DATA SOCIETY®

Advanced visualization in python

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	
Generate your first figure and add glyphs to it	
Transform Costa Rican data for visualizations	
Create simple plots using Bokeh	
Organize multiple visualization with layouts and configure plot tools	
Add interactivity and highlight data using labels	
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/Desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

```
# Create a plot directory to save our plots
plot_dir = main_dir + "/plots"
```

Loading packages

- Load the packages we will be using

```
import pandas as pd
import numpy as np
import os
```

```
from bokeh.io import output_notebook
from bokeh.plotting import figure, output_file, show, output_notebook, save
from bokeh.transform import factor_cmap, factor_mark
from bokeh.layouts import column, row, gridplot
from bokeh.models import HoverTool, ColumnDataSource, NumeralTickFormatter, GroupFilter, CDSView
import ipywidgets as widgets
from ipywidgets import interact, interact_manual
```

Working directory

- Set working directory to `data_dir`

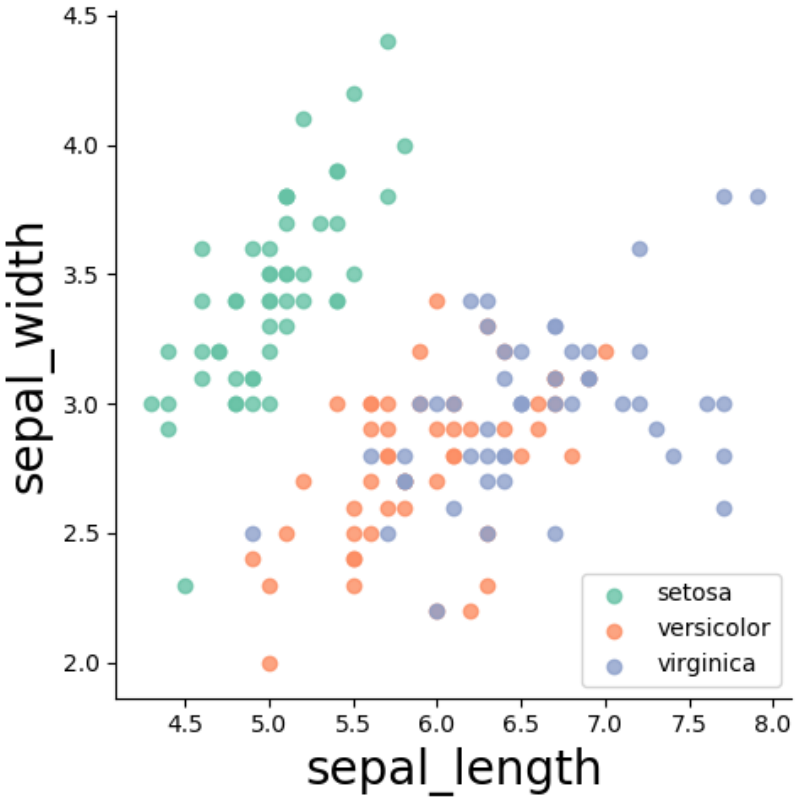
```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

Recap: why visualize data?

- To gain **insights on patterns, trends and correlations** that might not be detected otherwise
- Simple way to **convey concepts** and provide visual access to **large amounts of complex data**
- Python has multiple graphing libraries with a lot of great features!



Working with Costa Rica household poverty data

- Today we will:
  - Build simple plots on the dataset using Bokeh
  - Build interactive plots to analyze the data
  - Generate insights on the data by interpreting the plots

Dataset in your exercises

- Working with **Chicago census data**
  - Build simple plots to detect patterns in the data
  - Build complex plots to understand the socioeconomic indicators in the data

Visualizing data with Bokeh

- `bokeh` an interactive visualization library that targets modern web browsers for presentation
- Bokeh offers two interfaces to users:
  - `bokeh.models`: low-level interface with the most flexibility (most users will not use this level of interface to assemble plots directly)
  - `bokeh.plotting`: higher-level interface centered around composing visual glyphs
- The `bokeh.plotting` interface is handy when we need to customize the output more by adding more data series, glyphs, and so on

Plotting with Bokeh

- The basic steps to creating plots with the `bokeh.plotting` interface are:
- **Prepare some data:**
  - Could be numpy arrays or pandas series
- **Tell Bokeh where to generate output:**
  - In this case, it's `output_notebook()` for use in Jupyter notebooks
- **Call `figure()`**
  - This creates a plot with default options and easy customization of title, tools, and axes labels

Plotting with Bokeh

- **Add renderers**
  - We use functions specifying visual customizations like colors, legends, and widths
- **Ask Bokeh to `show()` or `save()` the results:**
  - These functions save the plot to an HTML file and optionally display it in a browser
- The last two steps can be repeated to create more than one plot

Output methods using Bokeh

- Common methods to view Bokeh plots are:
- `output_file()`
  - Generates HTML documents for Bokeh visualizations
- `output_notebook()`
  - Displays inline visualizations in Jupyter notebook

Using Bokeh with plotly

- The major concept of Bokeh is that graphs are built up **one layer at a time**
- **We start out by creating a figure**
- **Then we add elements, called glyphs, to the figure**
- Glyphs can take on many shapes depending on the desired use: circles, lines, patches, bars , arcs, and so on

Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	
Transform Costa Rican data for visualizations	
Create simple plots using Bokeh	
Organize multiple visualization with layouts and configure plot tools	
Add interactivity and highlight data using labels	
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Bokeh: simple plot

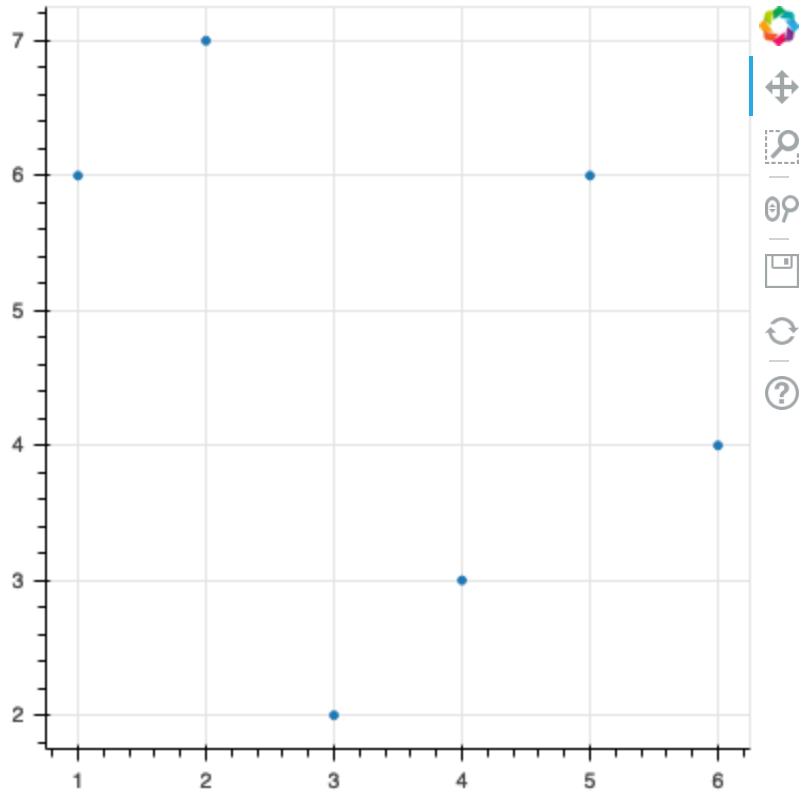
- We will create simple plots at first using data points assigned to variables `x_values` and `y_values`

