Interactive Visualization with Bokeh - Interactive Plots - 3

One should look for what is and not what he thinks should be. (Albert Einstein)

Warm up - Activity

- **Project - After Babylon** was developed to analyze the situation of linguistics in the world
- Visit their ***interactive visualizations*** and explore the distribution of different language groups
- As you explore the project answer the following question:
  - What two language families are present in India?

Recap

- This course has covered the following so far:
  - Organizing, transforming and visualizing data with Bokeh
  - Creating maps and simple plots with Bokeh

Module completion checklist

Objective	Complete
Organize multiple visualization with layouts and configure plot tools	
Add interactivity and highlight data using labels	

Laying out plots and plot tools

- Set the output method to display the plots in the notebook
- Organize the layout when you wish to render multiple plots together by specifying `show()`
- Add the tools we wish to add in `figure()` as shown below
- The code also shows an alternate method to label the axes

```
# Set the output method
output_notebook()

tools = ["box_select", "hover", "reset"]

# create a new plot
p1 = figure(title = "ppl_total vs num_adults",
            plot_width = 400, plot_height = 400,
            tools = tools)

p1.xaxis.axis_label = 'ppl_total'
p1.yaxis.axis_label = 'num_adults'

p1.diamond(costa_viz['ppl_total'],
            costa_viz['num_adults'],
            size = 20,
            color = "plum",
            alpha = 0.2)
```

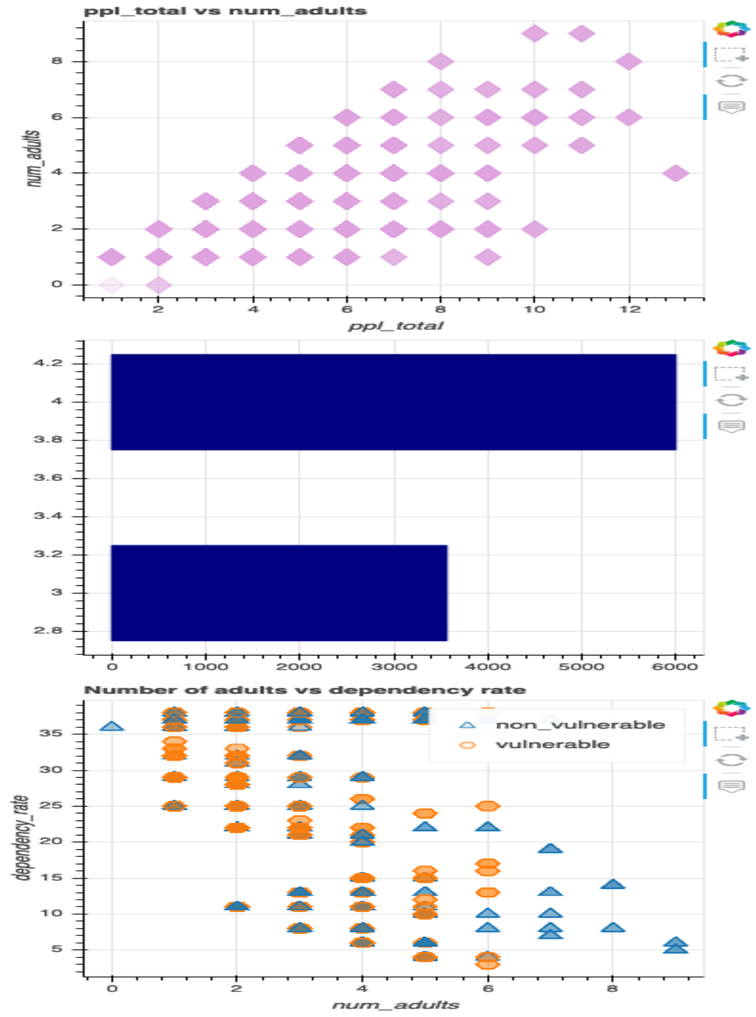
Laying out plots and widgets

```
# Create another one.
LEVELS = ['non_vulnerable', 'vulnerable']
MARKERS = ['triangle', 'hex']
p2 = figure(plot_width = 400, plot_height = 400, tools = tools)

p2.hbar(y=[4, 3, 2, 1],
        height = 0.5,
        left = 0,
       right = costa_viz.Target.value_counts(),
       color = "navy")

# Create another graph.
p3 = figure(title = "Number of adults vs dependency rate",
            plot_width = 400,
            plot_height = 400,
            tools = tools)
```

```
p3.xaxis.axis_label = 'num_adults'
p3.yaxis.axis_label = 'dependency_rate'

p3.scatter("num_adults", "dependency_rate",
            source = costa_viz,
            legend = "Target_class",
            fill_alpha = 0.1, size = 12,
            marker = factor_mark('Target_class',
                                 MARKERS, LEVELS),
            color = factor_cmap('Target_class',
                                'Category10_7',
                                 LEVELS))
```

