MASTER OF COMPUTER APPLICATION

SEMESTER 1

DATA VISUALISATION

Unit 14

Dashboard and Story Development in Python

Table of Contents

1. INTRODUCTION

Dashboards are dynamic visualisation tools that amalgamate data from various sources into an interactive and accessible format, employing charts, graphs, and other visual elements. Designed to facilitate quick and informed decision-making, dashboards are invaluable across various domains for monitoring key performance indicators and trends. Real-time data integration makes dashboards crucial for immediate analysis, while customisation options allow for tailored data presentations. With technologies like Dash and Plotly for Python, creating sophisticated dashboards has become more feasible, revolutionising data interaction and analysis.

1.1. Learning Objectives

After studying this unit, you will be able to:

- *Identify the principles of effective data dashboard design*
- *Learn the concepts of storyboarding*
- *Design a simple data dashboard layout*
- *Develop a sophisticated data dashboard with advanced interactive features*

2. INTRODUCTION TO DASHBOARDS

Dashboards are dynamic data visualisation tools that provide a concise and interactive way to present information, often in charts, graphs, and other visual elements. They are designed to enable users to access, understand, and analyse data from various sources quickly and easily. Dashboards are powerful decision support systems across various domains, from business intelligence and analytics to project management and monitoring.

At their core, dashboards offer a consolidated view of key performance indicators, metrics, or data points, allowing users to gain insights, detect trends, and make informed decisions. What sets dashboards apart is their ability to provide real-time or near-real-time information, making them indispensable for timely decision-making. Dashboards are customisable, allowing users to select and arrange the specific data visualisations and components most relevant to their needs. With the advent of dashboard-building tools and libraries, such as Dash and Plotly in Python, creating and deploying dashboards has become more accessible and flexible than ever. Dashboards have transformed how individuals and organisations interact with data, offering a dynamic and user-friendly approach to data analysis and information presentation.

2.1. Understanding Data Dashboards in Python

Understanding data dashboards in Python involves grasping the concepts and techniques for creating interactive visual displays of data for effective analysis and decision-making. Data dashboards are essential in data visualisation and are especially valuable for professionals across various domains. Let's break down the key aspects of understanding data dashboards in Python:

1. Components and Interactivity:

- Data dashboards consist of various components, including charts, graphs, tables, and other visual elements. These components represent data from diverse sources and are often interconnected to provide a holistic view.
- Interactivity is a fundamental feature of dashboards. Users can interact with the data through filters, dropdowns, sliders, or other input methods, allowing them to explore the data dynamically and tailor their views.

2. Designing Effective Dashboards:

- Design principles are crucial for creating effective dashboards. This involves considering the target audience, the data's context, and the goals of the dashboard.
- The layout and visual arrangement of components are critical for data communication. Well-designed dashboards should convey insights at a glance and lead users to explore further.

3. Building Interactive Dashboards with Dash/Plotly:

- Dash and Plotly, Python libraries, offer powerful tools for building interactive data dashboards. These libraries facilitate the creation of web-based dashboards with Python code.
- Users can develop dashboards with dynamic charts, maps, and text components and incorporate features like cross-filtering and real-time data updates.

