STAT3622 Data Visualization (with Python)

Lecture 9

Wenbin Du
The University of Hong Kong

29 March 2021

Dash

Dash is a productive Python framework for building web analytic applications.

Dash is ideal for building data visualization apps with highly custom user interfaces in pure Python. It's particularly suited for anyone who works with data in Python.

Dash is simple enough that you can bind a user interface around your Python code in an afternoon.

Dash apps are rendered in the web browser. You can deploy your apps to servers and then share them through URLs. Since Dash apps are viewed in the web browser, Dash is inherently cross-platform and mobile ready.

Dash is an open source library, released under the permissive MIT license. Plotly develops Dash and offers a platform for managing Dash apps in an enterprise environment. Dash may be installed using pip:

- $ pip install dash

Jupyter dash

Jupyter dash may be installed using pip:

- pip install jupyter-dash

The `jupyter-dash` package makes it easy to develop Plotly Dash apps from the Jupyter Notebook and JupyterLab.

Just replace the standard `dash.Dash` class with the `jupyter_dash.JupyterDash` subclass.

```python
from jupyter_dash import JupyterDash
#app = dash.Dash(__name__)
app = JupyterDash(__name__)
```

https://medium.com/plotly/introducing-jupyterdash-811f1f57c02e

JupyterDash

By default, `run_server` displays a URL that you can click on to open the app in a browser tab. The `mode` argument to `run_server` can be used to change this behavior. Setting `mode="inline"` will display the app directly in the notebook output cell.

```
# In jupyter notebook you can set the mode to "inline"
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
```

```
app._terminate_server_for_port("localhost", 8050)
# You can use this code to terminate the server in jupyter notebook
```

JupyterDash is not recommended because the version might be old.

Dash apps

Dash apps are composed of two parts.

- The first part is the "layout" of the app and it describes what the application looks like.
- The second part is the "callback" and it describes the interactivity of the application.

Dash provides Python classes for all of the visual components of the application. The components are maintained in the dash_core_components and the dash_html_components library but you can also build your own with JavaScript and React.js.

Note: Python code examples are meant to be executed using `python app.py`. You can also use Jupyter with the JupyterDash library.

Dash apps

Run the code in the commond line or using pycharm

```
$ python dash-iris-callback.py
```

```
...Running on http://127.0.0.1:8050/ (Press CTRL+C to quit)
```

Dash Layout

- Dash HTML Components

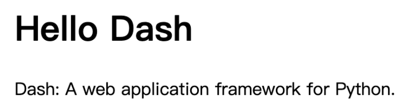
```
app = JupyterDash(__name__)
app.layout = html.Div(
                children=[html.H1(children='Hello Dash'),]
                )
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
# Address already in use-->
# lsof -i:8050
# kill pid_num
```

Hello Dash

Dash Layout

- Dash HTML Components

```
app = JupyterDash(__name__)
app.layout = html.Div(children=[
                  html.H1(children='Hello Dash'),
                  html.Div(children='''
                      Dash: A web application framework for Python.
                  '''),
])
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
```



Hello Dash

Dash: A web application framework for Python.

Dash Layout

- Dash HTML Components

```
html.Div()
```

> A Div component. Div is a wrapper for the `<div>` HTML5 element. For detailed attribute info see: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div

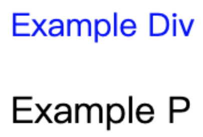
Keyword arguments:

> - children (a list of or a singular dash component, string or number; optional): The children of this component
> - id (string; optional): The ID of this component, used to identify dash components in callbacks. The ID needs to be unique across all of the components in an app.
> - n_clicks (number; default 0): An integer that represents the number of times that this element has been clicked on.
> - ...

Dash Layout

- Dash HTML Components

```
app = JupyterDash(__name__)
app.layout = html.Div([
                  html.Div('Example Div', style={'color': 'blue', 'fontSize': 14}),
                  html.P('Example P', className='my-class', id='my-p-element')
                  ], style={'marginBottom': 50, 'marginTop': 25})
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
```



Example Div

Example P

Other HTML Components: https://dash.plotly.com/dash-html-components

Dash Layout

- Dash HTML Components

If you're using HTML components, then you also have access to properties like style, class, and id. All of these attributes are available in the Python classes.

The HTML elements and Dash classes are mostly the same but there are a few key differences:

> - The style property is a dictionary
> - Properties in the style dictionary are camelCased
> - The class key is renamed as className
> - Style properties in pixel units can be supplied as just numbers without the px unit

Dash Layout

- Dash HTML Components

```
import dash_html_components as html

html.Div([
        html.Div('Example Div', style={'color': 'blue', 'fontSize': 14}),
        html.P('Example P', className='my-class', id='my-p-element')
], style={'marginBottom': 50, 'marginTop': 25})
```

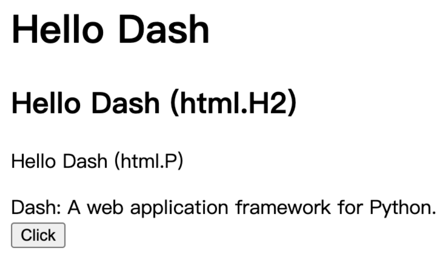
That Dash code will render this HTML markup:

```
<div style="margin-bottom: 50px; margin-top: 25px;">
    <div style="color: blue; font-size: 14px">
        Example Div
    </div>
    <p class="my-class", id="my-p-element">
        Example P
    </p>
</div>
```

Dash Layout

- Dash HTML Components

```
app = JupyterDash(__name__)
app.layout = html.Div(children=[
                          html.H1(children='Hello Dash'),
                          html.H2(children='Hello Dash (html.H2)'),
                          html.P(children='Hello Dash (html.P)'),
                          html.Div(children='''
                              Dash: A web application framework for Python.
                          '''),
                          html.Button(children='Click')
])
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
```


Hello Dash

Hello Dash (html.H2)

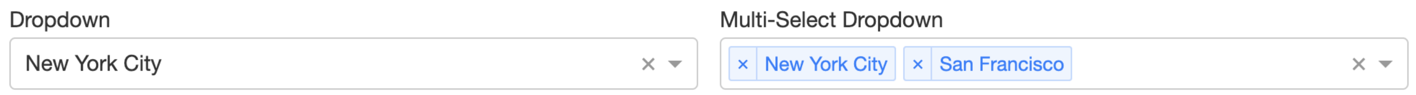
Hello Dash (html.P)

Dash: A web application framework for Python.

Dash Layout

- Dash Core Components

```
app.layout = html.Div([html.Label('Dropdown'),
                  dcc.Dropdown(options=[
                          {'label': 'New York City', 'value': 'NYC'},
                          {'label': 'San Francisco', 'value': 'SF'}
                          ],
                          value='NYC' ),
                  html.Label('Multi-Select Dropdown'),
                  dcc.Dropdown(options=[
                          {'label': 'New York City', 'value': 'NYC'},
                          {'label': 'San Francisco', 'value': 'SF'}
                          ],
                          value=['NYC', 'SF'],
                          multi=True),
                ], style={'columnCount': 2})
```



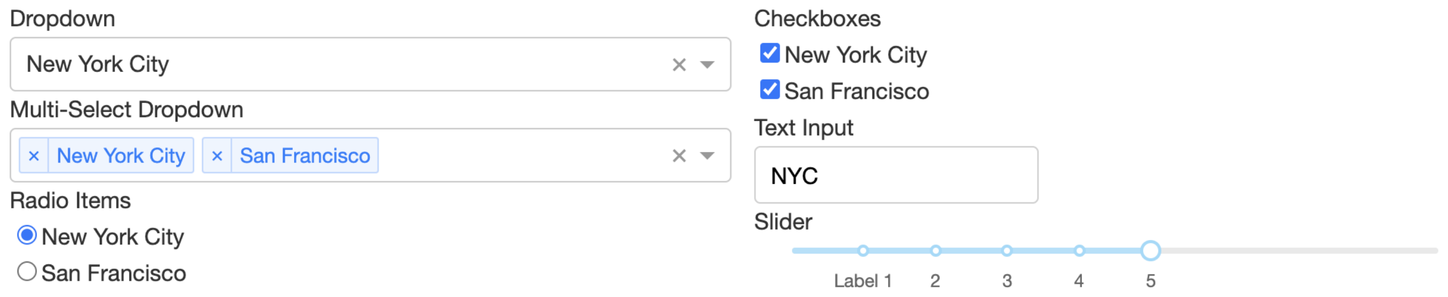
Dash Layout

- Dash Core Components

```
app.layout = html.Div([html.Label('Dropdown'),
                  dcc.Dropdown(options=[{'label': 'New York City', 'value': 'NYC'},
                          {'label': 'San Francisco', 'value': 'SF'}
                      ],value='NYC' ),
                  html.Label('Multi-Select Dropdown'),
                  dcc.Dropdown(options=[{'label': 'New York City', 'value': 'NYC'},
                          {'label': 'San Francisco', 'value': 'SF'}
                      ], value=['NYC', 'SF'], multi=True),
                  html.Label('Radio Items'),
                  dcc.RadioItems(options=[{'label': 'New York City', 'value': 'NYC'},
                          {'label': 'San Francisco', 'value': 'SF'}
                      ], value='NYC'),
                  html.Label('Checkboxes'),
                  dcc.Checklist(options=[{'label': 'New York City', 'value': 'NYC'},
                          {'label': 'San Francisco', 'value': 'SF'}],value=['NYC', 'SF']),
                  html.Label('Text Input'),
                  dcc.Input(value='NYC', type='text'),
                  html.Label('Slider'),
                  dcc.Slider(min=0,  max=9,
                      marks={i: 'Label {}'.format(i) if i == 1 else str(i) for i
                      in range(1, 6)},value=5,),
              ], style={'columnCount': 2})
```