```
# Input the sample data below.
x_values = [1, 2, 3, 4, 5, 6]
y_values = [6, 7, 2, 3, 6, 4]
```

Bokeh: simple plot

- First, we make a plot using the `figure()` method
- Then, we append our glyphs to the plot by calling the appropriate method and passing in data
- Finally, we show our plot

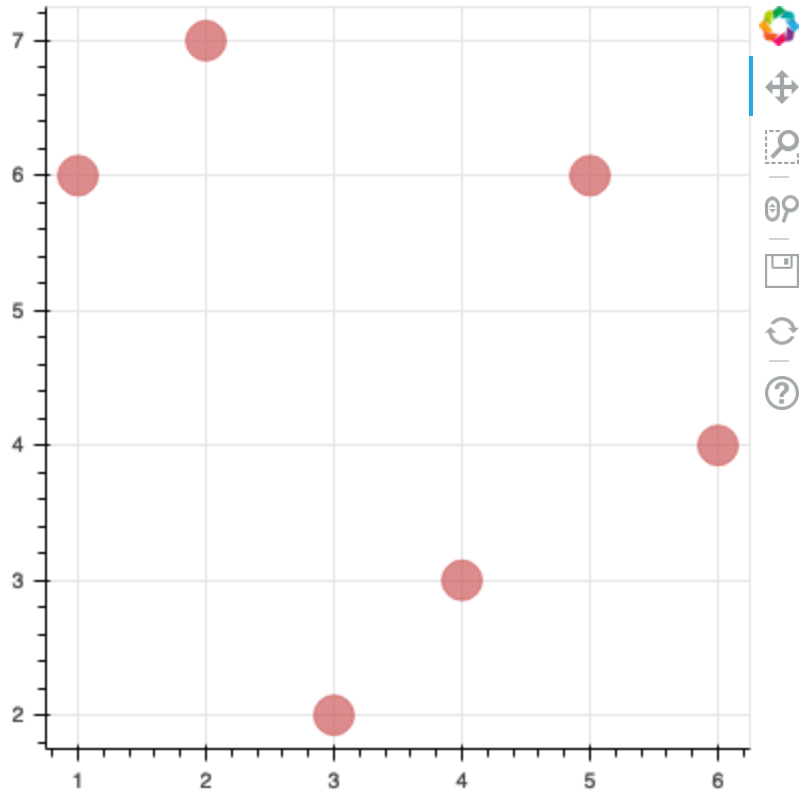
```
p = figure()
p.circle(x = x_values, y = y_values)
show(p)
```



Bokeh: add size,color and opacity

- We can now create the same circle glyph with a size, color and alpha

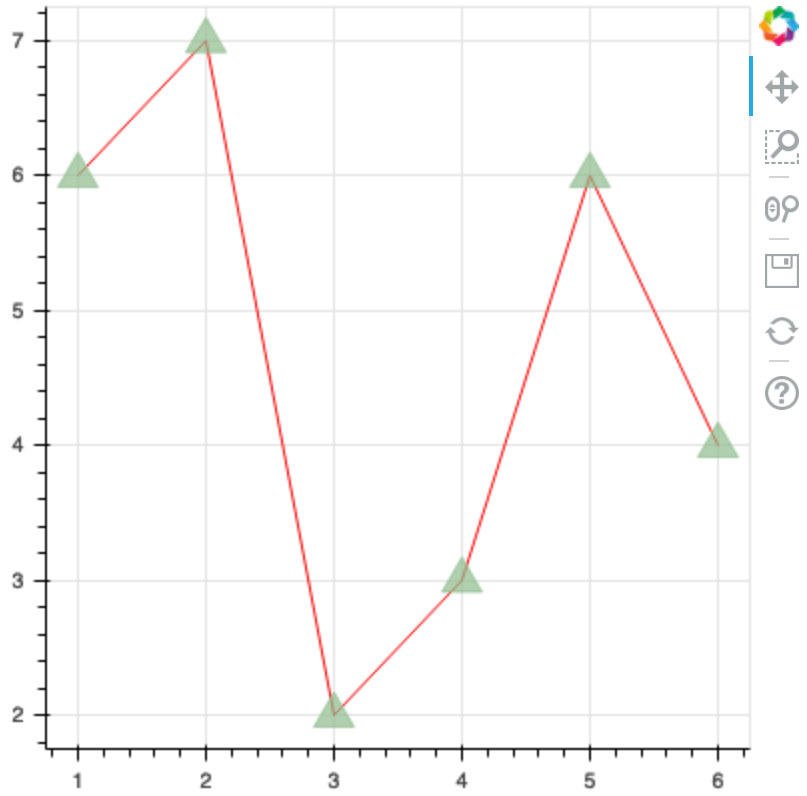
```
# Create the blank plot.
p = figure(plot_width = 400, plot_height = 400)

# Add a circle glyph with a size, color, and
alpha.
p.circle(x_values,
         y_values,
         size = 20,
         color = "indianred",
         alpha = 0.7)
show(p)
```



Bokeh: triangle glyph

- This time, two glyphs are added to the graph

```
p = figure(plot_width = 400, plot_height = 400)

p.line(x_values,
       y_values,
       color = 'red')

p.triangle(x_values,
           y_values,
           size = 20,
           color = "darkseagreen",
           alpha = 0.7)
show(p)
```



Bokeh: triangle glyph

- There are a lot more marker types you can try out
- You can see examples of plots with different markers ***here***

- `asterisk()`
- `circle()`
- `circle_cross()`
- `circle_x()`
- `cross()`
- `dash()`
- `diamond()`
- `diamond_cross()`
- `inverted_triangle()`
- `square()`
- `square_cross()`
- `square_x()`
- `triangle()`
- `x()`

Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	
Create simple plots using Bokeh	
Organize multiple visualization with layouts and configure plot tools	
Add interactivity and highlight data using labels	
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Load the dataset

- We will load in our Costa Rican poverty dataset as `household_poverty`
- Let's load the entire dataset
- For visualizations, we will be taking a specific subset
- We are now going to use the function `read_csv` to read in our `costa_rica_poverty` dataset

```
household_poverty = pd.read_csv("costa_rica_poverty.csv")
print(household_poverty.head())
```

```
  household_id        ind_id  rooms  ...  age  Target  monthly_rent
0    21eb7fcc1  ID_279628684      3  ...   43       4      190000.0
1    0e5d7a658  ID_f29eb3ddd      4  ...   67       4      135000.0
2    2c7317ea8  ID_68de51c94      8  ...   92       4           NaN
3    2b58d945f  ID_d671db89c      5  ...   17       4      180000.0
4    2b58d945f  ID_d56d6f5f5      5  ...   37       4      180000.0

[5 rows x 84 columns]
```

- The entire dataset consists of `9557` observations and `84` variables

Subsetting data

- Let's subset our data so that we have the variables we need
- Let's name this subset `costa_viz`

```
costa_viz = household_poverty[['household_id',
                               'ppl_total',
                               'dependency_rate',
                               'num_adults',
                               'rooms',
                               'age',
                               'monthly_rent',
                               'Target']]
print(costa_viz.head())
```