4. Adding Visual Components and Interactivity:

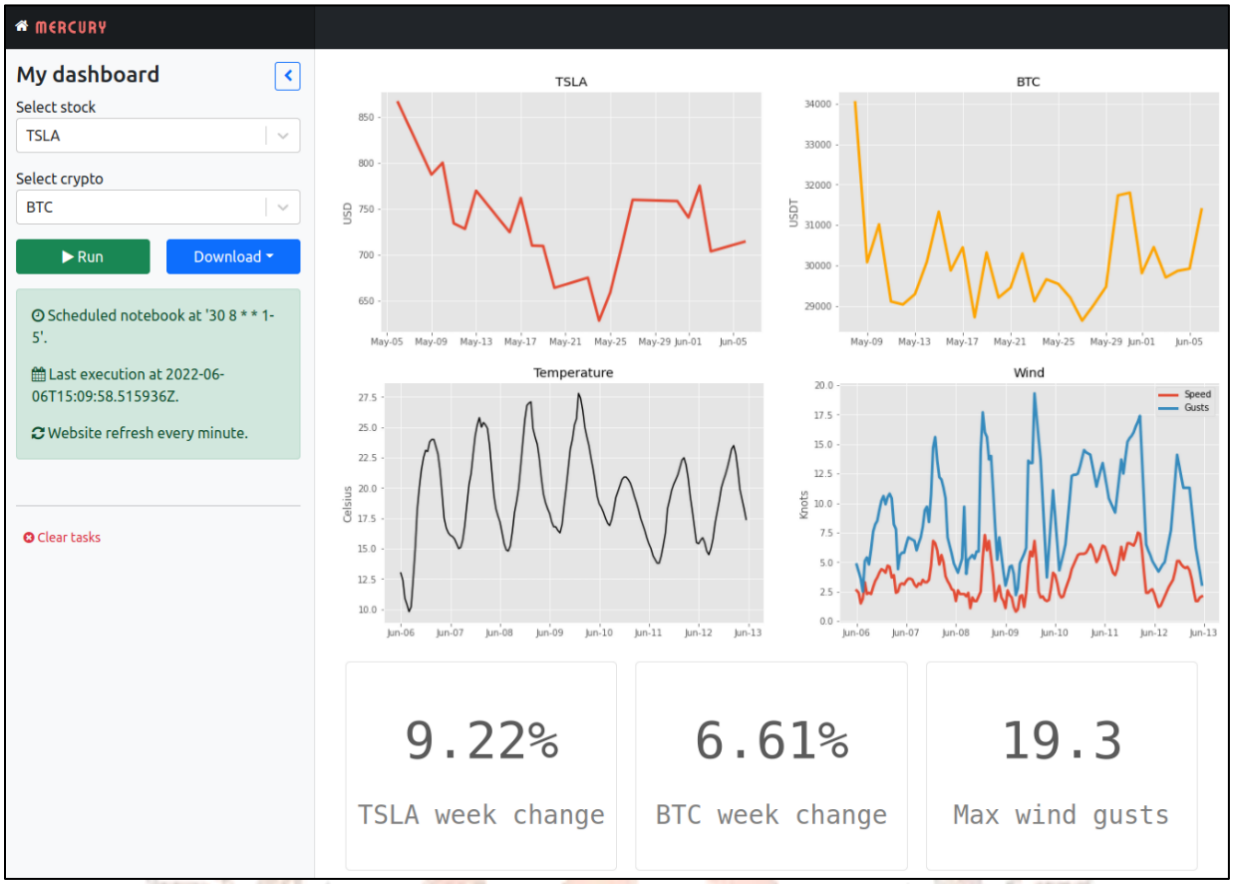
- Dashboards employ a range of visual components, such as bar charts, line plots, heatmaps, and more. These components are selected based on the type of data and insights desired.
- Interactivity is introduced through widgets like dropdowns and sliders, which allow users to explore data subsets or change parameters dynamically.

5. Crafting Data Stories with Visuals:

- Data dashboards can be used to tell data-driven stories. Sequencing and storytelling techniques are applied to guide users through data insights in a narrative format.
- Users can create data stories by combining visuals, text, and interactivity to present data compellingly and informatively.

6. Sequencing and Storytelling:

- Sequencing in dashboards involves arranging data components logically, making it easy for users to follow a narrative or explore data step by step.
- Storytelling focuses on conveying data insights in a narrative format, helping users understand the context and implications of the data.



Above is an example of a data dashboard. Understanding data dashboards in Python is key to unlocking the potential of data visualisation for data analysis and informed decision-making. It empowers individuals and organisations to harness the power of data and present it in a clear, engaging, and actionable manner. Whether used for business intelligence, project management, or any other application, data dashboards are vital for modern data-driven professionals.

2.2. Components and interactivity

Components and interactivity are two fundamental aspects of data dashboards that are pivotal in making these tools effective for data analysis and decision-making. Let's delve into these components in detail:

1. Components of Data Dashboards:

1. Data Visualisations:

- Data visualisations are at the core of data dashboards. They transform raw data into visual representations, making complex information easily understandable.

- Types of data visualisations include bar charts, line graphs, pie charts, scatter plots, heatmaps, and more. Each type is chosen based on the nature of the data and the insights to be conveyed.
- For instance, a bar chart might be used to compare sales figures for different products, while a heatmap can display patterns in temperature variations across regions.

2. Tables:

- Tables in data dashboards present structured data in rows and columns, similar to spreadsheets. They often display detailed information, summaries, or raw data for reference.
- Tables can be customised with features like sorting, filtering, and pagination, allowing users to interact with tabular data effectively.
- They are valuable for displaying data that requires precise values, such as financial data or a list of customer details.

3. Text and Labels:

- Text components in dashboards provide context, explanations, and descriptions. They are essential for guiding users and ensuring they understand the data and visualisations.
- Text components include titles, subtitles, captions, and annotations. They help users quickly grasp the purpose and significance of each dashboard section.
- Annotations can highlight key data points or provide explanations directly on the visualisations.

4. Images and Icons:

- Images and icons are integrated into dashboards for various purposes. They enhance the visual appeal of the dashboard, making it more engaging and informative.
- Images can be used for branding, company logos, or product images. Icons often convey specific messages or actions, such as download icons for reports.
- Visual elements like images and icons contribute to the overall user experience and help convey information beyond numerical data.

5. Widgets and Controls:

- Widgets and controls are interactive elements that empower users to manipulate the data displayed on the dashboard.
- Common widgets include dropdown menus, sliders, buttons, and checkboxes. These allow users to filter data, adjust time periods, or toggle between views.
- Widgets enhance the user's ability to explore data dynamically and extract insights based on their preferences.

