MASTER OF COMPUTER APPLICATIONS

SEMESTER 1

DATA VISUALIZATION

Unit 10

Advanced Visual Techniques in Python

Table of Contents

1. INTRODUCTION

A Bubble Chart is a dynamic method for visualising three data dimensions within a two-dimensional space. By utilising circles, or "bubbles", plotted on a grid, this technique showcases the relationships among variables effectively. The x and y coordinates of each bubble represent two of the data dimensions, while the bubble's size illustrates a third, providing a comprehensive view of the data landscape. An example using Plotly Express is presented, illustrating the ease of creating stylish figures with this high-level interface.

1.1 Learning Objectives

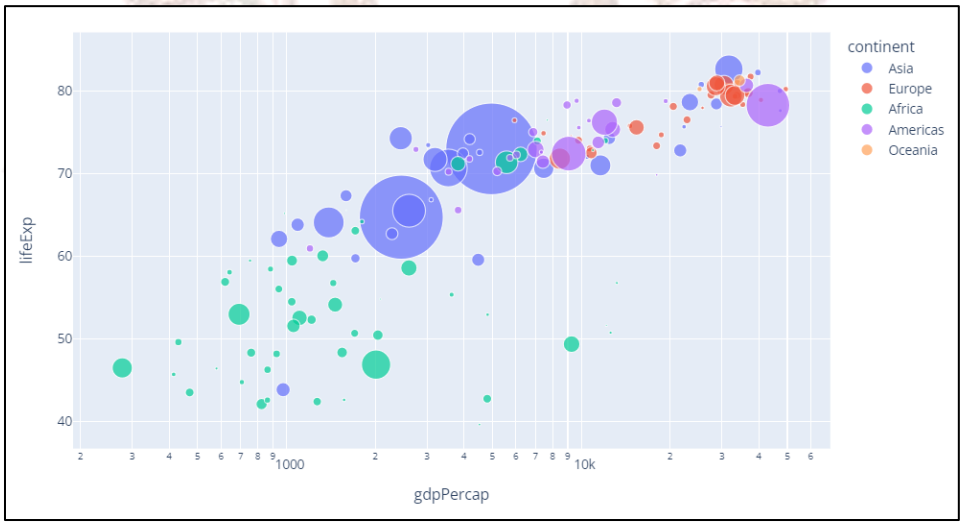
By the end of this chapter, you will be able to:

- *Recall and identify the fundamental principles and components of Bubble Charts, TreeMaps, and Interactive visuals.*
- *Demonstrate generating bubble charts, tree maps, and interactive visuals.*

2. BUBBLE CHART

A Bubble Chart is a data visualisation technique that allows you to display three data dimensions in a two-dimensional space. It is beneficial for illustrating relationships between variables and providing a visual representation of data points using circles (bubbles) on a grid. Each bubble is characterised by its x and y coordinates, which correspond to two data dimensions, and its size represents a third data dimension. We first show a bubble chart example using Plotly Express. Plotly Express is the easy-to-use, high-level interface to Plotly, which operates on various types of data and produces easy-to-style figures. The size of markers is set from the dataframe column given as the size parameter.

```
import plotly.express as px

df = px.data.gapminder()

fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp",

                 size="pop", color="continent",

                hover_name="country", log_x=True, size_max=60)

fig.show()
```