```
  household_id  ppl_total  dependency_rate  ...  age  monthly_rent  Target
0     21eb7fcc1          1               37  ...   43      190000.0       4
1     0e5d7a658          1               36  ...   67      135000.0       4
2     2c7317ea8          1               36  ...   92           NaN       4
3     2b58d945f          4               38  ...   17      180000.0       4
4     2b58d945f          4               38  ...   37      180000.0       4

[5 rows x 8 columns]
```

Remove labels

- Let's prepare the data for visualizations by removing any labels, removing the `household_id` variable, and keeping the remaining variables

```
costa_viz = costa_viz.drop('household_id', axis = 1)
print(costa_viz.head())
```

```
   ppl_total  dependency_rate  num_adults  rooms  age  monthly_rent  Target
0          1               37           1      3   43      190000.0       4
1          1               36           1      4   67      135000.0       4
2          1               36           1      8   92           NaN       4
3          4               38           2      5   17      180000.0       4
4          4               38           2      5   37      180000.0       4
```

Data prep: clean NAs

- Depending on **subject matter**, missing values might mean something
- Let's define how to handle **columns with NAs:**
  - Drop columns that contain any `NA`s
  - Drop columns with a certain % of `NA`s
  - Impute missing values
  - Convert column with missing values to categorical
- Let's look at the count of NAs by column first:

```
print(costa_viz.isnull().sum())
```

```
ppl_total              0
dependency_rate        0
num_adults             0
rooms                  0
age                    0
monthly_rent        6860
Target                 0
dtype: int64
```

Data cleaning: NAs

- We'll keep `monthly_rent` and impute missing values using the mean of the column

```
# Set the dataframe equal to the imputed dataset.
costa_viz = costa_viz.fillna(costa_viz.mean())
# Check how many values are null in monthly_rent.
print(costa_viz.isnull().sum())
```

```
ppl_total          0
dependency_rate    0
num_adults         0
rooms              0
age                0
monthly_rent       0
Target             0
dtype: int64
```

Converting the target variable

- Let's convert poverty to a target variable with two levels, which will help to balance it out
- The four original levels would also increase the complexity of the visualizations and the code
- For this reason, we will convert levels 1,2 and 3 to `vulnerable` and 4 to `non_vulnerable`
- The levels translate to 1, 2 and 3 as being **vulnerable** households
- Level 4 is **non vulnerable**

```
costa_viz['Target_class'] = np.where(costa_viz['Target'] <= 3, 'vulnerable','non_vulnerable')
```

```
print(costa_viz['Target_class'].head())
```

```
0    non_vulnerable
1    non_vulnerable
2    non_vulnerable
3    non_vulnerable
4    non_vulnerable
Name: Target_class, dtype: object
```

Knowledge check 1

Exercise 1

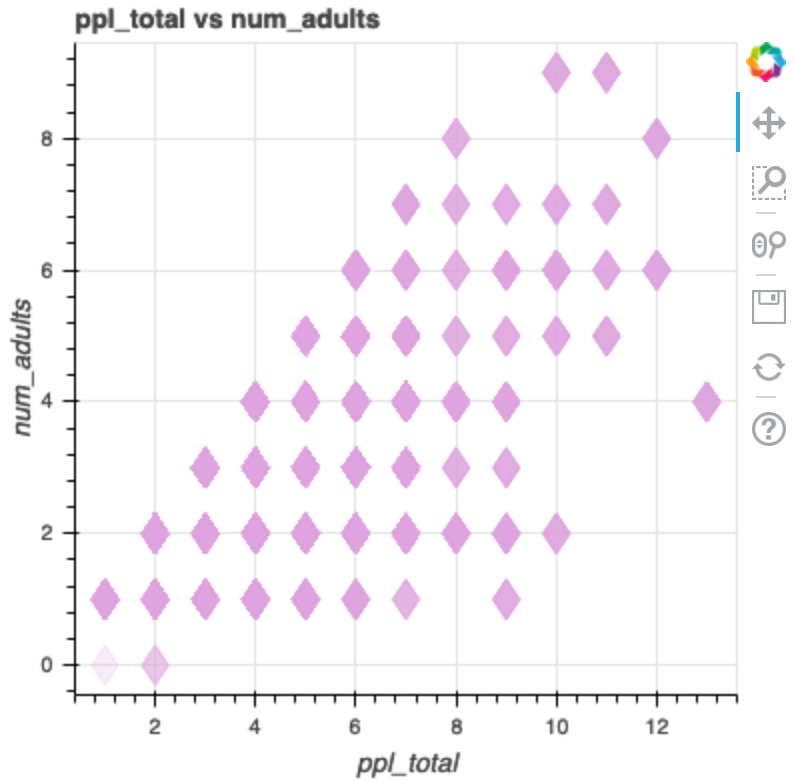
Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	✔
Create simple plots using Bokeh	
Organize multiple visualization with layouts and configure plot tools	
Add interactivity and highlight data using labels	
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Use Costa Rican data for plots

- We're ready to create plots with `costa_viz`
- First, let's investigate the relationship between the total number of people and the number of adults

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

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



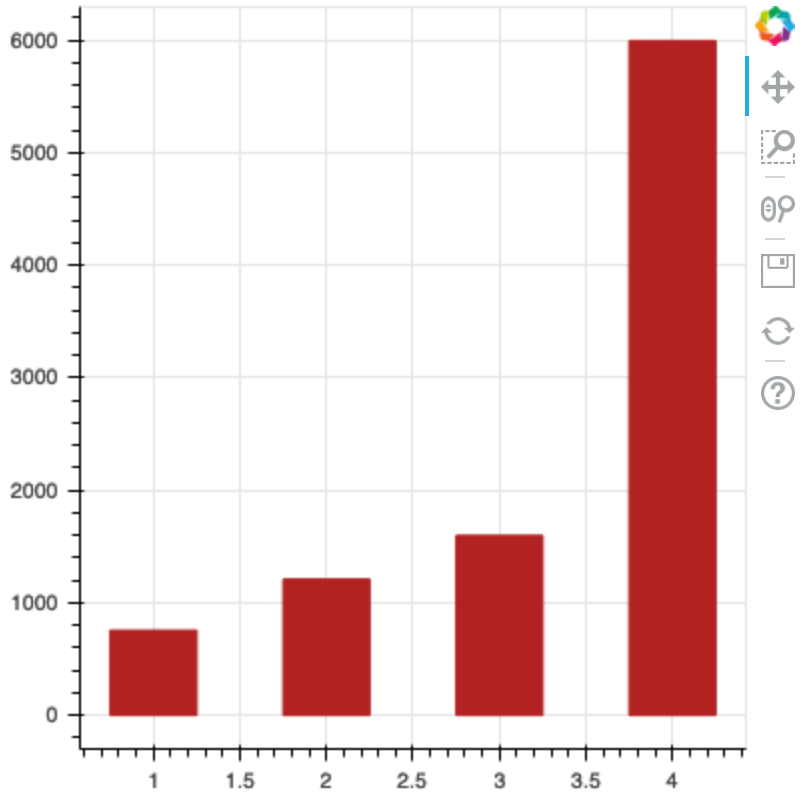
vbar() and hbar()

- To see the count of the four levels, we will use the original `Target` variable

