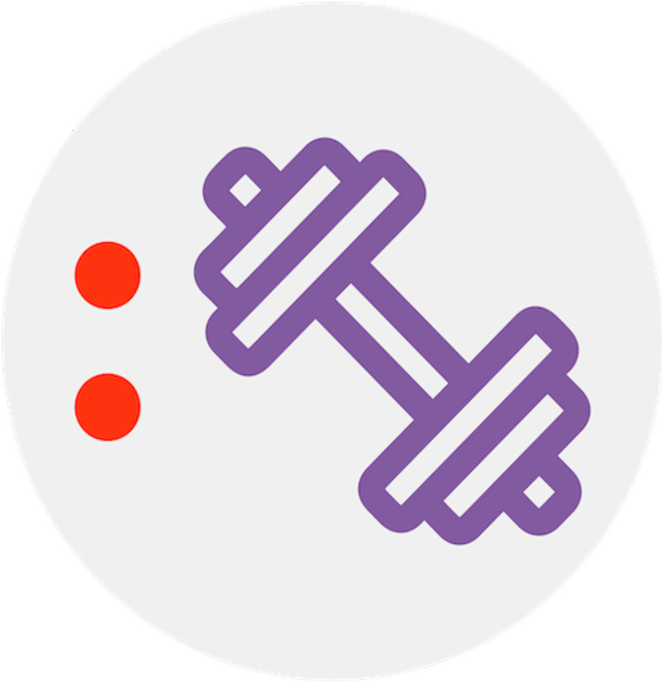
Interactive Visualization with Bokeh - 3

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

Warm up

- People can retain 65% of the information three days after watching an image with data compared to 10% of the information they hear!
- Interactive visualizations go a step further from regular visualization by allowing users to manipulate, explore, and filter data using sliders, buttons, menus, hover effects, etc.
- Let's explore an interactive viz displaying ***The Largest Vocabulary in Hip Hop***
- Take 5 minutes to analyze the data and then share your thought on the following:
  - Is this visualization **easy to interact with**? Does it **help the audience** understand the data?
  - What **kind of interaction** has been applied to this visualization?



Recap

- So far, we have covered the following:
  - Organizing, transforming, and visualizing data with Bokeh
  - Creating maps and simple plots with Bokeh

Module completion checklist

Objective	Complete
Discover different layouts for organizing multiple visualizations	
Demonstrate adding interactivity and highlighting 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 following 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 = "age vs avg_glucose_level",
            width = 400, height = 400,
            tools = tools)

p1.xaxis.axis_label = 'age'
p1.yaxis.axis_label = 'avg_glucose_level'

p1.scatter(df['age'],
            df['avg_glucose_level'],
            size = 20,
            color = "plum",
            alpha = 0.2,
            marker="diamond")
```

Laying out plots and widgets

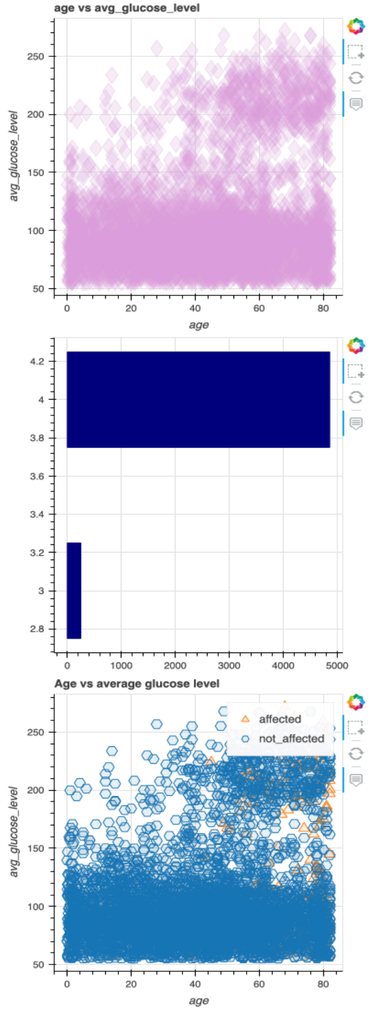
```
# Create another one.
LEVELS = ['not_affected', 'affected']
MARKERS = ['hex', 'triangle']
p2 = figure(width = 400, height = 400, tools = tools)

p2.hbar(y=[0, 1],
        height = 0.2,
        left = 0,
       right = df.stroke.value_counts(),
       color = "navy")

