

# Callbacks in Dash

BUILDING DASHBOARDS WITH DASH AND PLOTLY



**Alex Scriven**  
Data Scientist

# What are callbacks?

- Functionality triggered by interaction
  - A user interacts with an element
    - -> A Python function is triggered
      - --> Something is changed
- Why? Enhances interactivity

# Callbacks in Dash

- Start with the **decorator** function
  - Uses

```
from dash.dependencies import
Input, Output
```
- Output: Where to send the function return
  - `component_id`: Identify the component
  - `component_property`: What will be changed
- Input: What triggers the callback
  - `component_property`: What to use in triggered function

```
@app.callback(
    Output(component_id='my_plot',
           component_property='figure'),
    Input(component_id='my_input',
          component_property='value')
)
def some_function(data):
    # Subset Data
    # Recreate Figure
    return fig
```

# Dropdowns in Dash

```
dcc.Dropdown(id='title_dd',  
             options=[{'label': 'Title 1',  
                       'value': 'Title 1'},  
                      {'label': 'Title 2',  
                       'value': 'Title 2'}]),
```

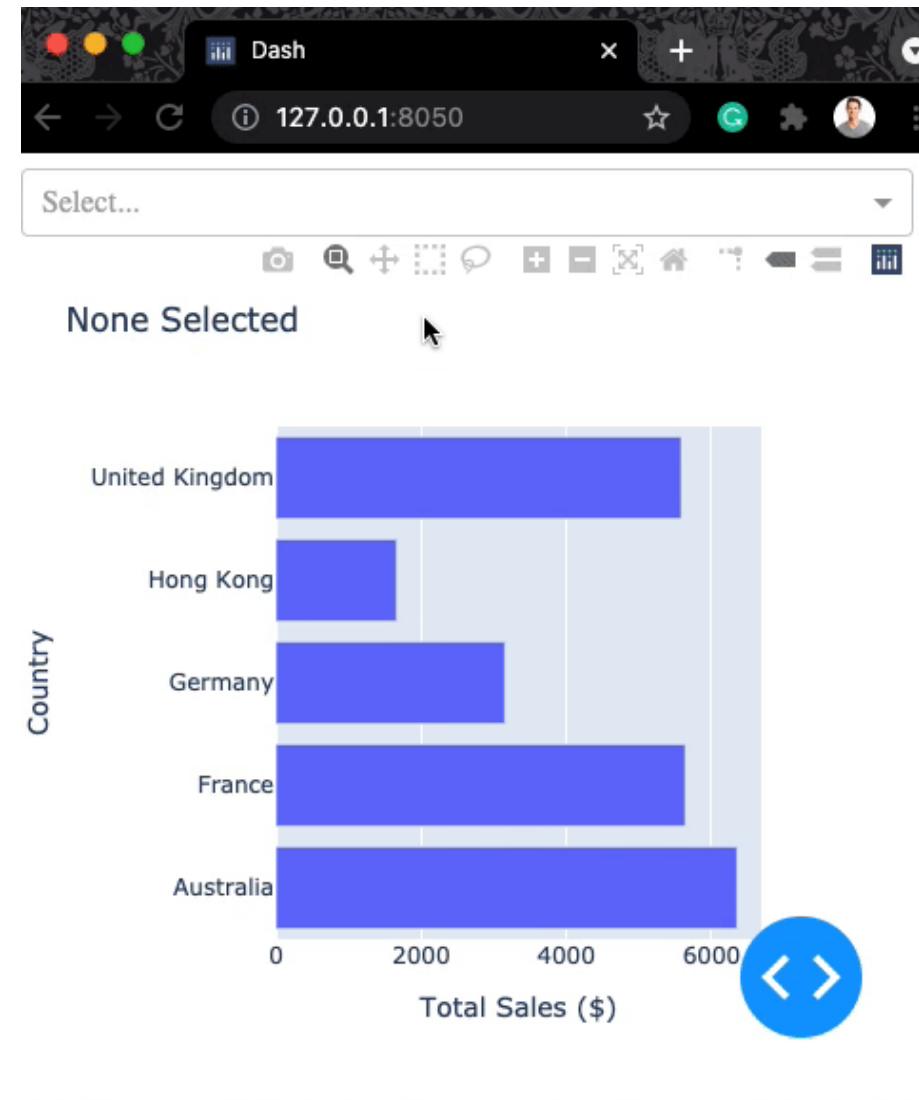
- List of label-value dictionaries

# A dropdown callback

```
app.layout = html.Div(children=[
    dcc.Dropdown(id='title_dd',
                 options=[{'label': 'Title 1',
                           'value': 'Title 1'},
                           {'label': 'Title 2',