6. Data Sources:

- Data dashboards often rely on data sources, which provide the data displayed on the dashboard. These sources include databases, spreadsheets, web APIs, and real-time data feeds.
- Connecting to data sources ensures the dashboard reflects up-to-date information, making it suitable for real-time monitoring or analysis.
- Integrating data sources is critical to dashboard development, enabling users to access live data without manual data entry or updates.

Each component serves a specific role in a data dashboard, collectively providing a comprehensive and interactive platform for data exploration and decision-making. The choice of components and their arrangement depends on the dashboard's objectives and the needs of its users.

Interactivity features in data dashboards in detail:

1. Filtering:

- Filtering is a key interactive feature that allows users to select specific subsets of data based on predefined criteria. These criteria can include date ranges, categories, numerical thresholds, or geographic regions.
- Filtering helps focus on relevant data and customise the view according to the user's needs. For example, in a sales dashboard, users can filter data to view sales for a particular time period or product category.

2. Drill-Down:

- Drill-down interactivity lets users explore data hierarchically. It starts with summary data and allows users to access more detailed levels of information progressively.
- This feature is valuable for in-depth analysis. For instance, users can begin with an overview of total company sales and then drill down to see sales figures for specific regions or individual products.

3. Cross-Filtering:

- Cross-filtering is an advanced interactive feature that enables changes made in one visualisation to update other related visualisations on the dashboard automatically.
- When a user selects or filters data in one chart, such as choosing a specific product category, other linked charts and visualisations adapt to reflect the selected criteria. This provides a synchronised view of data across the entire dashboard.

4. Hover and Tooltip:

- Hover and tooltip interactivity provide additional information when users hover their cursor over or click on a data point in a visualisation.
- Tooltips display relevant details about the specific data point, helping users understand the data behind the visuals. This is particularly useful when dealing with complex or detailed datasets.

5. Real-Time Updates:

- Dashboards can offer real-time data updates in cases where data changes frequently. Users can view the latest data without the need for manual refreshes.
- Real-time updates are crucial for applications such as monitoring systems or financial data analysis where timely information is essential for decision-making.

6. Linked Visuals:

- Linked visuals create connections between different visual components on the dashboard. When a user selects a specific data point or applies a filter in one visualisation, it simultaneously highlights or filters related visualisations.
- This coordination allows users to explore data synchronously, providing a more comprehensive view of the data and its relationships.

7. Export and Download:

- This feature enables users to export data or visualisations from the dashboard. Common export formats include PDF, CSV, or image files.
- Users can download the data or visuals for further analysis, reporting, or sharing with colleagues, making collaboration and external communication convenient.

Interactivity features in data dashboards enhance the user experience, enabling users to explore data dynamically and uncover insights based on their specific needs. These features are critical for making data dashboards powerful tools for data analysis and decision-making in various domains.

3. DASHBOARD DESIGN PRINCIPLES

Dashboard design principles are essential for creating effective and user-friendly data dashboards that communicate information clearly and facilitate decision-making.

- **User-Centred Design:** Focus on understanding the needs, roles, and goals of the dashboard's users. Conduct research, create user personas, and tailor the dashboard to meet these specific needs through iterative design and usability testing.
- **Clarity and Simplicity:** Emphasise clear, uncluttered presentations of data. Utilise simple designs, clear labels, and a hierarchy of information to guide users through the data. Ensure all elements are legible and easily understandable.
- **Consistency:** Maintain uniformity in visual and functional elements such as colour schemes, fonts, and layouts to enhance user experience and ease of use.
- **Information Hierarchy:** Organise content logically with visual cues and headings to guide users through the data meaningfully. Prioritise critical information and use visual emphasis to highlight key data points.
- **Whitespace:** Use negative space to separate and emphasise different dashboard components, enhancing readability and reducing cognitive load.
- **Data Integrity:** Ensure data accuracy and reliability through validation, cleansing, and secure practices. Provide transparent documentation and keep data up-to-date.
- **Visual Hierarchy:** Arrange visual elements to convey the relative importance of information, guiding users' attention to critical insights.
- **Interactivity:** Enhance user engagement with filters, drill-down capabilities, and dynamic updates. Balance complexity and ensure performance optimisation.
- **Mobile Responsiveness:** Design dashboards to be accessible and functional across various devices, using responsive frameworks and touch-friendly elements.
- **Feedback and Iteration:** Engage users in feedback and iteratively improve the dashboard based on their insights. This includes user testing, analysis of feedback, and iterative design adjustments.
- **Accessibility:** Ensure the dashboard is usable by individuals with disabilities by implementing best practices such as alt text for images, semantic HTML, and high-contrast colour schemes.

4. DASHBOARD DEVELOPMENT WITH DASH/PLOTLY

Dashboard development using Dash and Plotly is an approach that allows you to create interactive web-based data dashboards in Python. Dash is a Python web framework for building analytical web applications. Plotly is a graphing library that helps create interactive plots and charts. Together, they provide a powerful toolkit for developing data dashboards. Below is an overview of the process of building a dashboard with Dash and Plotly:

1. Installation: To get started, you must install the necessary libraries. You can install Dash and Plotly using pip:

pip install Dash plotly

2. Import Libraries: In your Python script or Jupyter Notebook, import the required libraries:

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
```

3. Initialise the App: Create a Dash web application by initialising it.