Laying out plots and widgets (cont'd)

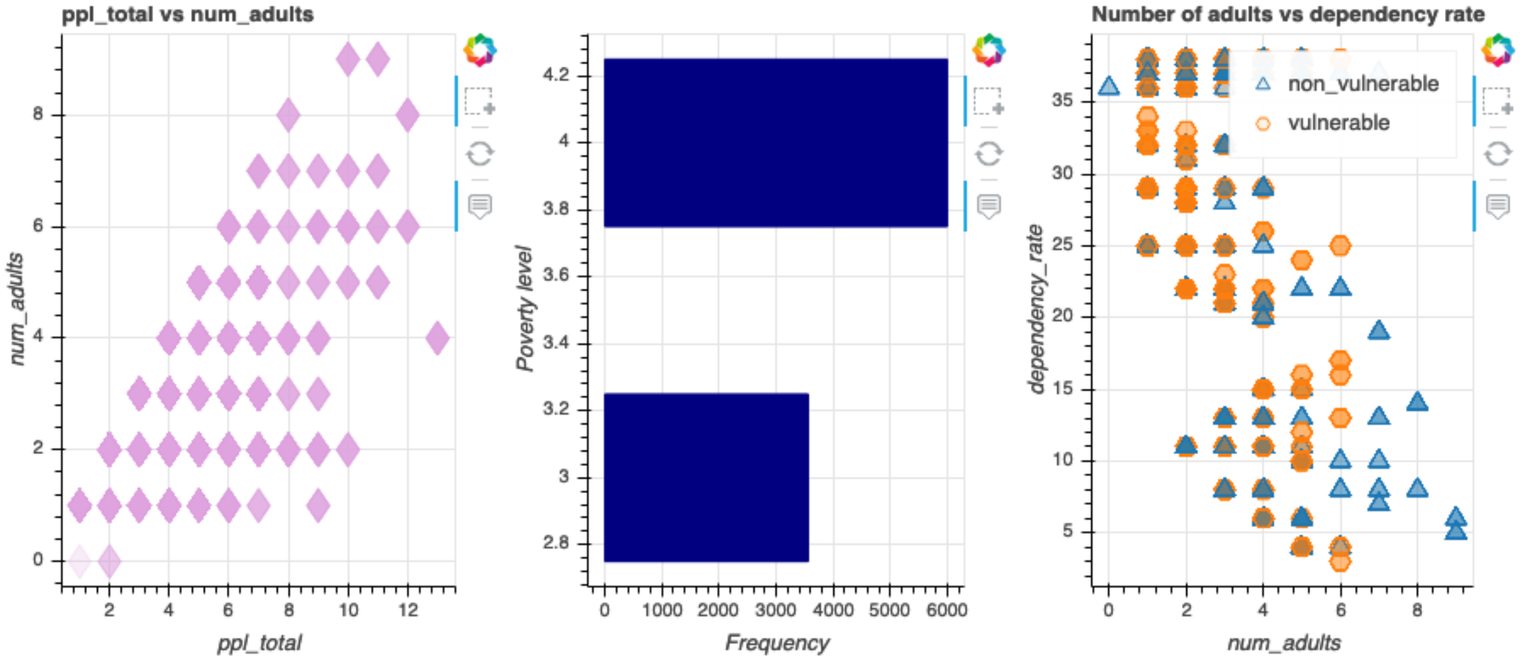
```
# Put the results in a column and show.
show(column(p1, p2, p3))
```