Dash Layout

- Dash Core Components



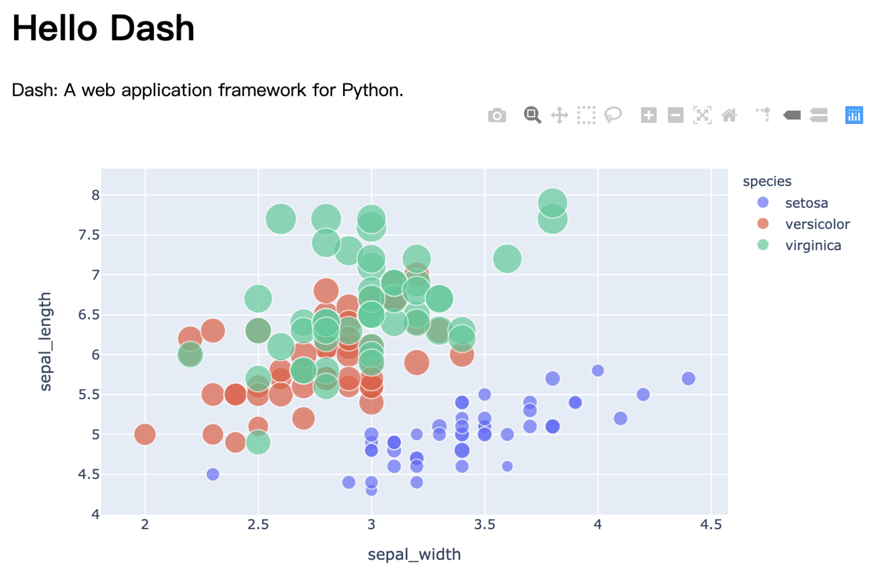
Dash Layout

- Dash Core Components

```
app = JupyterDash(__name__)
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species",
                 size='petal_length', hover_data=['petal_width'])
app.layout = html.Div(children=[
                    html.H1(children='Hello Dash'),
                    html.Div(children='''
                        Dash: A web application framework for Python.
                    '''),
                     dcc.Graph(
                        id='example-graph',
                        figure=fig
                    )
])
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
```

Dash Layout

- Dash Core Components



Hello Dash

Dash: A web application framework for Python.

Dash Layout

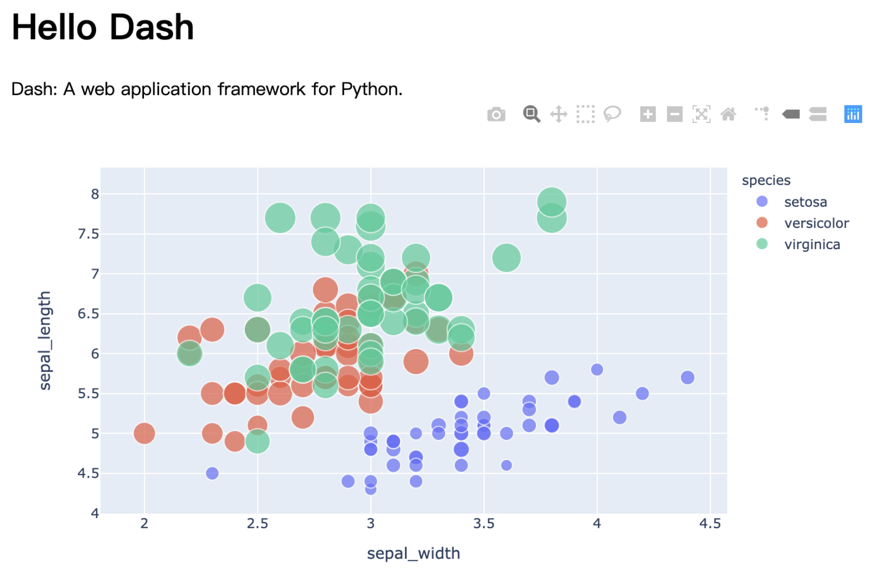
CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media.

- external_stylesheets: Additional CSS files to load with the page.

```
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
#app = JupyterDash(__name__)
app = JupyterDash(__name__,external_stylesheets=external_stylesheets)
df = px.data.iris()
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species",
                 size='petal_length', hover_data=['petal_width'])
app.layout = html.Div(children=[
                      html.H1(children='Hello Dash'),
                      html.Div(children='''
                          Dash: A web application framework for Python.
                      '''),
                      dcc.Graph(
                          id='example-graph',
                          figure=fig
                      )
])
app.run_server(mode="inline",port=int(os.getenv('PORT', '4444')))
```

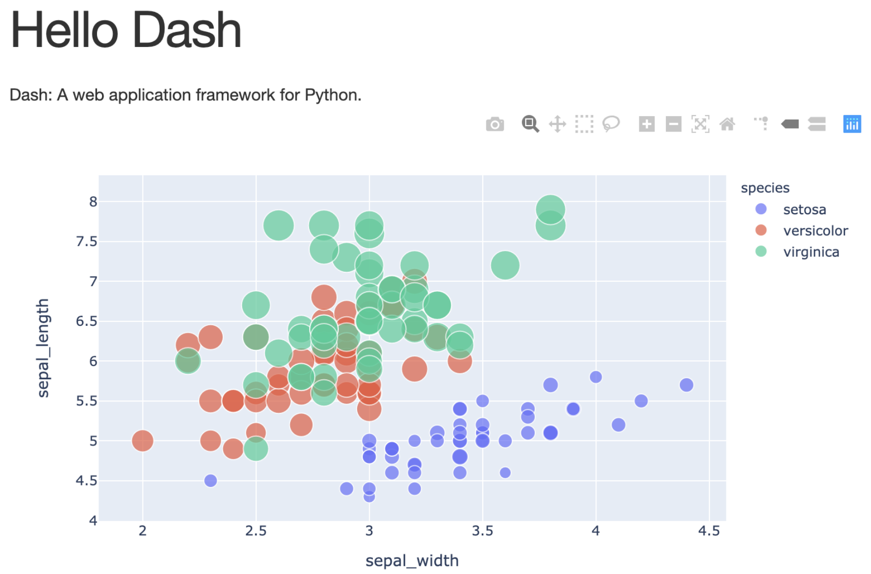
Dash Layout

- external_stylesheets



Hello Dash

Dash: A web application framework for Python.



Hello Dash

Dash: A web application framework for Python.

Dash Callbacks

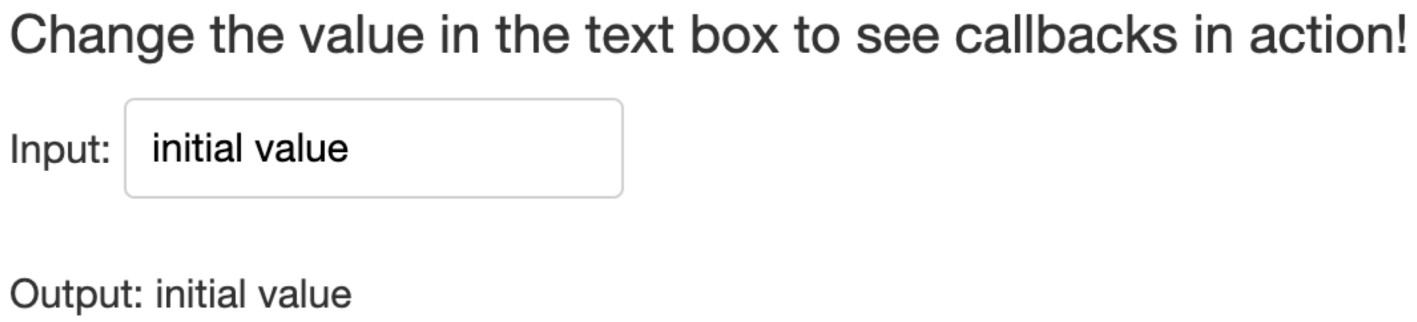
```python
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),
])
@app.callback(
     Output(component_id='my-output', component_property='children'),
     Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)
```

Dash Callbacks


Change the value in the text box to see callbacks in action!