```
app = dash.Dash(__name__)
```

4. Layout: Define the layout of your dashboard. This includes creating the web page's structure, including any HTML elements and components. You can use `html.Div`, `dcc.Graph`, and other components to structure your layout.

```
app.layout = html.Div([
    html.H1("My Data Dashboard"),
    dcc.Dropdown(  # Adding a missing Dropdown for demonstration
        id="dropdown-menu",
        options=[
            {'label': 'Option 1', 'value': 'opt1'},
            {'label': 'Option 2', 'value': 'opt2'}
```

```
        ],
        value='opt1'  # Default value
    ),
    dcc.Graph(id="my-graph"),
])
```

5. Callback Functions: You can use callback functions to make your dashboard interactive. These functions define the interactivity between components. For instance, you can create a

```
@app.callback(
    Output("my-graph", "figure"),
    [Input("dropdown-menu", "value")]
)  )
def update_graph(selected_value):
    # Example update logic using selected_value, assuming it impacts the data source
    df = px.data.gapminder()  # Example dataset
    if selected_value == 'opt1':
        fig = px.scatter(df, x="gdpPercap", y="lifeExp", size="pop", color="continent",
                    hover_name="country", log_x=True, size_max=60)
    else:
        fig = px.line(df, x="year", y="lifeExp", color="continent")
    return fig
```

6. Running the App: To run your dashboard, you'll need to add the following lines at the end of your script:

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

Output:

7. Testing and Deployment: You can test your dashboard locally by running the script. Once satisfied with the results, you can deploy your dashboard to a web server or cloud platform to make it accessible to others.

8. Styling and Customisation: You can further customise and style your dashboard using CSS or Dash's built-in styling options.

9. Data Integration: To make your dashboard truly useful, you'll need to integrate it with your data sources. This might involve reading data from databases, APIs, or files and updating the dashboard dynamically.

10. Documentation and Sharing: It's a good practice to provide documentation for your dashboard, explaining how to use it. You can also share it with others by hosting it online or sharing it as a standalone application.

Building a data dashboard with Dash and Plotly is a flexible and powerful way to create interactive data visualisations and provide insights to your audience. It's particularly useful for data scientists, analysts, and developers who want to create data-driven web applications without diving deep into web development.

Example :

Building blocks of Dash

Dash applications are made up of 2 building blocks :

1. Layout
2. Callbacks

The layout describes the look and feel of the app; it defines the elements such as graphs, dropdowns, etc., as well as the placement, size, colour, etc. Dash contains Dash HTML components, which we can create and style HTML content such as headings, paragraph, images, etc., using Python. Elements such as graphs, dropdowns, and sliders are created using Dash Core components.

Callbacks are used to bring interactivity to the dash applications. We use these functions to define the activity that would happen by clicking a button or a dropdown.

Layouts using Dash

Now, let's look at creating web-based layouts using plotly Dash. Before starting with the layout, let's install some required packages. (You can run the codes using Anaconda Spyder, a Python development environment.)

```
! pip install Dash
! pip install dash-html-components
! pip install dash-core-components
! pip install plotly
```

Now, we will import the dash package, dash_html_components for HTML classes, dash_core_components for elements such as graph, dropdown, etc., and plotly packages to create plots and read the stock prices dataset.

```
import dash
import dash_html_components as html
import dash_core_components as dcc
import plotly.graph_objects as go
import plotly.express as px
```

(In the below code) We are initialising our dash app using the dash package. Then, we read the stock price data for different companies from 2018 to 2019. We are creating a stock_prices function that returns the line chart for Google's stock prices.

