plotly

Star 23,446

Dash Python > ***Dash in 20 Minutes Tutorial***

Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. **Sign up for early access now.**

Dash in 20 Minutes

By the end of this tutorial, you will understand the basic building blocks of Dash and you will know how to build this app:

▸ View app

Hello World

Building and launching an app with Dash can be done with just 5 lines of code.

Open a Python IDE on your computer, create an `app.py` file with the code below and install Dash if you haven't done so already. To launch the app, type into your terminal the command `python app.py`. Then, go to the http link.

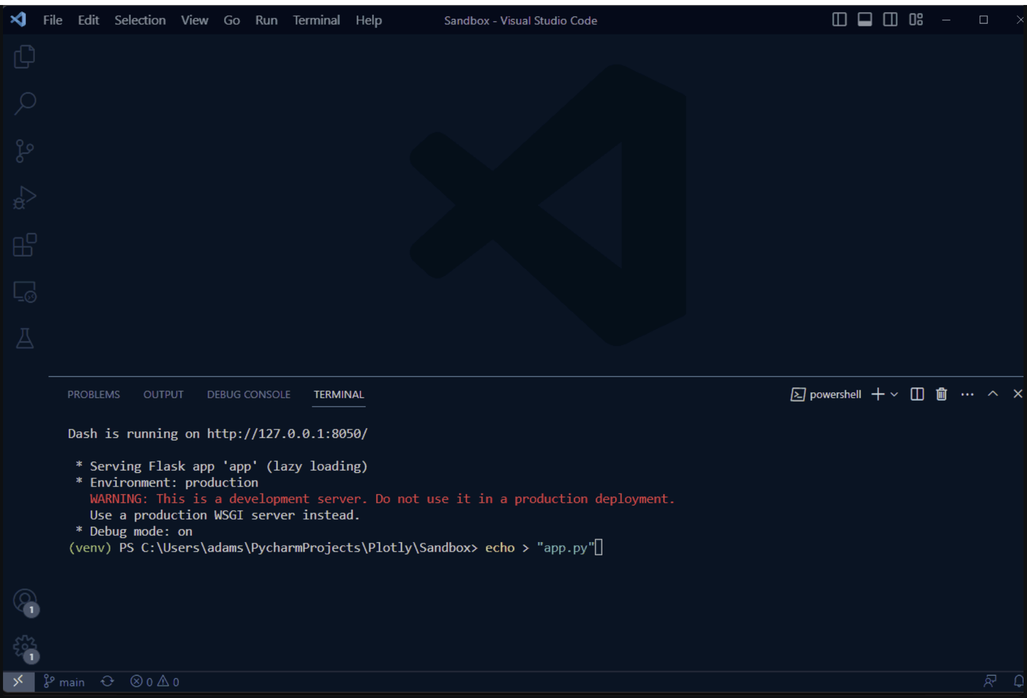
Alternatively, with Dash 2.11 or later, you can run this app and other examples from this documentation in a **Jupyter Notebook**.

The code below creates a very small "Hello World" Dash app.

```python
from dash import Dash, html

app = Dash()

# Requires Dash 2.17.0 or later
app.layout = [html.Div(children='Hello World')]

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

```
Hello World
```

Follow this example gif (using VS Code) if you are not sure how to set up the app:


File Edit Selection View Go Run Terminal Help
Sandbox - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
powershell
main

```
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
(venv) PS C:\Users\adams\PycharmProjects\Plotly\Sandbox> echo > "app.py"
```

Hello World: Code Breakdown

```
# Import packages
from dash import Dash, html
```

- When creating Dash apps, you will almost always use the import statement above. As you create more advanced Dash apps, you will import more packages.

```
# Initialize the app
app = Dash()
```

- This line is known as the Dash constructor and is responsible for initializing your app. It is almost always the same for any Dash app you create.

```
# App layout
app.layout = [html.Div(children='Hello World')]
```

- The app layout represents the app components that will be displayed in the web browser and here is provided as a `list`, though it could also be a Dash component. In this example, a single component was added to the list: an `html.Div`. The Div has a few properties, such as `children`, which we use to add text content to the page: "Hello World".