Laying out plots and widgets (cont'd)

- Row-wise layout

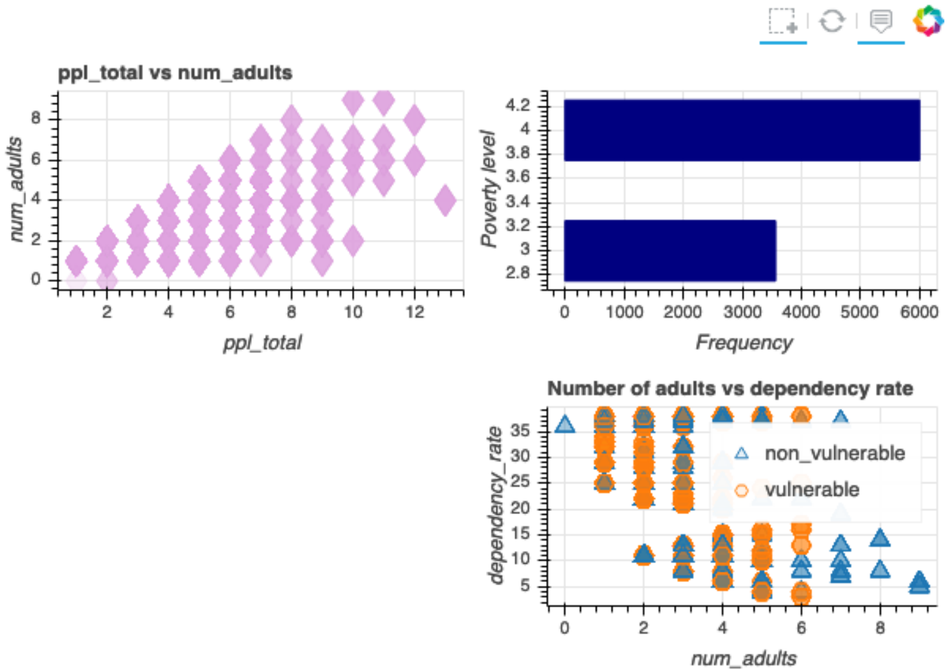
```
# Put the results in a row.
show(row(p1, p2, p3))
```



Laying out plots and widgets (cont'd)

- We can arrange graphs as subplots
- Notice that we have left the third quadrant empty

```
grid = gridplot([[p1, p2],
                 [None, p3]])

show(grid)
```



ColumnDataSource

- We can link our `pandas` DataFrame to Bokeh using object **ColumnDataSource**
- It is specifically used for plotting with several methods, and allows us to add annotations and interactivity to our graphs
- After it is created, the ColumnDataSource can then be passed to glyph methods via the `source` parameter and other parameters (such as `x` and `y` axes)

```
# Import the ColumnDataSource class.
from bokeh.models import ColumnDataSource

# Convert dataframe to column data source.
src = ColumnDataSource(costa_viz)
```