```
costa_viz.Target.value_counts()
```

```
4    5996
2    1597
3    1209
1     755
Name: Target, dtype: int64
```

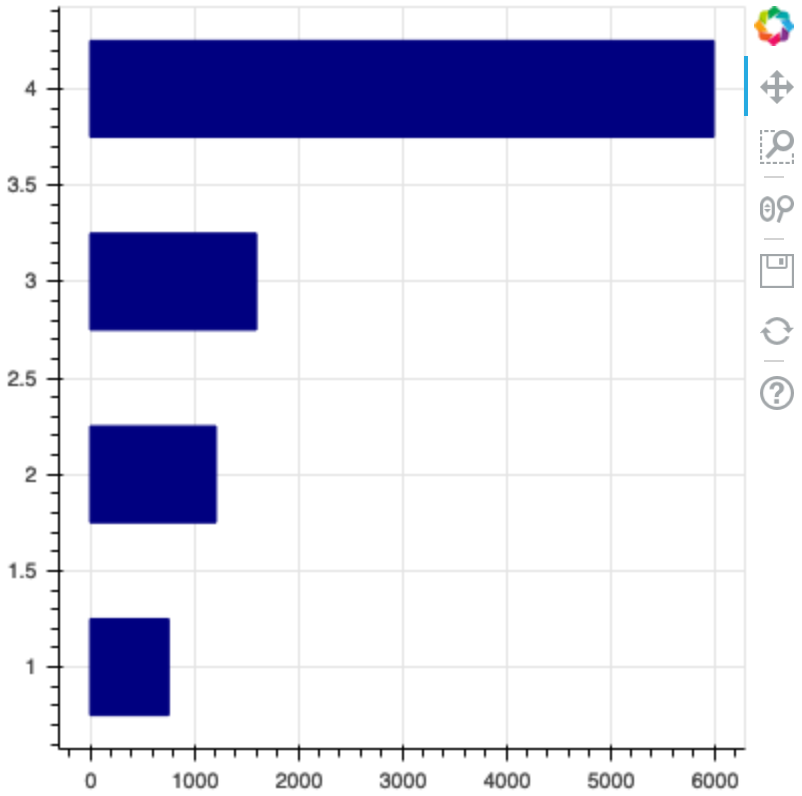
```
p = figure(plot_width=400, plot_height=400)

p.vbar(x = [4, 3, 2, 1],
       width = 0.5,
       bottom = 0,
       top =
costa_viz.Target.value_counts(),
       color = "firebrick")

show(p)
```



vbar() and hbar()

- Similarly, horizontal bar charts can be created using `.hbar()`
- Now, we can look at the distribution of levels, which is a best practice before analysis

```
p = figure(plot_width = 400, plot_height = 400)

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

show(p)
```



Markers for categorical data

- What if we want to see **three or more variables in one visualization**?
- We can map categorical data to marker types
- This example shows the use of `factor_mark()` to display different markers or different categories in the input data
- It also demonstrates the use of `factor_cmap()` to colormap those same categories

```
LEVELS = ['non_vulnerable', 'vulnerable']
MARKERS = ['triangle', 'hex']

p = figure(title = "Number of adults vs
dependency rate",
           x_axis_label = 'num_adults',
           y_axis_label = 'dependency_rate')
```

Markers for categorical data

```
p.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))

show(p)
```

Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	✔
Create simple plots using Bokeh	✔
Organize multiple visualization with layouts and configure plot tools	
Add interactivity and highlight data using labels	
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Laying out plots and plot tools

- 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

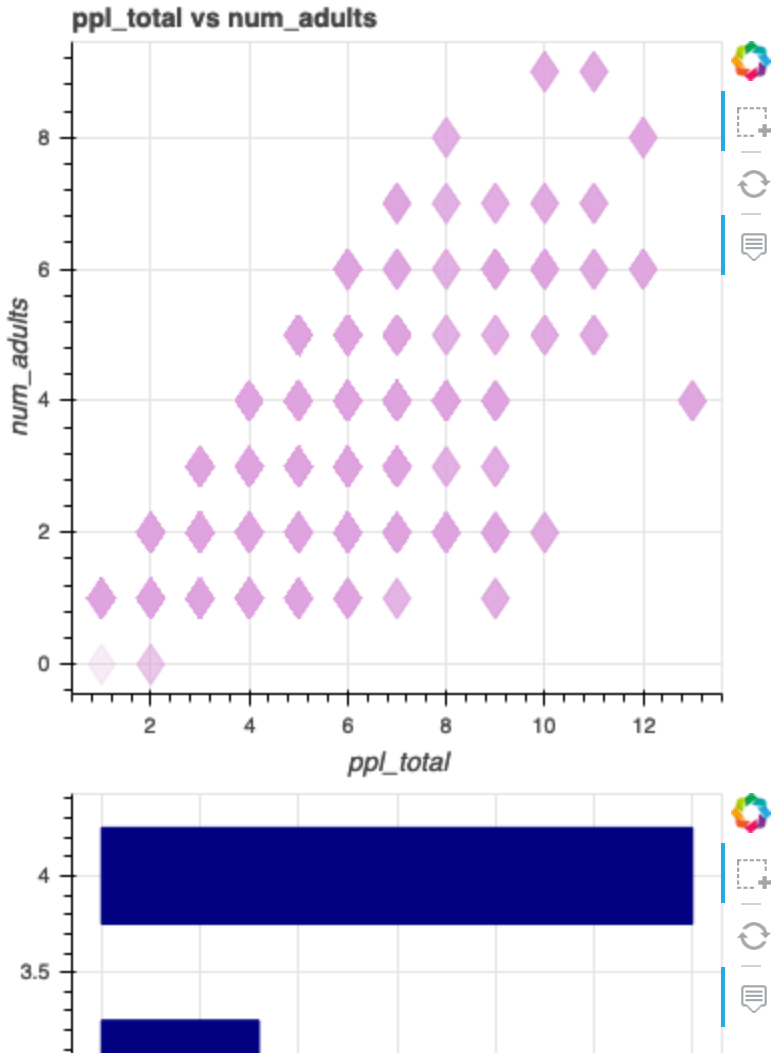
```
tools = ["box_select", "hover", "reset"]