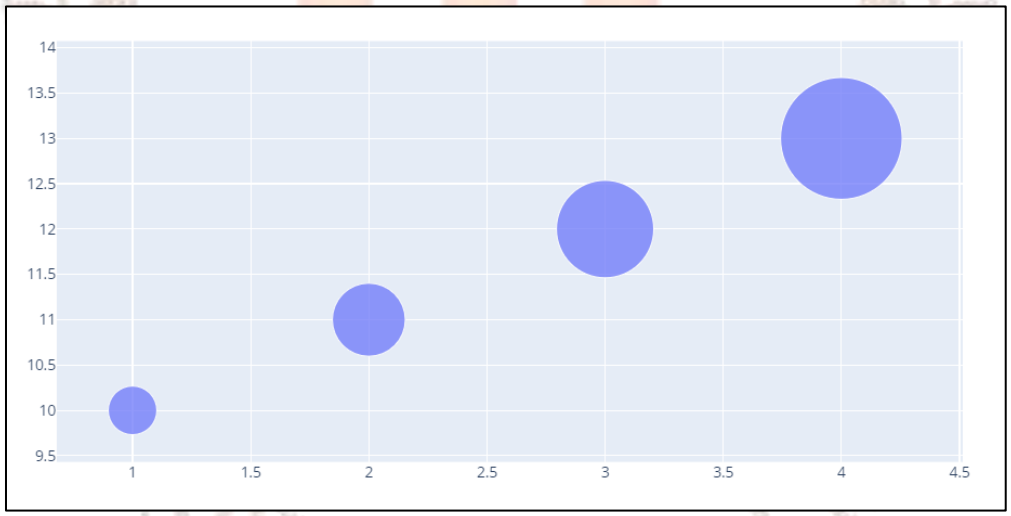
Bubble Chart with plotly.graph_objects

If Plotly Express does not provide a good starting point, it is also possible to use the more generic go. Scatter class from plotly.graph_objects, and define the size of markers to create a

bubble chart. All available options are described in the scatter section of the reference page: https://plotly.com/python/reference#scatter.

Simple Bubble Chart

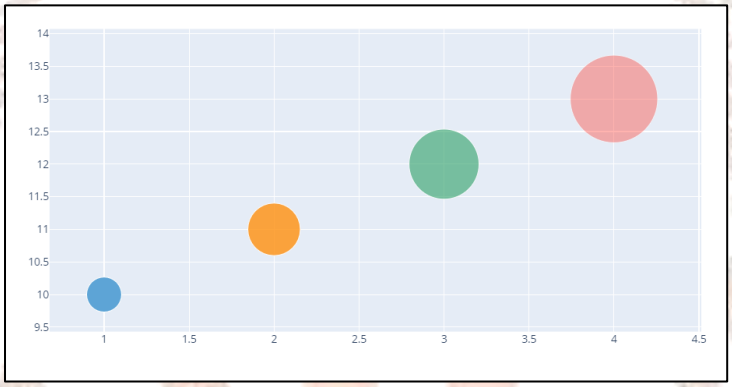
```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Scatter(
    x=[1, 2, 3, 4], y=[10, 11, 12, 13],
    mode='markers',
    marker_size=[40, 60, 80, 100])
])

fig.show()
```



Setting Marker Size and Color

```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Scatter(
    x=[1, 2, 3, 4], y=[10, 11, 12, 13],
    mode='markers',
```

```
    marker=dict(
        color=['rgb(93, 164, 214)', 'rgb(255, 144, 14)',
            'rgb(44, 160, 101)', 'rgb(255, 65, 54)'],
        opacity=[1, 0.8, 0.6, 0.4],
        size=[40, 60, 80, 100],    )])
fig.show()
```



Scaling the Size of Bubble Charts

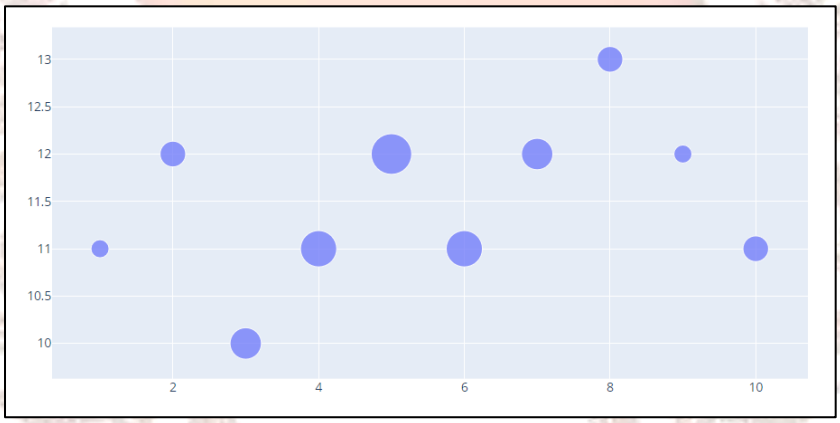
To scale the bubble size, use the attribute sizeref. We recommend using the following formula to calculate a sizeref value:

sizeref = 2. max(array of size values) / (desired maximum marker size 2)