# Create another graph.
p3 = figure(title = "Age vs average glucose level",
            width = 400,
            height = 400,
            tools = tools)
```

```
p3.xaxis.axis_label = 'age'
p3.yaxis.axis_label = 'avg_glucose_level'

p3.scatter("age", "avg_glucose_level",
            source = df,
            legend_group = "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)

- We can display the visualizations in a column format as demonstrated below:

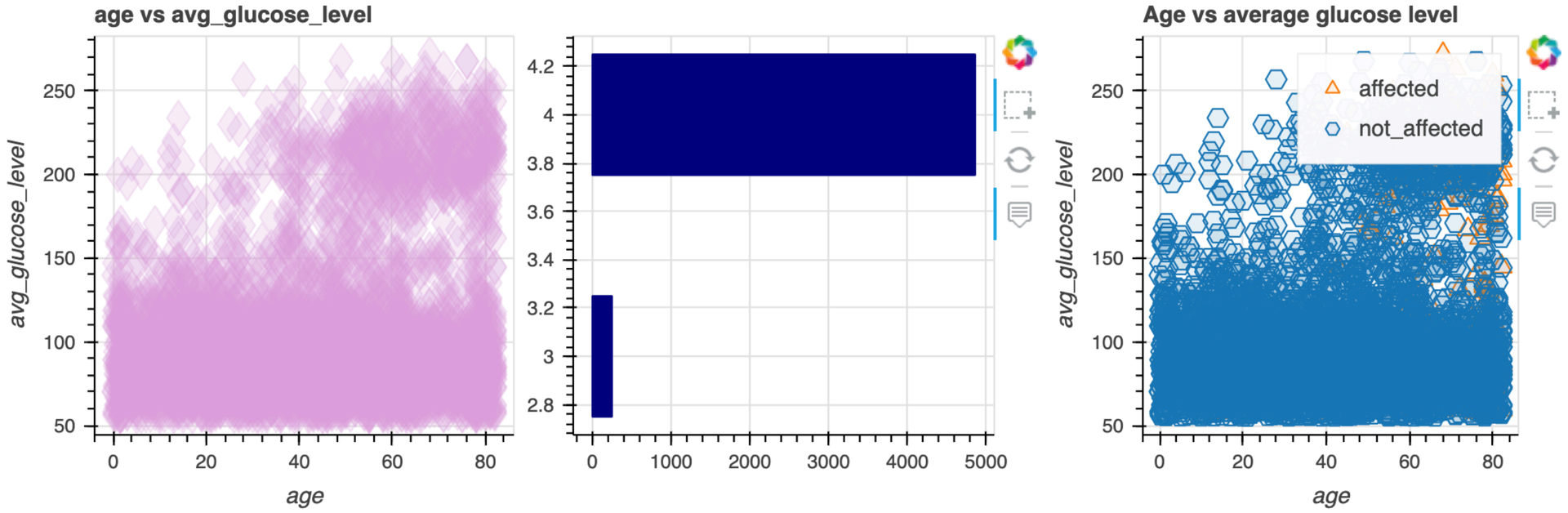
```
# Put the results in a column and show.
show(column(p1, p2, p3))
```



Laying out plots and widgets (cont'd)

- We can choose to organize the layout **row-wise** as demonstrated below:

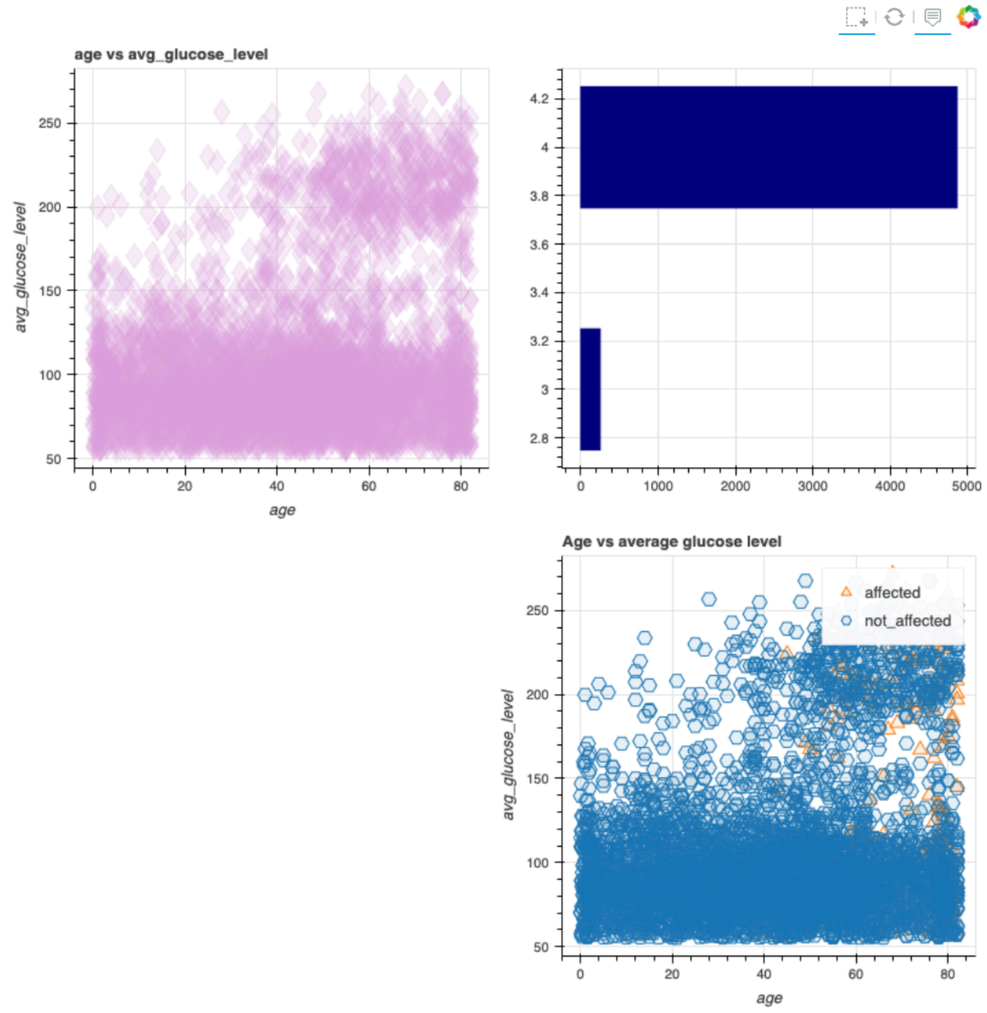
```
# Put the results in a row.
show(row(p1, p2, p3))
```



Laying out plots and widgets (cont'd)

- Alternatively, we can arrange the graphs as subplots
- Here 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 the 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 be passed to glyph methods via the `source` parameter and other parameters (such as the `x` and `y` axes)