```
# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

- These lines are for running your app, and they are almost always the same for any Dash app you create.

Connecting to Data

There are many ways to add data to an app: APIs, external databases, local `.txt` files, JSON files, and more. In this example, we will highlight one of the most common ways of incorporating data from a CSV sheet.

Replace the `app.py` code from the previous section with the code below. Then, install pandas (`pip install pandas`) and launch the app.

```
# Import packages
from dash import Dash, html, dash_table
```

```
import pandas as pd

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10)
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

My First App with Data

country	pop	continent	lifeExp	gdpPercap
Afghanistan	31889923	Asia	43.828	974.5803384
Albania	3600523	Europe	76.423	5937.029525999999
Algeria	33333216	Africa	72.301	6223.367465
Angola	12420476	Africa	42.731	4797.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	708573	Asia	75.635	29796.04834
Bangladesh	150448339	Asia	64.062	1391.253792
Belgium	10392226	Europe	79.441	33692.60508

« ‹ 1 / 15 › »

Connect to Data: Code Breakdown

```
# Import packages
from dash import Dash, html, dash_table
import pandas as pd
```

- We import the `dash_table` module to display the data inside a Dash DataTable. We also import the pandas library to read the CSV sheet data.

```
# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')
```

- Here we read the CSV sheet into a pandas dataframe. This will make it easier to slice, filter, and inspect the data.
- If you prefer to use a CSV sheet that is on your computer (and not online), make sure to save it in the same folder that contains the `app.py` file. Then, update the line of code to: `df = pd.read_csv('NameOfYourFile.csv')`
- If you're using an Excel sheet, make sure to `pip install openpyxl`. Then, update the line of code to: `df = pd.read_excel('NameOfYourFile.xlsx', sheet_name='Sheet1')`

> **Tip**: You can read the **pandas docs** on reading data if your data is in a different format, or consider using another Python library if you are connecting to a specific database type or file format. For example, if you're considering using Databricks as a backend for your Dash app, you may review their Python documentation for recommendations on how to connect.

```
# App layout
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10)
]
```

- In addition to the app title, we add the DataTable component and read the pandas dataframe into the table.

Visualizing Data

The Plotly graphing library has more than **50 chart types** to choose from. In this example, we will make use of the histogram chart.

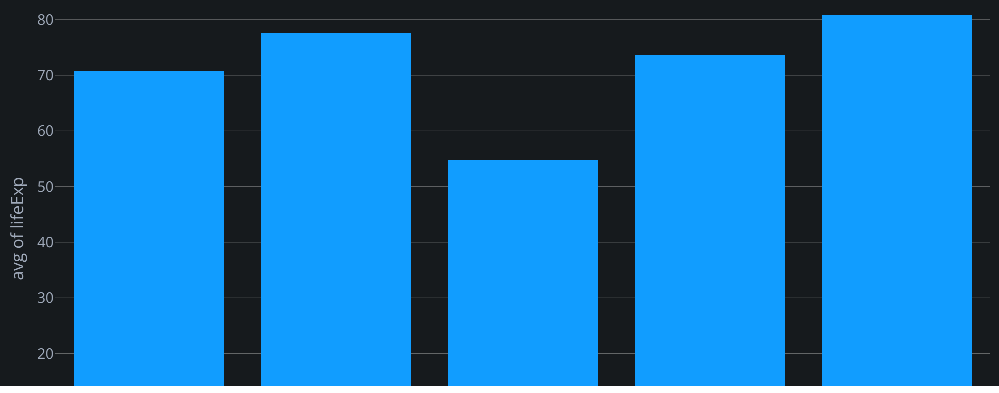
Replace the `app.py` code from the previous section with the code below.

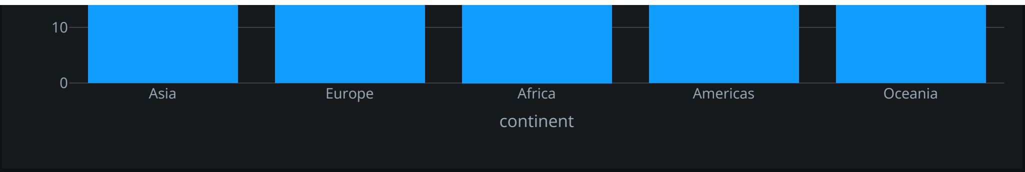
```
# Import packages
from dash import Dash, html, dash_table, dcc
import pandas as pd
import plotly.express as px