Note that setting 'sizeref' to a value greater than 1, decreases the rendered marker sizes, while setting 'sizeref' to less than 1, increases the rendered marker sizes. Additionally, we

```
import plotly.graph_objects as go
size = [20, 40, 60, 80, 100, 80, 60, 40, 20, 40]
fig = go.Figure(data=[go.Scatter(
    x=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    y=[11, 12, 10, 11, 12, 11, 12, 13, 12, 11],
    mode='markers',
```

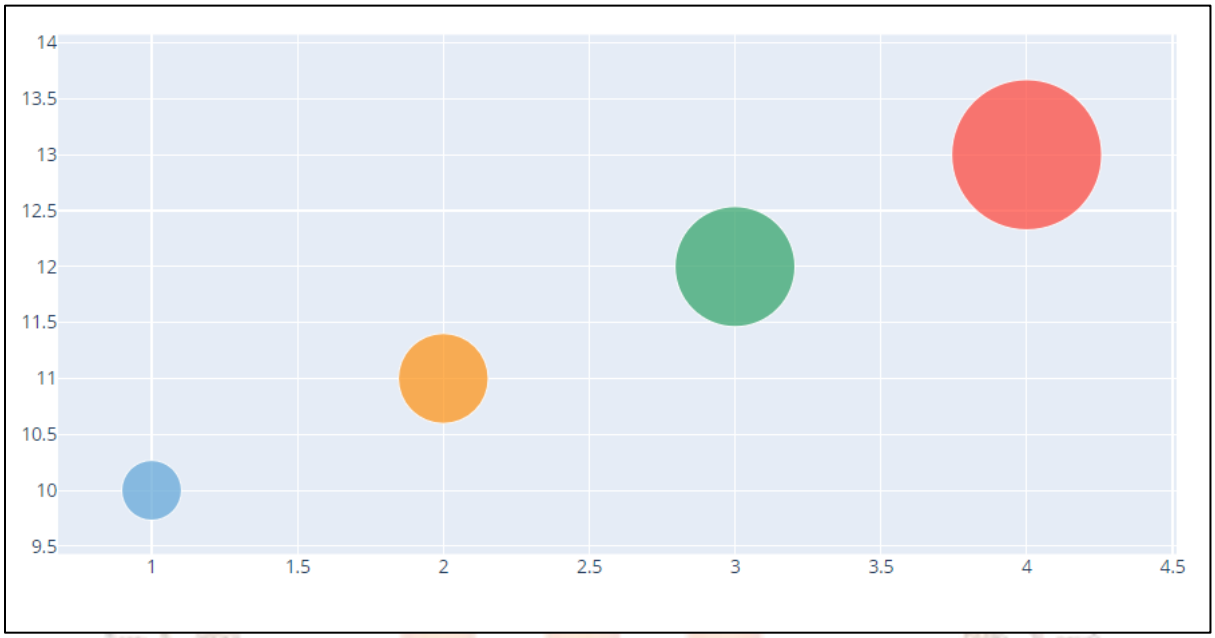
```
    marker=dict(
        size=size,
        sizemode='area',
        sizeref=max(size)/(40.2),
        sizemin=4
 )])]
fig.show()
```