                           'value': 'Title 2'}]),
    dcc.Graph(id='my_graph')])
@app.callback(
    Output(component_id='my_graph',
           component_property='figure'),
    Input(component_id='title_dd',
          component_property='value'))
```

```
#@app.callback()
def update_plot(selection):
    title = "None Selected"
    if selection:
        title = selection
    bar_fig = px.bar(
        data_frame=ecom_sales,
        title=f'{title}',
        x='Total Sales ($)', y='Country')
    return bar_fig
```

# Our first dropdown



# Dropdown as a filter

- Common use case - dropdown filters the plot DataFrame

```
#@app.callback()
def update_plot(input_country):
    input_country = 'All Countries'
    sales = ecom_sales.copy(deep=True)
    if input_country:
        sales = sales[sales['Country'] == input_country]
    bar_fig = px.bar(
        data_frame=sales, title=f"Sales in {input_country}",
        x='Total Sales ($)', y='Country')
    return bar_fig
```

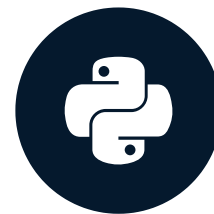
# Let's practice!

BUILDING DASHBOARDS WITH DASH AND PLOTLY



# Interactive components

BUILDING DASHBOARDS WITH DASH AND PLOTLY



**Alex Scriven**  
Data Scientist

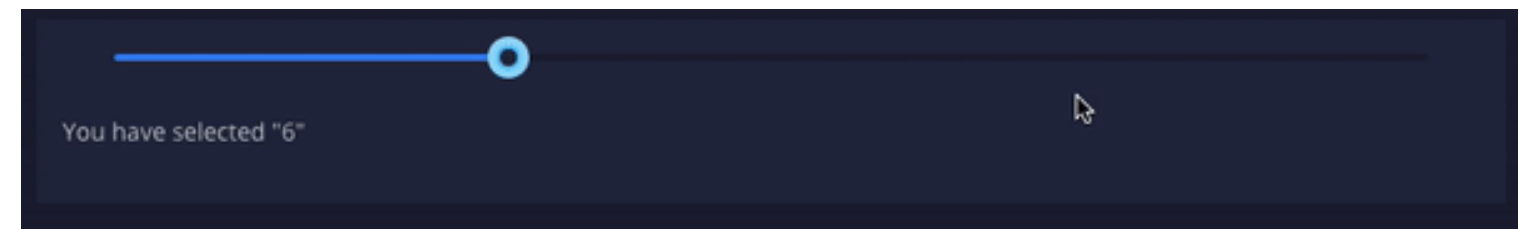
# Enhancing Interactivity

- Some useful interactive components:
  - `dcc.Checklist()` = Checkboxes
  - `dcc.RadioItems()` = Radio buttons
  - `dcc.Slider()` / `dcc.RangeSlider()` = Slider selectors
  - `dcc.DatePickerSingle()` / `dcc.DatePickerRange()` = Similar to sliders but for dates