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='avg'))
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

My First App with Data and a Graph

country	pop	continent	lifeExp	gdpPercap
Afghanistan	31889923	Asia	43.828	974.5803384
Albania	3600523	Europe	76.423	5937.029525999999
Algeria	33333216	Africa	72.301	6223.367465
Angola	12420476	Africa	42.731	4797.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	708573	Asia	75.635	29796.04834
Bangladesh	150448339	Asia	64.062	1391.253792
Belgium	10392226	Europe	79.441	33692.60508

« ‹ 1 / 15 › »





Visualize Data: Code Breakdown

```
# Import packages
from dash import Dash, html, dash_table, dcc
import pandas as pd
import plotly.express as px
```

- We import the `dcc` module (DCC stands for Dash Core Components). This module includes a Graph component called `dcc.Graph`, which is used to render interactive graphs.
- We also import the `plotly.express` library to build the interactive graphs.

```
# App layout
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='avg'))
]
```

- Using the `plotly.express` library, we build the histogram chart and assign it to the `figure` property of the `dcc.Graph`. This displays the histogram in our app.

Controls and Callbacks

So far you have built a static app that displays tabular data and a graph. However, as you develop more sophisticated Dash apps, you will likely want to give the app user more freedom to interact with the app and explore the data in greater depth. To achieve that, you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the callback to create the interaction between the radio buttons and the histogram chart.

```
# Import packages
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'], value='lifeExp', id='controls-and-
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
]

# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph', component_property='figure'),
    Input(component_id='controls-and-radio-item', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig
```

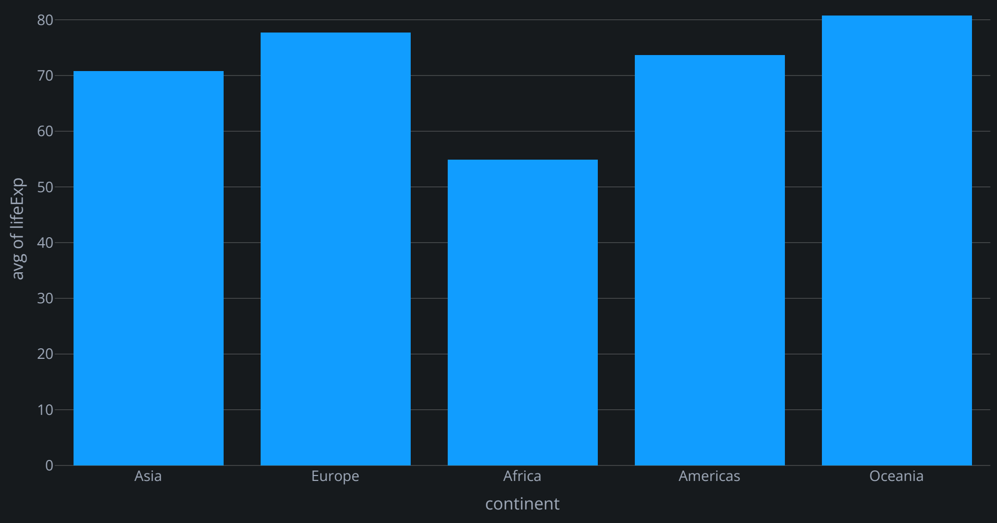
```python
# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```

My First App with Data, Graph, and Controls

( ) pop
(●) lifeExp
( ) gdpPercap

country	pop	continent	lifeExp	gdpPercap
Afghanistan	31889923	Asia	43.828	974.5803384
Albania	3600523	Europe	76.423	5937.029525999999
Algeria	33333216	Africa	72.301	6223.367465
Angola	12420476	Africa	42.731	4797.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995

« ‹ 1 / 24 › »



Controls: Code Breakdown

```python
# Import packages
from dash import Dash, html, dash_table, dcc, callback, Output, Input
```

- We import `dcc` like we did in the previous section to use `dcc.Graph`. In this example, we need `dcc` for `dcc.Graph` as well as the radio buttons component, `dcc.RadioItems`.
- To work with the callback in a Dash app, we import the `callback` module and the two arguments commonly used within the callback: `Output` and `Input`.