```
app = dash.Dash(__name__)  #initialising dash app
df = px.data.stocks() #reading stock price dataset
def stock_prices():
   # Function for creating line chart showing Google stock prices over time
   fig = go.Figure([go.Scatter(x = df['date'], y = df['GOOG'],\
            line = dict(color = 'firebrick', width = 4), name = 'Google')
            ])
   fig.update_layout(title = 'Prices over time',
            xaxis_title = 'Dates',
            yaxis_title = 'Prices'
            )
   return fig
app.layout = html.Div(id = 'parent', children = [
   html.H1(id = 'H1', children = 'Styling using html components', style =
{'textAlign':'center',\
                        'marginTop':40,'marginBottom':40}),
      dcc.Graph(id = 'line_plot', figure = stock_prices())
   ]
            )
```

(In the above code) In line 16, we are setting our layout using html Div component, a wrapper within which the layout's elements(heading, graph) will be created. The Div component contains arguments such as id (a unique identifier of the element), style (for setting the width, height, colour, etc.) and children (equal to the square bracket within which elements of the layout are initialised).

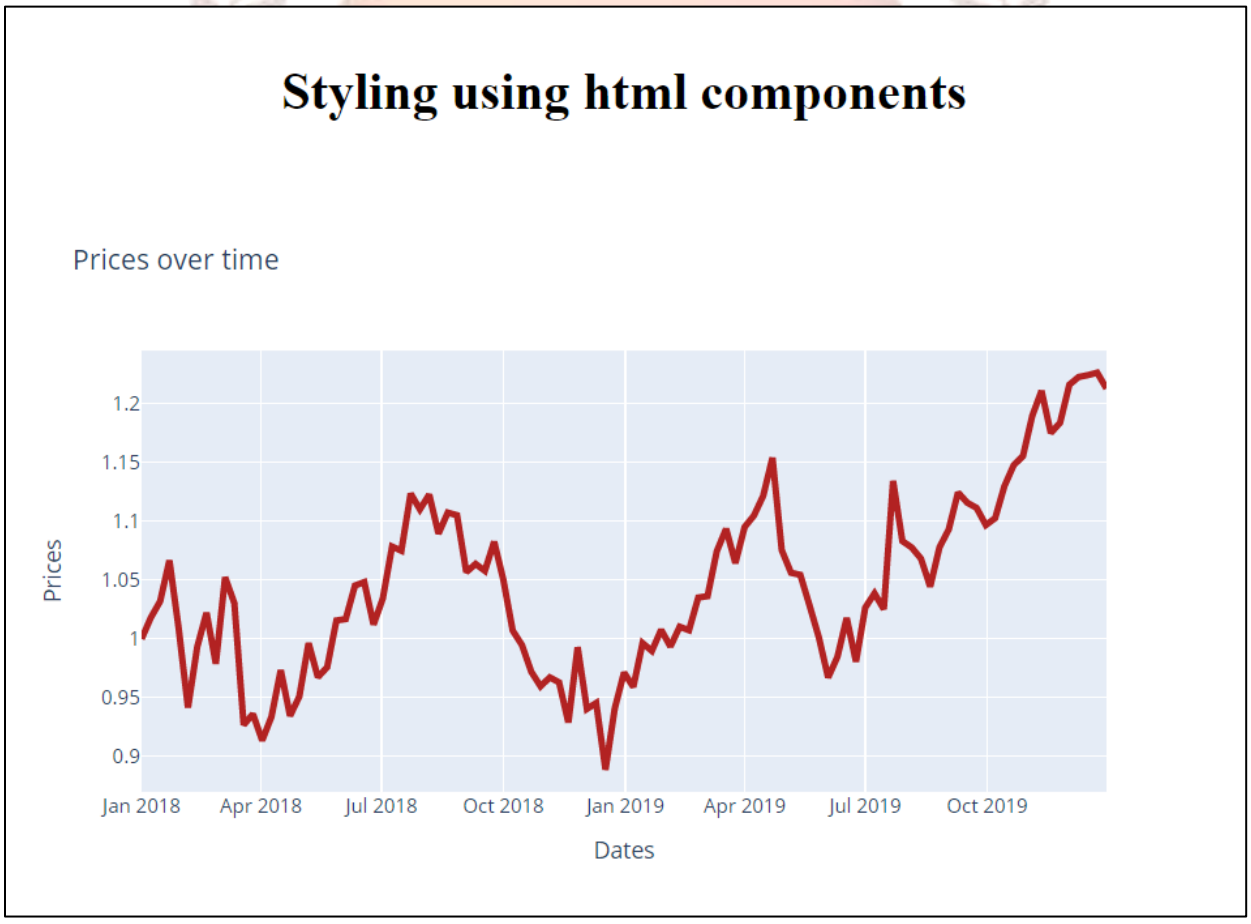
Inside the children component (of html. Div) we are creating the html H1 heading at line 17 using the H1 function. Inside the function, we are setting the unique id of the function (id = 'H1'), children property using which we set the text of the heading and style property as a dictionary within which we are setting styling such as centre aligning the text, setting the top and bottom margin to 40 pixels. At line 21, we are using the dash core component (dcc) to

create a graph, where we are setting the id of the graph and the figure argument, which is equal to the function call (stock_pricest()) that returns the plotly figure object.

We need to run our web server like in Flask to view our application. Remember, Dash is built on top of Flask.

```
if __name__ == '__main__':
    app.run_server()
```

On running the app, you will see that the app is running on http://127.0.0.1:8050/, which is your local server. Copy and paste this URL into your browser; see the visualisation below.



Now, we will create a dropdown using dash core components. Using the dropdown, we can select Google, Apple, or Amazon stocks.

