendrograms, treemaps, and sunburst charts. These visualizations help in understanding the hierarchical organization and relationships within the data. Dendrograms illustrate clustering arrangements, treemaps use nested rectangles to show proportions, and sunburst charts display hierarchical data with concentric circles.

Experiment No. 8

Title: Visualizing Textual Data to Reveal Patterns, Trends, and Insights

Aim: To analyze and visualize textual data using techniques such as word clouds, bar charts, and frequency distributions to uncover patterns, trends, and insights.

Theory:

Introduction:

Textual data analysis involves extracting meaningful information from text data. This can include identifying the most frequent words, understanding the distribution of words, and visualizing these patterns. Common techniques for visualizing textual data include word clouds, bar charts for word frequencies, and frequency distributions.

Techniques

- Word Cloud: A visual representation of text data where the size of each word indicates its frequency or importance.
- Bar Chart: A chart that presents categorical data with rectangular bars, where the length of each bar is proportional to the value it represents.
- Frequency Distribution: A representation of the number of occurrences of each word in the text.

Steps: -

Step 1: Import Necessary Libraries

First, import the libraries required for text manipulation and visualization.

Code: -

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from collections import Counter
import seaborn as sns
```

Step 2: Load and Preprocess Textual Data

Load and preprocess the textual data for analysis.

Code: -

```
# Sample textual data
text = """
```

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems
to extract knowledge and insights from structured and unstructured data. Data science is related to data mining,
machine learning and big data. Data science is a "concept to unify statistics, data analysis and their related methods"
in order to "understand and analyze actual phenomena" with data. It uses techniques and theories drawn from many fields
within the context of mathematics, statistics, computer science, domain knowledge and information science.
Turing Award winner Jim Gray imagined data science as a "fourth paradigm" of science (empirical, theoretical,
computational and now data-driven) and asserted that "everything about science is changing because of the impact
of information technology" and the data deluge.
"""

```
# Convert the text to lowercase
text = text.lower()

# Tokenize the text
words = text.split()
```

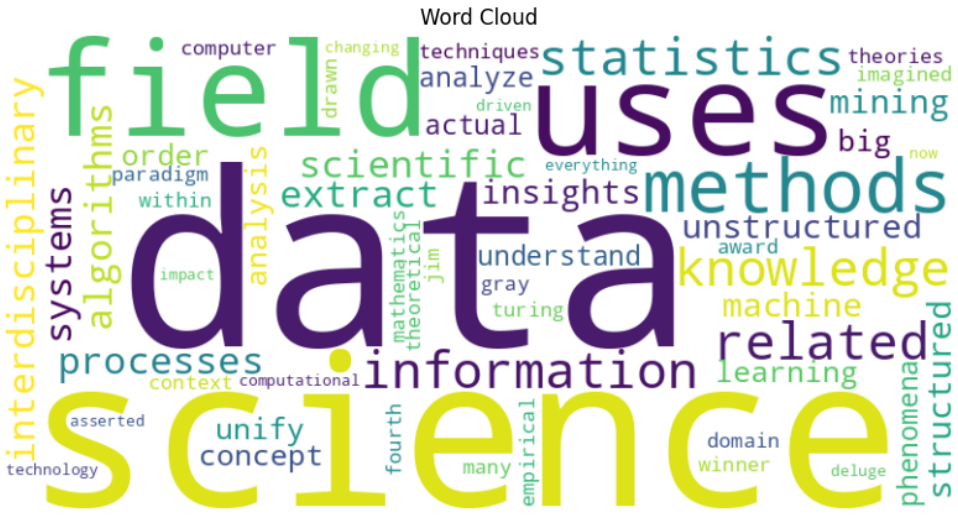
Step 3: Create a Word Cloud

Generate a word cloud to visualize the most frequent words in the textual data.

Code: -

```
# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(words))

# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud')
plt.show()
```

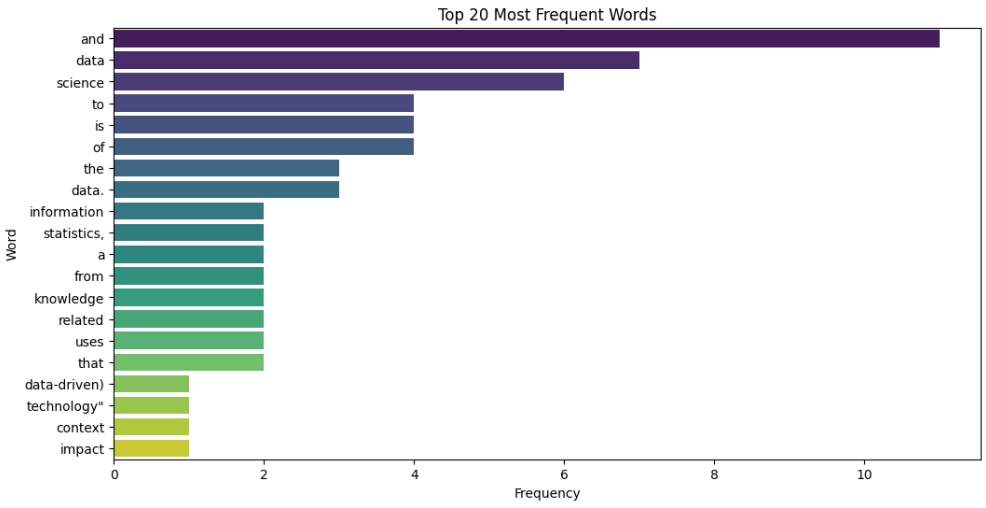


Step 4: Bar Chart of Word Frequencies

Create a bar chart to visualize the frequency of the most common words.

Code: -

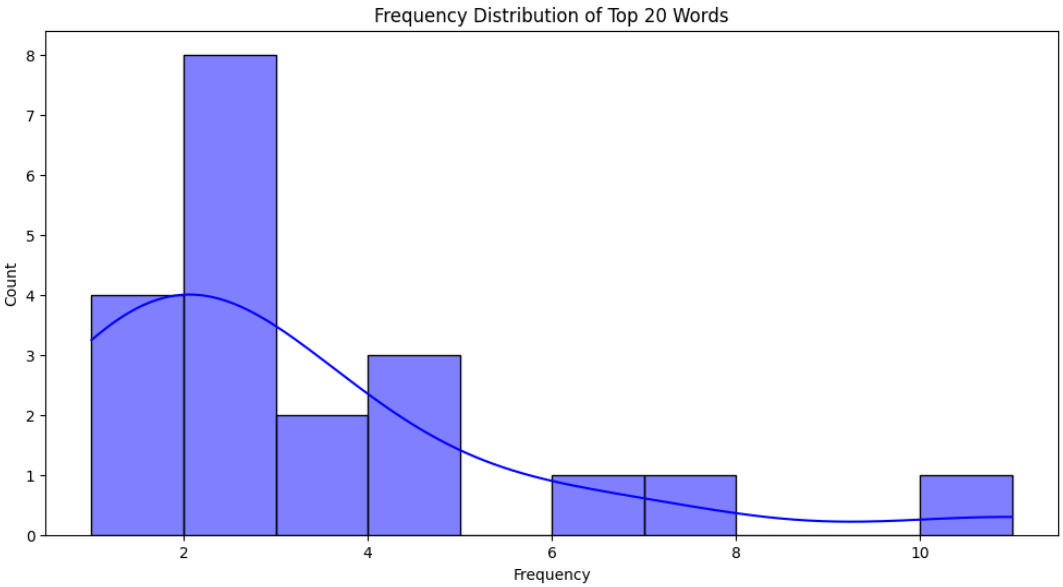
```
# Calculate word frequencies
word_freq = Counter(words)
# Create a DataFrame for the word frequencies
word_freq_df = pd.DataFrame(word_freq.items(), columns=['word', 'frequency']).sort_values(by='frequency', ascending=False).head(20)
# Create a bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x='frequency', y='word', data=word_freq_df, palette='viridis')
plt.title('Top 20 Most Frequent Words')
plt.xlabel('Frequency')
plt.ylabel('Word')
plt.show()
```



Step 5: Frequency Distribution
Create a frequency distribution plot to visualize the distribution of word frequencies.

Code: -