```python
# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'], value='lifeExp', id='controls-and-
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
]
```

- Notice that we add that `RadioItems` component to the layout, directly above the DataTable. There are three options, one for every radio button. The `lifeExp` option is assigned to the `value` property, making it the currently selected value.

- Both the `RadioItems` and the `Graph` components were given `id` names: these will be used by the callback to identify the components.

```
# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph', component_property='figure'),
    Input(component_id='controls-and-radio-item', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig
```

- The inputs and outputs of our app are the properties of a particular component. In this example, our input is the `value` property of the component that has the ID "controls-and-radio-item". If you look back at the layout, you will see that this is currently `lifeExp`. Our output is the `figure` property of the component with the ID "controls-and-graph", which is currently an empty dictionary (empty graph).
- The callback function's argument `col_chosen` refers to the component property of the input `lifeExp`. We build the histogram chart inside the callback function, assigning the chosen radio item to the y-axis attribute of the histogram. This means that every time the user selects a new radio item, the figure is rebuilt and the y-axis of the figure is updated.
- Finally, we return the histogram at the end of the function. This assigns the histogram to the `figure` property of the `dcc.Graph`, thus displaying the figure in the app.

Styling Your App

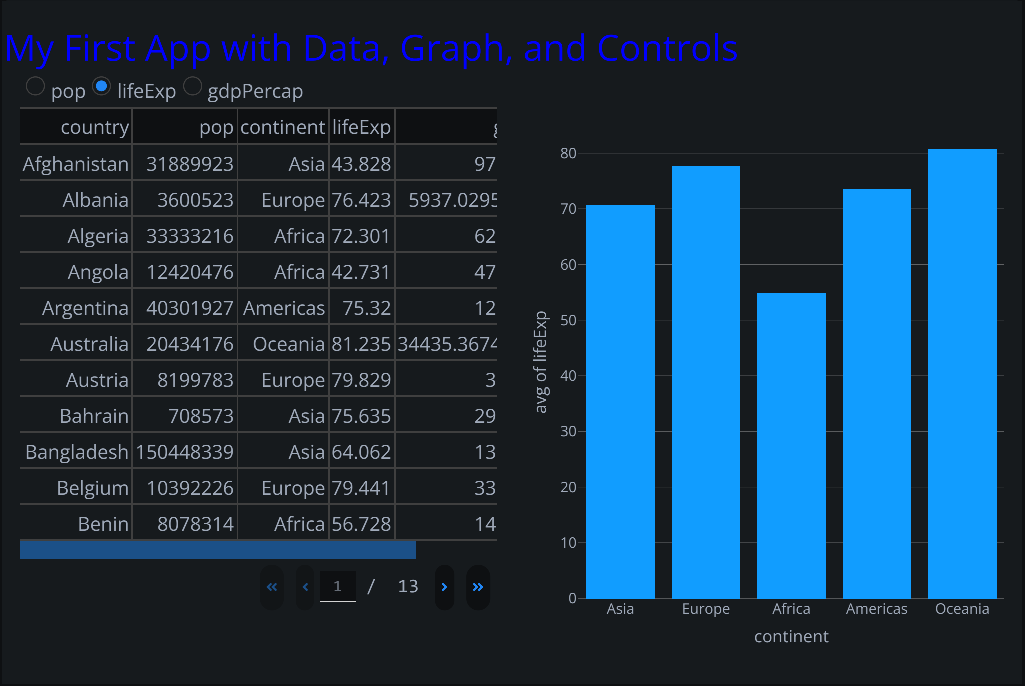
The examples in the previous section used Dash HTML Components to build a simple app layout, but you can style your app to look more professional. This section will give a brief overview of the multiple tools that you can use to enhance the layout style of a Dash app:

- HTML and CSS
- Dash Design Kit (DDK)
- Dash Bootstrap Components
- Dash Mantine Components

HTML and CSS

HTML and CSS are the lowest level of interface for rendering content on the web. The HTML is a set of components, and CSS is a set of styles applied to those components. CSS styles can be applied within components via the `style` property, or they can be defined as a separate CSS file in reference with the `className` property, as in the example below.