Input: initial value

Output: initial value

Dash Callbacks

> In Dash, the inputs and outputs of our application are simply the properties of a particular component. In this example, our input is the "value" property of the component that has the ID "my-input". Our output is the "children" property of the component with the ID "my-output".
>
> Whenever an input property changes, the function that the callback decorator wraps will get called automatically. Dash provides the function with the new value of the input property as an input argument and Dash updates the property of the output component with whatever was returned by the function.
>
> The component_id and component_property keywords are optional (there are only two arguments for each of those objects). They are included in this example for clarity and can be omitted for the sake of brevity and readability.

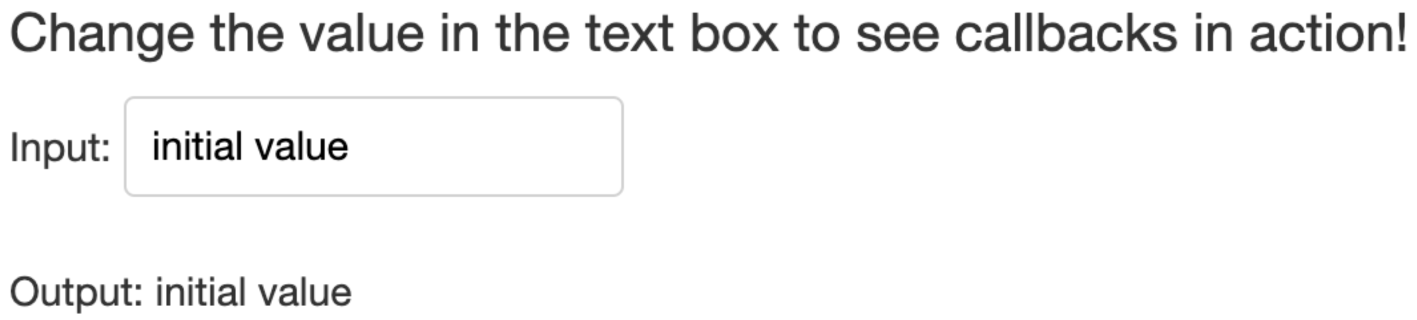
@app.callback decorator

```
app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),
])
@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)
```

a. By writing this decorator, we're telling Dash to call this function for us whenever the value of the "input" component (the text box) changes in order to update the `'children'` of the "output" component on the page (the HTML div).

b. You can use any name for the function that is wrapped by the @app.callback decorator. The convention is that the name describes the callback output(s).

Dash Callbacks


Change the value in the text box to see callbacks in action!

Input: initial value

Output: initial value

@app.callback decorator

```
app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),
])
@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)
```

c. You can use any name for the function arguments, but you must use the same names inside the callback function as you do in its definition, just like in a regular Python function. The arguments are positional: first the Input items and then any State items are given in the same order as in the decorator.

d. You must use the same id you gave a Dash component in the app.layout when referring to it as either an input or output of the @app.callback decorator.

@app.callback decorator

```python
app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div(["Input: ",
              dcc.Input(id='my-input', value='initial value', type='text')]),
    html.Br(),
    html.Div(id='my-output'),
])
@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)
```

e. The @app.callback decorator needs to be directly above the callback function declaration. If there is a blank line between the decorator and the function definition, the callback registration will not be successful.

Learn more about using the `@app.callback` decorator
https://stackoverflow.com/questions/739654/how-to-make-function-decorators-and-chain-

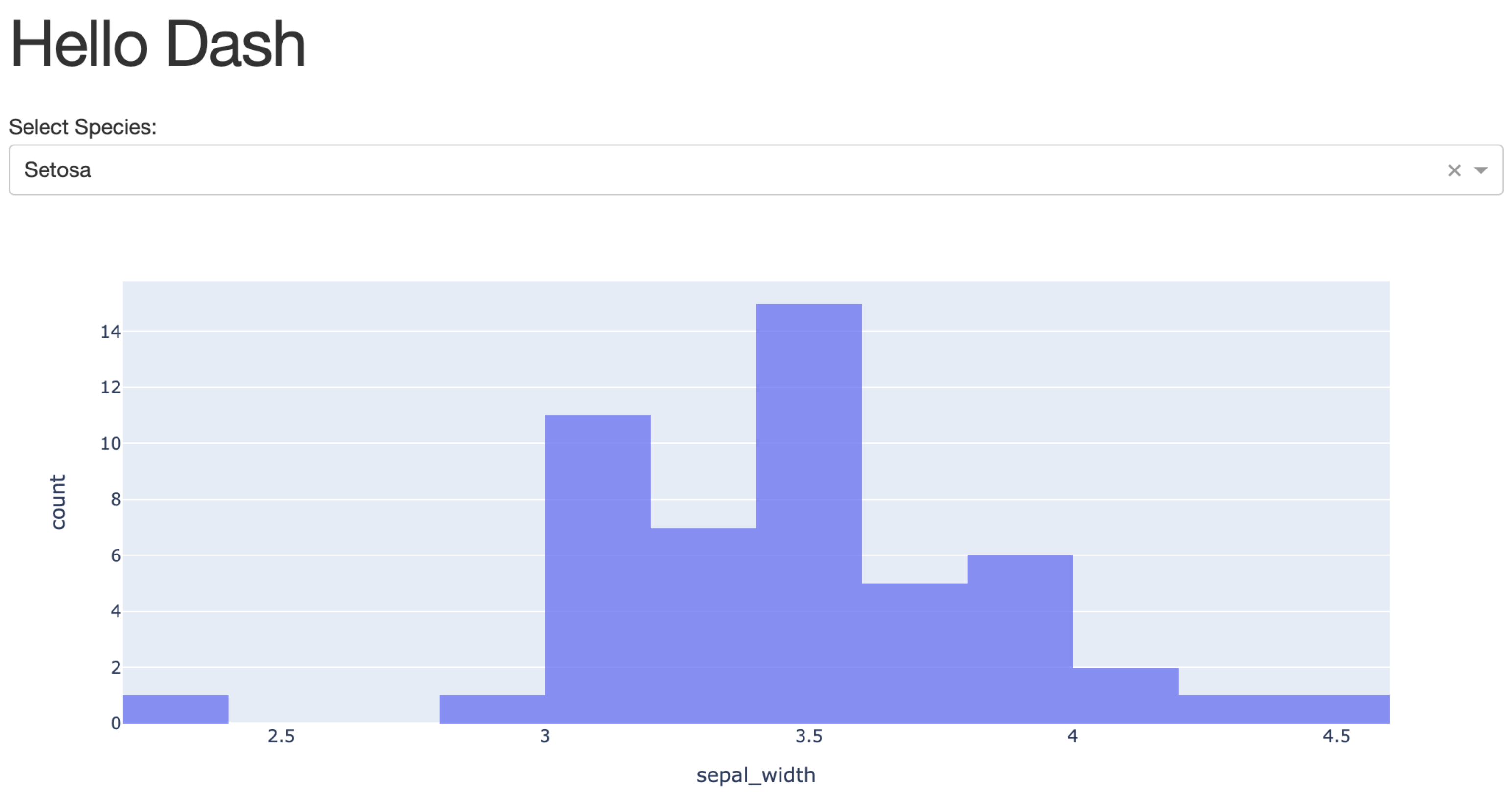
Dash Callbacks

```
import dash,os
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
import pandas as pd
import dash.dependencies as dependencies
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
df = px.data.iris()
app.layout = html.Div(children=[
                      html.H1(children='Hello Dash'),
                      html.Div(children=[
                              html.Label('Select Species:'),
                              dcc.Dropdown(
                                          id='select-species',
                                           options=[{'label': 'Setosa', 'value': 'setosa'},
                                          {'label': 'Versicolor', 'value': 'versicolor'},
                                          {'label':'Virginica','value':'virginica'}
                                          ],value='setosa')]),
                      dcc.Graph(
                           id='iris-graph',
                            figure={})],)
```

