

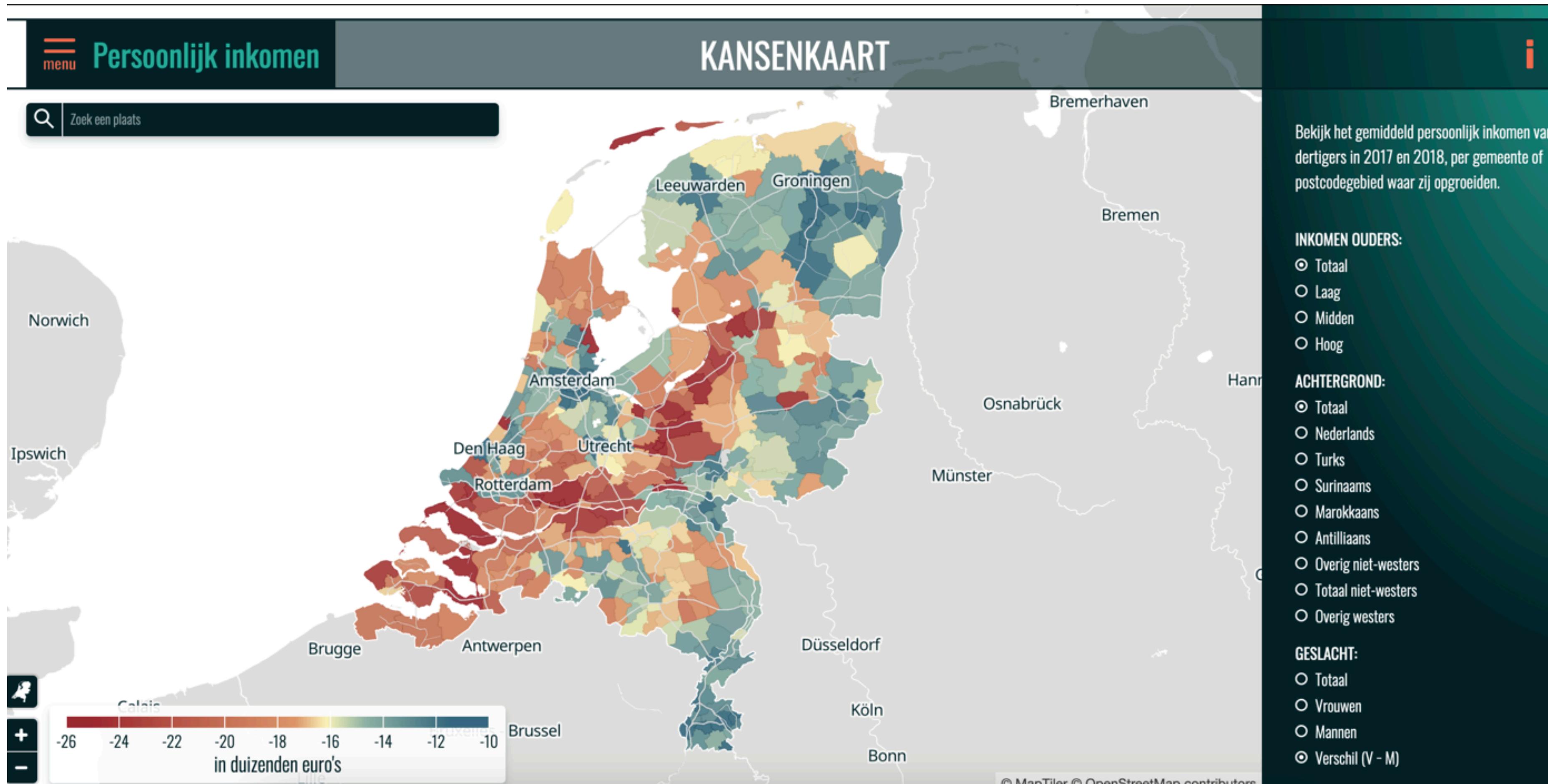
PART B

INTERACTIVE VISUALISATIONS AND (SERVERLESS) DASHBOARDS

WHY DO WE WANT INTERACTIVE VISUALISATIONS?

WHY DO WE WANT INTERACTIVE VISUALISATIONS?

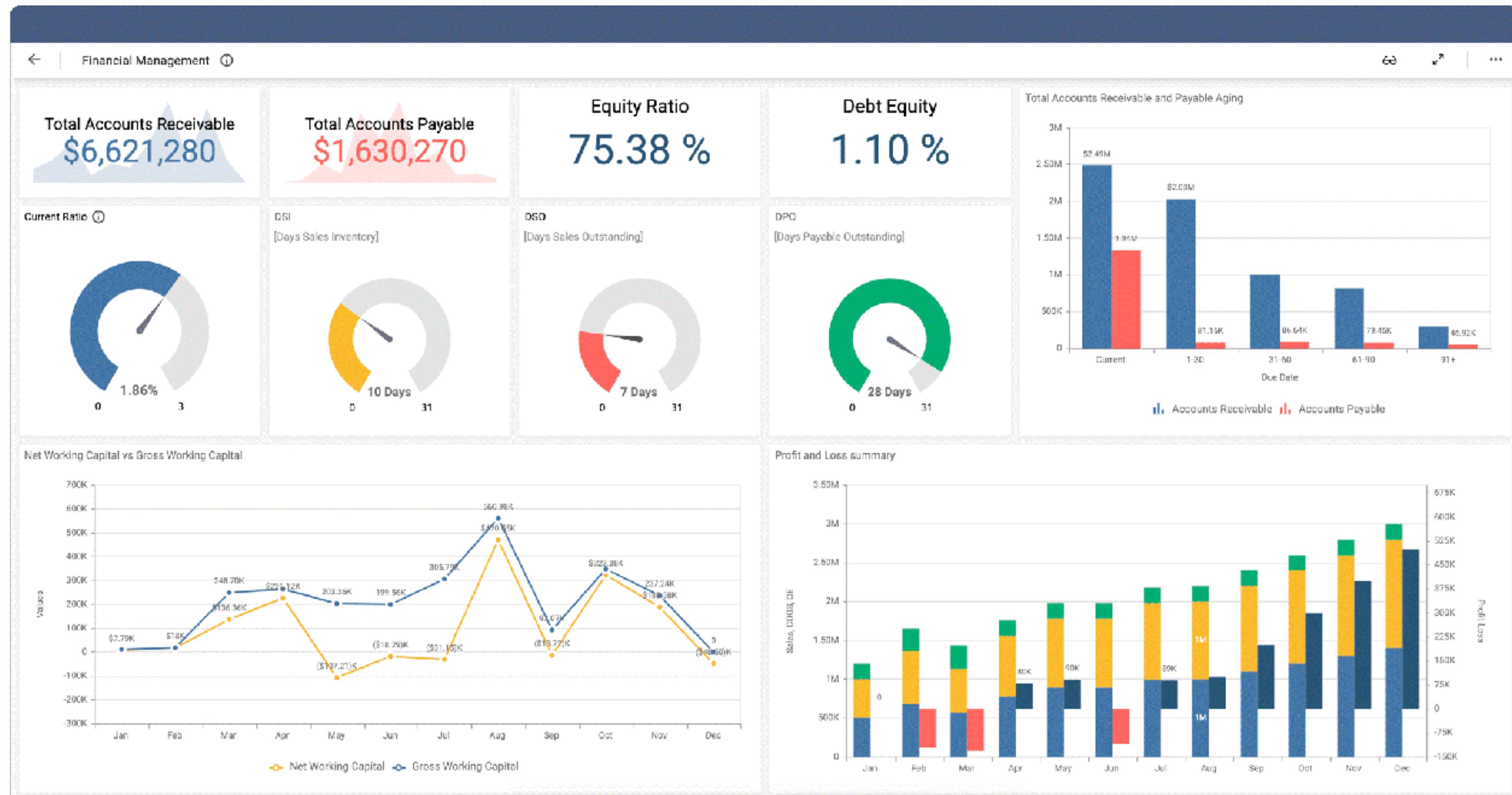
Main use: Explore data



<https://kansenkaart.nl/>

WHY DO WE WANT INTERACTIVE VISUALISATIONS?

Use 2: Monitor data



WHY DO WE WANT INTERACTIVE VISUALISATIONS?

Use 3: Tell a story

400 richest Americans (\$3.2 trillion)

Every single person in America could be lifted above the poverty line with a one-time cash subsidy of around \$10,000 per impoverished family (and about \$7,000 for impoverished individuals). The total cost would be \$170 billion, a little over 5% of the wealth currently controlled by 400 individuals.