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

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

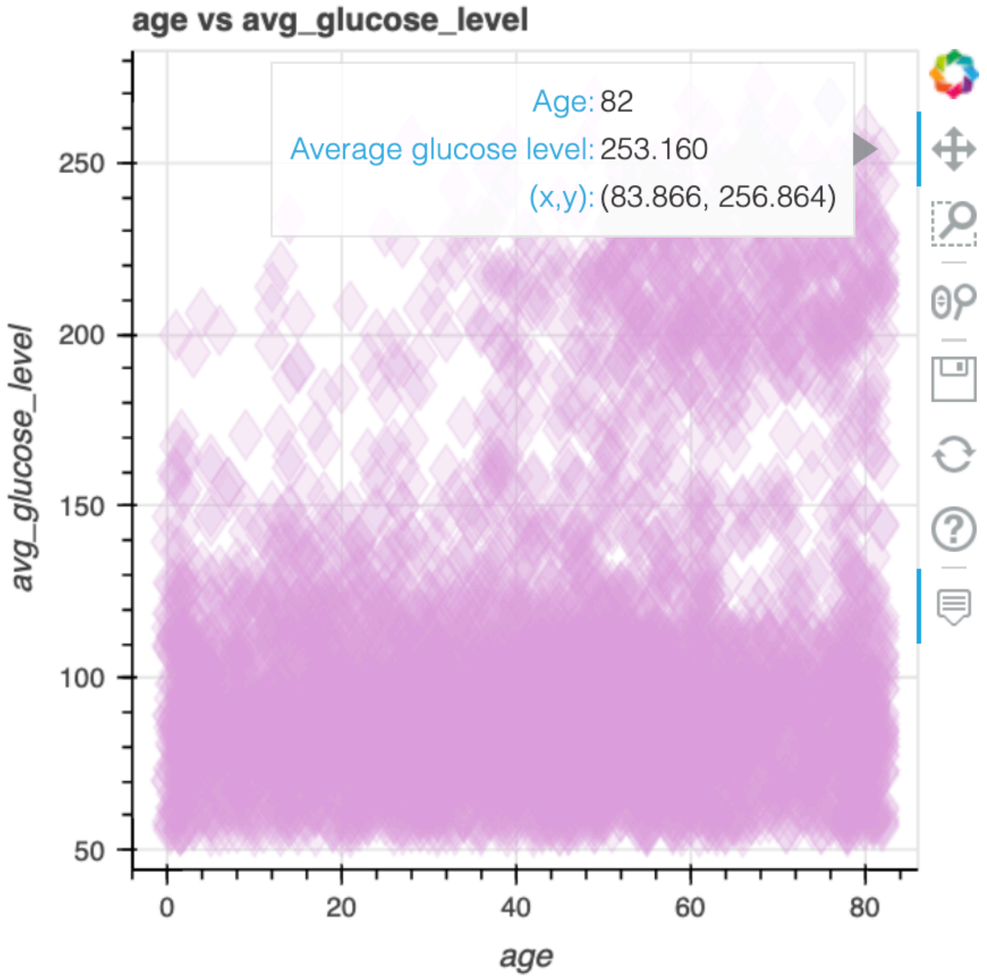
Customizing HoverTool

```
# Hover tool refers to our own data field using @ and
# a position on the graph using $.
hover = HoverTool(tooltips = [('Age', '@age'),
                              ('Average glucose level', '@avg_glucose_level'),
                              ('(x,y)', '($x, $y)')])

p = figure(title = "age vs avg_glucose_level",
           width=400, height=400,
           x_axis_label = 'age',
           y_axis_label = 'avg_glucose_level')

p.scatter('age','avg_glucose_level', source = src, size=20, color = "plum", alpha=0.2,
marker="diamond")
```

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

Customizing HoverTool (cont'd)

```
show(p)
```



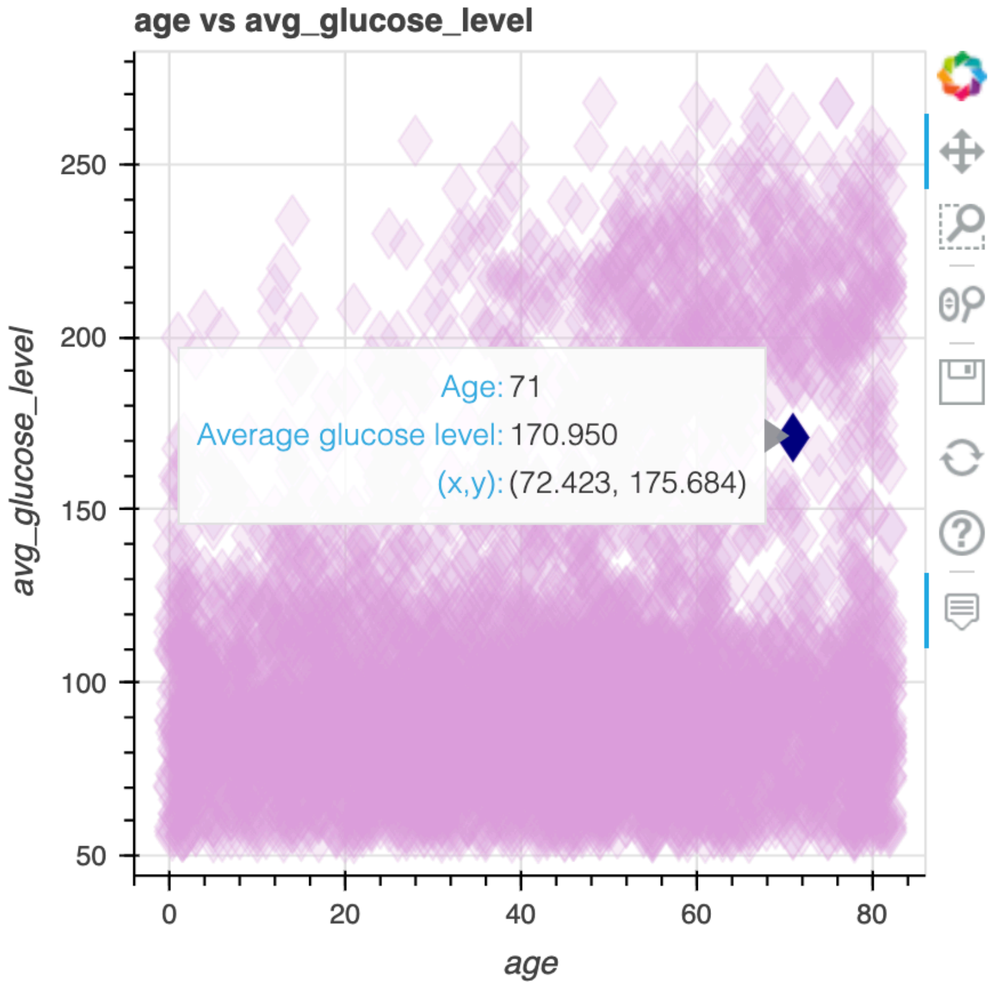
Customizing HoverTool (cont'd)