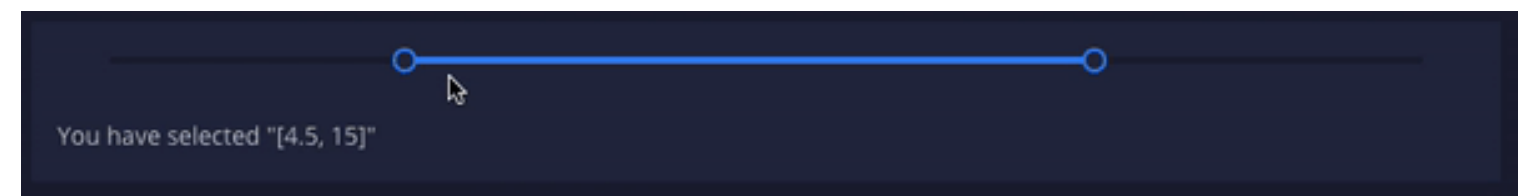
# Sliders

- Slider: drag and move for a single value
- Range Slider: drag and move for two values
- Reminder: Can link to callback
  - Update plots or components

A slider:



A range slider:



# Sliders in Dash

```
dcc.Slider(  
    min=10,  
    max=50,  
    value=45,  
    step=5,  
    vertical=False  
)
```

## Key arguments:

- `min` / `max` : Bounds of slider
- `value` : Starting selection
- `step` : Increment for each notch
- `vertical` : To make horizontal or vertical

# Date pickers in Dash

`DatePickerSingle` : Select a single date

```
dcc.DatePickerSingle(  
    date=date(2021, 7, 1),  
    initial_visible_month=datetime.now(),  
)
```

- `date` = starting selection
- `initial_visible_month` = month shown in popup
- Optionally limit `min_date_allowed` and `max_date_allowed`

07/01/2021

You have selected: July 01, 2021

# Date Range Picker

- Similar to `DatePickerSingle`

```
dcc.DatePickerRange(  
    initial_visible_month=datetime.now(),  
    start_date=date(2021, 7, 1),  
    end_date=date(2021, 7, 14),  
)
```

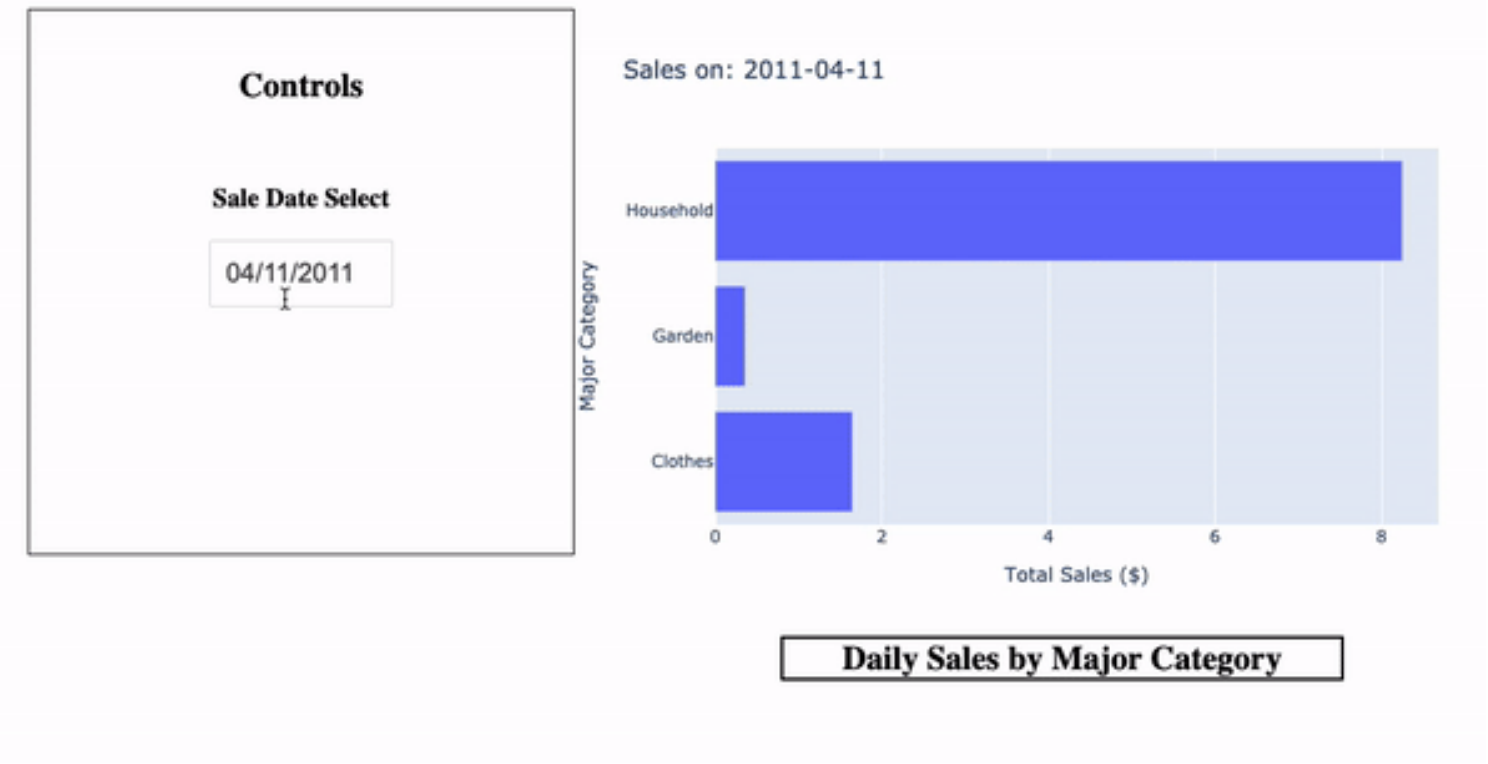
- Set an initial `start_date` and `end_date`