```
dcc.Dropdown( id = 'dropdown',
options = [
   {'label':'Google', 'value':'GOOG' },
   {'label': 'Apple', 'value':'AAPL'},
   {'label': 'Amazon', 'value':'AMZN'},
   ],
value = 'GOOGL'
)
```

Dropdowns are created using the Dropdown () function, which has the following arguments

1. **id** — Unique identifier of the dropdown.
2. **options** — Sets the 'label' (the text visible in the dropdown) and 'value' (used by Dash to communicate with callbacks) as a key-value pair.
3. **value** — default selection for the dropdown.

Output:

Callbacks in Dash: Making it interactive

Let's create the callback connecting the dropdown and stock prices line chart.

A callback is initialised using @app.callback(), followed by a function definition. Within this function, we define what happens when changing the dropdown's value.

from Dash.dependencies import Input, Output

```
@app.callback(Output(component_id='line_plot', component_property= 'figure'),
        [Input(component_id='dropdown', component_property= 'value')])
def graph_update(dropdown_value):
   print(dropdown_value)
```

```
fig = go.Figure([go.Scatter(x = df['date'], y = df['{}'.format(dropdown_value)],\
        line = dict(color = 'firebrick', width = 4))
        ])
  fig.update_layout(title = 'Stock prices over time',
        xaxis_title = 'Dates',
        yaxis_title = 'Prices'
        )
return fig
```

Let's have a look at the arguments within the callback function :

1. **Output:** This defines the components within the layout, which will be updated when the function below the callback (graph_update()) returns some object. The output function takes 2 arguments — 1) component_id defines the id of the component we want to update with our function graph_update. We want to update the stock prices chart within the dcc. Graph so that we will set the component id to 'line_plot', id of our graph component. 2) Component property defines the property of the component that will be updated, which is the figure property of dcc.Graph in our layout.
2. **Input:** This defines the components, the change in whose value will trigger the callback. The input function also takes component_id and component_property as arguments. We want the callback to get triggered based on the change in the value of our dropdown, so we set the component_property to the 'value' property of the dropdown. Please note that input is defined in a list.

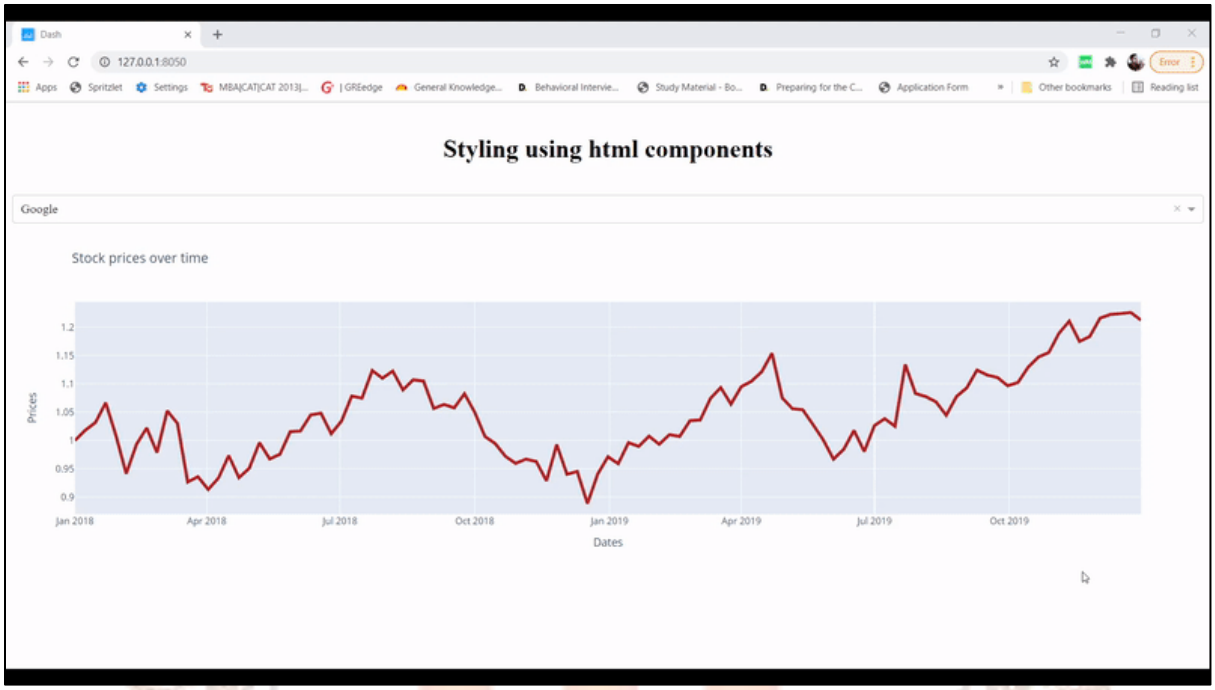
The component property of the Input function, which is the 'value' of the dropdown, goes as an argument within the function graph_update. Inside the function, we create a scatter plot and return the figure object fig, which is passed to the figure property of dcc.Graph using the Output function of the callback.

Bringing it all together

Now let's combine the layout, the dropdown and the callback in the below code :