- **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 $.
hover = HoverTool(tooltips = [('Age', '@age'),
                              ('Average glucose level', '@avg_glucose_level'),
                              ('(x,y)', '($x, $y)')])

p = figure(title = "age vs avg_glucose_level",
           width = 400, height = 400,
          x_axis_label = 'ppl_total',
          y_axis_label = 'num_adults')


p.scatter('age','avg_glucose_level', source = src, size=20, color="plum", alpha=0.2,
          marker="diamond", 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
Discover different layouts for organizing multiple visualizations	✔
Demonstrate adding interactivity and highlighting data using labels	

Highlighting data using HoverTool

- Using `ColumnDataSource()` *(as used for the previous visualization)* sometimes can throw an error, so we will create a new one for each graph

```
# Store the data in a ColumnDataSource.
stroke_cds = ColumnDataSource(df)
```

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

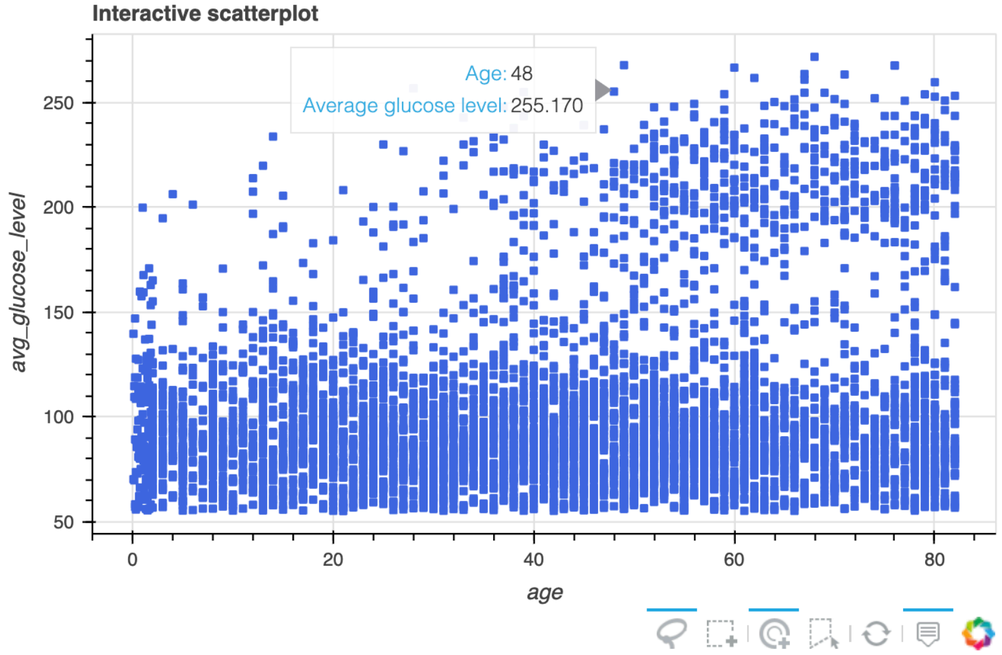
```
# Add square representing each layer.
fig.scatter(
    x='age',
    y='avg_glucose_level',
    source=df,
    color='royalblue',
    marker='square',
    selection_color='deepskyblue',
    nonselection_color='lightgray',
    nonselection_alpha=0.3
)
```