```
# Import packages
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app - incorporate css
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = Dash(external_stylesheets=external_stylesheets)

# App layout
app.layout = [
    html.Div(className='row', children='My First App with Data, Graph, and Controls',
             style={'textAlign': 'center', 'color': 'blue', 'fontSize': 30}),

    html.Div(className='row', children=[
        dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'],
                       value='lifeExp',
                       inline=True,
                       id='my-radio-buttons-final')
    ]),
    html.Div(className='row', children=[
        html.Div(className='six columns', children=[
            dash_table.DataTable(data=df.to_dict('records'), page_size=11, style_table={'overfl
        ]),
        html.Div(className='six columns', children=[
            dcc.Graph(figure={}, id='histo-chart-final')
        ])
    ])
]

# Add controls to build the interaction
@callback(
    Output(component_id='histo-chart-final', component_property='figure'),
    Input(component_id='my-radio-buttons-final', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```



My First App with Data, Graph, and Controls

pop [✓ lifeExp] gdpPercap

country	pop	continent	lifeExp	g
Afghanistan	31889923	Asia	43.828	97
Albania	3600523	Europe	76.423	5937.0295
Algeria	33333216	Africa	72.301	62
Angola	12420476	Africa	42.731	47
Argentina	40301927	Americas	75.32	12
Australia	20434176	Oceania	81.235	34435.3674
Austria	8199783	Europe	79.829	3
Bahrain	708573	Asia	75.635	29
Bangladesh	150448339	Asia	64.062	13
Belgium	10392226	Europe	79.441	33
Benin	8078314	Africa	56.728	14

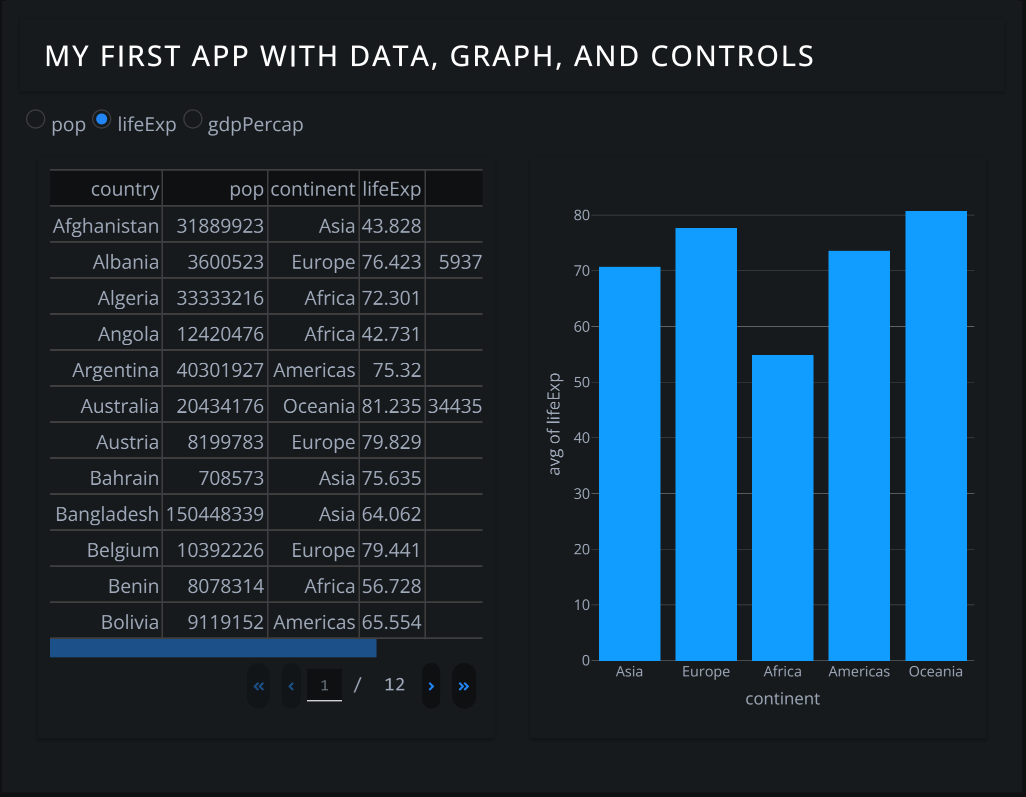
Dash Design Kit (DDK)