Dash Callbacks

```
@app.callback(
     dependencies.Output(component_id='iris-graph',component_property='figure'),
     dependencies.Input(component_id='select-species', component_property='value')
)
def update_fig(species):
    filtered_df = df[df['species']==species]
    fig = px.histogram(filtered_df, x="sepal_width", hover_data=['petal_width'],
          opacity=0.75, barmode='overlay')
    return fig
if __name__ == '__main__':
    app.run_server(debug=False,port=int(os.getenv('PORT', '4544')))
```

Dash Callbacks



Hello Dash

Select Species:

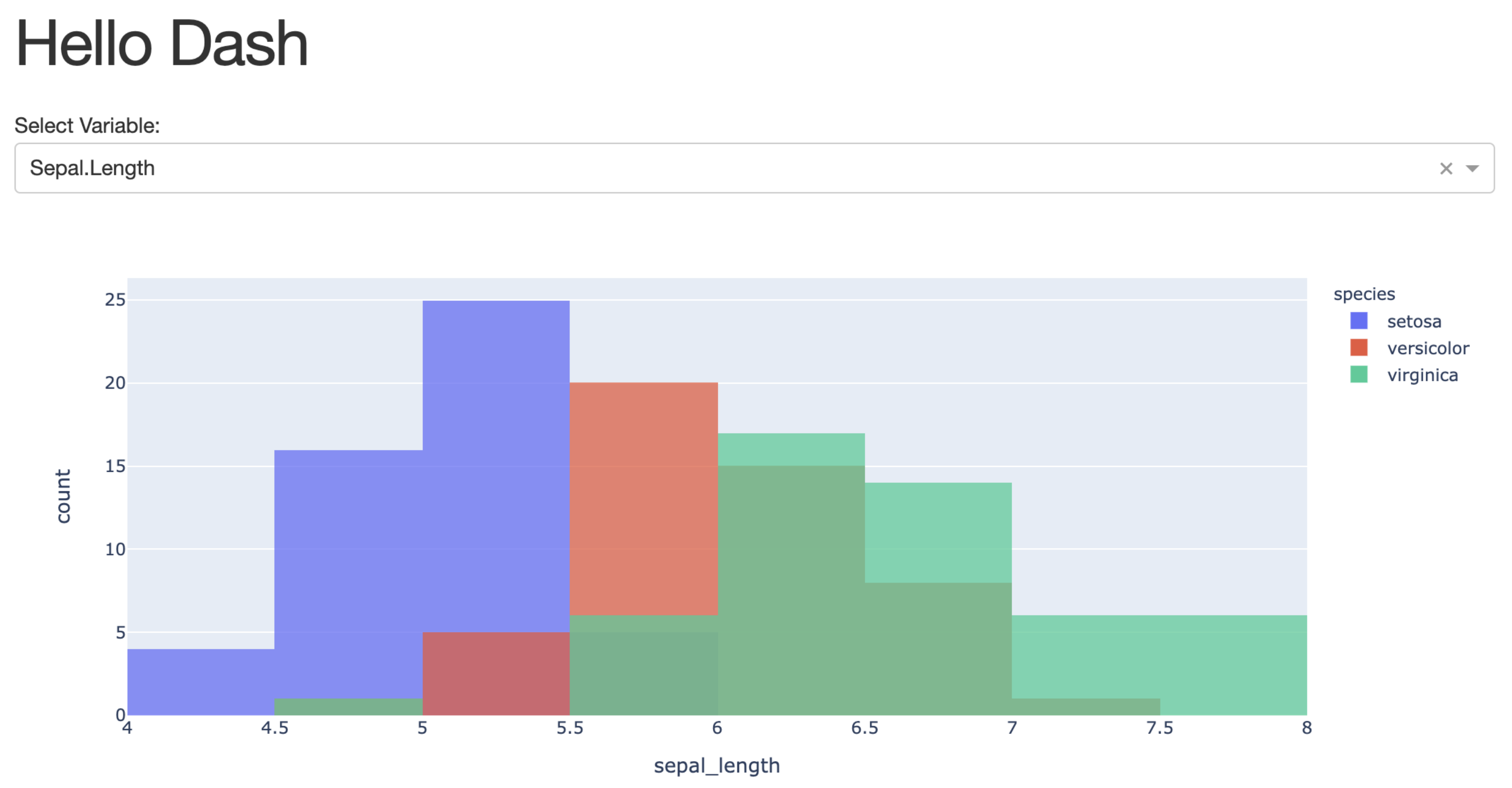
Setosa

Dash Callbacks

```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),
    html.Div(
        children=[
            html.Label('Select Variable:'),
             dcc.Dropdown(id='select-variable',
                         options=[{'label':'Sepal.Length','value':'sepal_length'},
                                {'label':'Sepal.Width','value':'sepal_width'}
                        ],
                         value='sepal_length')]),
    dcc.Graph(
        id='iris-graph',
        figure={})
],)

@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
     dependencies.Input(component_id='select-variable', component_property='value')
)
def update_fig(iris_variable):
  fig = px.histogram(df, x=iris_variable, color='species',opacity=0.75, barmode='overlay')
    return fig
```

Dash Callbacks



Hello Dash

Select Variable:

Sepal.Length

Dash App with Multiple Inputs

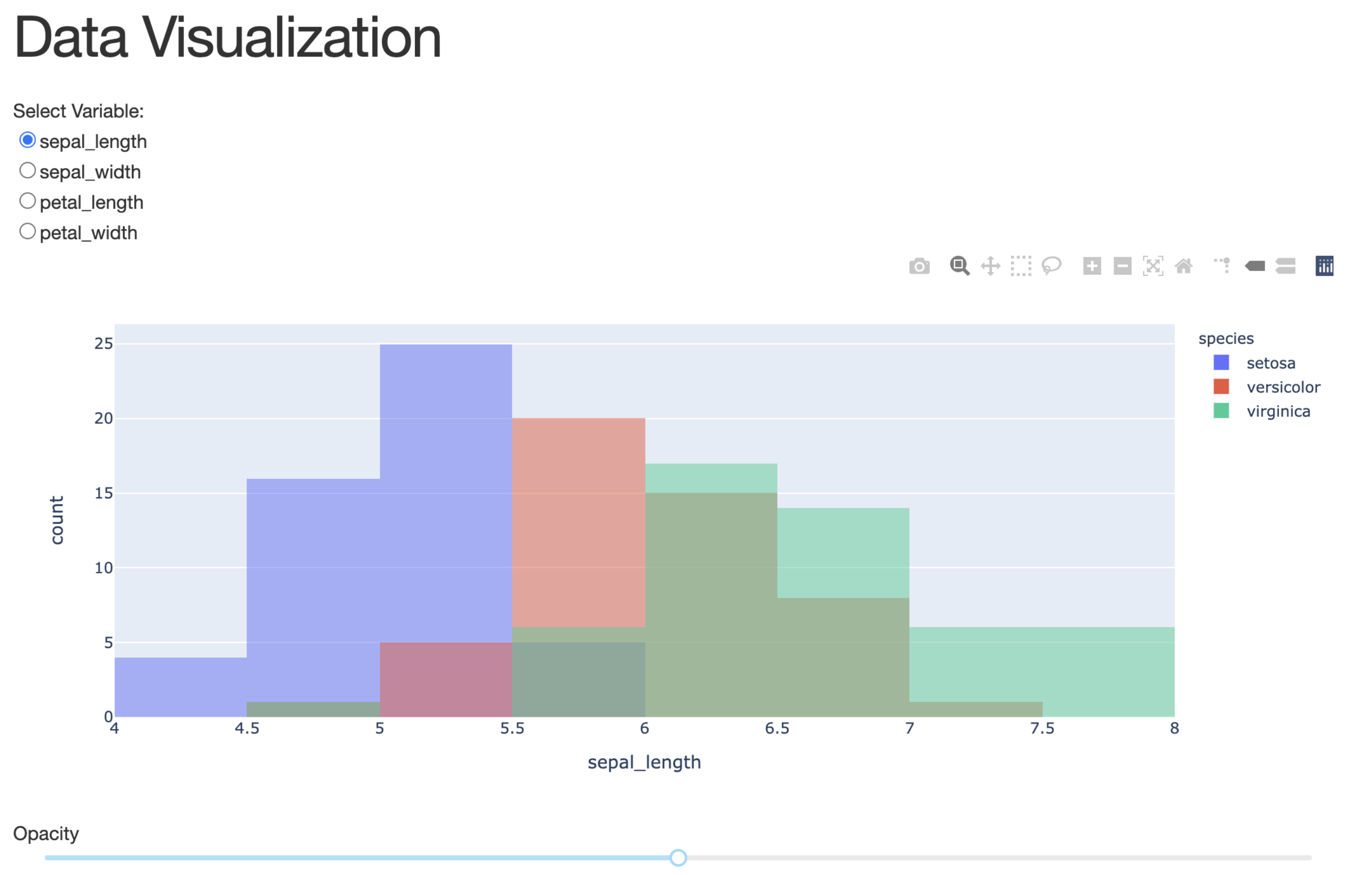
```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(children='Data Visualization'),
    html.Div(
        children=[
            html.Label('Select Variable:'),
            dcc.RadioItems(id='select-variable',
                          options=[{'label': var, 'value': var} for var in
                          ['sepal_length','sepal_width','petal_length','petal_width']
                         ],
                          value='sepal_length')]),
    dcc.Graph(id='iris-graph',
              figure={}),
    html.Label('Opacity'),
     dcc.Slider(id='select-opacity',min=0,  max=1,step=0.1,
              value=0.5,),
],)
```

Dash App with Multiple Inputs

```
@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
     dependencies.Input(component_id='select-variable', component_property='value'),
     dependencies.Input(component_id='select-opacity', component_property='value')
)
def update_fig(iris_variable,opacity):
    fig = px.histogram(df, x=iris_variable, color='species',opacity=opacity, barmode='overlay
    return fig
```

Note: in jupyter dash, you may need to put the Inputs in square brackets `[ ]`.