<https://mkorostoff.github.io/1-pixel-wealth/>

<https://www.theguardian.com/uk-news/ng-interactive/2023/apr/05/revealed-royals-took-more-than-1bn-income-from-controversial-estates-king-charles-queen-duchies-cornwall-lancaster>

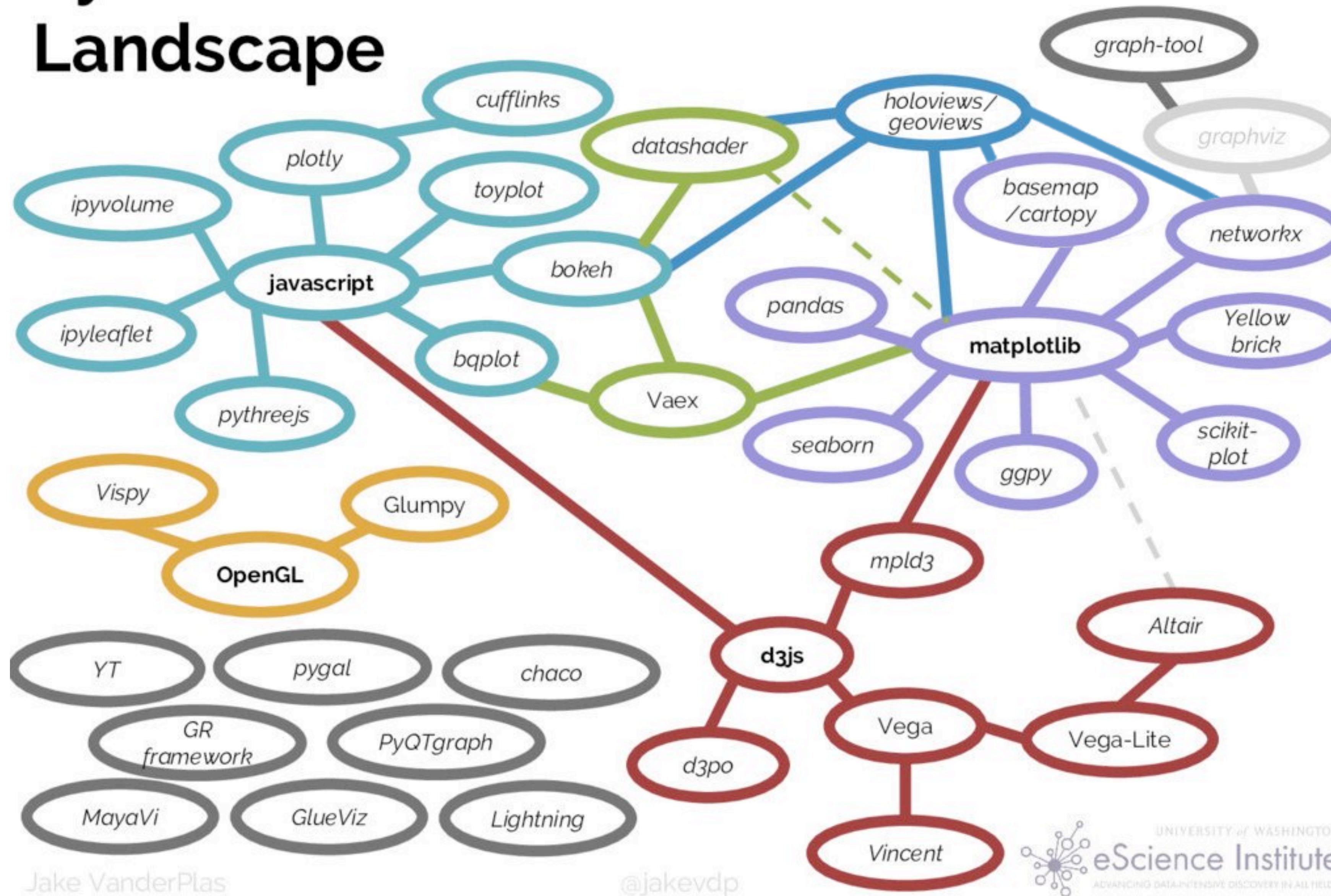
IN R



IN R



Python's Visualization Landscape