Dash Design Kit is our high level UI framework that is purpose-built for Dash. With Dash Design Kit, you don't need to use HTML or CSS. Apps are mobile responsive by default and everything is themeable. Dash Design Kit is licensed as part of Dash Enterprise and officially supported by Plotly.

Here's an example of what you can do with Dash Design Kit (note that you won't be able to run this example without a Dash Enterprise license).

```
# Import packages
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px
import dash_design_kit as ddk

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app
app = Dash()

# App layout
```

```
app.layout = ddk.App([
    ddk.Header(ddk.Title('My First App with Data, Graph, and Controls')),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'],
                    value='lifeExp',
                    inline=True,
                    id='my-ddk-radio-items-final'),
    ddk.Row([
        ddk.Card([
            dash_table.DataTable(data=df.to_dict('records'), page_size=12, style_table={'overf
        ], width=50),
        ddk.Card([
            ddk.Graph(figure={}, id='graph-placeholder-ddk-final')
        ], width=50),
    ]),

])

# Add controls to build the interaction
@callback(
    Output(component_id='graph-placeholder-ddk-final', component_property='figure'),
    Input(component_id='my-ddk-radio-items-final', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```



MY FIRST APP WITH DATA, GRAPH, AND CONTROLS

pop [✓ lifeExp] gdpPercap

country	pop	continent	lifeExp	
Afghanistan	31889923	Asia	43.828	
Albania	3600523	Europe	76.423	5937
Algeria	33333216	Africa	72.301	
Angola	12420476	Africa	42.731	
Argentina	40301927	Americas	75.32	
Australia	20434176	Oceania	81.235	34435
Austria	8199783	Europe	79.829	
Bahrain	708573	Asia	75.635	
Bangladesh	150448339	Asia	64.062	
Belgium	10392226	Europe	79.441	
Benin	8078314	Africa	56.728	
Bolivia	9119152	Americas	65.554	

« ‹ 1 / 12 › »

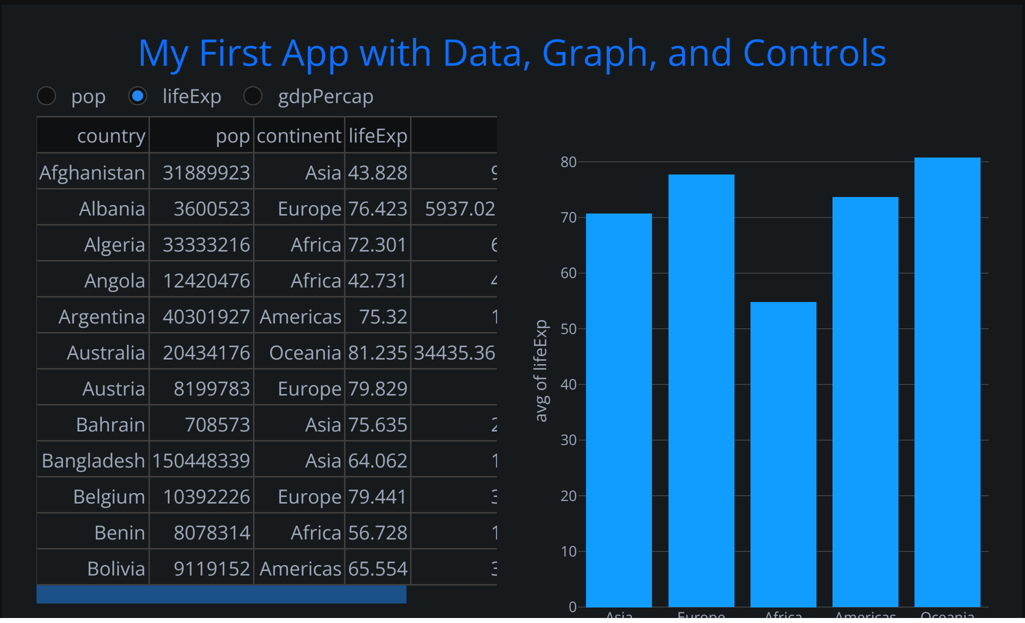
Dash Bootstrap Components

Dash Bootstrap is a community-maintained library built off of the bootstrap component system. Although it is not officially maintained or supported by Plotly, Dash Bootstrap is a powerful way of building elegant app layouts. Notice that we first define a row and then the width of columns inside the row, using the `dbc.Row` and `dbc.Col` components.

For the app below to run successfully, make sure to install the Dash Bootstrap Components library: `pip install dash-bootstrap-components`

> Read more about the Dash Bootstrap Components in the **third-party documentation**.