A UI element for a date range picker. It consists of two rectangular input boxes with thin borders. The first box contains the text '07/01/2021' and the second box contains '07/14/2021'. Between the two boxes is a right-pointing arrow '→'.

You have selected: Start Date: July 01, 2021 | End Date: July 14, 2021

# Updating plots

```
# dcc.DatePickerSingle(id='sale_date')
# dcc.Graph(id='sales_cat')
@app.callback(
    Output(component_id='sales_cat',
            component_property='figure'),
    Input(component_id='sale_date',
           component_property='date'))
def update_plot(input_date):
    sales = ecom_sales.copy(deep=True)
    if input_date:
        sales = sales[sales['InvoiceDate'] == input_date]
    # Create fig
    return fig
```



# Let's practice!

BUILDING DASHBOARDS WITH DASH AND PLOTLY



# Reusable Dash components

BUILDING DASHBOARDS WITH DASH AND PLOTLY



**Alex Scriven**  
Data Scientist

# DRY Code

- DRY = Don't Repeat Yourself (or refactoring)
  - Remove unnecessary code
- In Python: Often using functions

# DRY Code example

```
sales_country = ecom_sales\  
    .groupby('Country')['OrderValue']\  
    .sum()\  
    .reset_index(name='Total Sales ($)')\  
    .sort_values('Total Sales ()',  
                 ascending=False)  
  
sales_ma_cat = ecom_sales\  
    .groupby('Major Category')['OrderValue']\  
    .sum()\  
    .reset_index(name='Total Sales ($)')\  
    .sort_values('Total Sales ()',  
                 ascending=False)
```

Refactored:

```
def sales_by(col):  
    df = ecom_sales\  
        .groupby(col)['OrderValue']\  
        .sum()\  
        .reset_index(name='Total Sales ($)')\  
        .sort_values('Total Sales ()',  
                     ascending=False)  
  
    return df  
  
# Call many times  
sales_country = sales_by('Country')  
sales_ma_cat = sales_by('Major Category')  
sales_mi_cat = sales_by('Minor Category')
```