# 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.
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

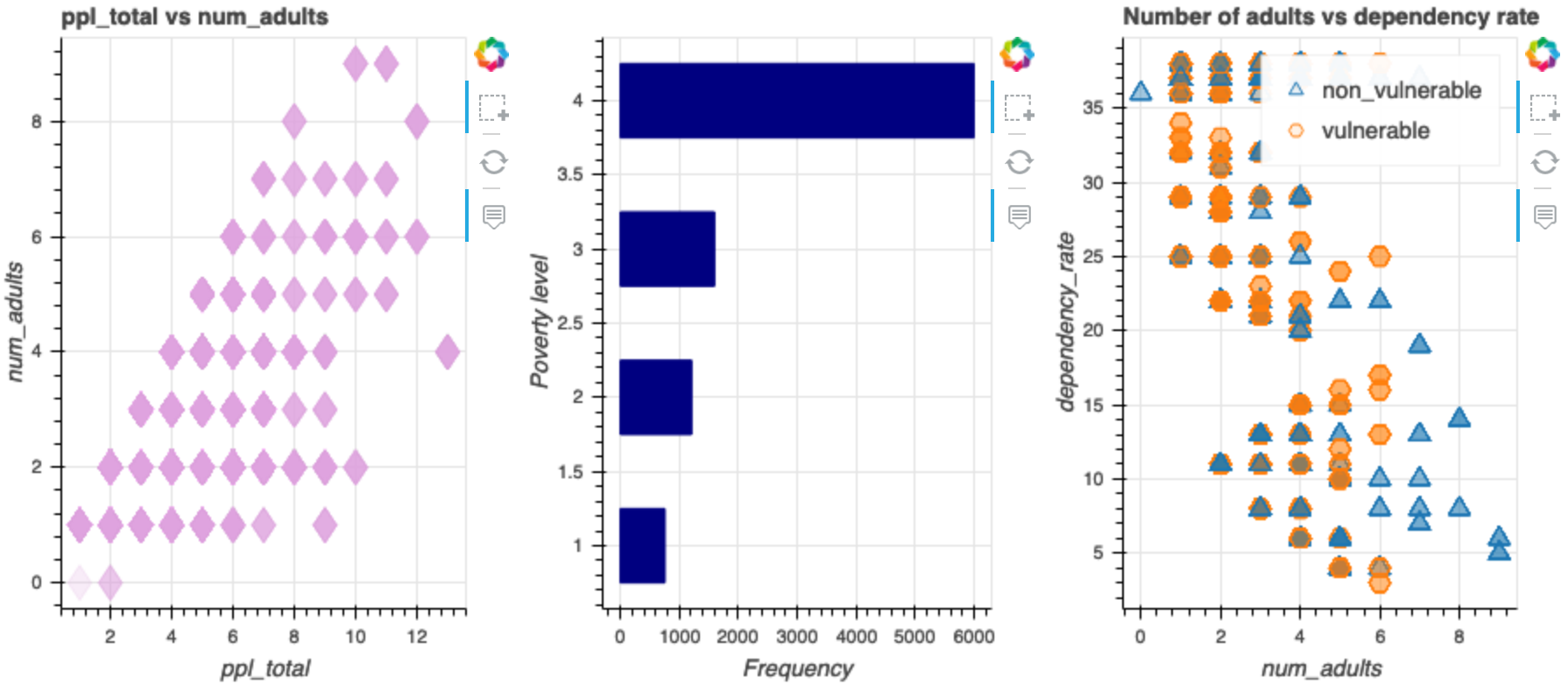
```
# Put the results in a column and show.
show(column(p1, p2, p3))
```



Laying out plots and widgets

- Row-wise layout

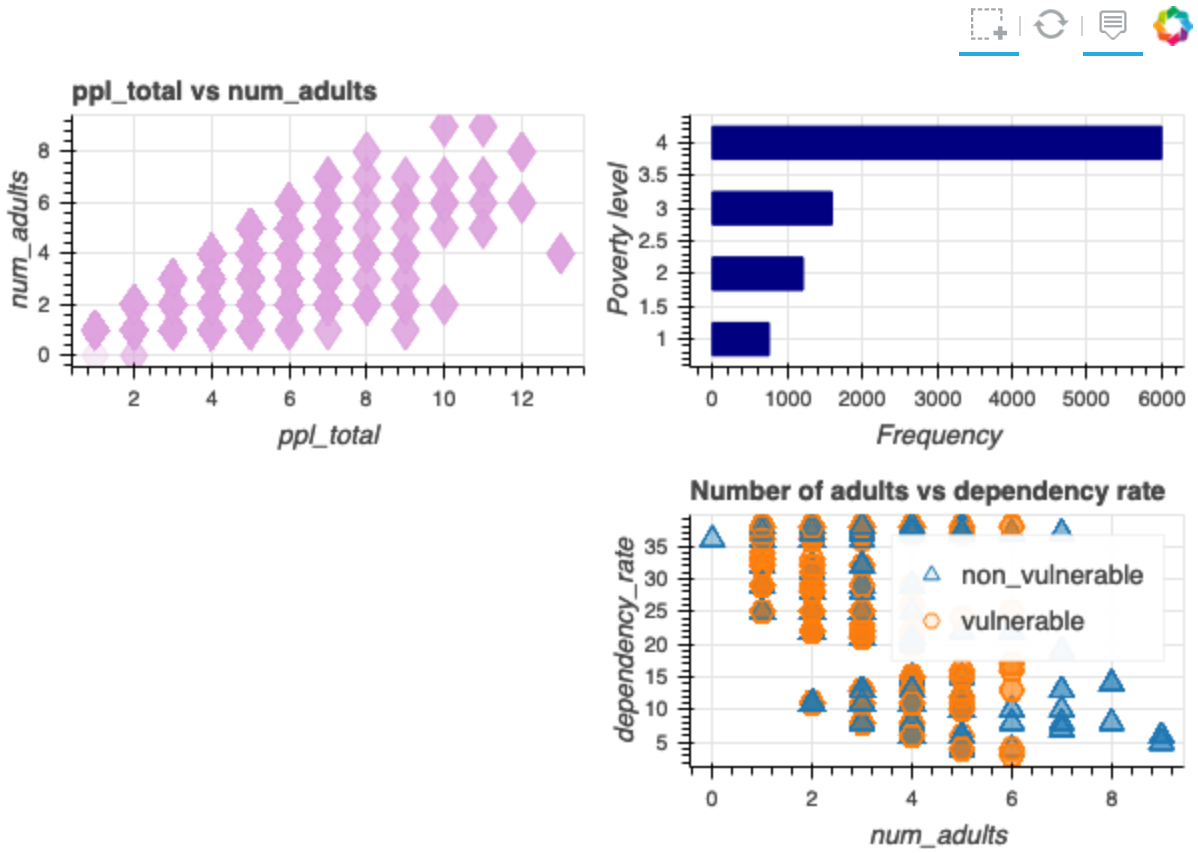
```
# Put the results in a row.
show(row(p1, p2, p3))
```



Laying out plots and widgets

- 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

```
# The Hovertool 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

```
show(p)
```

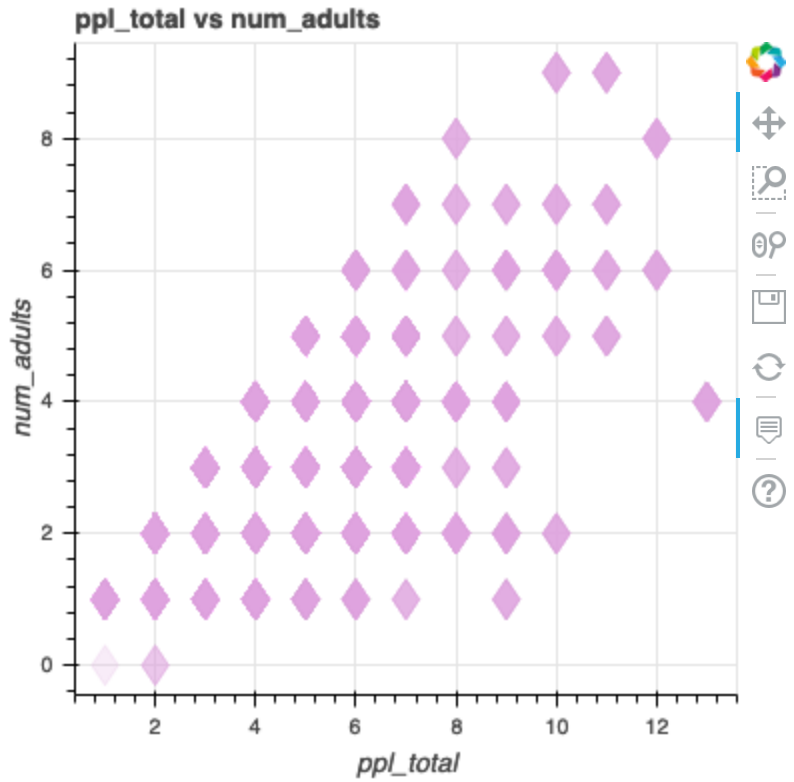
Customizing Hovertool

- 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