```
import dash
from dash import dcc, html  # Updated for Dash 1.11.0+ compatibility
from dash.dependencies import Input, Output  # Corrected 'Dash' to 'dash'
import plotly.graph_objects as go
import plotly.express as px
app = dash.Dash(__name__)
df = px.data.stocks()
app.layout = html.Div(id='parent', children=[
   html.H1(id='H1', children='Styling using HTML components',
style={'textAlign':'center', 'marginTop':40, 'marginBottom':40}),
   dcc.Dropdown(id='dropdown',
            options=[
               {'label': 'Google', 'value': 'GOOG'},
               {'label': 'Apple', 'value': 'AAPL'},
               {'label': 'Amazon', 'value': 'AMZN'},
            ],
            value='GOOG'),
   dcc.Graph(id='bar_plot')
])
@app.callback(Output(component_id='bar_plot', component_property='figure'),
          [Input(component_id='dropdown', component_property='value')])
def graph_update(dropdown_value):
   fig = go.Figure(go.Scatter(x=df['date'], y=df[dropdown_value],
                     line=dict(color='firebrick', width=4)))
   fig.update_layout(title='Stock prices over time',
                xaxis_title='Dates',
                yaxis_title='Prices')
   return fig
if __name__ == '__main__':
   app.run_server(debug=True)
```

The image below shows how the change in the value of the dropdown updates our stock prices line chart.



5. STORY DEVELOPMENT

Data storytelling is transforming data into a compelling and easily understandable narrative using the power of visualisation. It uses data visualisation, infographics, and various tools to make data accessible and engaging to a broad audience.

The primary objective of data storytelling is to convey insights and discoveries in a manner that is informative and captivating. By creating a structured storyline around the data, data storytellers assist their audience in comprehending intricate data sets, drawing meaningful conclusions, and making informed decisions.

Data storytelling combines visualisation with narrative techniques to make complex information understandable and engaging. It requires a clear message, a coherent narrative, visual aids, and an understanding of the audience's context. Key skills include data comprehension, communication, and presentation.

Steps for Crafting a Data Story

- **Define Objectives:** Determine the key insights or messages you want to convey. What story does the data tell, and what should the audience take away?
- **Select Data and Visualisations:** Choose relevant data and the types of visualisations (line charts, bar graphs, pie charts, heatmaps) that best illustrate your story.
- **Organise the Sequence:** Arrange visualisations and data presentations to create a logical flow, guiding the audience through the narrative.
- **Use Annotations and Text:** Employ text annotations and descriptions to provide context and explanations, highlighting key points or trends.
- **Implement Transitions and Interactivity:** Smooth transitions between dashboard elements and interactive features like dropdowns or sliders engage users and allow exploration.
- **Maintain Consistency in Design:** Use a consistent colour scheme, fonts, and styling for a cohesive narrative flow.
- **Create Focus Points:** Emphasise crucial findings or story turns with visual cues to draw attention.

- **Test and Iterate:** Seek feedback and refine the dashboard based on user interactions and understanding.
- **Documentation and Sharing:** Include documentation for transparency and share the final story with your audience, prepared to guide them through the narrative.

Visuals in Data Storytelling

- **Select the Right Visualisations:** Match plot types to your data store (e.g., line plots for changes over time).
- **Minimise Clutter:** Edit plots to focus on the data, removing distracting non-data elements.
- **Effective Use of Text:** Incorporate text to highlight insights and provide context without overwhelming the visual.
- **Colour Theory:** Apply colour effectively to emphasise different aspects of your visualisation, considering hue, chroma, and luminance.

Narrative Techniques

- **Know Your Audience:** Tailor your message based on the audience's knowledge, priorities, and constraints.
- **Choose the Best Medium:** Decide on the most suitable format for your story (presentations, reports, notebooks, dashboards) based on the audience and setting.

Sequencing and Storytelling in Dashboards

Implement sequencing by structuring data and visualisations to narrate the data story logically. Use Python libraries like Dash and Plotly to build interactive, engaging dashboards that guide users through the narrative, enhancing their understanding and engagement.

Here's an example of how to implement sequencing and storytelling in a Python dashboard using Dash and Plotly:

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
```

```
# Sample data
import pandas as pd
data = pd.DataFrame({
    'Year': [2010, 2011, 2012, 2013, 2014, 2015],
    'Sales': [100, 120, 150, 130, 170, 200],
    'Expenses': [80, 90, 100, 110, 120, 130]
})
# Initialise the Dash app
app = dash.Dash(__name__)
# Define the layout
app.layout = html.Div([
    dcc.Graph(id='sales-expenses-plot'),
    dcc.Slider(
        id='year-slider',
        min=data['Year'].min(),
        max=data['Year'].max(),
        value=data['Year'].min(),
        marks={str(year): str(year) for year in data['Year'].unique()},
    ),
    html.Div(id='story-text')
])
# Define callback for updating the plot
@app.callback(
    [Output('sales-expenses-plot', 'figure'),
     Output('story-text', 'children')],
    [Input('year-slider', 'value')]
)
def update_plot(selected_year):
    filtered_data = data[data['Year'] == selected_year]
        # Create a plot using Plotly Express
```

```
    # Create a plot using Plotly Express
    fig = px.bar(filtered_data, x='Year', y=['Sales', 'Expenses'],
                 labels={'value': 'Amount in USD'})
    fig.update_layout(title=f'Sales and Expenses in {selected_year}')
    # Create a narrative text
    story_text = f"In {selected_year}, we had sales of ${filtered_data['Sales'].values[0]:.2f} and expenses of ${filtered_data['Expenses'].values[0]:.2f}."
    return fig, story_text
if __name__ == '__main__':
    app.run_server(debug=True)
```

Output:

In this example:

1. We import the necessary libraries and create a sample dataset representing sales and expenses for multiple years.
2. We initialise a Dash app and define its layout. The layout includes a Plotly graph, a slider for selecting the year, and a text element for the story.
3. We set up a callback function that responds to changes in the selected year. When the user moves the slider, the callback function filters the data for the selected year and updates the Plotly graph and the story text accordingly.
4. The Plotly Express library creates a bar chart showing sales and expenses for the selected year. The story text is dynamically generated based on the selected year and the corresponding data.
5. We run the Dash app.