Customizing HoverTool

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

```
# Format the tooltip.
tooltips = [
            ('Age','@age'),
            ('Average glucose level', '@avg_glucose_level')
           ]

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

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

Customizing HoverTool (cont'd)

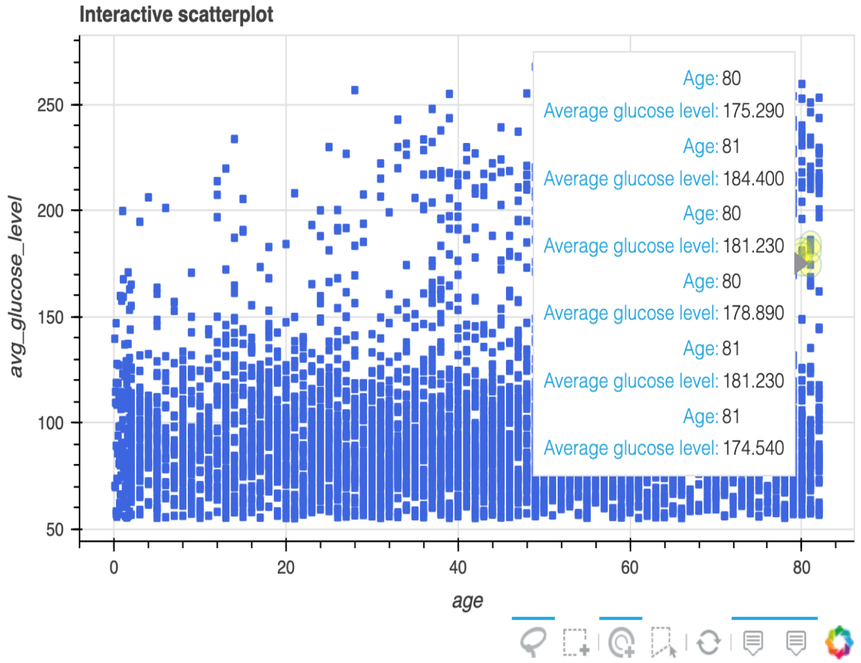


Customizing HoverTool (cont'd)

- 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

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

```
# Format the tooltip.
tooltips = [
            ('Age','@age'),
            ('Average glucose level', '@avg_glucose_level')
           ]

hover_glyph = fig.scatter(
    x='age',
    y='avg_glucose_level',
    source=costa_cds,
    size=15,
    marker='circle',
    alpha=0,
    hover_fill_color='yellow',
    hover_alpha=0.2
)

# Add the HoverTool to the figure.
```



Highlighting data using labels

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

```
stroke_labels = ColumnDataSource(df)

# Create a view for the 'affected' group
vul_view = CDSView(filter=GroupFilter(column_name='Target_class', group='affected'))
```

```
# Create a view for the 'not_affected' group
nonvul_view = CDSView(filter=GroupFilter(column_name='Target_class', group='not_affected'))
```

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 = {
    'width': 400,
    'height':500,
    'x_axis_label': 'age',
    'y_axis_label' : 'avg_glucose_level',
    'toolbar_location': None}
common_circle_kwargs = {
    'x': 'age',
    'y': 'avg_glucose_level',
    'source': stroke_labels,
    'size': 12,
    'alpha': 0.7,}
common_vul_kwargs = {
    'view': vul_view,
    'color': '#002859',
    'legend_label': 'affected'}
common_non_kwargs = {
    'view': nonvul_view,
```

Highlighting data using labels (cont'd)