```
show(p)
```



Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	✔
Create simple plots using Bokeh	✔
Organize multiple visualization with layouts and configure plot tools	✔
Add interactivity and highlight data using labels	
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Highlighting data using HoverTool()

- Using 'ColumnDataSource()' from your previous visualization can cause an error, so let's create a new one for each graph

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

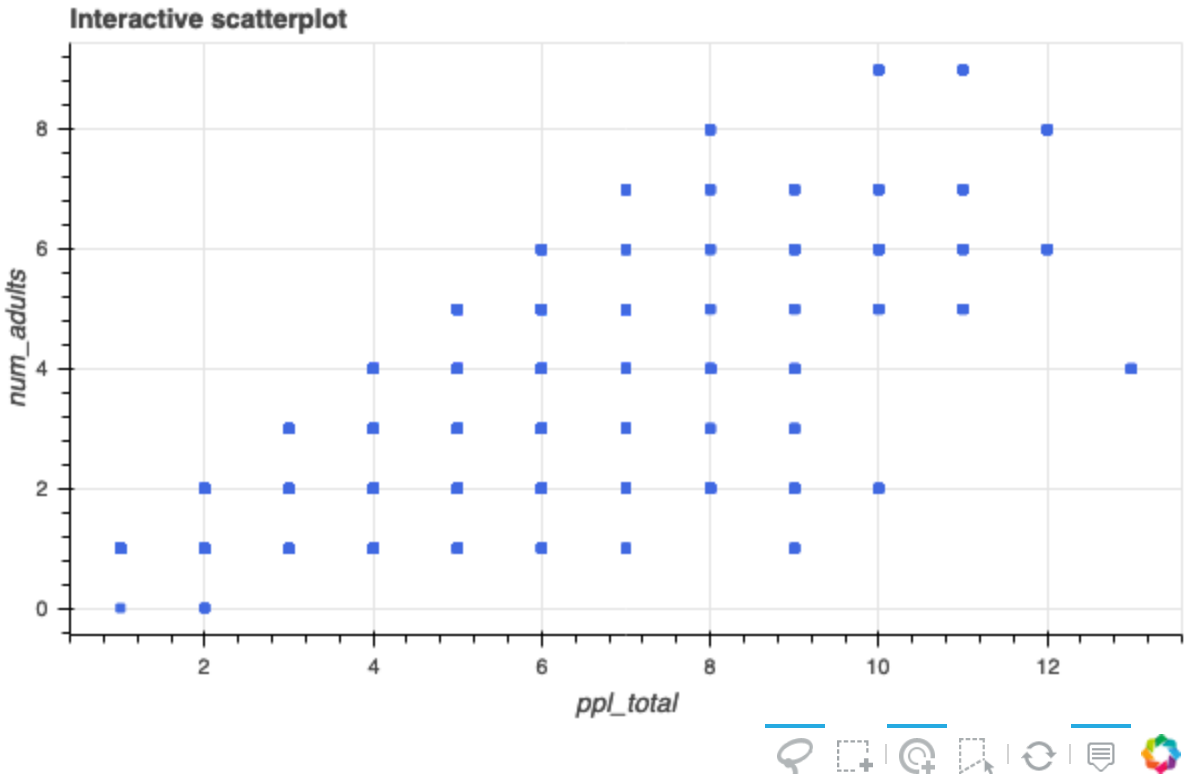
```
# Specify the selection tools to be made available.
select_tools = ['box_select', 'lasso_select', 'poly_select', 'tap', 'reset']

# 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)
```

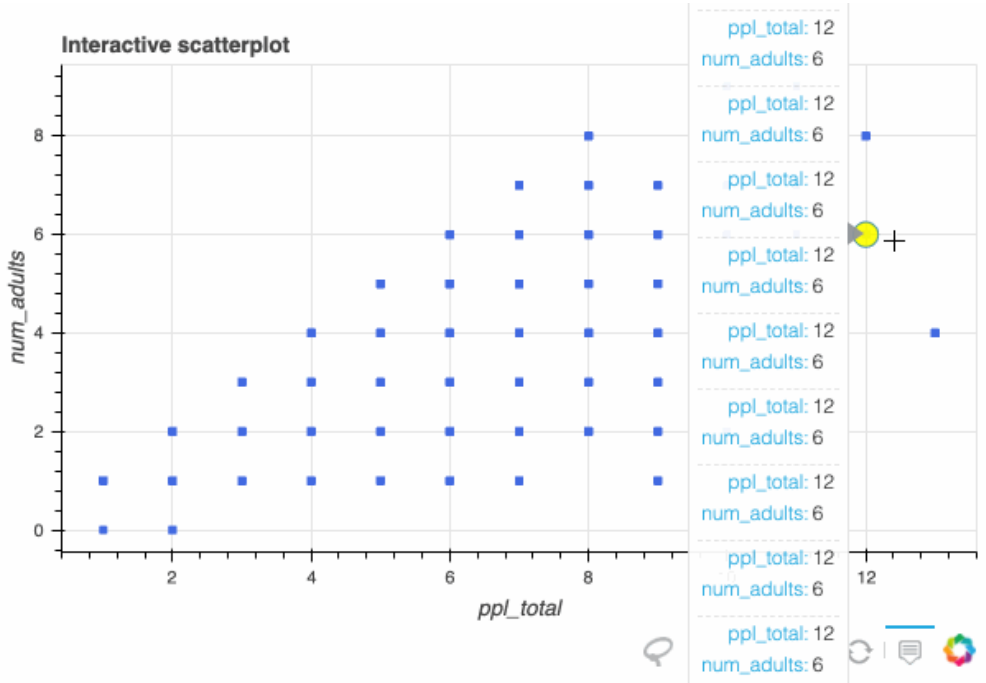
Customizing Hovertool



Customizing Hovertool

- 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

- 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,
    '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

- Create the 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)
```