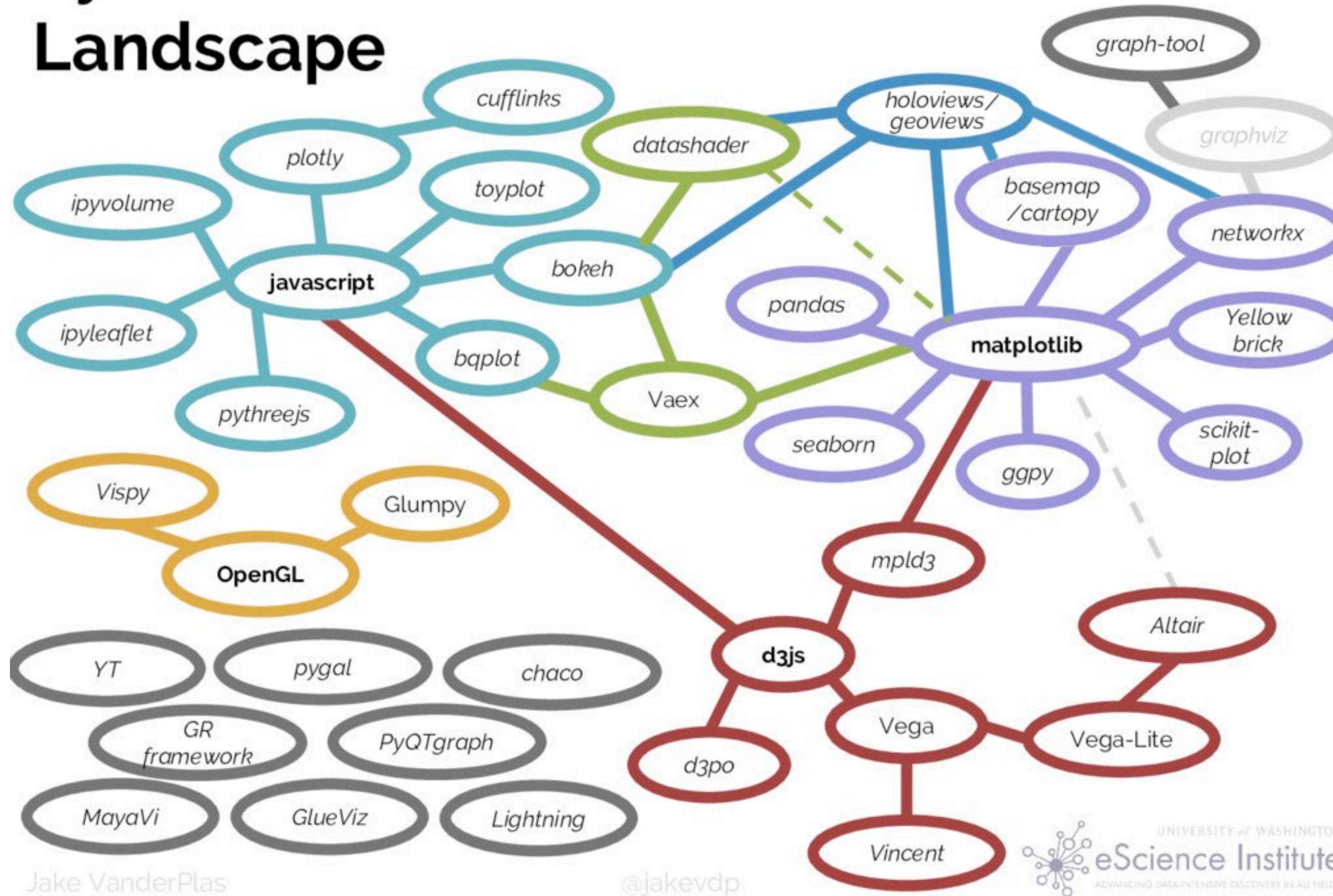


Jake VanderPlas

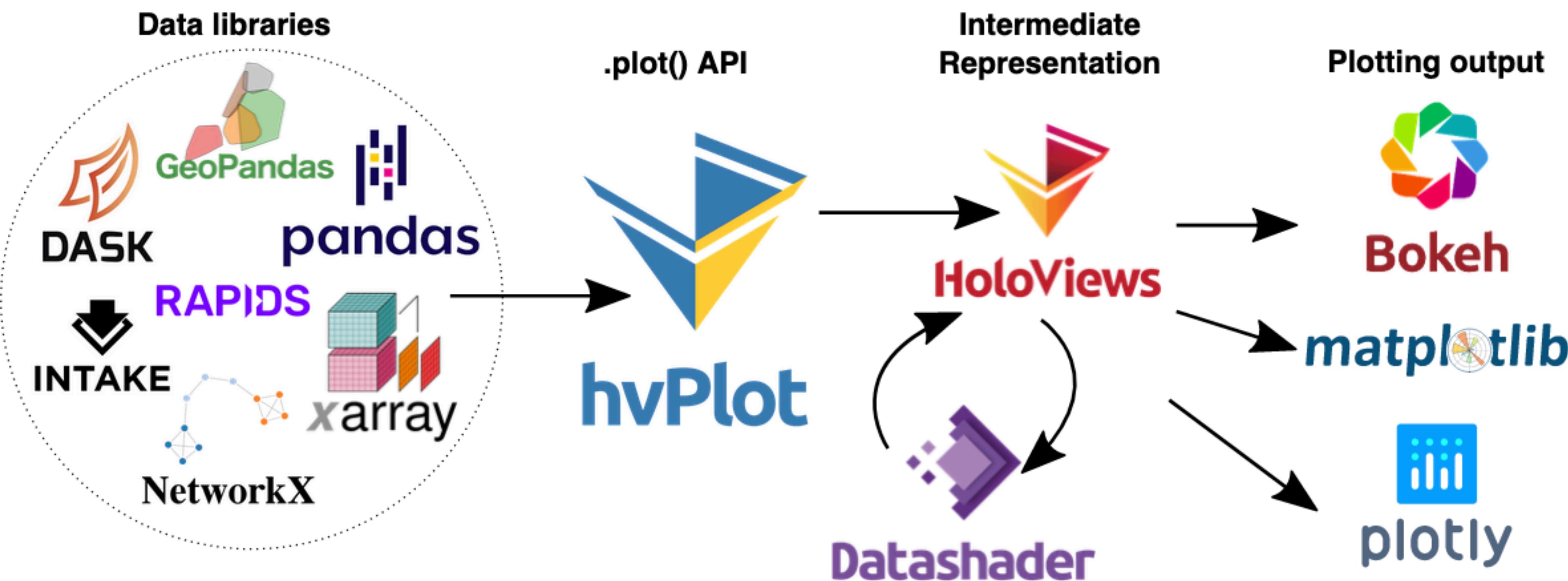
@jakevdp

<https://holoviz.org/tutorial/Overview.html>

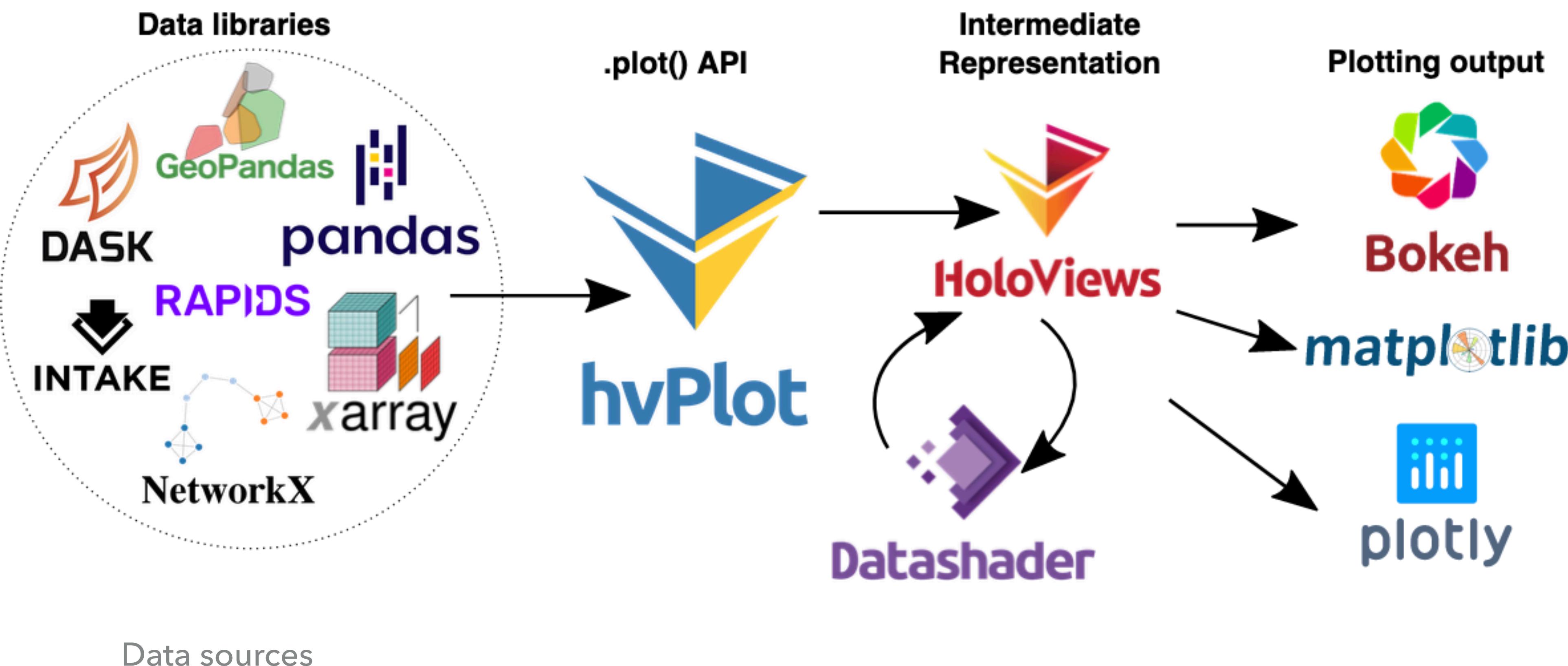
Python's Visualization Landscape



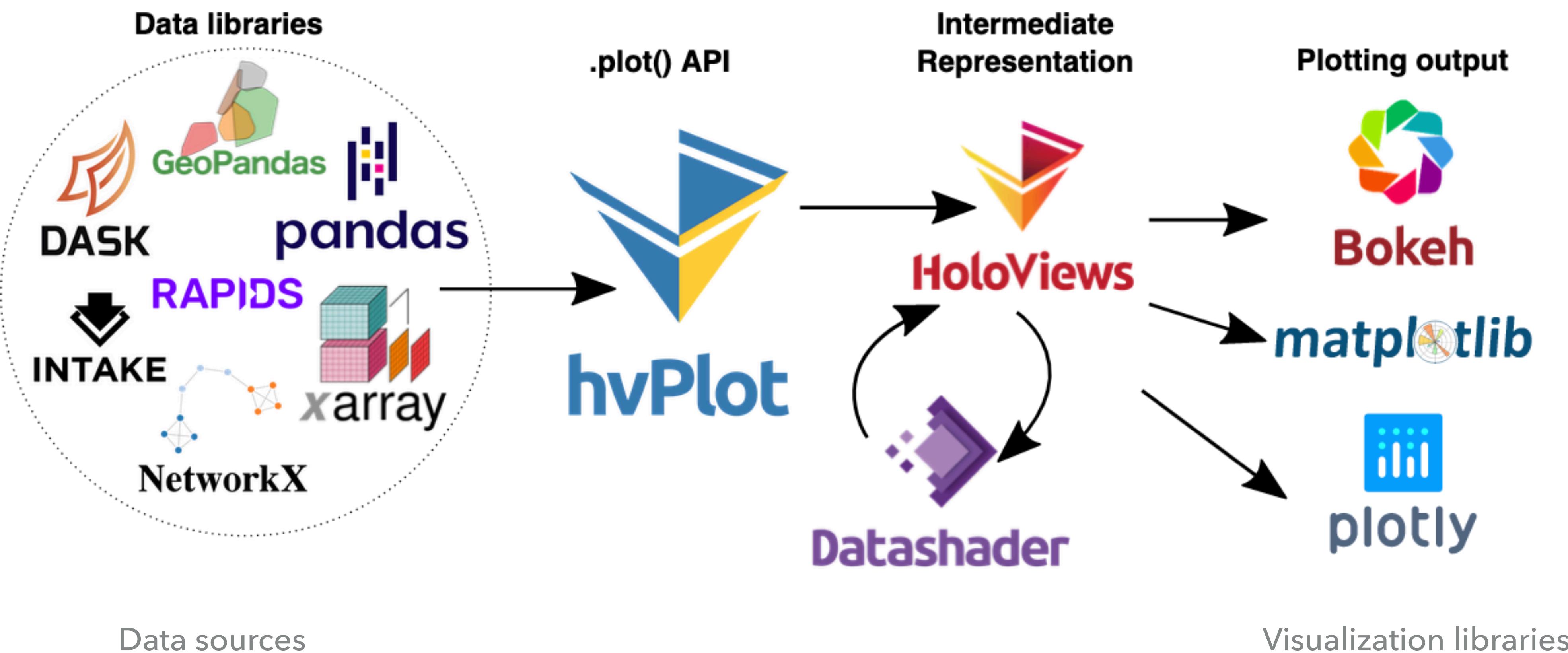
HOLOVIZ



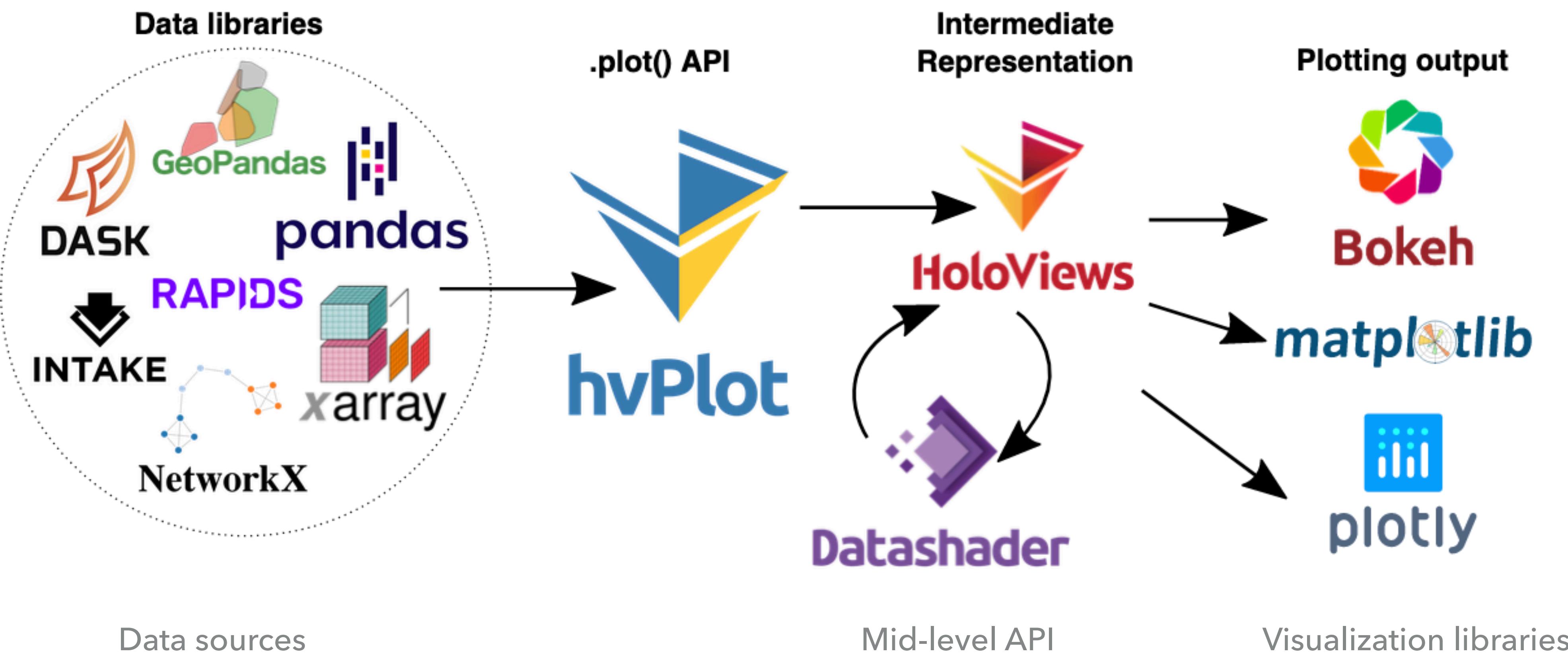
