**Creating Dashboards with Plotly and Dash**

Dashboards and interactive data applications are crucial tools for data visualization and analysis because they provide a consolidated view of key data and metrics in a visually appealing and understandable format. In this module, you will explore the benefits of dashboards and identify the different web-based dashboarding tools in Python. You will learn about Plotly and discover how to use Plotly graph objects and Plotly express to create charts. You will gain insight into Dash, an open-source user interface Python library, and its two components. Finally, you will gain a clear understanding of the callback function and determine how to connect core and HTML components using callback.

**Learning Objectives**

* Discover Dash and its two components
* Identify different web-based dashboarding tools available in Python
* Use Plotly graph objects and Plotly express to create charts
* Explore Plotly and its two sub-modules
* Determine the process of connecting core and HTML components using callback
* Describe the callback function

# **Creating Dashboards with Plotly**

## **Dashboarding Overview**

* **Benefits of Interactive Data Applications**:
  + Improve business performance.
  + Provide real-time visuals for better understanding of business dynamics.
* **Types of Reports**:
  + **Type 1**: Presented through tables with documented inferences.
  + **Type 2**: Presented in a dashboard format, allowing for interactive exploration of data.
* **Importance of Dashboards**:
  + Simplifies the presentation of dynamic business data.
  + Helps stakeholders understand performance and make informed decisions.
* **Web-Based Dashboarding Tools in Python**:
  + **Dash**: Framework for building web analytic applications.
  + **Panel**: Works with various Python plotting libraries and is suitable for Jupyter Notebooks.
  + **Voila**: Turns Jupyter notebooks into standalone web applications.
  + **Streamlit**: Converts data scripts into shareable web apps.
  + **Bokeh**: Plotting library that also serves as a dashboard server.
  + **Matplotlib**: Comprehensive library for creating various visualizations.

Conclusion

Dashboards are essential for presenting data in a visually appealing and comprehensible manner, enabling better decision-making in business contexts.

Additional Resources for Dashboards

For more information about Dashboards, visit the following links:

[Python dashboarding tools](https://pyviz.org/dashboarding/)

[John Snow's data journalism](https://www.theguardian.com/news/datablog/2013/mar/15/john-snow-cholera-map)

## **Introduction to Plotly**

1. **What is Plotly?**
   * An **interactive, open-source plotting library** that supports over **40 unique chart types**.
   * Available in **Python** and **JavaScript**.
   * Useful for creating **web-based visualizations** that can be displayed in Jupyter Notebook or saved as HTML files.
2. **Sub-modules of Plotly:**
   * **Plotly Graph Objects**: A low-level interface for creating figures, traces, and layouts.
   * **Plotly Express**: A high-level wrapper that simplifies the creation of common figures.

Code Example: Creating a Simple Line Chart

Using Plotly Graph Objects

import plotly.graph\_objects as go

import numpy as np

# Generate sample data

np.random.seed(10) # For reproducibility

x = np.arange(12) # X values (0 to 11)

y = np.random.randint(50, 500, size=12) # Random Y values

# Create a figure

fig = go.Figure(data=go.Scatter(x=x, y=y))

# Update layout

fig.update\_layout(title='Simple Line Plot', xaxis\_title='Month', yaxis\_title='Sales')

# Show the figure

fig.show()

**Explanation:**

* **Importing Libraries**: We import plotly.graph\_objects as go and numpy for data generation.
* **Data Generation**: We create an array of 12 elements for the x-axis and random integers for the y-axis.
* **Creating the Figure**: We use go.Figure to create a scatter plot.
* **Updating Layout**: We set titles for the chart and axes.
* **Displaying the Plot**: Finally, we call fig.show() to display the plot.

Using Plotly Express

import plotly.express as px

# Create the same line chart in one command

fig = px.line(x=x, y=y, title='Simple Line Plot', labels={'x': 'Month', 'y': 'Sales'})

# Show the figure

fig.show()

**Explanation:**

* **Simplified Syntax**: Plotly Express allows us to create the same line chart with a single command, making it easier to use.
* **Interactive Visualization**: The chart created is automatically interactive.

Summary

* **Plotly** is a powerful tool for data visualization, allowing for both simple and complex visualizations.
* **Graph Objects** provide detailed control, while **Plotly Express** offers a more straightforward approach for common visualizations.

# **Additional Resources for Plotly**

* To learn more about using Plotly to create dashboards, explore
* [Plotly python](https://plotly.com/python/getting-started/)
* [Plotly graph objects with example](https://plotly.com/python/graph-objects/)
* [Plotly express](https://plotly.com/python/plotly-express/)
* [API reference](https://plotly.com/python-api-reference/)
* Here are additional useful resources:
* [Plotly cheatsheet](https://images.plot.ly/plotly-documentation/images/plotly_js_cheat_sheet.pdf)
* [Plotly community](https://community.plotly.com/c/api/5)
* [Related blogs](https://plotlygraphs.medium.com/)
* [Open-source datasets](https://developer.ibm.com/exchanges/data/)

# **Working with Dash**

## **Introduction to Dash**

* **What is Dash?**
  + An open-source Python library for creating reactive web-based applications.
  + Utilizes Flask for web servers and communicates via JSON packets over HTTP.
  + Front end uses React.js for rendering components.
* **Components of Dash:**
  + **Core Components (DCC):** Higher-level interactive components (e.g., sliders, input areas).
  + **HTML Components (HTML):** Components for every HTML tag, allowing layout composition using Python.
* **Key Features:**
  + Declarative and reactive design.
  + Mobile and cross-platform ready.
  + Simplifies the process of building graphical user interfaces.
* **Application Development Steps:**
  + Determine the application layout.
  + Decide on charts and their placements.
  + Add interactivity to the application.