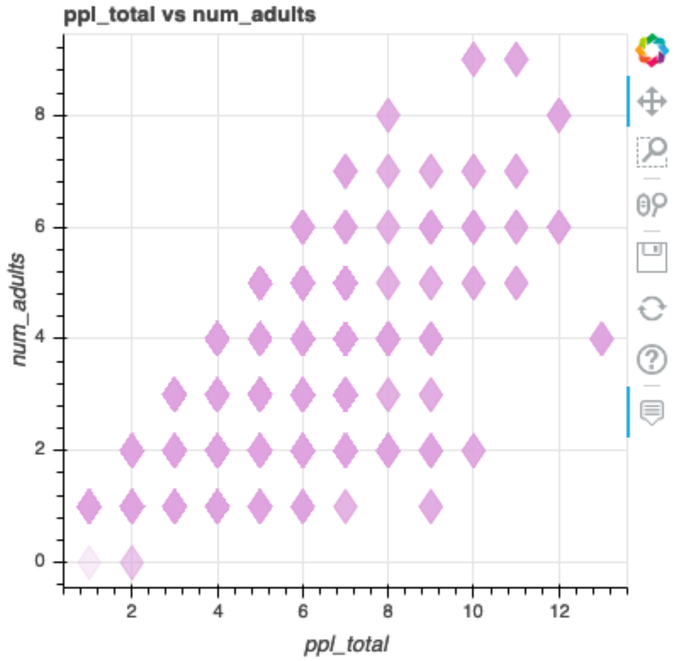
Customizing HoverTool

```
# Hover tool refers to our own data field using @ and
# a position on the graph using $.
hover = HoverTool(tooltips = [('Total number of people', '@ppl_total'),
                              ('Number of adults', '@num_adults'),
                              ('(x,y)', '($x, $y)')])

p = figure(title = "ppl_total vs num_adults",
           plot_width=400, plot_height=400,
           x_axis_label = 'ppl_total',
           y_axis_label = 'num_adults')

p.diamond('ppl_total',
          'num_adults',
          source = src,
          size = 20,
          color = "plum",
          alpha = 0.2)
```

```
# Add the hover tool to the graph.
p.add_tools(hover)
```

Customizing HoverTool (cont'd)

```
show(p)
```



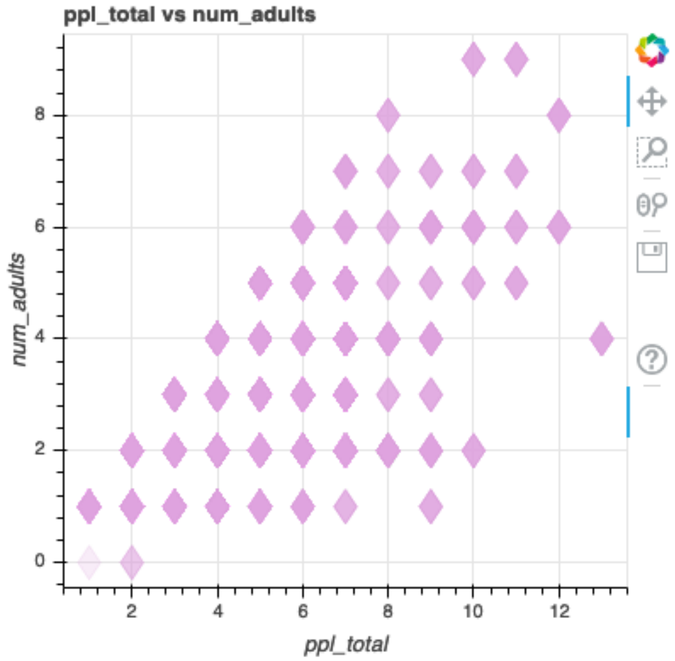
Customizing HoverTool (cont'd)

- Hover attributes can be customized in the glyphs as shown below
- The data point hovered over will change its color and opacity level

```
# Hover tool refers to our own data field using @ and
# a position on the graph using $.
hover = HoverTool(tooltips = [('Total number of people', '@ppl_total'),
                              ('Number of adults', '@num_adults'),
                              ('(x,y)', '($x, $y)')])

p = figure(title = "ppl_total vs num_adults",
           plot_width = 400, plot_height = 400,
          x_axis_label = 'ppl_total',
          y_axis_label = 'num_adults')


p.diamond('ppl_total','num_adults', source = src, size=20, color = "plum", alpha=0.2,
         hover_fill_alpha = 1.0, hover_fill_color = 'navy')

# Add the hover tool to the graph.
p.add_tools(hover)
```