Hover Text with Bubble Charts

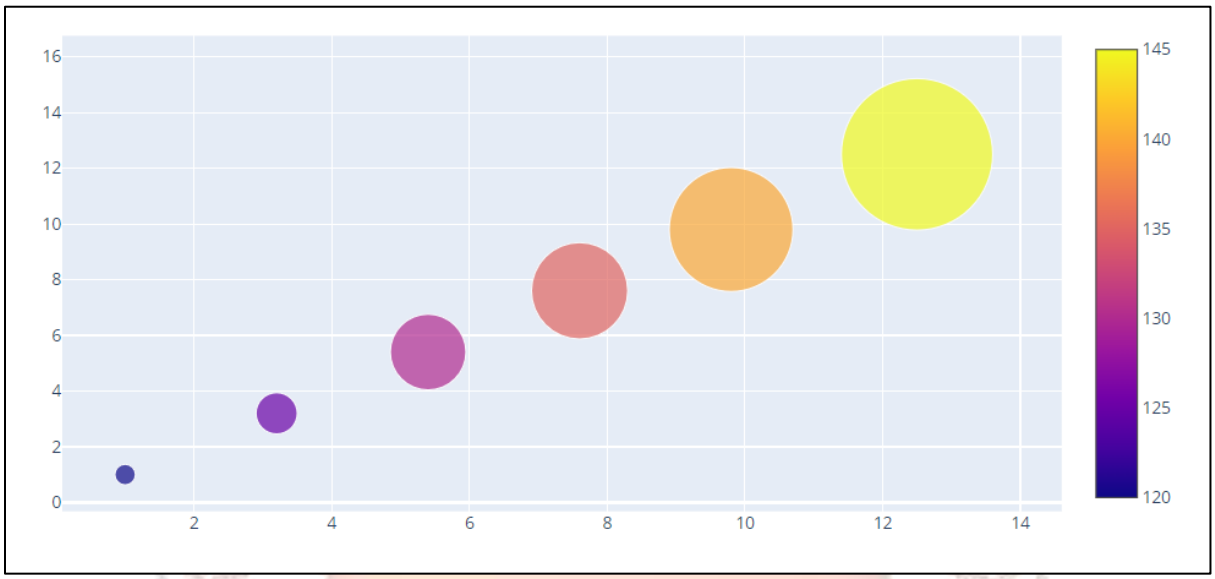
```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Scatter(
    x=[1, 2, 3, 4], y=[10, 11, 12, 13],
    text=['A<br>size: 40', 'B<br>size: 60', 'C<br>size: 80', 'D<br>size: 100'],
    mode='markers',
    marker=dict(
        color=['rgb(93, 164, 214)', 'rgb(255, 144, 14)',  'rgb(44, 160, 101)', 'rgb(255, 65, 54)'],
        size=[40, 60, 80, 100],
```

```
    )])
fig.show()
```



Bubble Charts with Colorscale

```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Scatter(
    x=[1, 3.2, 5.4, 7.6, 9.8, 12.5],
    y=[1, 3.2, 5.4, 7.6, 9.8, 12.5],
    mode='markers',
    marker=dict(
        color=[120, 125, 130, 135, 140, 145],
        size=[15, 30, 55, 70, 90, 110],
        showscale=True
        )])
fig.show()
```