```python
# Import packages
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px
import dash_bootstrap_components as dbc

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Initialize the app - incorporate a Dash Bootstrap theme
external_stylesheets = [dbc.themes.CERULEAN]
app = Dash(__name__, external_stylesheets=external_stylesheets)

# App layout
app.layout = dbc.Container([
    dbc.Row([
        html.Div('My First App with Data, Graph, and Controls', className="text-primary text-ce
    ]),

    dbc.Row([
        dbc.RadioItems(options=[{"label": x, "value": x} for x in ['pop', 'lifeExp', 'gdpPercap
                       value='lifeExp',
                       inline=True,
                       id='radio-buttons-final')
    ]),

    dbc.Row([
        dbc.Col([
            dash_table.DataTable(data=df.to_dict('records'), page_size=12, style_table={'overfl
        ], width=6),

        dbc.Col([
            dcc.Graph(figure={}, id='my-first-graph-final')
        ], width=6),
    ]),

], fluid=True)

# Add controls to build the interaction
@callback(
    Output(component_id='my-first-graph-final', component_property='figure'),
    Input(component_id='radio-buttons-final', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True)
```



My First App with Data, Graph, and Controls

pop [✓ lifeExp] gdpPercap

country	pop	continent	lifeExp	
Afghanistan	31889923	Asia	43.828	9
Albania	3600523	Europe	76.423	5937.02
Algeria	33333216	Africa	72.301	6
Angola	12420476	Africa	42.731	4
Argentina	40301927	Americas	75.32	1
Australia	20434176	Oceania	81.235	34435.36
Austria	8199783	Europe	79.829	
Bahrain	708573	Asia	75.635	2
Bangladesh	150448339	Asia	64.062	1
Belgium	10392226	Europe	79.441	3
Benin	8078314	Africa	56.728	1
Bolivia	9119152	Americas	65.554	3

« ‹ 1 / 12 › »

continent

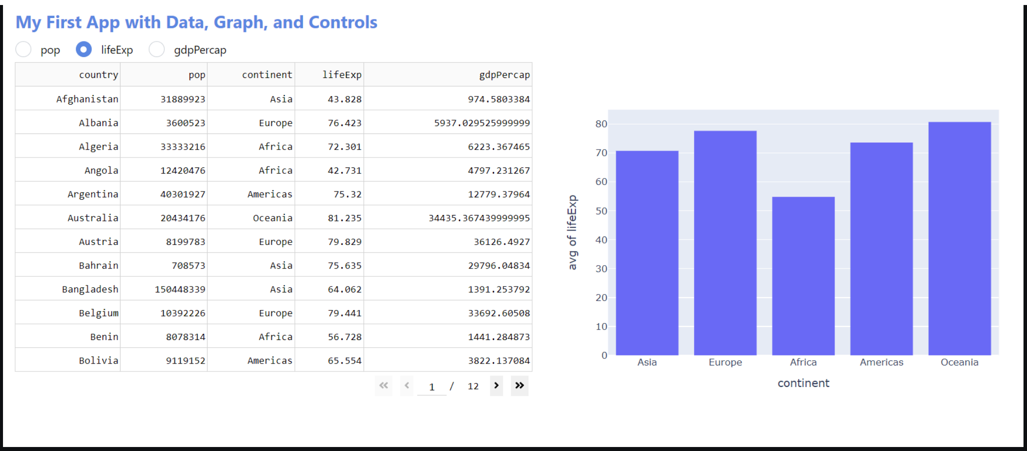
Dash Mantine Components

Dash Mantine is a community-maintained library built off of the Mantine component system. Although it is not officially maintained or supported by the Plotly team, Dash Mantine is another powerful way of customizing app layouts. The Dash Mantine Components uses the Grid module to structure the layout. Instead of defining a row, we define a `dmc.Grid`, within which we insert `dmc.Col`s and define their width by assigning a number to the `span` property.

For the app below to run successfully, make sure to install the Dash Mantine Components library: `pip install dash-mantine-components==0.12.1`

> Read more about the Dash Mantine Components in the **third-party documentation**.