Customizing HoverTool (cont'd)

```
show(p)
```



Module completion checklist

Objective	Complete
Organize multiple visualization with layouts and configure plot tools	✔
Add interactivity and highlight data using labels	

Highlighting data using HoverTool

- Using `ColumnDataSource()` for a previous visualization sometimes throws an error, so let's create a new one for each graph

```
# Store the data in a ColumnDataSource.
costa_cds = ColumnDataSource(costa_viz)
```

```
# Specify the selection tools to be made available.
select_tools = ['box_select', 'lasso_select', 'poly_select', 'tap', 'reset']
# Create the figure.
fig = figure(plot_height = 400,
             plot_width = 600,
             x_axis_label = 'ppl_total',
             y_axis_label = 'num_adults',
             title = 'Interactive scatterplot',
             toolbar_location = 'below',
             tools = select_tools)
```

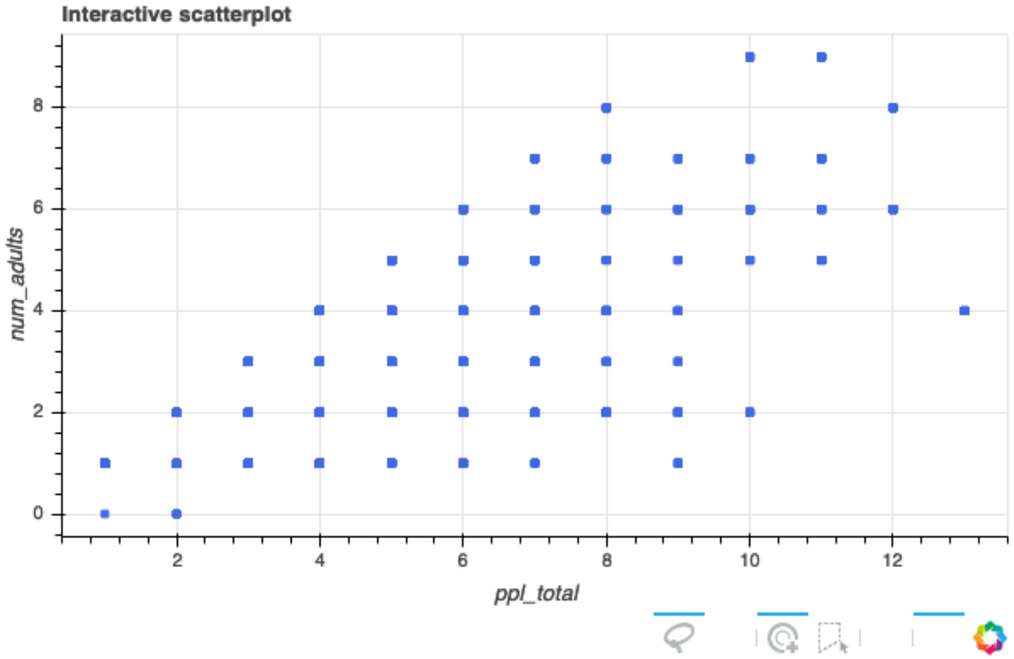
```
# Add square representing each layer.
fig.square(x = 'ppl_total',
           y = 'num_adults',
           source = costa_cds,
           color = 'royalblue',
           selection_color = 'deepskyblue',
           nonselection_color = 'lightgray',
           nonselection_alpha = 0.3)
```

Customizing HoverTool

- `tooltips` from `HoverTool()` accepts input data and allows us to select data with the cursor

```
# Format the tooltip.
tooltips = [
            ('ppl_total','@ppl_total'),
            ('num_adults', '@num_adults')
           ]

# Add the HoverTool to the figure.
fig.add_tools(HoverTool(tooltips=tooltips))

# Visualize the graph.
show(fig)
```