Dash App with Multiple Inputs



Data Visualization

Select Variable:

- [✓ sepal_length]
- sepal_width
- petal_length
- petal_width

Opacity

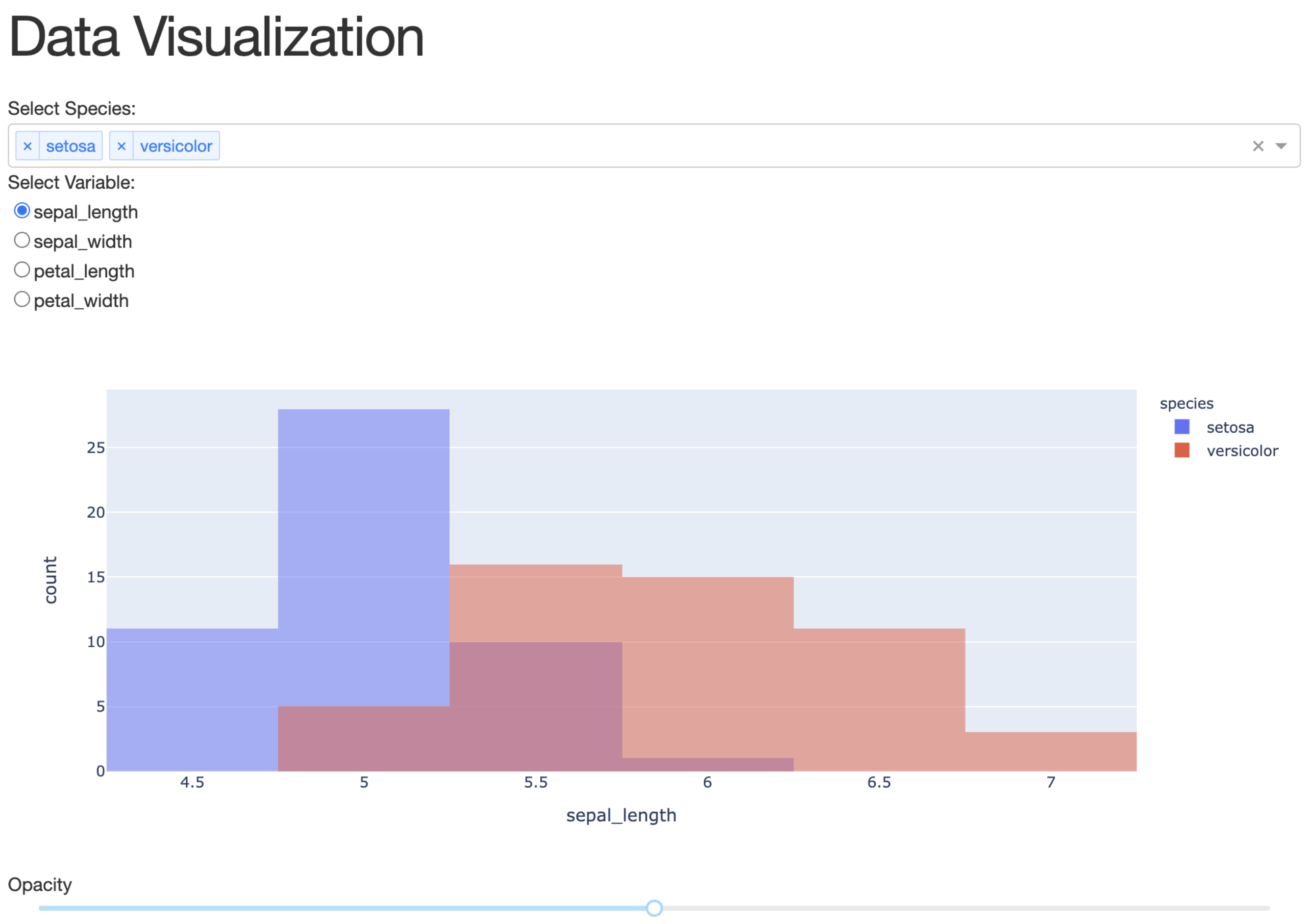
Dash App with Multiple Inputs

```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(children='Data Visualization'),
    html.Div(
        children=[
            html.Label('Select Species:'),
            dcc.Dropdown(id='select-species',
                         options=[{'label': var, 'value': var} for var in df.species.unique()
                         value=['setosa','versicolor'],
                          multi=True),
            html.Label('Select Variable:'),
            dcc.RadioItems(id='select-variable',
                         options=
                         [{'label': var, 'value': var} for var in
                         ['sepal_length','sepal_width','petal_length','petal_width']
                        ],
                         value='sepal_length',)]),
    dcc.Graph(
        id='iris-graph',
        figure={}),
    html.Label('Opacity'),
    dcc.Slider(id='select-opacity',min=0,  max=1,step=0.1,
        value=0.5,),],)
```

Dash App with Multiple Inputs

```
@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
    dependencies.Input(component_id='select-species', component_property='value'),
    dependencies.Input(component_id='select-variable', component_property='value'),
    dependencies.Input(component_id='select-opacity', component_property='value')
)
def update_fig(iris_species,iris_variable,opacity):
     flag =[True if item in iris_species else False for item in df.species]
    filtered_df = df[flag]
    fig = px.histogram(filtered_df, x=iris_variable, color='species',opacity=opacity, barmode=
    return fig
```

Dash App with Multiple Inputs



Data Visualization

Select Species:

× setosa × versicolor

Select Variable:

[✓ sepal_length]
sepal_width
petal_length
petal_width

Opacity

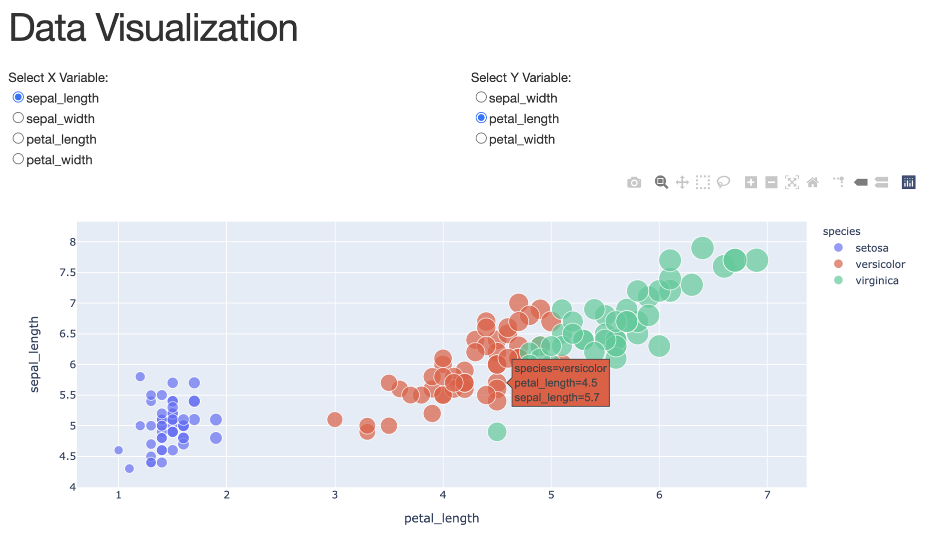
Dash Callbacks

```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(children='Data Visualization'),
    html.Div(
        children=[
            html.Label('Select X Variable:'),
            dcc.RadioItems(id='select-x-variable',
                         options=[{'label': var, 'value': var} for var in
                         ['sepal_length','sepal_width','petal_length','petal_width']
                        ],
                         value='sepal_length',),
            html.Label('Select Y Variable:'),
                        dcc.RadioItems(id='select-y-variable',
                                     options=[{'label': var, 'value': var} for var in
                                     ['sepal_length','sepal_width','petal_length','petal_widtl
                                    ],
                                     value='sepal_width',)],style={'columnCount': 2}),
    dcc.Graph(
        id='iris-graph',
        figure={}),
],)
```