HOLOVIZ



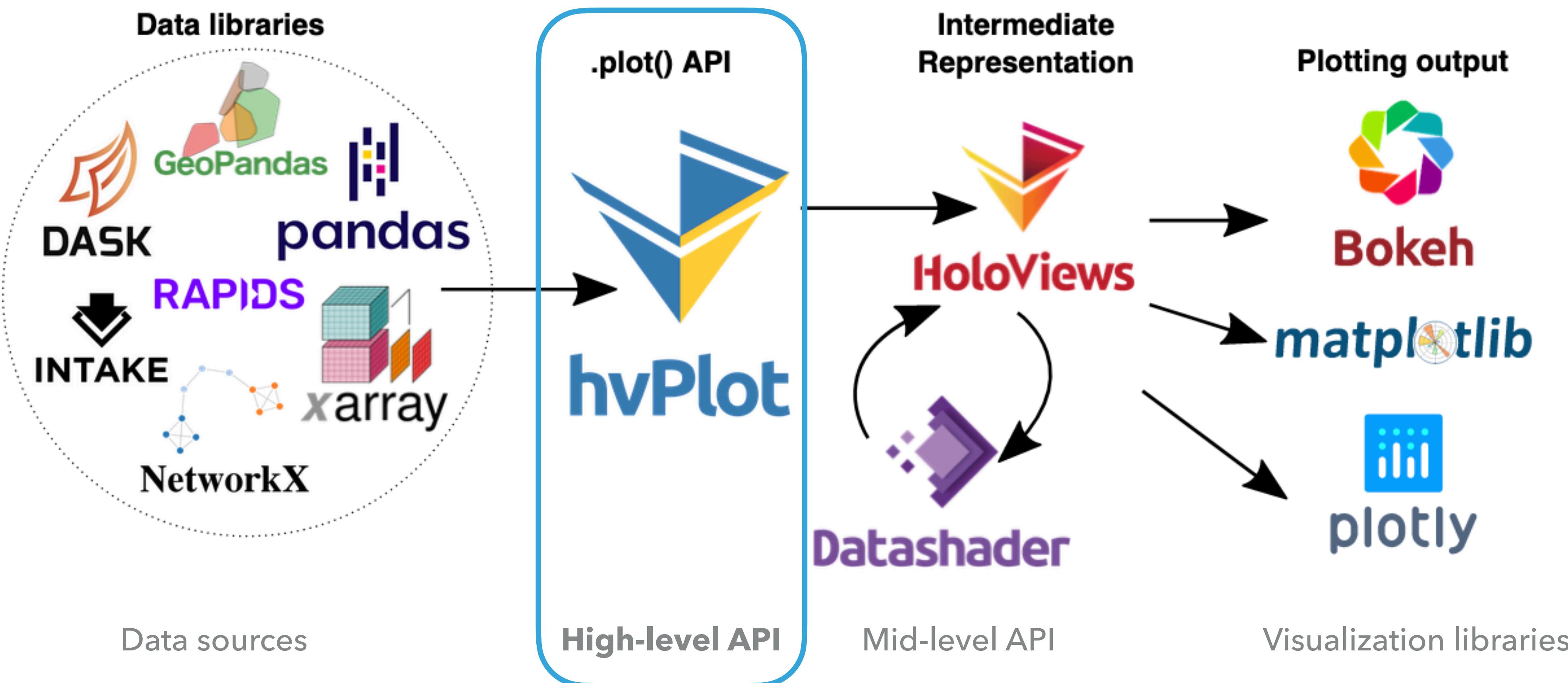
HOLOVIZ



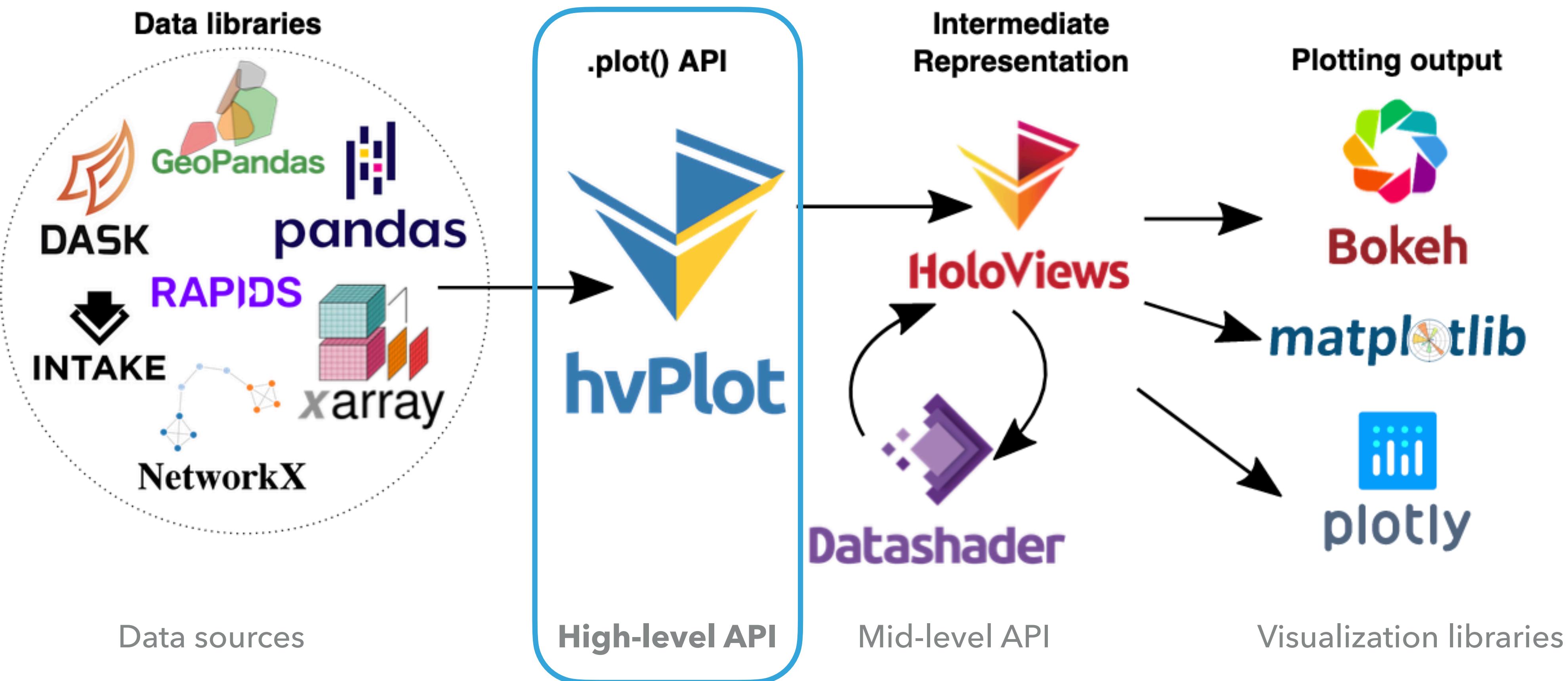
HOLOVIZ



HOLOVIZ



HOLOVIZ

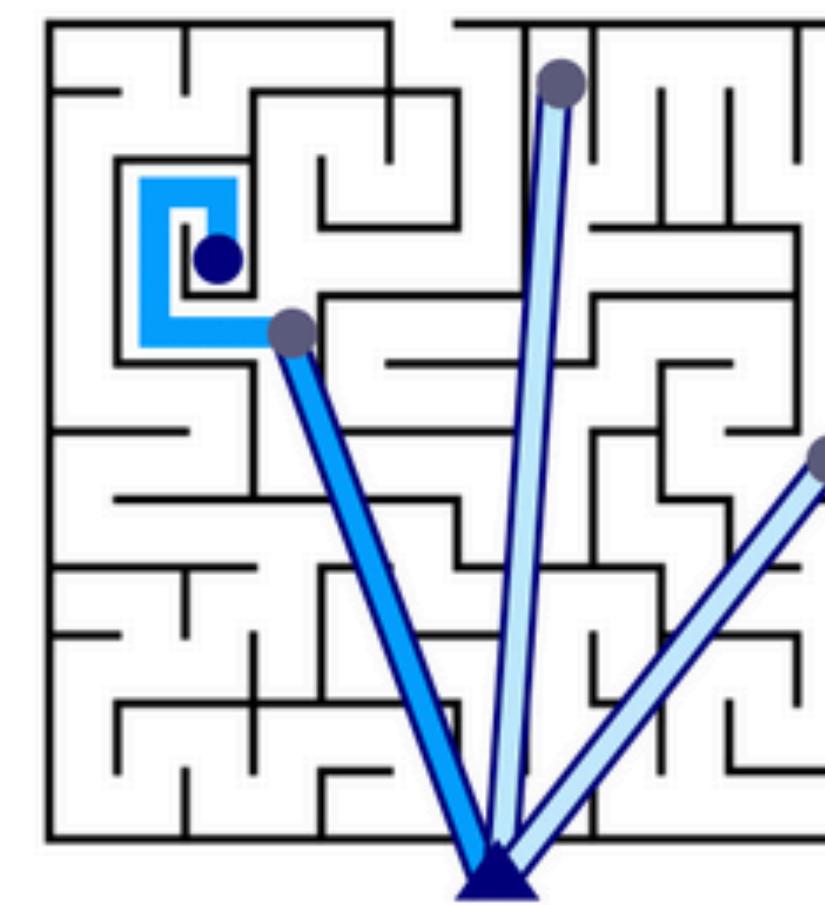


Low level



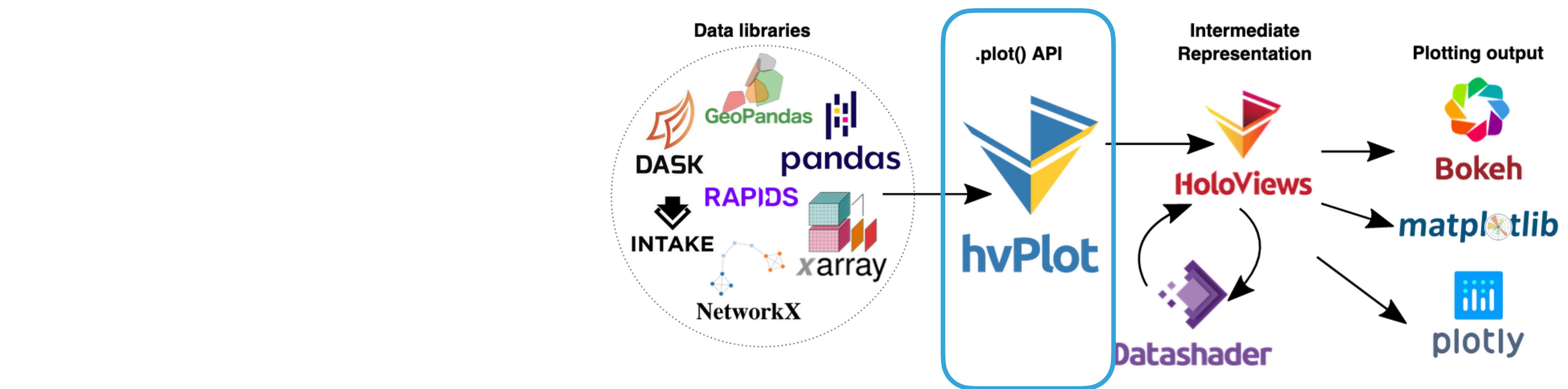
- ✓ Can precisely choose where to go
- Requires expertise to make decisions all along the way