# DRY in Dash

- In Dash: use functions to refactor code
- Use cases (using functions):
  - Re-using HTML (or any) component
  - Adding consistent styling (CSS can be fiddly!)
  - Ease of updating code

# Re-using components

E.g., Heavily styled logo;

```
def create_logo():
```

```
    logo=html.Img(src=logo_link, style={
        'margin': '30px 0px 0px 0px',
        'padding': '50px 50px',
        'border': '3px dotted lightblue',
        'background-color': 'rgb(230,131,247)'
    })
```

```
    return logo
```

```
app.layout = html.Div([
    create_logo(),
    html.Div(),
    # More components
    create_logo(),
    dcc.Graph(id='my_graph')
    create_logo()
])
```

The logo is inserted 3 times!

# Generating a component list

Before:

```
app.layout = html.Div([
    html.Img(src=logo_link),
    html.Br(),
    html.Br(),
    html.H1("Sales breakdowns"),
    html.Br(),
    html.Br(),
    html.Br(),
    html.Div(children=[
        html.Div(children=[
```

After:

```
def make_break(num_breaks):
    br_list = [html.Br()] * num_breaks
    return br_list
app.layout = html.Div([
    html.Img(src=logo_link),
    *make_break(2),
    html.H1("Sales breakdowns"),
    *make_break(3),
    html.Div(children=[
        html.Div(children=[
```

# Reusing styling

- Have some common styling we want added
- Python dictionary `.update()` used (warning: unique keys!)

```
d1 = {'Country': 'Australia'}  
d2 = {'City': 'Sydney'}  
d1.update(d2)  
print(d1)
```

```
{'Country': 'Australia', 'City': 'Sydney'}
```

# Styling functions in Dash

Set up the function:

```
def style_c():  
    corp_style={  
        'margin': '0 auto',  
        'border': '2px solid black',  
        'display': 'inline-block',  
    }  
    return corp_style
```

Call in Dash layout:

```
app.layout = html.Div([  
    html.Img(src=logo_link,  
        style=style_c()),  
    dcc.DatePickerSingle(  
        style={'width': '200px'}.update(style_c())  
    )  
])
```



# Let's practice!

BUILDING DASHBOARDS WITH DASH AND PLOTLY

# User inputs in Dash components

BUILDING DASHBOARDS WITH DASH AND PLOTLY



**Alex Scriven**  
Data Scientist

# Why user input?

Some useful applications of user inputs:

- Filtering across a large number range (`number` inputs)
  - Consider a dropdown with 2000 'year' values!
- Filtering based on text-matching (`search` or `text` inputs)
- Creating a login (`password` and `email` / `text` input)
  - Beyond this course

# User input in Dash

A user input is a `dash_core_components`  
`Input` type (`dcc.Input`)

- An `id` is required for callback usage
- The `type` will default to text (more on this later!)
- The `placeholder` appears faded in the input box

```
dcc.Input(  
    id='my_input',  
    type='text',  
    placeholder="Enter your text")
```

# Using the input

Similar to previous input work:

- Input becomes a Python variable
- Used with the callback
  - Typically to **filter** a DataFrame
- For example (right):
  - `df` subset using input

```
#@app.callback()
def update_plot(my_input):
    df = data.copy(deep=True)
    df = df[df['column'] == my_input]
    fig = px.scatter(data_frame=df)
    return fig
```

# User input types

Dash offers useful input types:

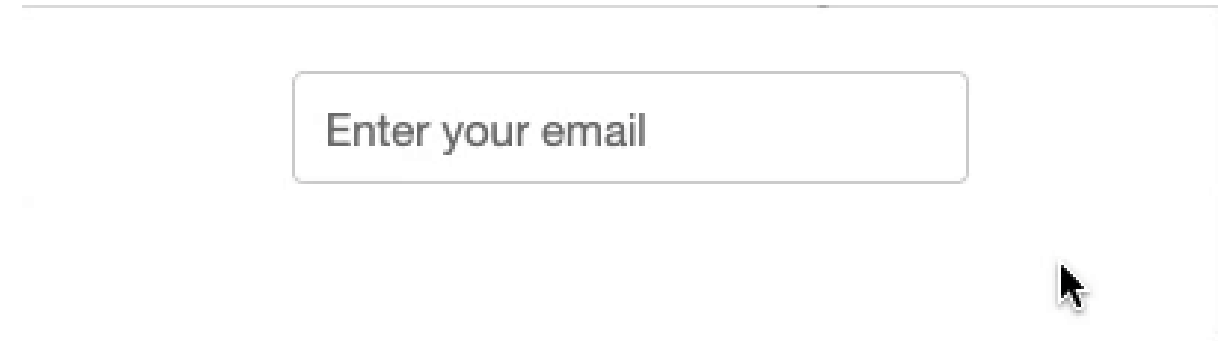
- Some are straightforward: `'text'` , `'number'` , `'password'` , `'email'`
- Some are more specialized:
  - `'range'` produces a range slider
  - `'tel'` and `'url'` are for telephone numbers and website urls
- Some are advanced
  - `'search'` and `'hidden'` involve advanced browser interaction

# Restricting user input

The `type` argument automatically sets some limitations.

- E.g., an `email` type requires **something@something.com** format

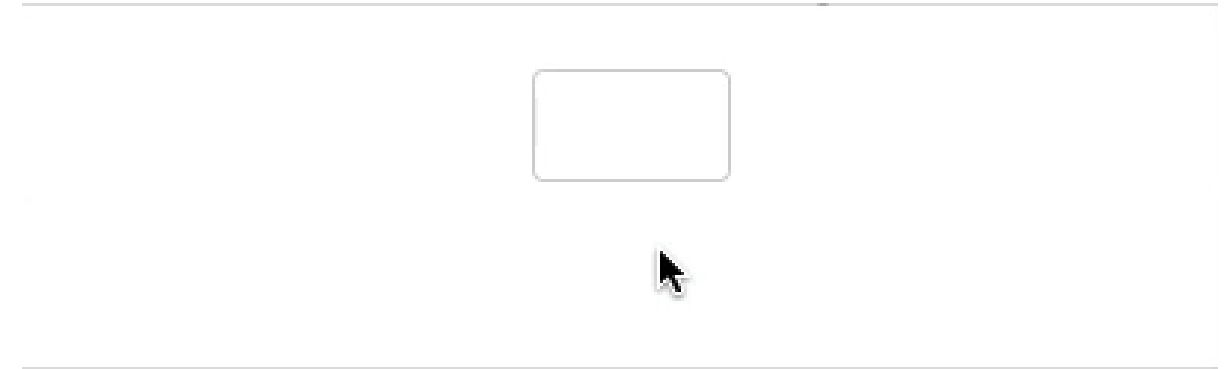
```
dcc.Input(  
    id='my_input',  
    type='email',  
    placeholder="Enter your email")
```



# Additional restrictions

Additional arguments for specific types help control input

- E.g., a `number` type only allows numbers
  - Additionally: `min` and `max` set numerical limit
  - `minLength` / `maxLength` for `text` inputs
- E.g., a `text` type also has `pattern` for regex validation



```
dcc.Input(  
    id='my_input',  
    type='number',  
    max=250)
```



# Toggling an input

We can turn off an input programmatically with `disabled`

A disabled input

```
dcc.Input(id='my_input', disabled=True)
```

Or we can force its usage with `required`

A required input

(Both are True/False arguments of `dcc.Input()` )