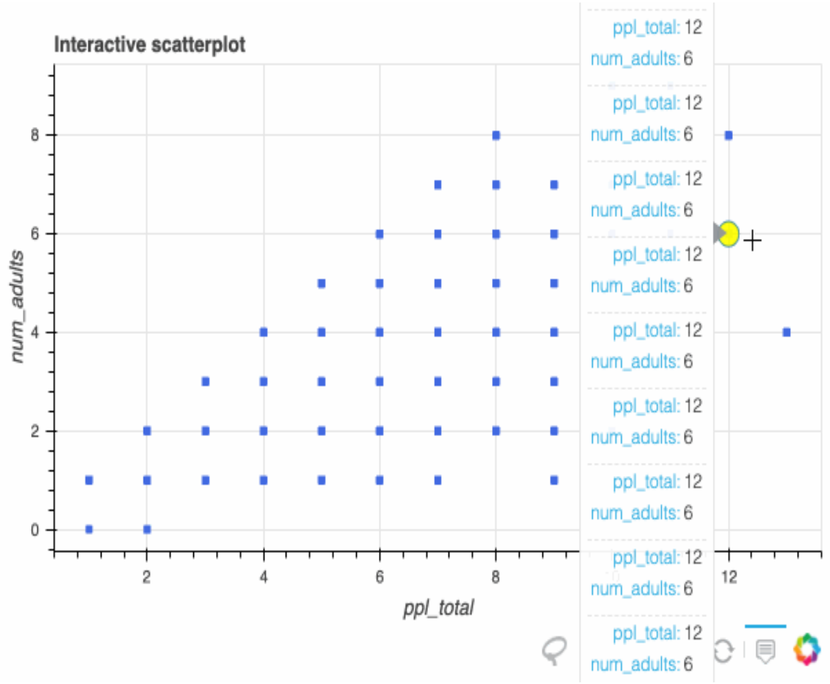
Customizing HoverTool (cont'd)



Customizing HoverTool (cont'd)

- Creating a new circle glyph named `hover_glyph` and adding it as `renderers` to `.add_tools()` will display the data point hovered over as a yellow circle instead

```
# Format the tooltip.
tooltips = [
            ('ppl_total','@ppl_total'),
            ('num_adults', '@num_adults')
           ]

hover_glyph = fig.circle(x = 'ppl_total', y = 'num_adults',
                         source = costa_cds,
                         size = 15, alpha = 0,
                         hover_fill_color = 'yellow',
                         hover_alpha = 0.2)

# Add the HoverTool to the figure.
fig.add_tools(HoverTool(tooltips = tooltips,  renderers =
[hover_glyph]))

# Visualize the graph.
show(fig)
```



Highlighting data using labels

- We can select data points using the labels of `Target_class` by creating filters and views for both labels

```
costa_labels = ColumnDataSource(costa_viz)


# Create a view for each label.
vul_filters = [GroupFilter(column_name='Target_class', group = 'vulnerable')]

vul_view = CDSView(source = costa_labels,
                   filters = vul_filters)
```

```
# Create a view for each label.
nonvul_filters = [GroupFilter(column_name='Target_class', group = 'non_vulnerable')]

nonvul_view = CDSView(source = costa_labels,
                   filters = nonvul_filters)
```

Highlighting data using labels (cont'd)

- The common parameters used across the whole graph can be consolidated into dictionaries so we can reuse them later, instead of defining them every time

```
# Consolidate the common keyword arguments in dictionaries.
common_figure_kwargs = {
    'plot_width': 400,
    'plot_height':500,
    'x_axis_label': 'num_adults',
    'y_axis_label' : 'dependency_rate',
    'toolbar_location': None}
common_circle_kwargs = {
    'x': 'ppl_total',
    'y': 'num_adults',
    'source': costa_labels,
    'size': 12,
    'alpha': 0.7,}
```

```
common_vul_kwargs = {
    'view': vul_view,
    'color': '#002859',
    'legend': 'vulnerable'}
common_non_kwargs = {
    'view': nonvul_view,
    'color': '#FFC324',
    'legend': 'non_vulnerable'}
```