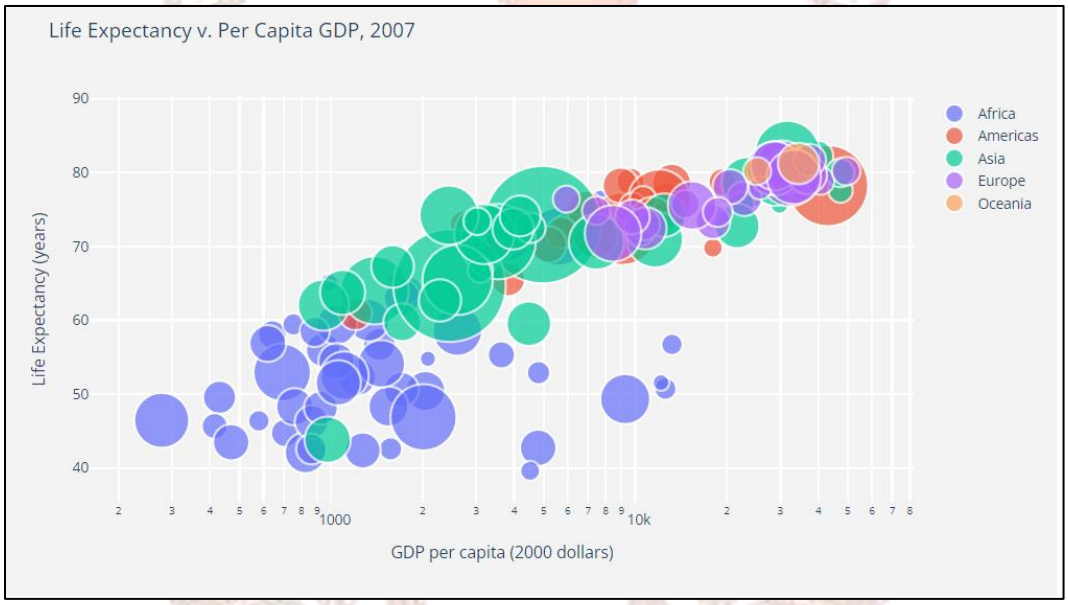


Categorical Bubble Charts

```
import plotly.graph_objects as go
import plotly.express as px
import pandas as pd
import math
# Load data, define hover text and bubble size
data = px.data.gapminder()
df_2007 = data[data['year']==2007]
df_2007 = df_2007.sort_values(['continent', 'country'])
hover_text = []
bubble_size = []
for index, row in df_2007.iterrows():
    hover_text.append(('Country: {country}<br>'+
                'Life Expectancy: {lifeExp}<br>'+
                'GDP per capita: {gdp}<br>'+
                'Population: {pop}<br>'+
                'Year: {year}').format(country=row['country'],
                            lifeExp=row['lifeExp'],
                            gdp=row['gdpPercap'],
                            pop=row['pop'],
```

```
                                year=row['year']))
    bubble_size.append(math.sqrt(row['pop']))
df_2007['text'] = hover_text
df_2007['size'] = bubble_size
sizeref = max(df_2007['size'])/(1002)
# Dictionary with dataframes for each continent
continent_names = ['Africa', 'Americas', 'Asia', 'Europe', 'Oceania']
continent_data = {continent:df_2007.query("continent == '%s'" %continent)
                  for continent in continent_names}
# Create figure
fig = go.Figure()

for continent_name, continent in continent_data.items():
    fig.add_trace(go.Scatter(
        x=continent['gdpPercap'], y=continent['lifeExp'],
        name=continent_name, text=continent['text'],
        marker_size=continent['size'],
        ))
# Tune marker appearance and layout
fig.update_traces(mode='markers', marker=dict(sizemode='area',
                                sizeref=sizeref, line_width=2))
fig.update_layout(
    title='Life Expectancy v. Per Capita GDP, 2007',
    xaxis=dict(
        title='GDP per capita (2000 dollars)',
        gridcolor='white',
        type='log',
        gridwidth=2,
    ),
    yaxis=dict(
        title='Life Expectancy (years)',
```

```
        gridcolor='white',
        gridwidth=2,
    ),
    paper_bgcolor='rgb(243, 243, 243)',
    plot_bgcolor='rgb(243, 243, 243)',
)

fig.show()
```



3. TREE MAPS

A tree map is a visualisation technique to display hierarchical data using nested rectangles. Each tree branch is given a rectangle, tiled with smaller rectangles representing sub-branches. A leaf node's rectangle has an area proportional to a specified data dimension. Often, tree maps use colour and size to show different dimensions of data, making it easy to compare sections of the hierarchy at a glance. Treemaps are particularly useful for displaying the structure of data and helping to identify patterns in a large data set, such as the distribution of file sizes in a file system or the financial performance of different departments within a company.

Advantages of Tree Maps:

- Efficient Use of Space: Treemaps can display thousands of items simultaneously in a compact space.
- Immediate Data Insight: They provide a quick overview of the structure and relative sizes of the various segments within the data.
- Comparative Analysis: Enables easy comparison of sections through size and colour.

Disadvantages of Tree Maps:

- Complexity with Large Hierarchies: This can become cluttered and less effective if the hierarchy is too deep or complex.
- Detail Overload: This may require interactive features like zooming to navigate and understand dense tree maps.

Example:

To illustrate how tree maps work, we can use the matplotlib library in combination with squarify, a Python library that generates tree maps.

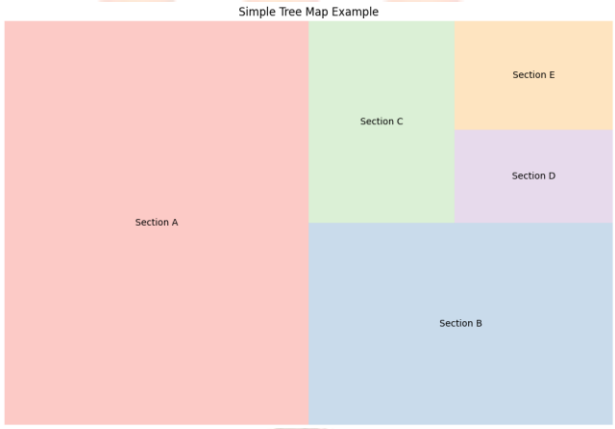
```
! pip install squarify
```

Below is a simple example that demonstrates creating a tree map for a small set of data:

```
import matplotlib.pyplot as plt
import squarify
# Sample data: sizes of different sections
sizes = [50, 25, 12, 6, 7]
```

```
labels = ['Section A', 'Section B', 'Section C', 'Section D', 'Section E']
# Create a color palette
colors = plt.cm.Pastel1(range(len(labels)))
# Create the tree map
plt.figure(figsize=(12, 8))
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=0.7)
# Add a title
plt.title('Simple Tree Map Example')
# Remove axes
plt.axis('off')
# Show the plot
plt.show()
```