```
from dash import Dash, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px
import dash_mantine_components as dmc

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

app = Dash()

app.layout = dmc.Container([
    dmc.Title('My First App with Data, Graph, and Controls', color="blue", size="h3"),
    dmc.RadioGroup(
            [dmc.Radio(i, value=i) for i in  ['pop', 'lifeExp', 'gdpPercap']],
            id='my-dmc-radio-item',
            value='lifeExp',
            size="sm"
        ),
    dmc.Grid([
        dmc.Col([
            dash_table.DataTable(data=df.to_dict('records'), page_size=12, style_table={'overfl
        ], span=6),
        dmc.Col([
            dcc.Graph(figure={}, id='graph-placeholder')
        ], span=6),
    ]),

], fluid=True)

@callback(
    Output(component_id='graph-placeholder', component_property='figure'),
    Input(component_id='my-dmc-radio-item', component_property='value')
)
def update_graph(col_chosen):
    fig = px.histogram(df, x='continent', y=col_chosen, histfunc='avg')
    return fig

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



My First App with Data, Graph, and Controls

pop [✓ lifeExp] gdpPercap

country	pop	continent	lifeExp	gdpPercap
Afghanistan	31889923	Asia	43.828	974.5803384
Albania	3600523	Europe	76.423	5937.029525999999
Algeria	33333216	Africa	72.301	6223.367465
Angola	12420476	Africa	42.731	4797.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	708573	Asia	75.635	29796.04834
Bangladesh	150448339	Asia	64.062	1391.253792
Belgium	10392226	Europe	79.441	33692.60508
Benin	8078314	Africa	56.728	1441.284873
Bolivia	9119152	Americas	65.554	3822.137084

Next Steps

Congratulations! You have learned to build and style your first Dash app. This is the first step of an exciting Dash learning path that will give you the skills and tools to build sophisticated Python analytics apps.

To continue your learning journey, we recommend you check out the following resources:

- **100 simple examples** of Dash components interacting with Plotly graphs
- **Dash Core Components**

Write, deploy, and scale Dash apps on Dash Enterprise.

Learn More | **Pricing** | **Dash Enterprise Demo** | **Dash Enterprise Overview**

Dash Python ***> Dash in 20 Minutes Tutorial***

Products

Dash

Consulting and Training

Pricing

Enterprise Pricing

About Us

Careers

Resources

Blog

Support

Community Support

Graphing Documentation

Join our mailing list

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

SUBSCRIBE

Terms of Service Privacy Policy