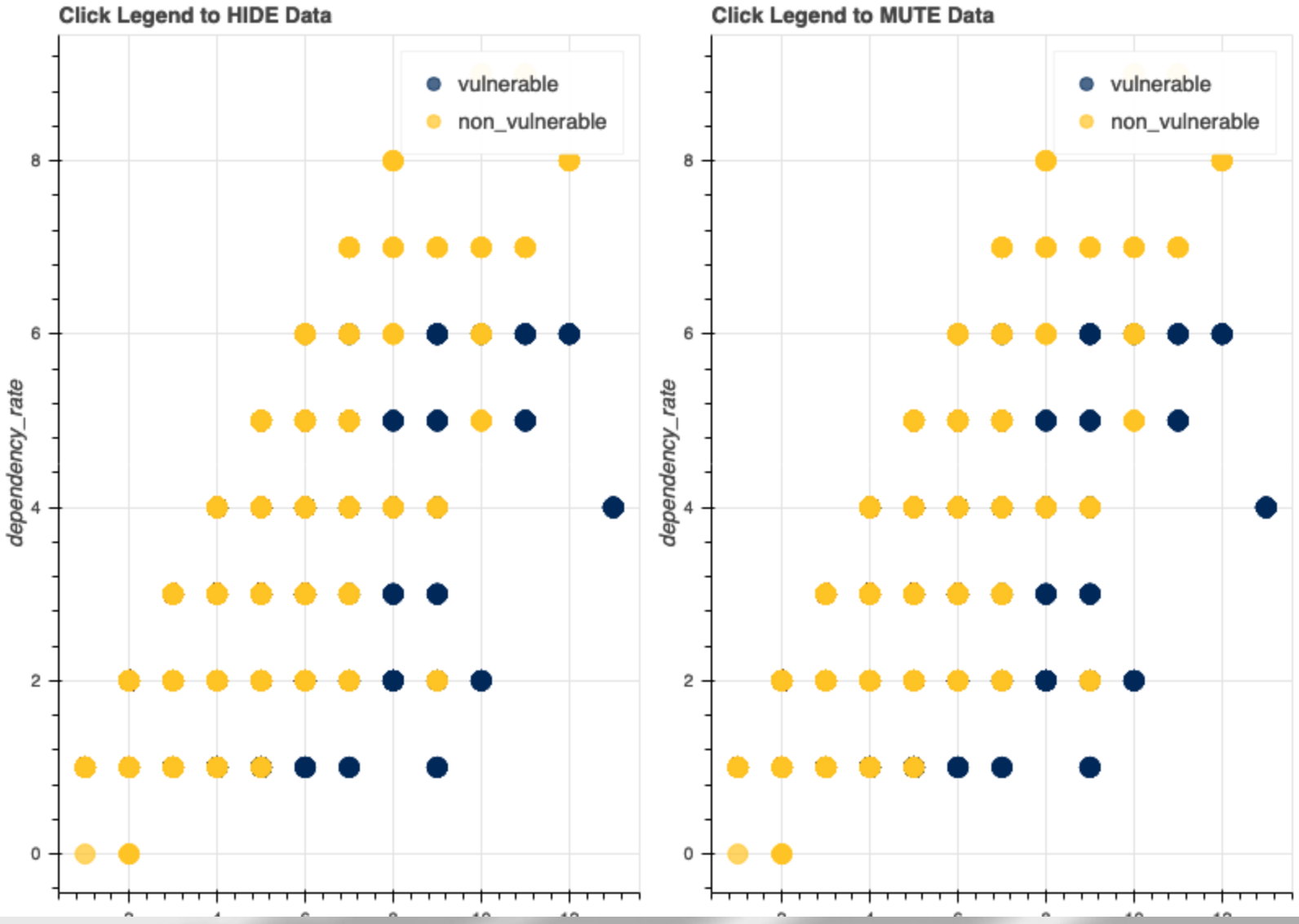
```
mute_fig.circle(**common_circle_kwargs, **common_non_kwargs,
                muted_alpha = 0.1)
```

Highlighting data using labels

- 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



Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	✔
Create simple plots using Bokeh	✔
Organize multiple visualization with layouts and configure plot tools	✔
Add interactivity and highlight data using labels	✔
Integrate widgets to Bokeh and plotly graphs	
Save your graphs	

Adding widgets to graphs

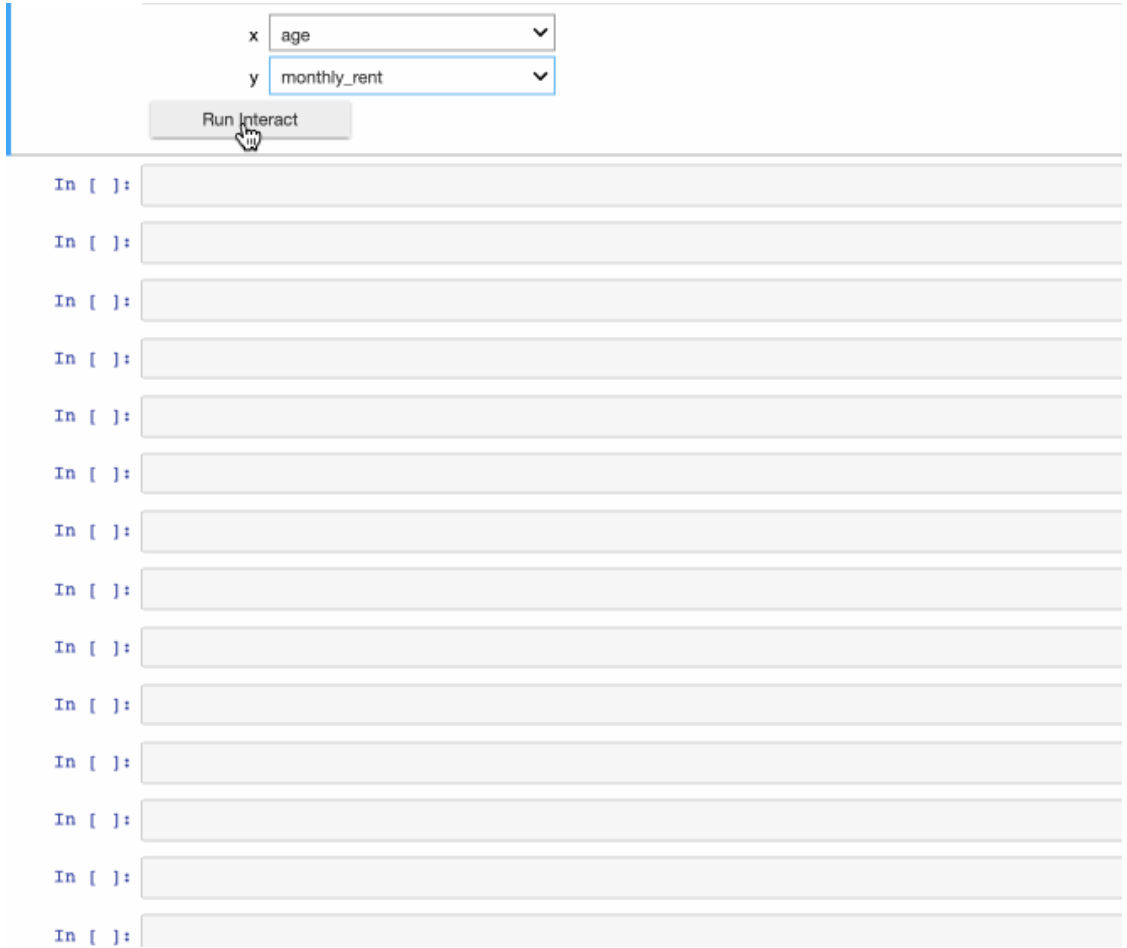
- `ipywidgets` library allows us to turn Jupyter Notebooks from static documents into interactive dashboards
- Widgets are handy when we wish to **change inputs without needing to rewrite or rerun code**
- You can read the documentation on `ipywidgets` ***here***