Additional Resources for Dash

To learn more about Dash, explore

[Complete dash user guide](https://dash.plotly.com/)

[Dash core components](https://dash.plotly.com/dash-core-components)

[Dash HTML components](https://dash.plotly.com/dash-html-components)

[Dash community forum](https://community.plotly.com/c/python/25)

[Related blogs](https://medium.com/plotly/tagged/dash)

## **Make Dashboards Iinteractive**

Key Points from the Lecture

* **Callback Function**: A Python function that Dash automatically calls whenever an input component's property changes.
* **@app.callback Decorator**: This decorator tells Dash to call the function whenever there's a change in the input component value.
* **Parameters**:
  + **Output**: Sets the results returned from the callback function to a component ID.
  + **Input**: Provides input to the callback function from a component ID.

Callback Function Structure

1. **Define the Function**: Create a function that performs operations to return the desired result for the output component.
2. **Decorate the Function**: Use the @app.callback decorator to connect input and output components.
3. **Connect Input and Output**: Specify which properties of the components are being used.

Example Code

Here’s a simplified version of the code structure discussed in the lecture:

import pandas as pd

from dash import Dash, html, dcc

from dash.dependencies import Input, Output

# Initialize the Dash app

app = Dash(\_\_name\_\_)

# Load the airline data

df = pd.read\_csv('airline\_data.csv')

# Define the layout of the app

app.layout = html.Div([

html.H1("Airline Data Visualization"),

dcc.Input(id='input-yr', value='2010', type='text'),

dcc.Graph(id='bar-plot')

])

# Define the callback function

@app.callback(

Output('bar-plot', 'figure'),

Input('input-yr', 'value')

)

def get\_graph(input\_year):

# Filter data based on the input year

filtered\_data = df[df['year'] == int(input\_year)]

# Create a bar plot (example)

figure = {

'data': [{

'x': filtered\_data['carrier'],

'y': filtered\_data['flights'],

'type': 'bar'

}],

'layout': {

'title': f'Top Airlines in {input\_year}'

}

}

return figure

# Run the app

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

Explanation of the Code

* **Imports**: Necessary libraries are imported, including Dash components and Pandas for data manipulation.
* **App Initialization**: The Dash app is initialized.
* **Data Loading**: Airline data is loaded into a Pandas DataFrame.
* **Layout Definition**: The layout includes an input field for the year and a graph to display the results.
* **Callback Definition**: The @app.callback decorator connects the input and output. The function get\_graph filters the data based on the input year and creates a bar plot.
* **Running the App**: The app is run in debug mode.

Additional Resources for Interactive Dashboards

To learn more about making interactive dashboards in Dash, visit

[Python decorators reference 1](https://realpython.com/primer-on-python-decorators/)

[Python decorators reference 2](https://www.python.org/dev/peps/pep-0318/#current-syntax)

[Callbacks with example](https://dash.plotly.com/basic-callbacks)

[Dash app gallery](https://dash-gallery.plotly.host/Portal/)

[Dash community components](https://plotly.com/dash-community-components/)

## **Understanding of Lab Environment**

1. **Skills Network Labs Cloud IDE**:
   * A web-based environment for completing course-related labs.
   * Comprises two sides: instructions on the right and the Integrated Development Environment (IDE) on the left.
2. **Data Analysis**:
   * Analyzing flight on-time data using a dataset from the Bureau of Transportation Statistics.
   * Creating line graphs for various types of delays.
3. **Creating a Dash Application**:
   * Steps to set up the application, including installing required packages and writing code.

Coding Steps:

1. **Install Required Packages**:
   * Use terminal commands to install necessary libraries.
2. **Create a New Script File**:
   * Name the file flight\_delay.py.
3. **Import Libraries and Load Dataset**:
4. import pandas as pd

airline\_data = pd.read\_csv('path\_to\_dataset.csv')

* + **Explanation**:
    - import pandas as pd: Imports the pandas library for data manipulation.
    - pd.read\_csv(...): Reads the CSV file containing flight data.

1. **Design the Layout of the Dash App**:
2. from dash import Dash, html, dcc
3. app = Dash(\_\_name\_\_)
4. app.layout = html.Div([
5. dcc.Input(id='input-year', value=2010, type='number'),
6. dcc.Graph(id='graph-output')

])

* + **Explanation**:
    - from dash import Dash, html, dcc: Imports necessary components from the Dash library.
    - app = Dash(\_\_name\_\_): Initializes the Dash application.
    - app.layout: Defines the layout of the app, including an input field and a graph.

1. **Callback Function**:
2. @app.callback(
3. Output('graph-output', 'figure'),
4. Input('input-year', 'value')
5. )
6. def update\_graph(selected\_year):

# Code to compute and return the figure

* + **Explanation**:
    - @app.callback(...): Decorator to link input and output components.
    - update\_graph(selected\_year): Function that updates the graph based on user input.

1. **Run the Application**:
2. if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

* + **Explanation**:
    - app.run\_server(...): Starts the Dash server, allowing the app to be accessed in a web browser.

Summary: Creating Dashboards with Plotly and Dash

Congratulations! You have completed this module. At this point in the course, you know:

* Dash is an Open-Source User Interface Python library for creating reactive, web-based applications.
* It is easy to build Graphical User Interfaces using Dash as it abstracts all technologies required to make the applications.
* There are two components of Dash: Core and HTML components.
* The dash\_core\_components describe higher-level interactive components generated with JavaScript, HTML, and CSS through the React.js library.
* The dash\_html\_components library has a component for every HTML tag.
* A callback function is a python function that is automatically called by Dash whenever an input component's property changes.
* The @app.callback decorator decorates the callback function in order to tell Dash to call it whenever there is a change in the input component value.
* The callback function takes input and output components as parameters and performs operations to return the desired result for the output component.