Dash

```python
@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
    dependencies.Input(component_id='select-x-variable', component_property='value'),
    dependencies.Input(component_id='select-y-variable', component_property='value'),
)
def update_fig(iris_x_variable,iris_y_variable):
    fig = px.scatter(df, x=iris_x_variable,y=iris_y_variable, color='species')
    return fig
```

Dash Callbacks



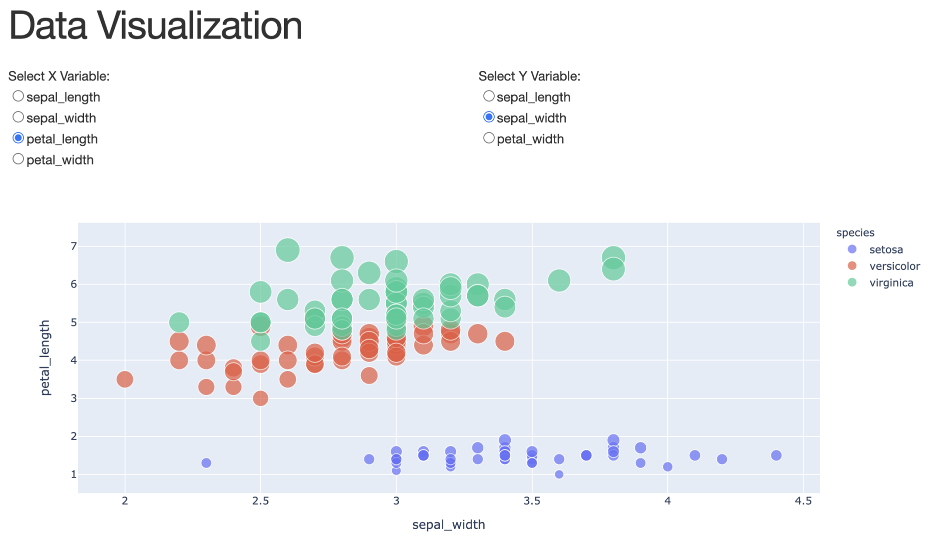
Data Visualization

Select X Variable:

- [✓ sepal_length]
- sepal_width
- petal_length
- petal_width

Select Y Variable:

- sepal_width
- [✓ petal_length]
- petal_width



Data Visualization

Select X Variable:

- sepal_length
- sepal_width
- [✓ petal_length]
- petal_width

Select Y Variable:

- sepal_length
- [✓ sepal_width]
- petal_width

Multiple Outputs&Chained Callbacks

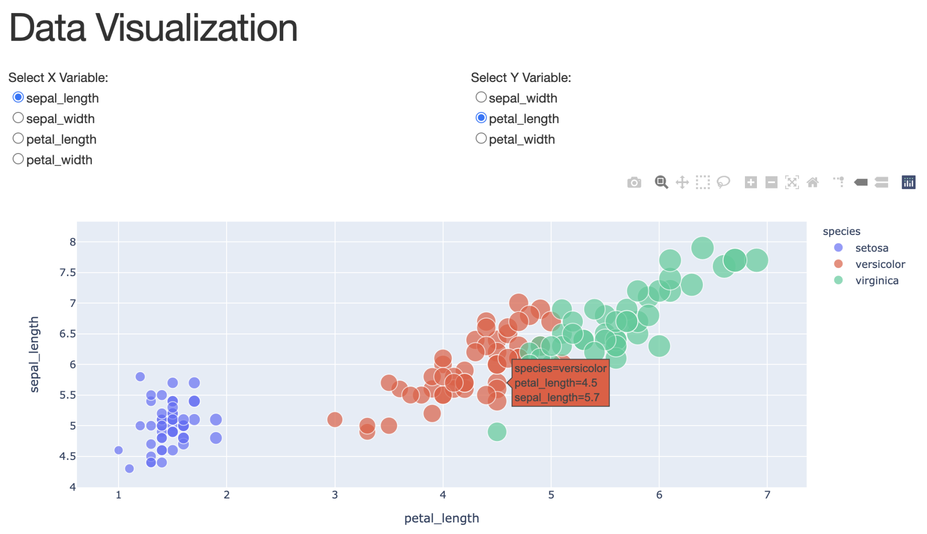
```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(children='Data Visualization'),
    html.Div(
        children=[
            html.Label('Select X Variable:'),
            dcc.RadioItems(id='select-x-variable',
                          options=[{'label': var, 'value': var} for var in
                          ['sepal_length','sepal_width','petal_length','petal_width']
                         ],
                          value='sepal_length',),
            html.Label('Select Y Variable:'),
                         dcc.RadioItems(id='select-y-variable',
                                      options=[],
                                      value='sepal_length',)],
                                       style={'columnCount': 2}),
    dcc.Graph(
        id='iris-graph',
        figure={}),],)
```

Multiple Outputs&Chained Callbacks

```
@app.callback(
     dependencies.Output(component_id='select-y-variable', component_property='options'),
     dependencies.Output(component_id='select-y-variable', component_property='value'),
    dependencies.Input(component_id='select-x-variable', component_property='value'),)
def update_y_options(iris_x_variable):
    iris_variable_list=['sepal_length','sepal_width','petal_length','petal_width']
    iris_variable_list.remove(iris_x_variable)
    opt =[{'label': var, 'value': var} for var in iris_variable_list]
    value = iris_variable_list[0]
    return opt,value

@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
    dependencies.Input(component_id='select-x-variable', component_property='value'),
    dependencies.Input(component_id='select-y-variable', component_property='value'),)
def update_fig(iris_x_variable,iris_y_variable):
    fig = px.scatter(df, x=iris_x_variable,y=iris_y_variable,size='petal_length', color='spec
    return fig
```

Multiple Outputs&Chained Callbacks



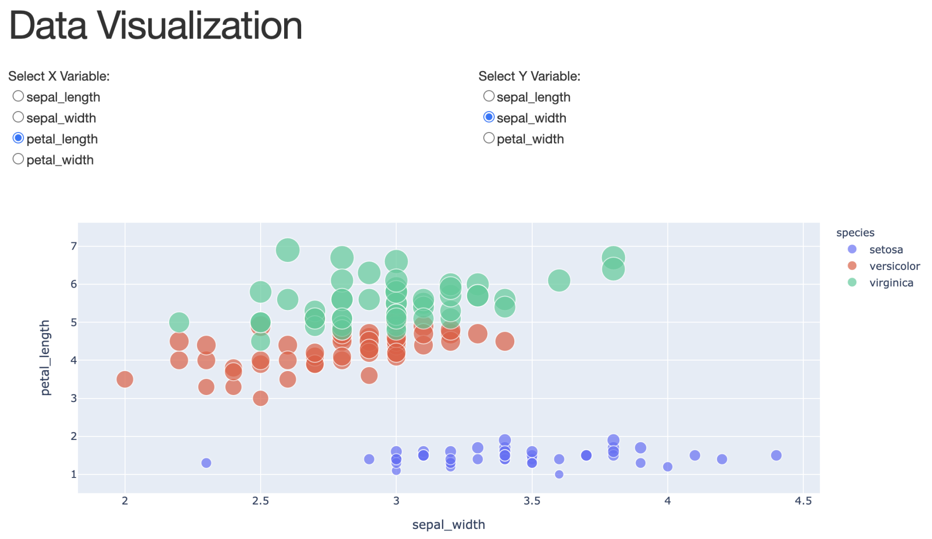
Data Visualization

Select X Variable:

[✓ sepal_length]
sepal_width
petal_length
petal_width

Select Y Variable:

sepal_width
[✓ petal_length]
petal_width



Data Visualization

Select X Variable:

sepal_length
sepal_width
[✓ petal_length]
petal_width

Select Y Variable:

sepal_length
[✓ sepal_width]
petal_width

Dash App with State

In some cases, you might have a "form"-type pattern in your application. In such a situation, you might want to read the value of the input component, but only when the user is finished entering all of his or her information in the form.

```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(id='head',children='Hello Dash1'),
    html.Div(
        children=[
            html.Label('Select Species:'),
            dcc.Dropdown(id='select-species',
                          options=[{'label': 'Setosa', 'value': 'setosa'},
                         {'label': 'Versicolor', 'value': 'versicolor'},
                         {'label':'Virginica','value':'virginica'}
                         ],value='setosa')]),
    dcc.Input(id='graph-title',value='Data Visualization'),
    html.Button(id='button',children='Update Title',n_clicks=0),
    dcc.Graph(
        id='iris-graph',
        figure={})
],)
```