```
# Create a frequency distribution plot
plt.figure(figsize=(12, 6))
sns.histplot(word_freq_df['frequency'], bins=10, kde=True, color='blue')
plt.title('Frequency Distribution of Top 20 Words')
plt.xlabel('Frequency')
plt.ylabel('Count')
plt.show()
```



Conclusion:
In this experiment, we analyzed and visualized textual data to reveal patterns, trends, and insights. We used a word cloud to highlight the most frequent words, a bar chart to visualize the frequency of the top 20 words, and a frequency distribution plot to understand the distribution of word frequencies. These visualizations are essential for gaining insights from textual data and identifying key terms and patterns.

Experiment No. 9

Title: Visualizing Geographic Data to Understand Spatial Relationships and Patterns

Aim: To understand and visualize geographic data to reveal spatial relationships and patterns using geospatial analysis techniques.

Theory:

Introduction:

Geospatial data analysis involves the collection, display, and manipulation of imagery, GPS, satellite photography, and historical data, described explicitly in terms of geographic coordinates. This experiment will use geographic data to understand spatial relationships and patterns.

Techniques

- GeoPandas: A Python library used for working with geospatial data.
- Matplotlib and Seaborn: Libraries used for data visualization.

Steps: -

Step 1: Import Necessary Libraries

First, import the libraries required for geospatial data manipulation and visualization.

Code: -

```
import geopandas as gpd
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 2: Download and Unzip the Geographic Dataset

Download the naturalearth_lowres dataset manually from the Natural Earth website and unzip it.

- Download the dataset from Natural Earth (naturalearthdata.com).
- Extract the contents of the zip file to a known location on your local machine.

Step 3: Load the Dataset

Load the dataset using GeoPandas.

Code: -

```
# Load the dataset from the extracted files
world = gpd.read_file(r'path/to/natural_earth_vector.shp')
```

```
# Display the first few rows of the dataset
print(world.head())
```

Step 4: Plot the World Map
Use GeoPandas and Matplotlib to plot the world map.

Code: -

```
# Plot the world map
world.plot(figsize=(15, 10))
plt.title('World Map')
plt.show()
```

Step 5: Analyze Population Distribution
Visualize the population distribution by country.

Code: -

```
# Plot the population distribution
world.plot(column='pop_est', cmap='OrRd', legend=True, figsize=(15, 10))
plt.title('Population Distribution by Country')
plt.show()
```

Step 6: Analyze GDP Distribution
Visualize the GDP distribution by country.

Code: -

```
# Plot the GDP distribution
world.plot(column='gdp_md_est', cmap='YlGnBu', legend=True, figsize=(15, 10))
plt.title('GDP Distribution by Country')
plt.show()
```

Step 7: Combine Population and GDP Analysis
Visualize the relationship between population and GDP.

Code: -

```
# Scatter plot for population vs. GDP
plt.figure(figsize=(15, 10))
```

```
sns.scatterplot(data=world, x='pop_est', y='gdp_md_est', hue='continent', size='pop_est', sizes=(10, 200))
plt.title('Population vs. GDP by Country')
plt.xlabel('Population Estimate')
plt.ylabel('GDP Estimate (in million USD)')
plt.xscale('log')
plt.yscale('log')
plt.show()
```

Conclusion:

In this experiment, we visualized geographic data to understand spatial relationships and patterns. We used GeoPandas to load and manipulate geographic data and Matplotlib and Seaborn for visualization. This analysis helps in understanding the distribution of population and GDP across different countries and their spatial relationships.

Experiment No. 10

Title: Visualizing Hierarchical Data Structures using Dendrograms, Treemaps, and Sunburst Charts

Aim: To visualize hierarchical data structures using various techniques such as dendrograms, treemaps, and sunburst charts to understand the relationships and hierarchical organization within the data.