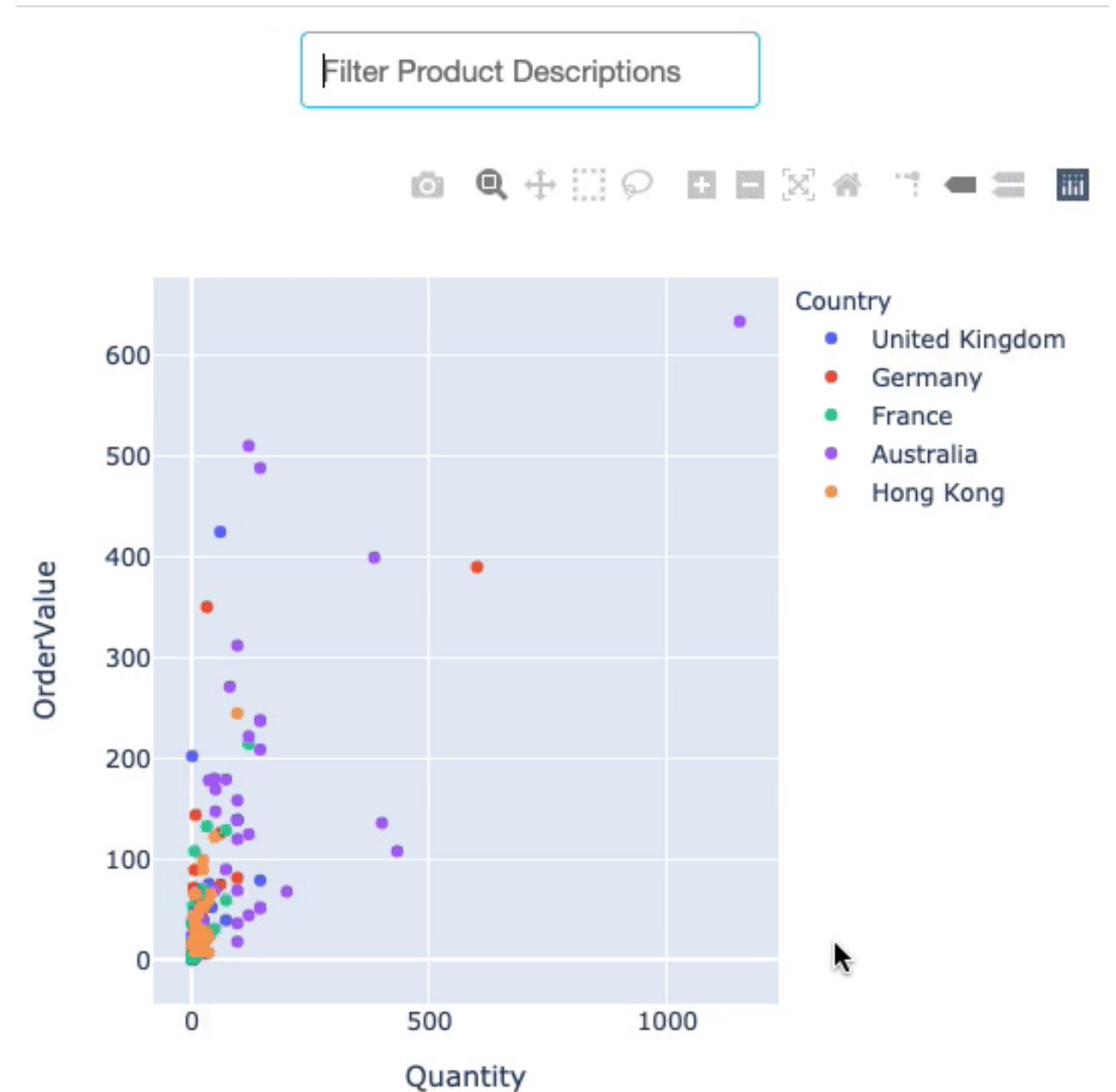
```
dcc.Input(id='my_input', required=True)
```

# When to update

A vital argument is `debounce` : Determines callback trigger (on unfocus or 'Enter') versus as-you-type

- Here `debounce` is `False` (callback as you type)
  - Filtering for `R` , `Re` , `Red` , `Redd` in turn

```
dcc.Input(id='my_input', type='text',  
         debounce=False)
```



# Let's practice!

BUILDING DASHBOARDS WITH DASH AND PLOTLY