Output:



4. INTERACTIVE VISUALS

Interactive visuals are a powerful way to explore and understand complex datasets, allowing users to manipulate the data presentation in real time to uncover deeper insights. These visuals can range from simple hover effects showing more information to complex dashboards that update based on user input or filters. They are instrumental in data analysis, business intelligence, educational tools, and anywhere data needs to be accessible and understandable to a broad audience.

Making Visualisations Interactive

Interactive visualisations extend beyond static charts by enabling user interaction, such as clicking, dragging, zooming, or filtering, to change what the visualisation displays dynamically. This interaction provides a more engaging experience and allows for deeper data exploration.

Key Features:

- Tooltips: Hovering over parts of the visualisation can display tooltips with more information.
- Zoom and Pan: Users can focus on specific areas of the data, especially useful in maps or detailed charts.
- Dynamic Updates: The visualisation can update in real time based on user interactions, such as selecting a category, moving a slider, or entering a value.
- Drill Down: Users can click on chart elements for more detailed data about the selected item.

Technologies and Libraries:

Several Python libraries facilitate the creation of interactive visuals, including:

- Plotly: A graphing library that makes interactive, publication-quality graphs online.
- Bokeh: Designed to create interactive plots and dashboards that can be embedded in web applications.
- Dash: Built on Plotly, Dash is a productive framework for building web analytic applications.

- Streamlit: An app framework specifically for Machine Learning and Data Science projects, allowing the creation of interactive apps quickly.

Example:

Bokeh Installation

```
!pip install bokeh
```

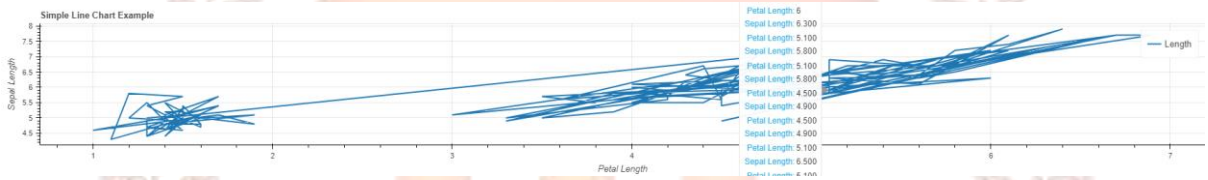
Interactive Line Chart Example with Bokeh

In this example, we'll plot a simple line chart that allows users to hover over the data points to see their values. This type of interactivity is beneficial for time series data or any dataset where you want to observe precise values at specific points.

```
from bokeh.plotting import figure, show, output_notebook

from bokeh.models import HoverTool

from bokeh.sampledata.iris import flowers
# Prepare some data
x = flowers['petal_length']
y = flowers['sepal_length']

# Output to notebook

output_notebook()

# Create a new plot with a title and axis labels
p = figure(title="Simple Line Chart Example", x_axis_label='Petal Length', y_axis_label='Sepal Length', sizing_mode="stretch_width", height=250)

# Add a line renderer with legend and line thickness
p.line(x, y, legend_label="Length", line_width=2)

# Add a hover tool
# Hover tool with vline mode
hover = HoverTool()
hover.tooltips=[
```

```
    ('Petal Length', '@x'),
    ('Sepal Length', '@y'),
]

hover.mode = 'vline'

p.add_tools(hover)

# Show the results
show(p)
```

Output:



5. SUMMARY

The exploration began with an introduction to Bubble Charts, showcasing their utility in displaying multidimensional data engagingly and informatively. Following this, various examples demonstrated the creation of bubble charts using Plotly Express and plotly.graph_objects, highlighting customisation options for marker size, colour, and interactivity. The discussion expanded to include techniques for scaling bubble sizes and enhancing charts with hover text and colour scales, culminating in a comprehensive view of categorical bubble charts. Additionally, tree maps and interactive visuals were discussed, emphasising the power of visualisation in data analysis. Each example reinforced the versatility and impact of effectively presenting data.