- Create two figures and draw the data

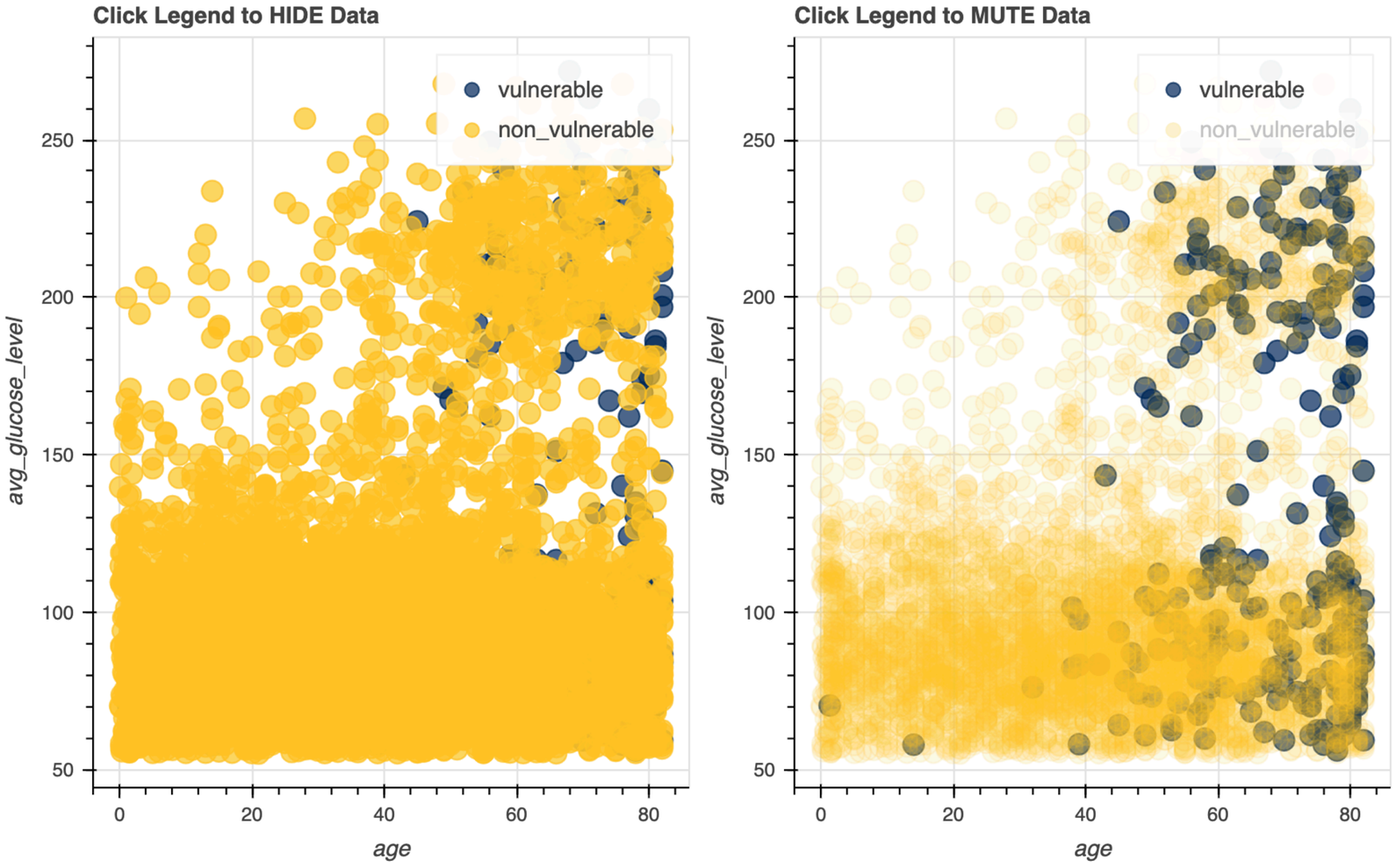
```
hide_fig = figure(**common_figure_kwargs,
                  title = 'Click Legend to HIDE Data')
hide_fig.scatter(**common_circle_kwargs, **common_vul_kwargs)
hide_fig.scatter(**common_circle_kwargs, **common_non_kwargs)

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

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

Module completion checklist

Objective	Complete
Discover different layouts for organizing multiple visualizations	✔
Demonstrate adding interactivity and highlighting data using labels	✔

Congratulations on completing this module!

You are now ready to try tasks 9-21 in the Exercise for this topic