Theory:
Introduction:
Hierarchical data structures represent data organized in a tree-like structure, where each node is a subset of its parent node. Visualizing such structures helps in understanding the relationships and organization within the data. Common techniques for visualizing hierarchical data include dendrograms, treemaps, and sunburst charts.

Techniques

- Dendrogram: A tree diagram used to illustrate the arrangement of clusters produced by hierarchical clustering.
- Treemap: A method for displaying hierarchical data using nested rectangles, where each rectangle represents a category, and its size is proportional to a specific measure.
- Sunburst Chart: A radial chart that represents hierarchical data using concentric circles, where each level of the hierarchy is represented by a ring.

Steps: -
Step 1: Import Necessary Libraries
First, import the libraries required for data manipulation and visualization.

Code: -

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import dendrogram, linkage
import plotly.express as px
import squarify
```

Step 2: Create Sample Hierarchical Data
Generate sample hierarchical data points for visualization.

Code: -

```
# Sample hierarchical data
data = {
    'category': ['A', 'B', 'C', 'D', 'E', 'F'],
    'value': [10, 20, 30, 40, 50, 60],
    'subcategory': ['A1', 'B1', 'C1', 'D1', 'E1', 'F1']
}
```

df = pd.DataFrame(data)

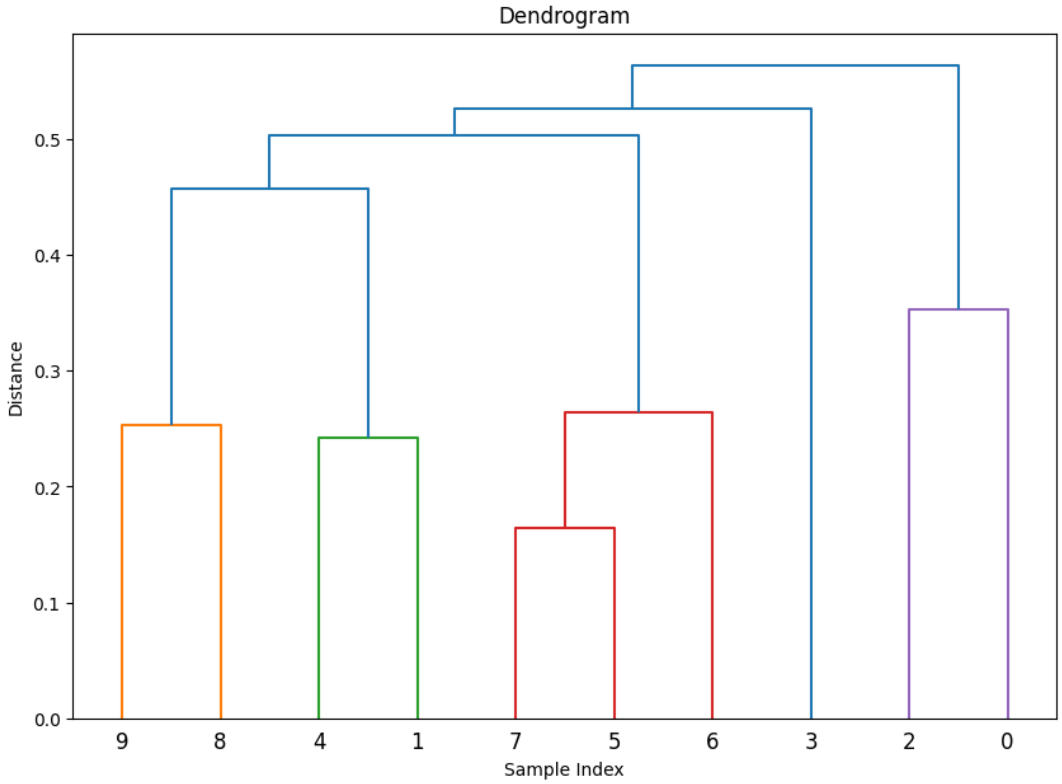
Step 3: Dendrogram
Create a dendrogram to visualize hierarchical clustering.

Code: -

```
# Generate sample data for clustering
np.random.seed(42)
sample_data = np.random.rand(10, 3)