Layered

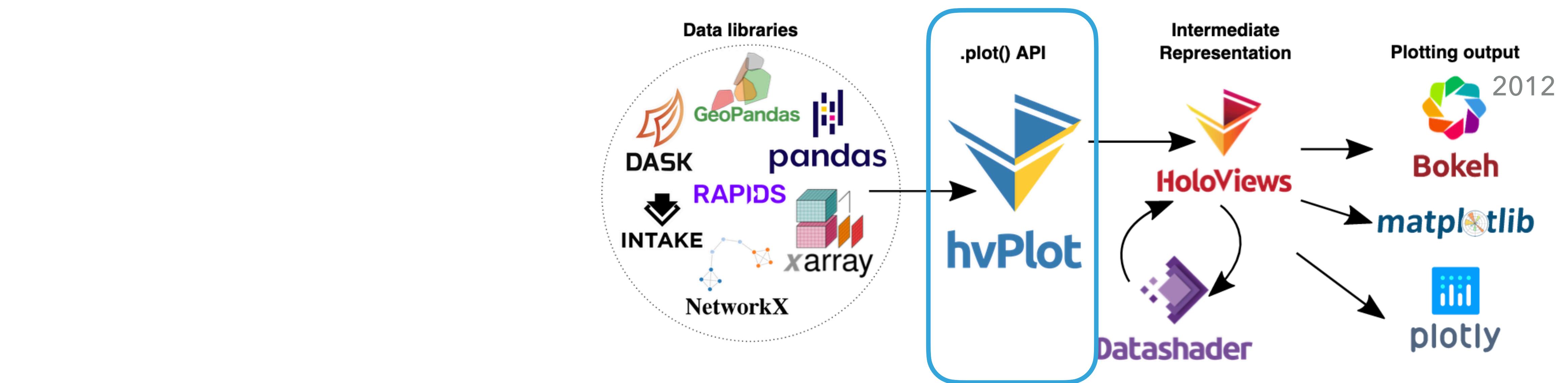


- ✓ Can quickly go far
- ✓ Can precisely choose where to go
- ✓ Only need to study where shortcut got you

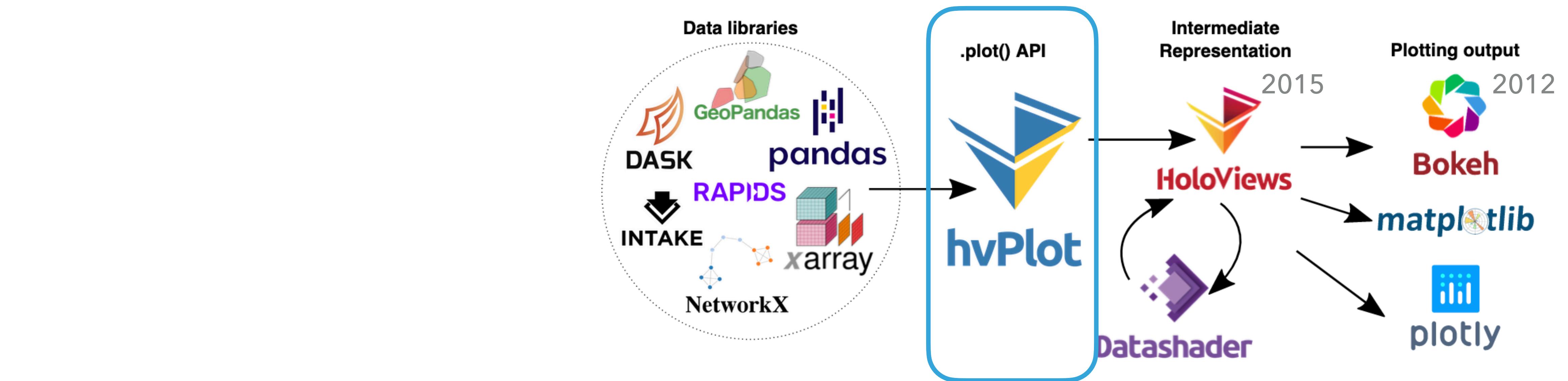
VISUALIZATION



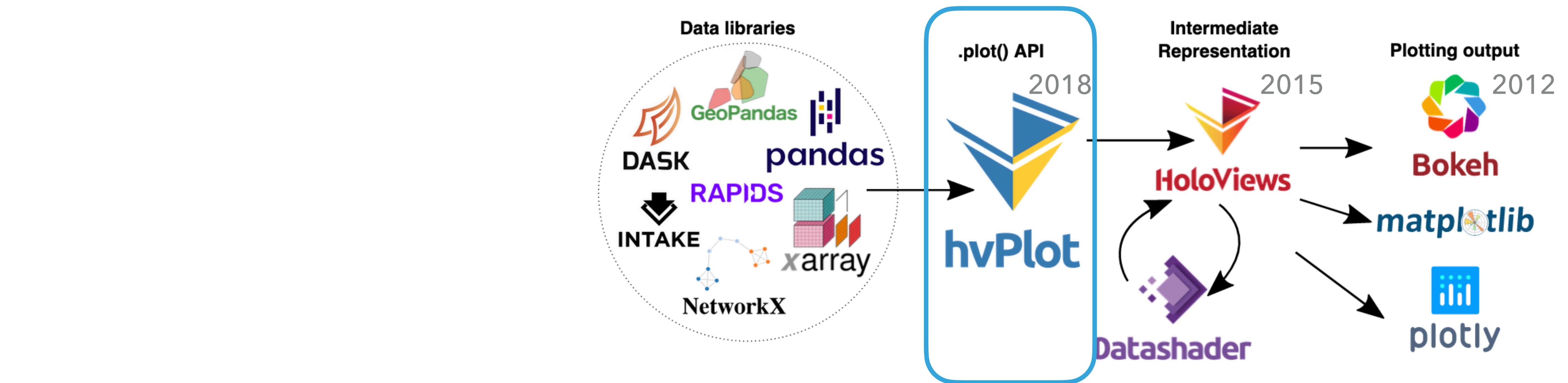
VISUALIZATION



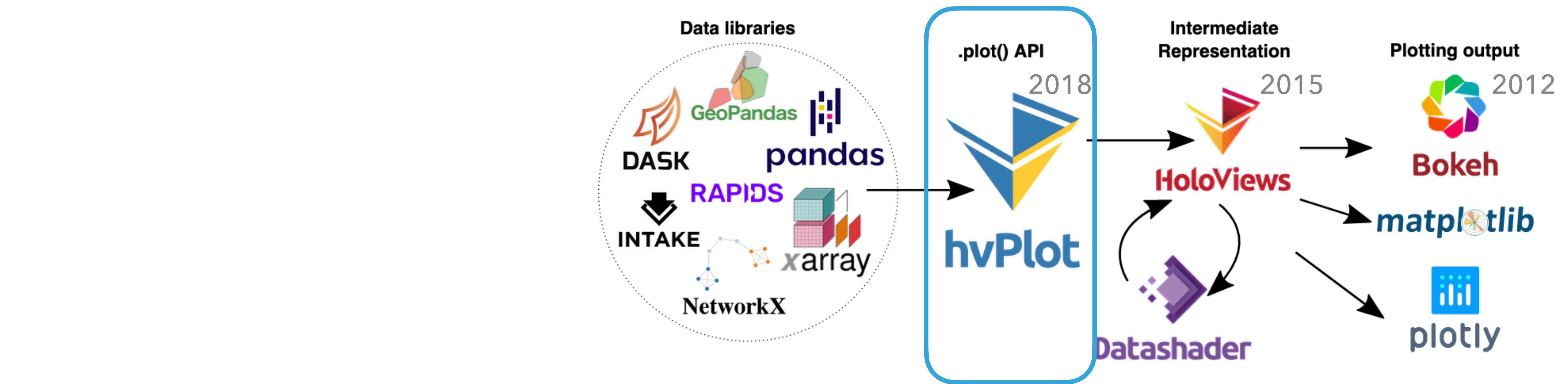
VISUALIZATION



VISUALIZATION



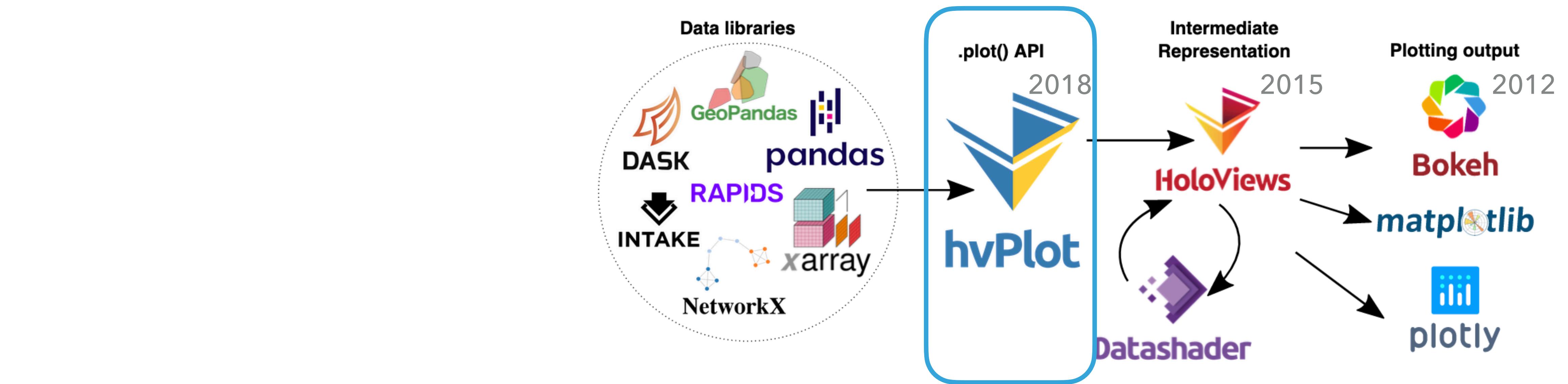
VISUALIZATION



Advantages:

- * Easy and powerful
- * Uses bokeh as default (!)
- * General and consistent API
- * Connects very well with panel

VISUALIZATION



Advantages:

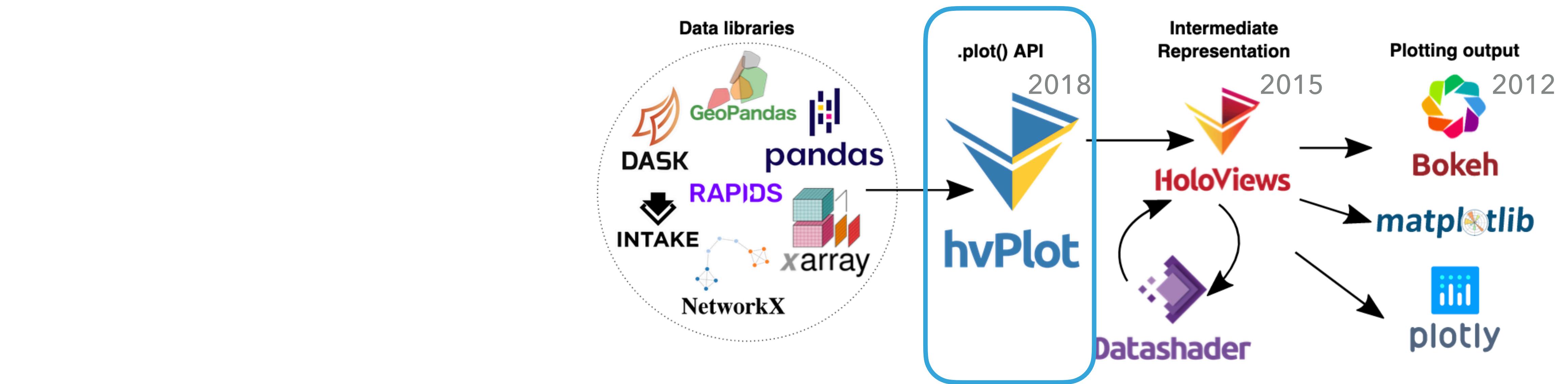
- * Easy and powerful
- * Uses bokeh as default (!)
- * General and consistent API
- * Connects very well with panel

Disadvantages:

- * Fine-tuning harder than in bokeh
- * New technologies
- * Small user-base
- * But you can always use bokeh!

MAKING APPS

VISUALIZATION



Advantages:

- * Easy and powerful
- * Uses bokeh as default (!)
- * General and consistent API
- * Connects very well with panel

Disadvantages:

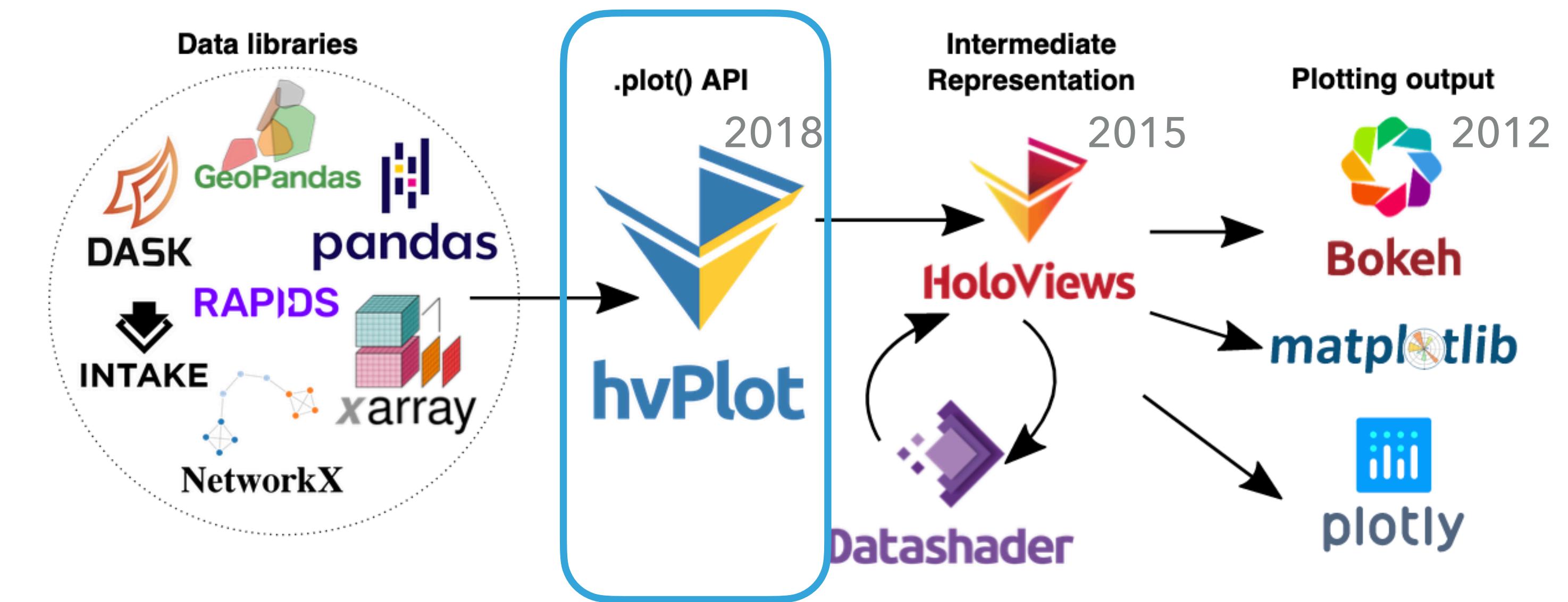
- * Fine-tuning harder than in bokeh
- * New technologies
- * Small user-base
- * But you can always use bokeh!

MAKING APPS



Panel

VISUALIZATION



Advantages:

- * Easy and powerful
- * Uses bokeh as default (!)
- * General and consistent API
- * Connects very well with panel

Disadvantages:

- * Fine-tuning harder than in bokeh
- * New technologies
- * Small user-base
- * But you can always use bokeh!

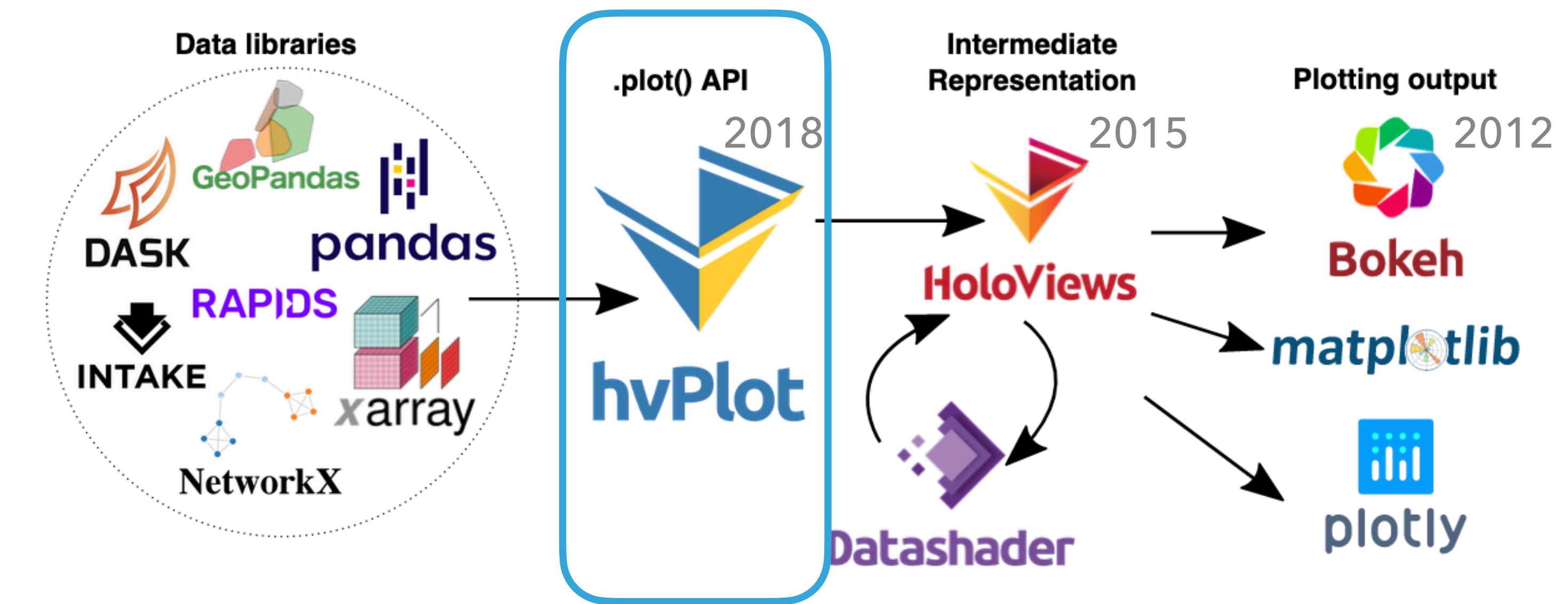
MAKING APPS



2018

Panel

VISUALIZATION



Advantages:

- * Easy and powerful
- * Uses bokeh as default (!)
- * General and consistent API
- * Connects very well with panel

Disadvantages:

- * Fine-tuning harder than in bokeh
- * New technologies
- * Small user-base
- * But you can always use bokeh!