This example demonstrates how you can use Python, Dash, and Plotly to create a dashboard that allows users to explore a data story by sequencing through different visualisations and accompanying text.

6. SUMMARY

Understanding data dashboards, particularly in Python, involves recognising their components, interactivity, and the principles behind effective dashboard design. Dashboards consist of visualisations, tables, text, and widgets, making data exploration dynamic and insightful. Interactivity features such as filtering and cross-filtering enhance user engagement. Key design principles include clarity, consistency, and responsiveness, ensuring dashboards are accessible and informative. Story development in dashboards uses sequencing and narrative techniques to guide users through data insights, which is made possible by Python libraries like Dash and Plotly. This approach simplifies complex data analysis and democratises data understanding, making it an essential skill for modern professionals.

7. QUESTIONS

Self-Assessment Questions

1. What are the key components of data dashboards discussed in the chapter?
2. How does interactivity enhance the user experience in data dashboards?
3. Why is user-centered design an essential principle when designing dashboards?
4. What is the significance of maintaining consistency in dashboard design?
5. What are the advantages of using Dash and Plotly for building interactive dashboards?
6. How can visual components and interactivity improve data exploration in dashboards?
7. What role does data storytelling play in effective dashboard communication?
8. What are some common layout options for data dashboards mentioned in the chapter?
9. How can data integrity be ensured in data dashboards?
10. Why is feedback and iteration important in maintaining the relevance and effectiveness of dashboards?

Terminal questions

1. How do data visualisations, tables, text and labels, images and icons, widgets and controls, and data sources contribute to the components of data dashboards?
2. Explain the importance of maintaining consistency in dashboard design, including the use of color schemes, fonts, and styles.
3. Describe the process of building interactive dashboards with Dash/Plotly in Python. What are the key advantages of using Dash and Plotly for dashboard development?
4. Discuss the principles of adding visual components and interactivity to data dashboards.
5. How does data storytelling play a crucial role in effective dashboard communication?
6. Explain the key elements of data storytelling and how they help users understand complex data sets and draw meaningful conclusions.
7. Explore the concept of sequencing and storytelling in data dashboards.
8. How can storytelling techniques be employed to guide users through data narratives, and why is sequencing important in dashboard design?
9. Describe the importance of user-centered design in dashboard development.
10. How does interactivity enhance the utility of data dashboards?

8. ANSWERS

Self-Assessment Questions

1. The key components of data dashboards discussed in the chapter include data visualisations, tables, text and labels, images and icons, widgets and controls, and data sources.
2. Interactivity enhances the user experience in data dashboards by allowing users to filter, sort, and manipulate data, providing real-time updates, enabling drill-down into data details, and supporting cross-filtering.
3. User-centered design is essential when designing dashboards because it ensures that the dashboard meets the needs and preferences of the end-users, tailoring it to their expectations and supporting their decision-making processes.
4. Maintaining consistency in dashboard design is crucial because it ensures that the design elements, such as color schemes, fonts, and styles, remain uniform throughout the dashboard, enhancing professionalism and ease of navigation.
5. Dash and Plotly are advantageous for building interactive dashboards because they provide user-friendly tools and libraries for creating web-based interactive data visualisations and dashboards in Python.
6. Visual components and interactivity can improve data exploration in dashboards by allowing users to customise their view, apply filters, and interact with the data, making it easier to derive insights.
7. Data storytelling plays a significant role in effective dashboard communication by presenting data in a narrative format that helps users understand complex data sets, draw conclusions, and make informed decisions.
8. Some common layout options for data dashboards include standard rectangular dashboards, long-form (vertical and horizontal) dashboards, and multiple levels (landing page + investigational layers).
9. Data integrity can be ensured in data dashboards by validating data sources and calculations, and by including data source references or timestamps to inform users of data reliability and freshness.
10. Feedback and iteration are important in maintaining the relevance and effectiveness of dashboards because they allow for ongoing improvements based on user feedback and

evolving user needs. This iterative process ensures that the dashboard remains useful and up-to-date.

Terminal Questions

1. Refer section 3.1
2. Refer section 3.2
3. Refer section 4.1
4. Refer section 4.2
5. Refer section 5.1
6. Refer section 5.1
7. Refer section 5.2
8. Refer section 5.2
9. Refer section 5
10. Refer section 4