# Perform hierarchical clustering
linked = linkage(sample_data, 'single')

# Create dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title('Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

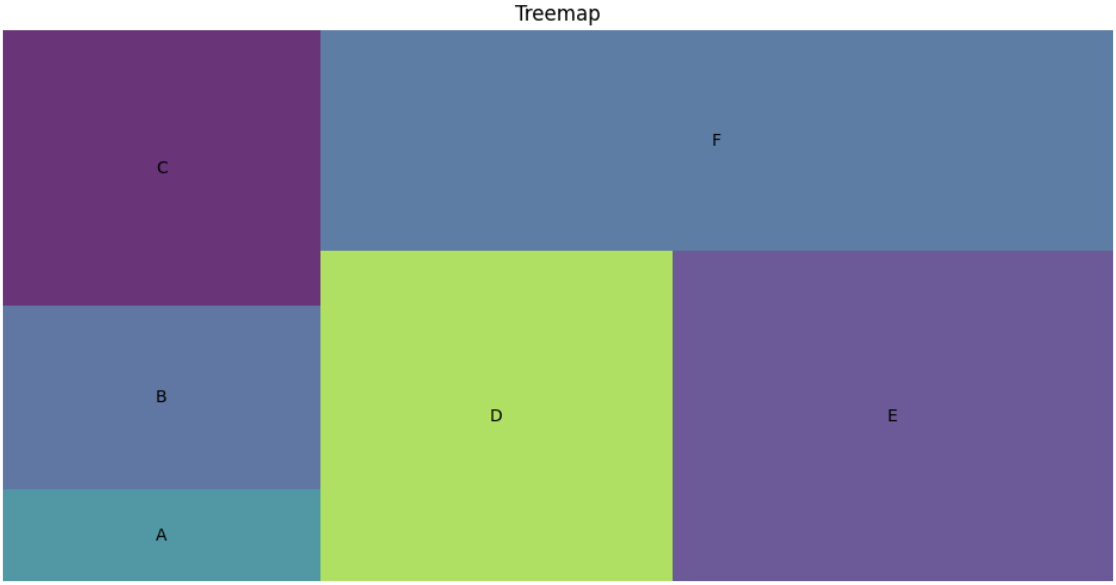


Hierarchical clustering is performed on sample data, and a dendrogram is created to visualize the resulting cluster arrangement.

Step 4: Treemap
Create a treemap to visualize hierarchical data.

Code: -

```
# Create a treemap
plt.figure(figsize=(12, 6))
squarify.plot(sizes=df['value'], label=df['category'], alpha=0.8)
plt.axis('off')
plt.title('Treemap')
plt.show()
```



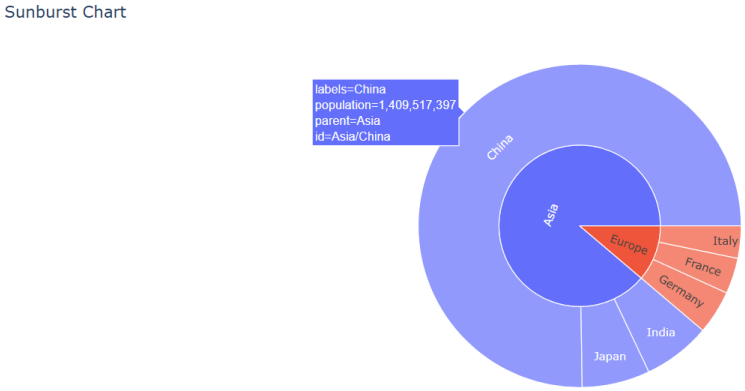
A treemap is created using the squarify library, displaying hierarchical data with rectangles proportional to the values.

Step 5: Sunburst Chart
Create a sunburst chart to visualize hierarchical data.

Code: -

```
# Create a sample hierarchical dataset for sunburst chart
sunburst_data = pd.DataFrame({
    'region': ["Asia", "Asia", "Asia", "Europe", "Europe", "Europe"],
    'country': ["China", "Japan", "India", "Germany", "France", "Italy"],
    'population': [1409517397, 127484450, 126576461, 82114224, 67106161, 60483973]
})

# Create sunburst chart
fig = px.sunburst(sunburst_data, path=['region', 'country'], values='population', title='Sunburst
Chart')
fig.show()
```



A sunburst chart is created using the plotly.express library, displaying hierarchical data with concentric circles.