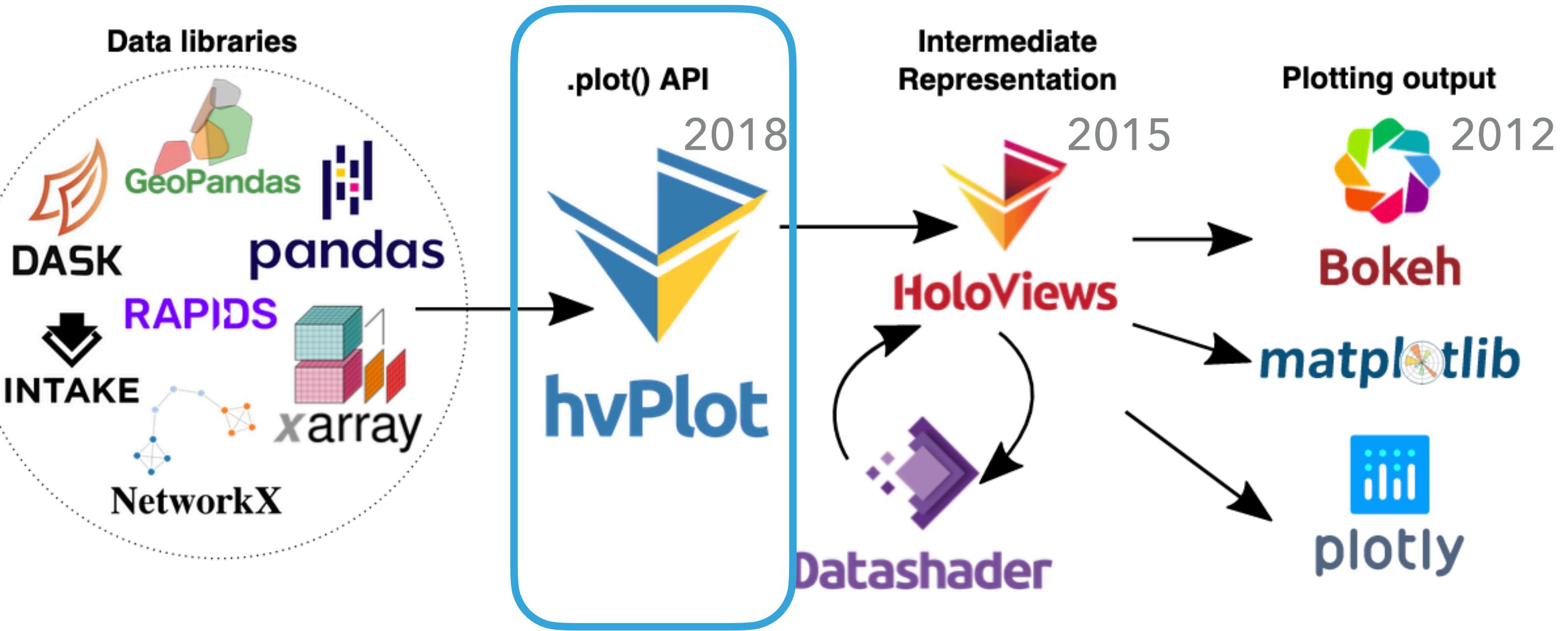
MAKING APPS



Panel

Provides components, widgets, and layouts

2018



Advantages:

- * Easy and powerful
- * Uses bokeh as default (!)
- * General and consistent API
- * Connects very well with panel

Disadvantages:

- * Fine-tuning harder than in bokeh
- * New technologies
- * Small user-base
- * But you can always use bokeh!

1. HVPILOT

1. HV PLOT

SIMPLE PLOT

1. HV PLOT

SIMPLE PLOT

x	y	class
-1.353869	-1.584364	class A
0.497119	0.546156	class B
-0.615940	-0.379600	class A
-0.142613	0.328385	class A
0.750646	0.203923	class B
-0.922876	-0.677276	class A
-1.763000	-1.978123	class A
1.263385	1.887111	class B
0.308715	1.615190	class B
0.780764	0.285989	class B

1. HVPLLOT

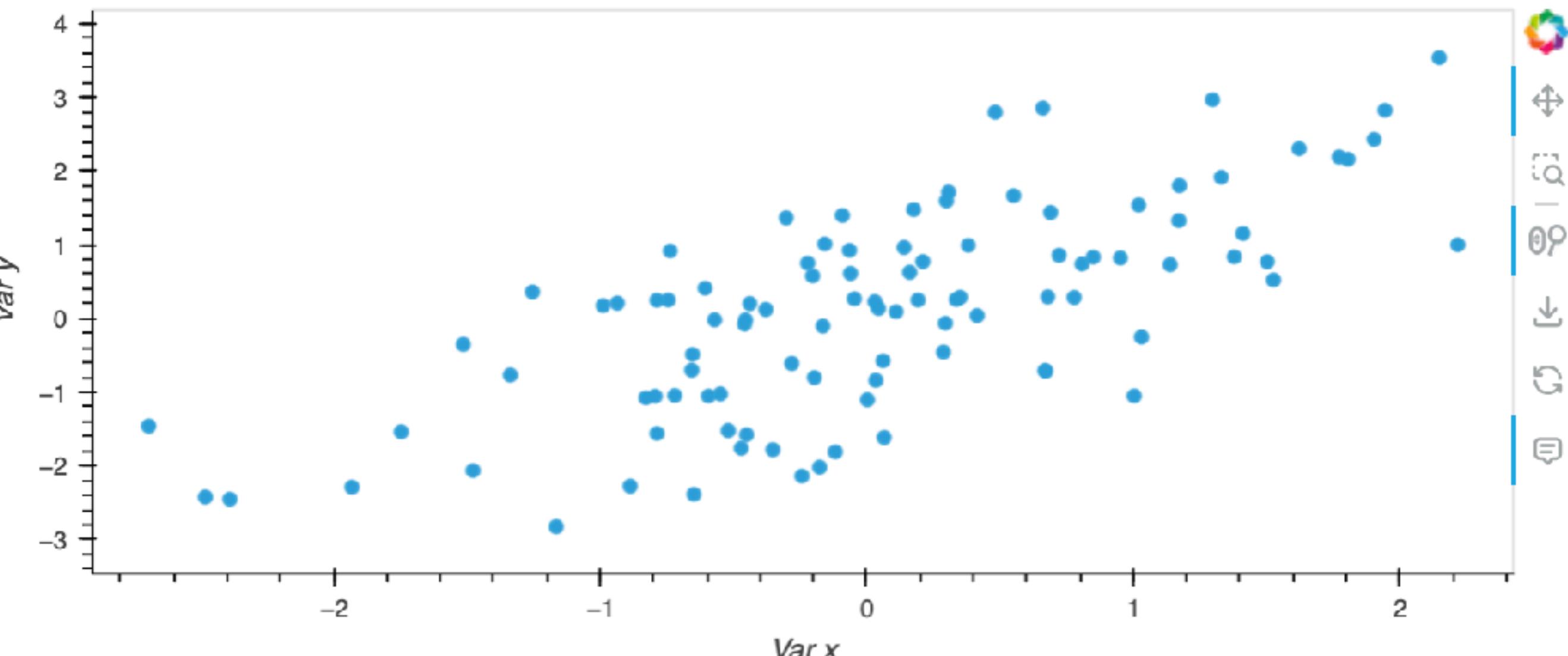
SIMPLE PLOT

x	y	class
-1.353869	-1.584364	class A
0.497119	0.546156	class B
-0.615940	-0.379600	class A
-0.142613	0.328385	class A
0.750646	0.203923	class B
-0.922876	-0.677276	class A
-1.763000	-1.978123	class A
1.263385	1.887111	class B
0.308715	1.615190	class B
0.780764	0.285989	class B

```
df = create_data(100)

# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             )

# Show
scatter
```