Adding widgets to graphs

- The Costa Rican dataset has many inputs which can slow the plot rendering, so let's subset the variables we wish to use

```
@interact_manual
def scatter_plot(x = list(costa_viz.columns),
                 y = list(costa_viz.columns)):
    p = figure(title = f'{x} vs {y}',
               x_axis_label = x,
               y_axis_label = y)

    p.circle(x = x,
             y = y,
             source = df,
             size = 20, color = "thistle", alpha = 0.2)
    show(p)
```

Adding widgets to graphs



Adding widgets to plotly graphs

- We can add `colorscale` and `themes` to our widget in cufflinks graphs
- First, we need to run the initialization steps for offline plotting

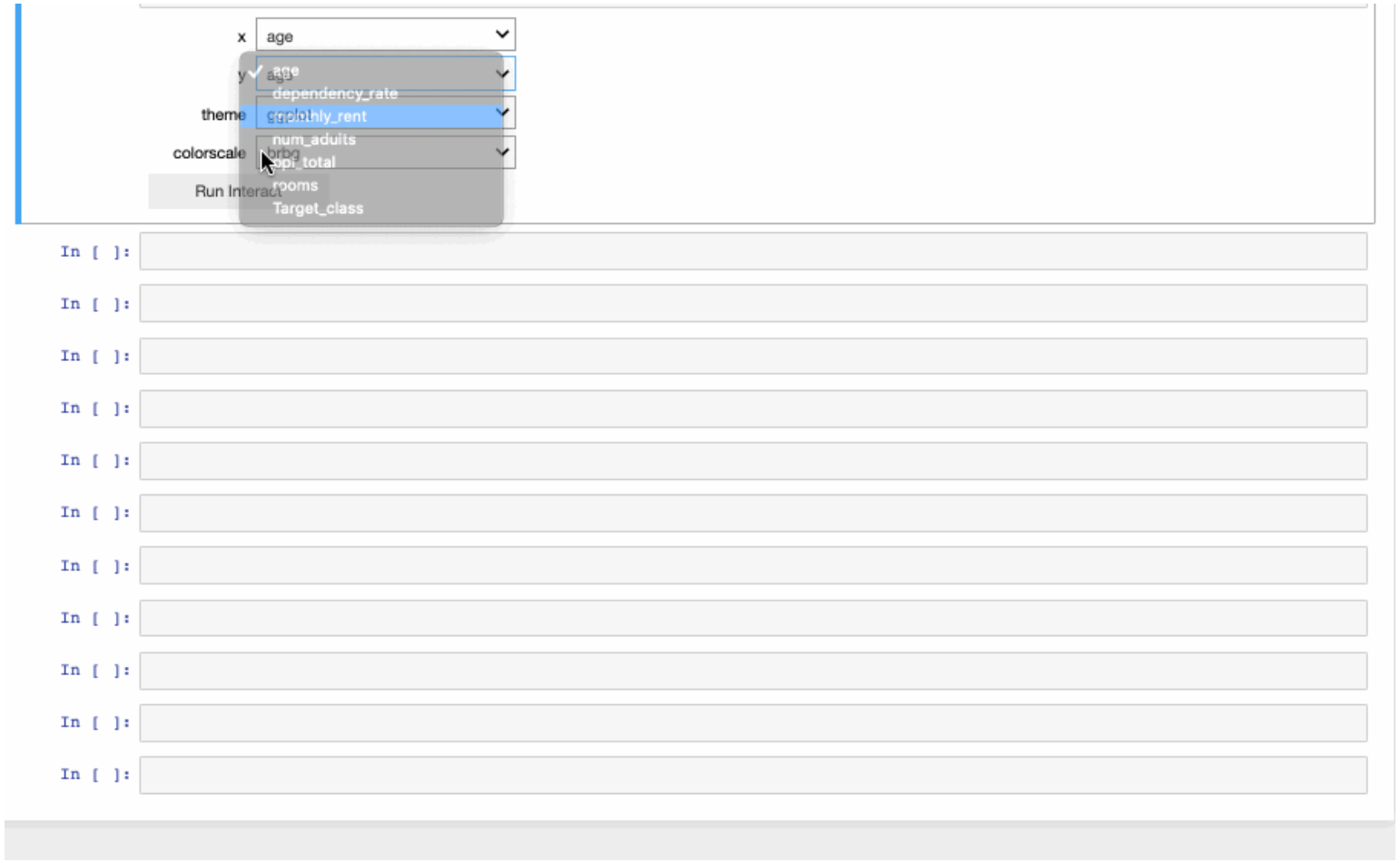
```
import plotly
import cufflinks as cf
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
cf.go_offline()
```

Adding widgets to plotly graphs

- `@interact` is another common method which will automatically output the widgets and the plot, but it may take longer to update
- We decided to use `@interact_manual` to prevent the notebook from freezing and to select all desired parameters before running the scatterplot

```
@interact_manual
def scatter_plot(x = list(costa_viz.columns),
                 y = list(costa_viz.columns),
                 theme = list(cf.themes.THEMES.keys()),
                 colorscale = list(cf.colors._scales_names.keys())):

    df.iplot(kind = 'scatter', x = x, y = y, mode='markers',
             categories = 'Target_class',
             xTitle = x, yTitle = y,
             title = f'{x} vs {y}',
             theme = theme, colorscale = colorscale)
```

Adding widgets to plotly graphs



Module completion checklist

Objective	Complete
Introducing Bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	✔
Create simple plots using Bokeh	✔
Organize multiple visualization with layouts and configure plot tools	✔
Add interactivity and highlight data using labels	✔
Integrate widgets to Bokeh and plotly graphs	✔
Save your graphs	

Saving your graph

- You will be saving all of your graphs as an `HTML` file in the plots folder
- Change your directory to `plot_dir` and save your `bokeh` plots as shown below:

```
os.chdir(plot_dir)
```

```
# Create figure.
p = figure(plot_width = 400, plot_height = 400)

# Add glyphs to it.
p.triangle(x_values, y_values, size = 20, color = "darkseagreen", alpha = 0.7)

# Save your plot.
output_file("bokeh-simple-plot.html", mode = 'inline')
save(p)
```

Plotly vs Bokeh

- **Bokeh is ideal to create charts with multiple glyphs and has various options for customization**
- But it undergoes a lot of development, hence the code we write today may change in future
- On the other hand, the **plotly syntax is also simple and can be embedded as an HTML in applications**
- The main limitation of plotly is that the online community version plots are public and there is a limit of plots which can be created per day
- Choosing an interactive visualization package depends on your personal preference and ease of use!

Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Introducing bokeh and its applications	✔
Generate your first figure and add glyphs to it	✔
Transform Costa Rican data for visualizations	✔
Create simple plots using Bokeh	✔
Organize multiple visualization with layouts and configure plot tools	✔
Add interactivity and highlight data using labels	✔
Integrate widgets to bokeh and plotly graphs	✔
Save your graphs	✔

Workshop: next steps!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today, you can try out the concepts covered in each module
  - Create basic plots using bokeh and then head on to plots with multiple metrics
  - Add interactions to your plots using `HoverTool()` and legends
  - Create widgets and run them with `bokeh` and `plotly` graphs
  - Save the above plots as a `.html` file in the `plots` folder

This completes our module

Congratulations!