Dash App with State

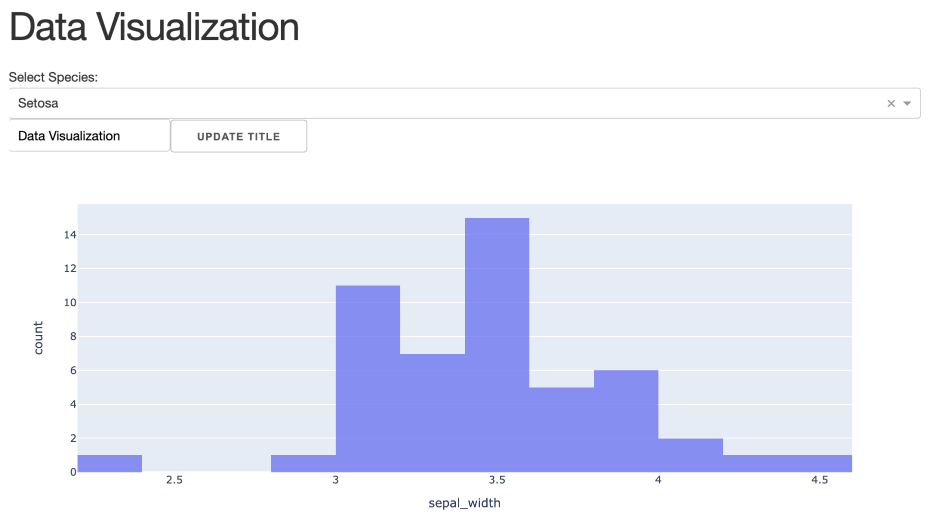
In some cases, you might have a "form"-type pattern in your application. In such a situation, you might want to read the value of the input component, but only when the user is finished entering all of his or her information in the form.

```
@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
    dependencies.Input(component_id='select-species', component_property='value')
)
def update_fig(species):
    filtered_df = df[df['species']==species]
    fig = px.histogram(filtered_df,x="sepal_width", hover_data=['petal_width'], opacity=0.75,
    return fig

@app.callback(
    dependencies.Output(component_id='head',component_property='children'),
    dependencies.Input(component_id='button', component_property='n_clicks'),
     dependencies.State(component_id='graph-title', component_property='value')
)

def update_fig_title(n_clicks,iris_title):
    return iris_title
```

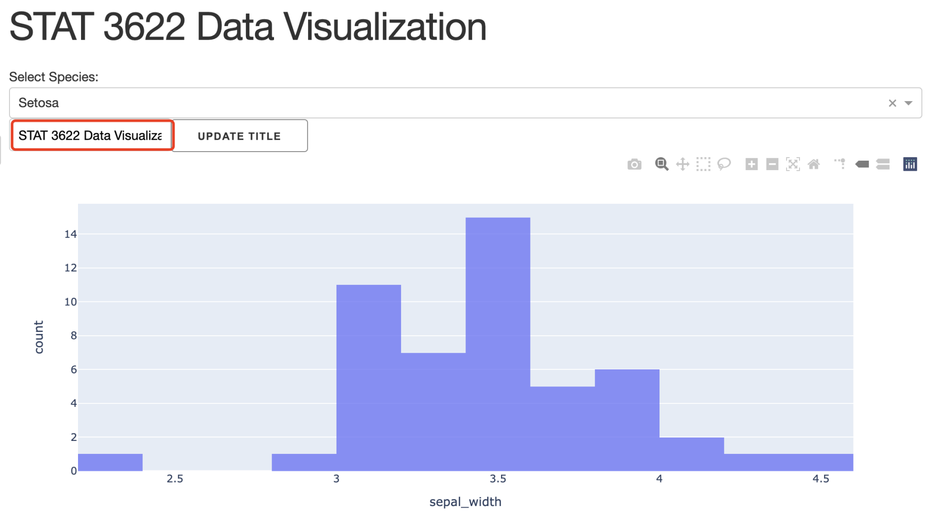
Dash App with State


Data Visualization

Select Species:

Setosa

Data Visualization


STAT 3622 Data Visualization

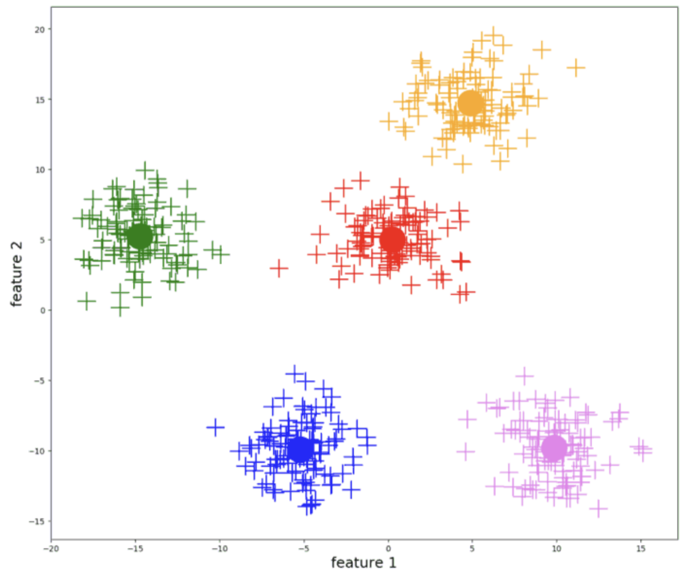
Select Species:

Setosa

STAT 3622 Data Visualiza

K-means Clustering for Iris Data

K-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.



https://en.wikipedia.org/wiki/K-means_clustering

K-means Clustering for Iris Data

K-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

The algorithm works in the following way:

- initialize K clusters with their centers at random positions
- determine the distance between every data point and the center
- assign this point to the cluster with the nearest center
- go through every point in the sample
- move the cluster center to the “center of mass” between the assigned points
- repeat these steps until convergence

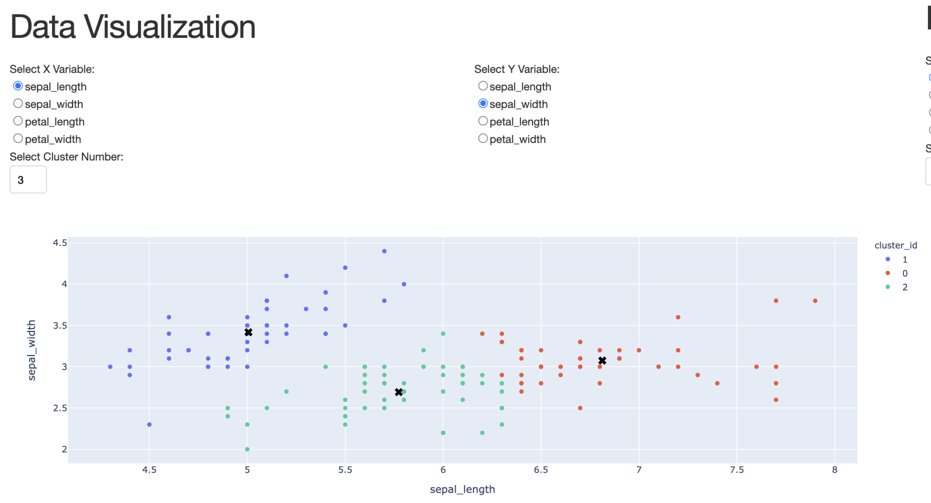
```
df = px.data.iris()
app.layout = html.Div(children=[
    html.H1(children='Data Visualization'),
    html.Div(
        children=[
            html.Label('Select X Variable:'),
            dcc.RadioItems(id='select-x-variable',
                          options=[{'label': var, 'value': var} for var in
                          ['sepal_length','sepal_width','petal_length','petal_width']
                         ],
                          value='sepal_length',),
            html.Label('Select Y Variable:'),
                        dcc.RadioItems(id='select-y-variable',
                                      options=[{'label': var, 'value': var} for var in
                                      ['sepal_length','sepal_width','petal_length','petal_widt
                                     ],
                                      value='sepal_width',)]),

            html.Label('Select Cluster Number:'),
                        dcc.Input(
                            id="select-k", type="number",
                            min=1, max=8, step=1,value=3
                        ),
    dcc.Graph(
        id='iris-graph',
        figure={}),
],)
```