Highlighting data using labels (cont'd)

- Create two figures and draw the data

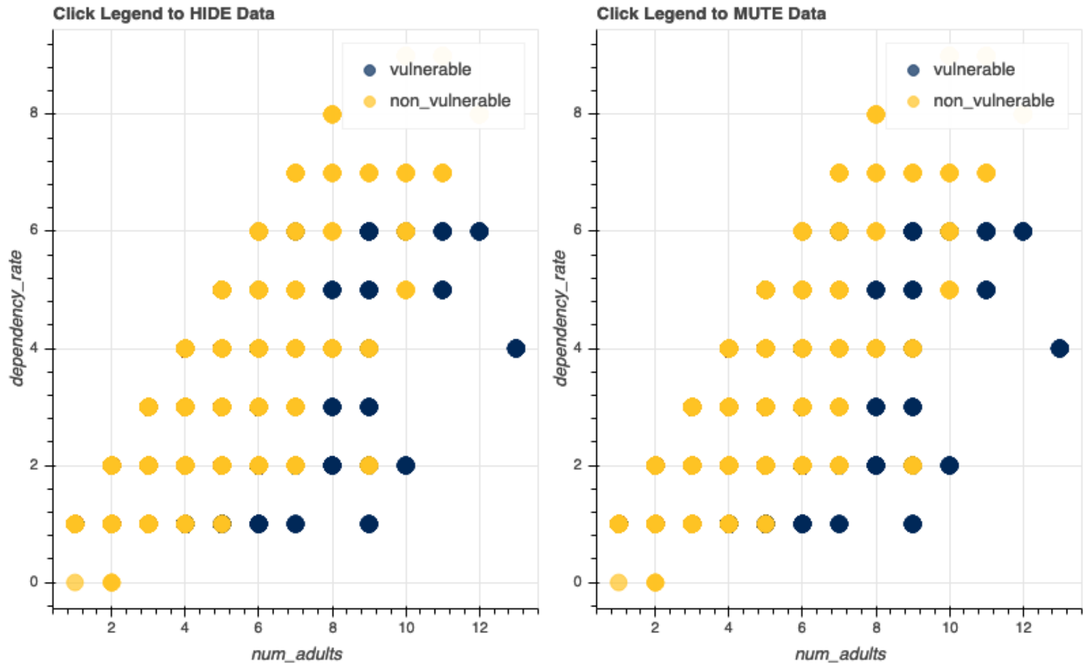
```
hide_fig = figure(**common_figure_kwargs,
                  title = 'Click Legend to HIDE Data')
hide_fig.scatter(**common_circle_kwargs, **common_vul_kwargs)
hide_fig.scatter(**common_circle_kwargs, **common_non_kwargs)

mute_fig = figure(**common_figure_kwargs, title = 'Click Legend to MUTE Data')
mute_fig.circle(**common_circle_kwargs, **common_vul_kwargs,
                muted_alpha = 0.1)
mute_fig.circle(**common_circle_kwargs, **common_non_kwargs,
                muted_alpha = 0.1)
```

Highlighting data using labels (cont'd)

- Add interactivity to the legend

```
hide_fig.legend.click_policy = 'hide'
mute_fig.legend.click_policy = 'mute'

# Visualize the graph.
show(row(hide_fig, mute_fig))
```

Highlighting data using labels (cont'd)



Knowledge check

Link: ***Click here to complete the knowledge check***

Module completion checklist

Objective	Complete
Organize multiple visualization with layouts and configure plot tools	✔
Add interactivity and highlight data using labels	✔

This completes our module

You are now ready to try Tasks 11-23 in the Exercise for this topic