Conclusion:

In this experiment, we visualized hierarchical data structures using dendrograms, treemaps, and sunburst charts. These visualizations help in understanding the hierarchical organization and relationships within the data. Dendrograms illustrate clustering arrangements, treemaps use nested rectangles to show proportions, and sunburst charts display hierarchical data with concentric circles.

Experiment No. 11

Title: Implementing Web Crawling to Automatically Navigate and Extract Information from Websites

Aim: To understand and implement web crawling techniques to automatically navigate and extract information from websites.

Theory:
Introduction:
Web crawling, also known as web scraping, is the process of automatically navigating web pages and extracting information from them. This process is useful for collecting data from websites for analysis. Web crawlers or spiders are automated scripts that browse the web systematically.

Techniques

- HTTP Requests: Using libraries like requests to send HTTP requests to web servers and receive responses.
- HTML Parsing: Using libraries like BeautifulSoup to parse HTML documents and extract data.
- Automation Tools: Using tools like Selenium to automate browser actions for dynamic content extraction.

Steps: -
Step 1: Import Necessary Libraries
First, import the libraries required for web crawling.

Code: -

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
```

Step 2: Define the Target Website and Send HTTP Request
Specify the URL of the website you want to scrape and send an HTTP request to the server.

Code: -

```
# Define the target URL
url = 'https://example.com'

# Send an HTTP request to the server
response = requests.get(url)
```

```
# Check the status code of the response
if response.status_code == 200:
    print("Successfully accessed the website")
else:
    print("Failed to access the website")
```

Step 3: Parse the HTML Content
Parse the HTML content of the web page using BeautifulSoup.

Code: -

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

# Print the title of the web page
print("Title of the page:", soup.title.string)
```

Step 4: Extract Information
Extract specific information from the web page. For example, extract all the headings.

Code: -

```
# Extract all headings (h1 tags)
headings = soup.find_all('h1')

# Print the extracted headings
for heading in headings:
    print(heading.text)
```

Step 5: Extract Data from Multiple Pages
To extract data from multiple pages, you can loop through the pages by modifying the URL.

Code: -

```
# Define a list to store the extracted data
data = []

# Loop through the first 5 pages
for page in range(1, 6):
    # Modify the URL to access the next page
    url = f'https://example.com/page/{page}'

    # Send an HTTP request to the server
    response = requests.get(url)

    # Parse the HTML content
```

```
    soup = BeautifulSoup(response.content, 'html.parser')

    # Extract specific data (e.g., paragraphs)
    paragraphs = soup.find_all('p')

    # Store the extracted data
    for paragraph in paragraphs:
        data.append(paragraph.text)

    # Wait for a few seconds before the next request
    time.sleep(2)

# Convert the data to a DataFrame
df = pd.DataFrame(data, columns=['Paragraphs'])
print(df)
```

Step 6: Use Selenium for Dynamic Content
For websites with dynamic content, use Selenium to automate browser actions.

Code: -

```
# Initialize the WebDriver
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

# Open the target website
driver.get('https://example.com')

# Wait for the page to load
time.sleep(5)

# Extract specific elements (e.g., buttons)
buttons = driver.find_elements(By.TAG_NAME, 'button')

# Click on the first button
if buttons:
    buttons[0].click()

# Wait for the new content to load
time.sleep(5)

# Extract updated content
new_content = driver.find_elements(By.TAG_NAME, 'p')

# Print the extracted content
for content in new_content:
    print(content.text)

# Close the browser
driver.quit()
```

Conclusion:
In this experiment, we implemented web crawling to automatically navigate and extract information from websites. We used the requests library to send HTTP requests, BeautifulSoup to parse HTML content, and Selenium to automate browser actions for dynamic content extraction. These techniques are essential for collecting data from the web for analysis and further processing.