K-means Clustering for Iris Data

```
from sklearn.cluster import KMeans
def cluster(n_clusters,data):
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(data)
    Z = kmeans.predict(data)
    return kmeans, Z
@app.callback(
    dependencies.Output(component_id='iris-graph',component_property='figure'),
    dependencies.Input(component_id='select-x-variable', component_property='value'),
    dependencies.Input(component_id='select-y-variable', component_property='value'),
    dependencies.Input(component_id='select-k', component_property='value'),
)
def update_fig(iris_x_variable,iris_y_variable,n_clusters):
    data_iris= df[[iris_x_variable, iris_y_variable]]
    kmeans,Z =cluster(n_clusters,data_iris.values)
    data_iris["cluster_id"]=[str(i) for i in Z]
    fig = px.scatter(data_iris, x=iris_x_variable,y=iris_y_variable, color='cluster_id')
    fig.add_trace(
        go.Scatter(x=kmeans.cluster_centers_[:, 0],
                   y=kmeans.cluster_centers_[:, 1],
                   mode='markers', marker_symbol='x',marker=dict(color='black',size=10,),
                   showlegend=False))
    return fig
```

K-means Clustering for Iris Data

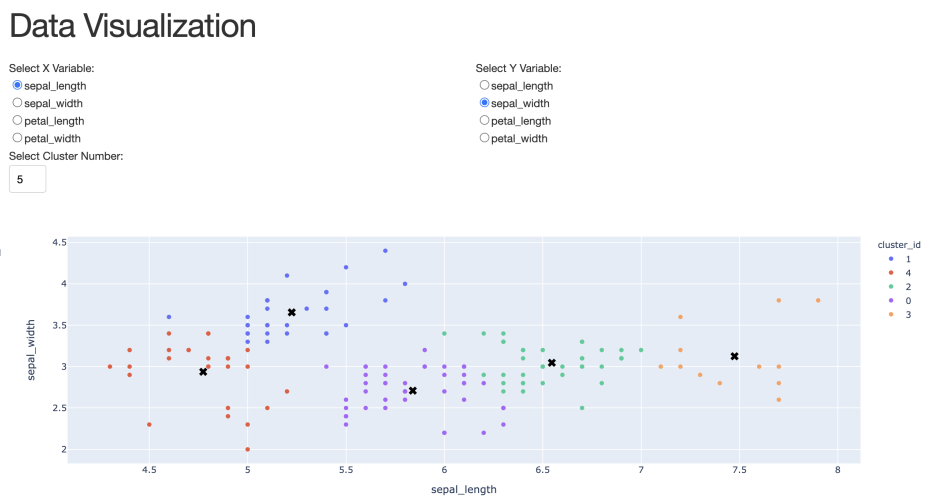


Data Visualization

Select X Variable:
[✓ sepal_length] sepal_width petal_length petal_width

Select Y Variable:
sepal_length [✓ sepal_width] petal_length petal_width

Select Cluster Number:
3



Data Visualization

Select X Variable:
[✓ sepal_length] sepal_width petal_length petal_width

Select Y Variable:
sepal_length [✓ sepal_width] petal_length petal_width

Select Cluster Number:
5

Dash Callbacks

> custom_data (list of str or int, or Series or array-like) – Either names of columns in data_frame, or pandas Series, or array_like objects Values from these columns are extra data, to be used in widgets or Dash callbacks for example. This data is not user-visible but is included in events emitted by the figure (lasso selection etc.)

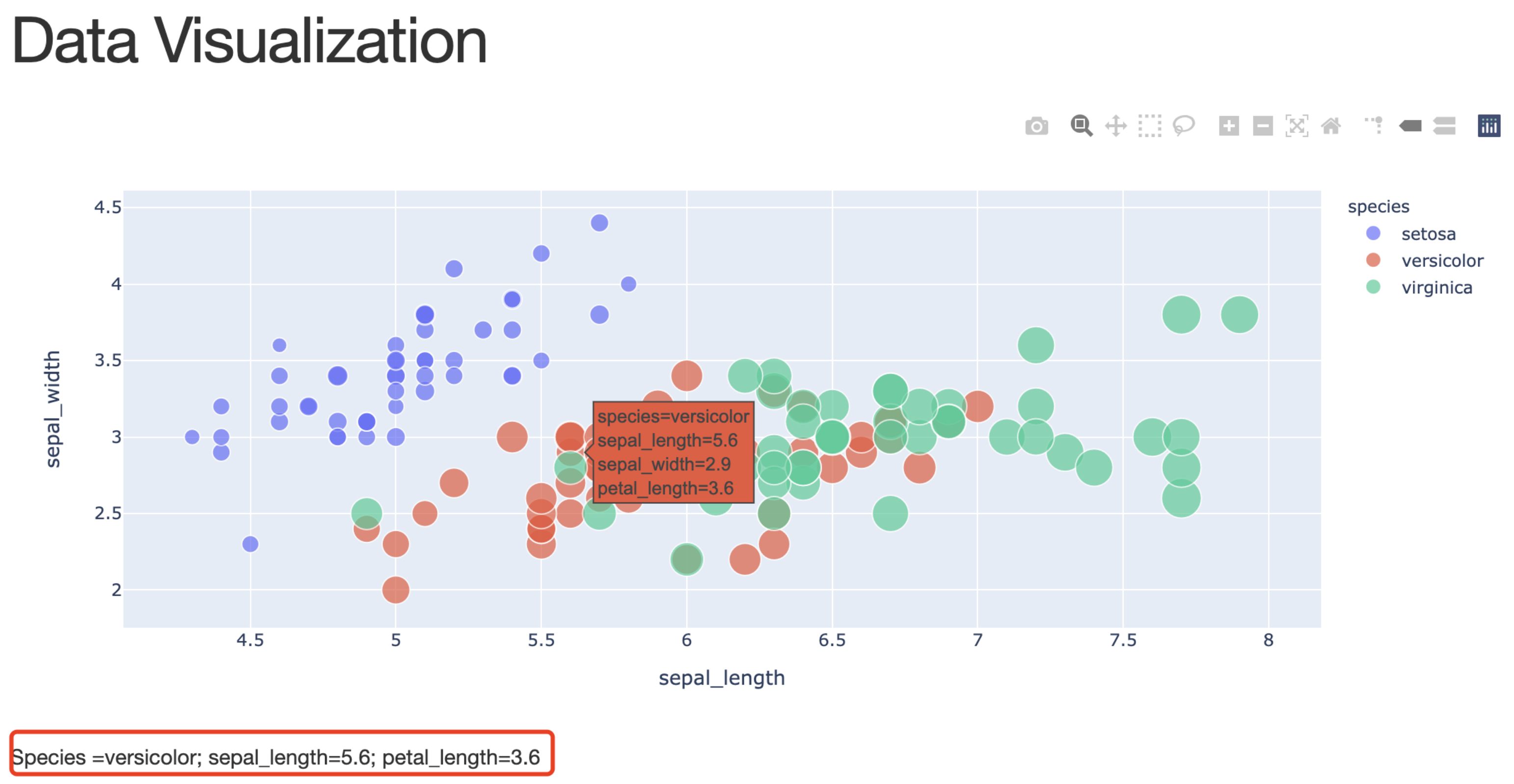
```
df = px.data.iris()
fig = px.scatter(df, x='sepal_length',y='sepal_width',
                 size='petal_length',color='species',
                  custom_data=['species','sepal_length','petal_length'])
app.layout = html.Div(children=[
    html.H1(children='Data Visualization'),
        dcc.Graph(
            id='iris-graph',
            figure=fig,),
        html.P(id='hover-data',children='',)
],)
```

Dash Callbacks

> custom_data (list of str or int, or Series or array-like) – Either names of columns in data_frame, or pandas Series, or array_like objects Values from these columns are extra data, to be used in widgets or Dash callbacks for example. This data is not user-visible but is included in events emitted by the figure (lasso selection etc.)

```python
@app.callback(
    dependencies.Output(component_id='hover-data', component_property='children'),
    dependencies.Input(component_id='iris-graph', component_property='hoverData'))
def display_hover_data(hoverData):
    if hoverData is None:
        return ''
    else:
        #print(hoverData)
        # {'points': [{'curveNumber': 1, 'pointNumber': 9, 'pointIndex': 9, 'x': 5.2, 'y': 2.
        # 'marker.size': 3.9, 'customdata': ['versicolor', 5.2, 3.9]}]}
        info=hoverData['points'][0]['customdata']
        text = 'Species ={}; sepal_length={}; petal_length={}'.format(info[0],info[1],info[2]
        return text
```

Dash Callbacks



Species =versicolor; sepal_length=5.6; petal_length=3.6

Thank you!

Q&A or Email wbdu@hku.hk