5. QUESTIONS

Self-Assessment Questions

SELF-ASSESSMENT QUESTIONS – 1

1. What is the primary purpose of using Bubble Charts in data visualisation?
2. Name one Python library commonly used to create Bubble Charts.
3. In a Bubble Chart, what do the sizes of the bubbles typically represent?
4. How do you customise the appearance of bubbles in a Bubble Chart?
5. What is the core objective of using Tree Maps in data visualisation?
6. Which Python library is often used to create tree maps?
7. Explain the concept of hierarchical data as it pertains to Tree Maps.
8. How does 'squarify' assist in generating Tree Maps in Python?
9. What can you customise in a Tree Map visualisation to make it more informative?
10. Give an example of a practical use case for Tree Maps in a specific domain or industry.

Terminal Questions

1. Explain the key components of a Bubble Chart. How are Bubble Charts useful in visualising three-dimensional data? Provide a step-by-step example of creating a Bubble Chart in Python using a specific dataset and library.
2. Compare and contrast the advantages and disadvantages of Bubble Charts with other data visualisation techniques, such as bar charts and scatter plots. When is it most appropriate to use a Bubble Chart, and what considerations should be considered when designing one for effective data communication?
3. Describe the process of preparing hierarchical data for TreeMap visualisation. How does the 'squarify' library assist in generating Tree Maps in Python, and what customisation options are available for improving the clarity and interpretability of Tree Maps? Provide a real-world use case of Tree Maps in a business or organisational context.

4. In the context of Bubble Charts, discuss the significance of choosing appropriate scaling and colour schemes. How can these choices impact the visual representation and understanding of the data? Provide examples of good and poor design choices and explain their rationale.
5. Explain how the principles of data storytelling can be applied to both Bubble Charts and Tree Maps. Discuss the role of titles, labels, and annotations in effectively conveying insights from these visualisations. Provide practical examples of how data storytelling techniques can make the visualisations more informative and engaging.
6. Discuss the limitations and potential pitfalls of using Tree Maps for hierarchical data visualisation in data exploration and analysis. How can these limitations be mitigated, and what alternative visualisation methods can be considered for conveying hierarchical data effectively? Provide recommendations for best practices in utilising Tree Maps while addressing their constraints.
7. What makes interactive visuals particularly powerful in data analysis and business intelligence?
8. How does Plotly Express facilitate the creation of bubble charts?
9. How do you customise the appearance of markers in a bubble chart using plotly.graph_objects?
10. Explain different ways of creating bubble charts.

6. ANSWERS

Self-Assessment Questions

1. The primary purpose of using Bubble Charts in data visualisation is to represent three data dimensions in a two-dimensional space, where the size of the bubbles represents a third variable.
2. Explain the commonly used Python libraries to create Bubble Charts.
3. In a Bubble Chart, the sizes of the bubbles typically represent a third variable or dimension of data, often indicating magnitude or importance.
4. You can customise the appearance of bubbles in a Bubble Chart by adjusting their colours, labels, and sizes to enhance clarity and visual appeal.
5. The core objective of using Tree Maps in data visualisation is to represent hierarchical data structures as nested rectangles, where the size of each rectangle is proportional to a specific attribute or variable.
6. Explain that the 'squarify' library is often used to create tree maps in Python.
7. Hierarchical data in the context of Tree Maps refers to data organised in a tree-like structure with parent-child relationships, where each node can have sub-nodes.
8. 'Squarify' assists in generating Tree Maps in Python by calculating and plotting nested rectangles to represent hierarchical data effectively.
9. In a Tree Map visualisation, you can customise rectangle colours, labels, and sizes to make it more informative and tailored to the specific data.
10. A practical use case for Tree Maps can be found in financial portfolio management, where the hierarchy represents the breakdown of investments in different asset classes, and the size of each rectangle corresponds to the value of the investments in each class.

Terminal questions

1. Refer Section 2
2. Refer Section 2
3. Refer Section 3
4. Refer Section 2
5. Refer Section 3
6. Refer Section 3
7. Refer Section 4
8. Refer Section 2
9. Refer Section 3
10. Refer Section 2