1. HV PLOT

SIMPLE PLOT

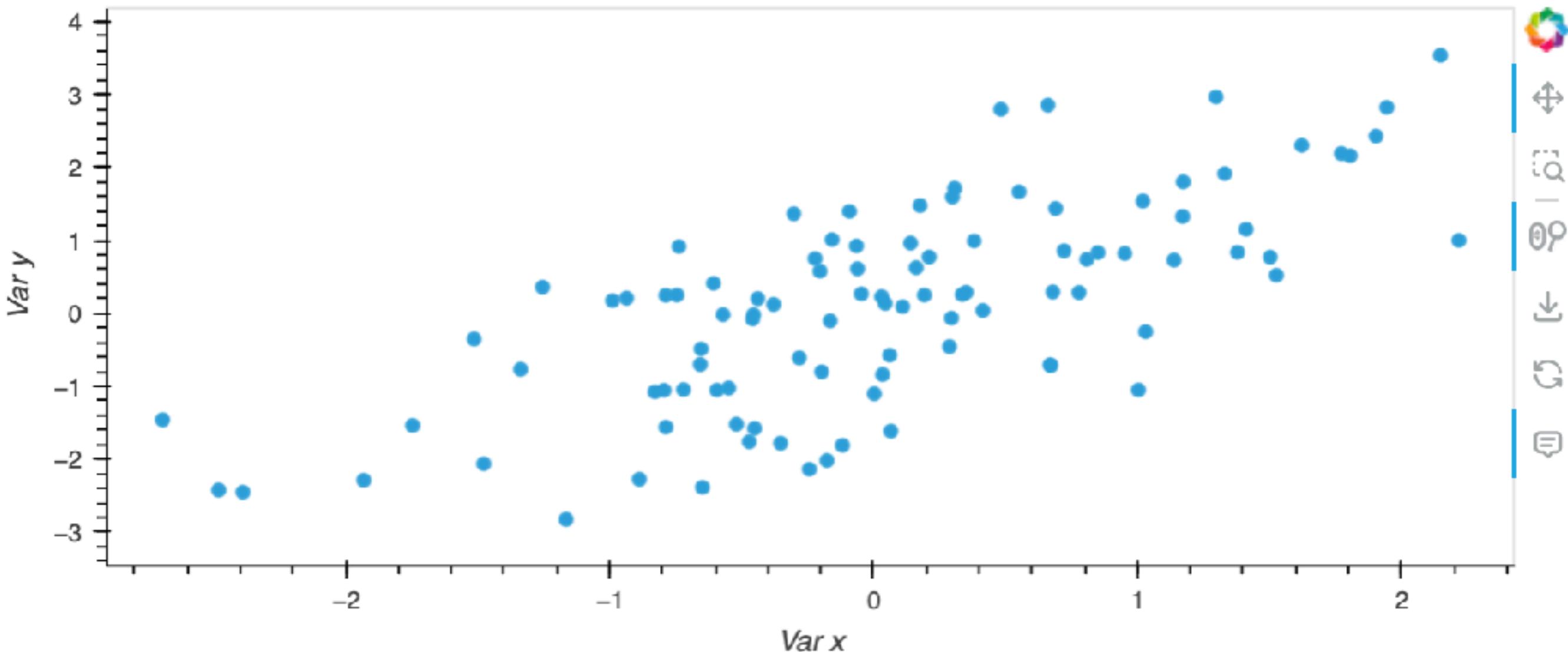
Makes plot with some labels

x	y	class
-1.353869	-1.584364	class A
0.497119	0.546156	class B
-0.615940	-0.379600	class A
-0.142613	0.328385	class A
0.750646	0.203923	class B
-0.922876	-0.677276	class A
-1.763000	-1.978123	class A
1.263385	1.887111	class B
0.308715	1.615190	class B
0.780764	0.285989	class B

```
df = create_data(100)

# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             )

# Show
scatter
```



1. HVPLLOT

SIMPLE PLOT

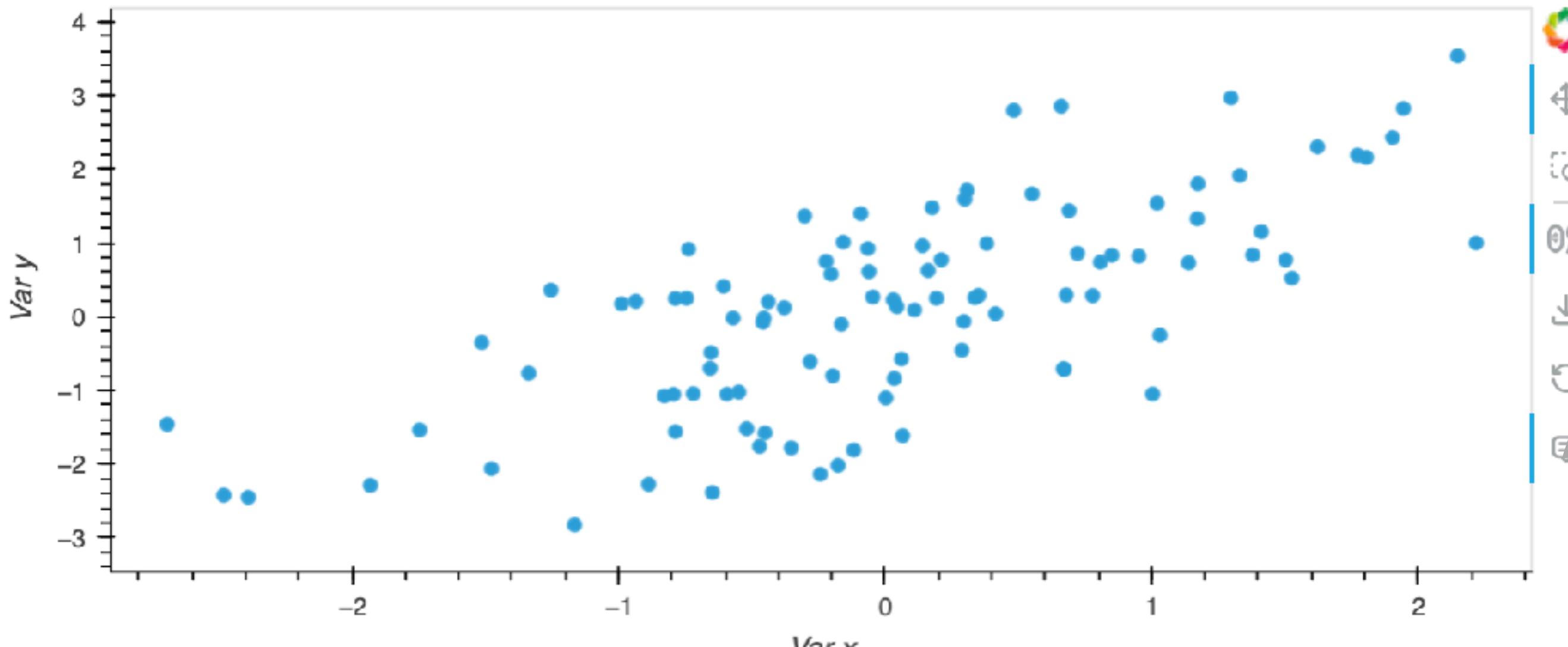
Makes plot with some labels

x	y	class
-1.353869	-1.584364	class A
0.497119	0.546156	class B
-0.615940	-0.379600	class A
-0.142613	0.328385	class A
0.750646	0.203923	class B
-0.922876	-0.677276	class A
-1.763000	-1.978123	class A
1.263385	1.887111	class B
0.308715	1.615190	class B
0.780764	0.285989	class B

```
df = create_data(100)

# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             )

# Show
scatter
```



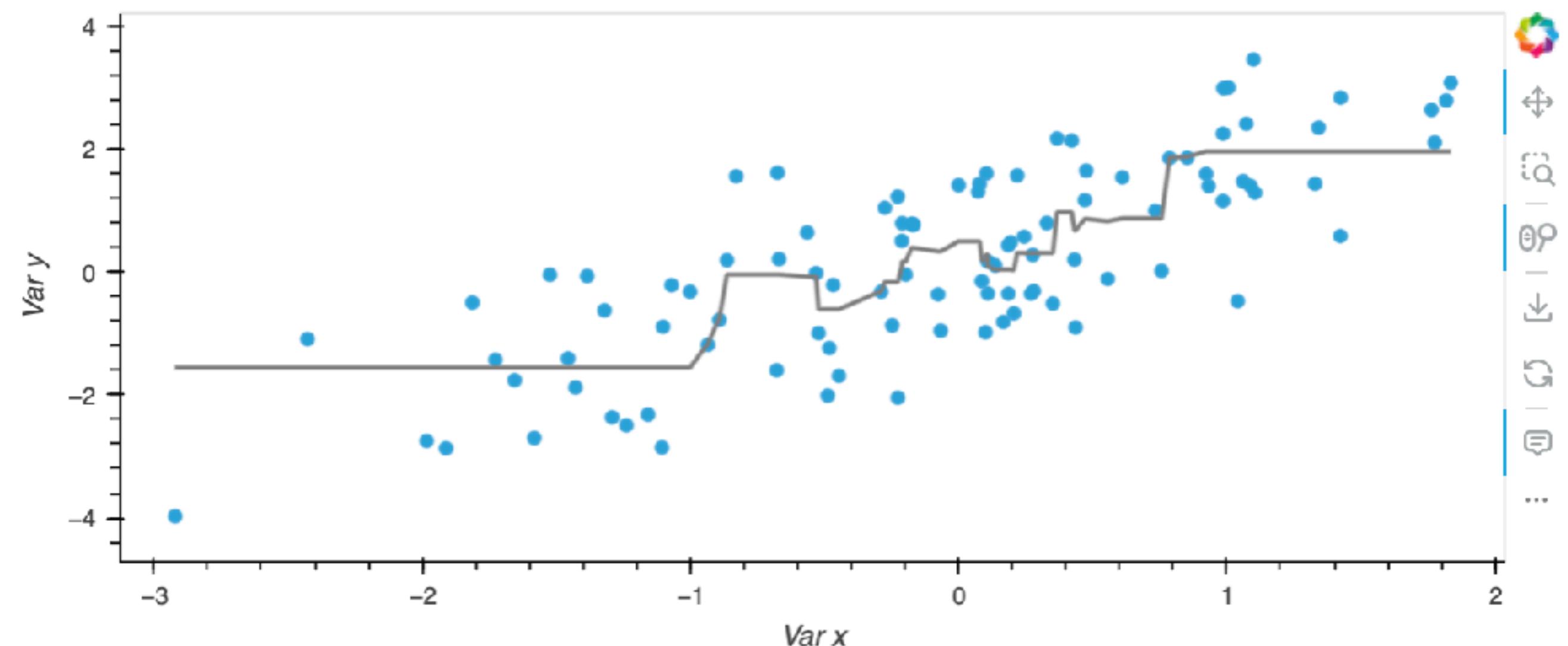
Bokeh tools

1. HV PLOT

COMPOSITE PLOT

```
# Fit a fancy model on the data
fancy_model = HistGradientBoostingRegressor().fit(df[["x"]], df["y"])
df["pred"] = fancy_model.predict(df[["x"]])
# Plot the predicted values (careful, this is in-sample)
s = df.hvplot.line(x='x',
                    y='pred',
                    color="gray"
                    )

# Show
scatter * s
```



1. HV PLOT

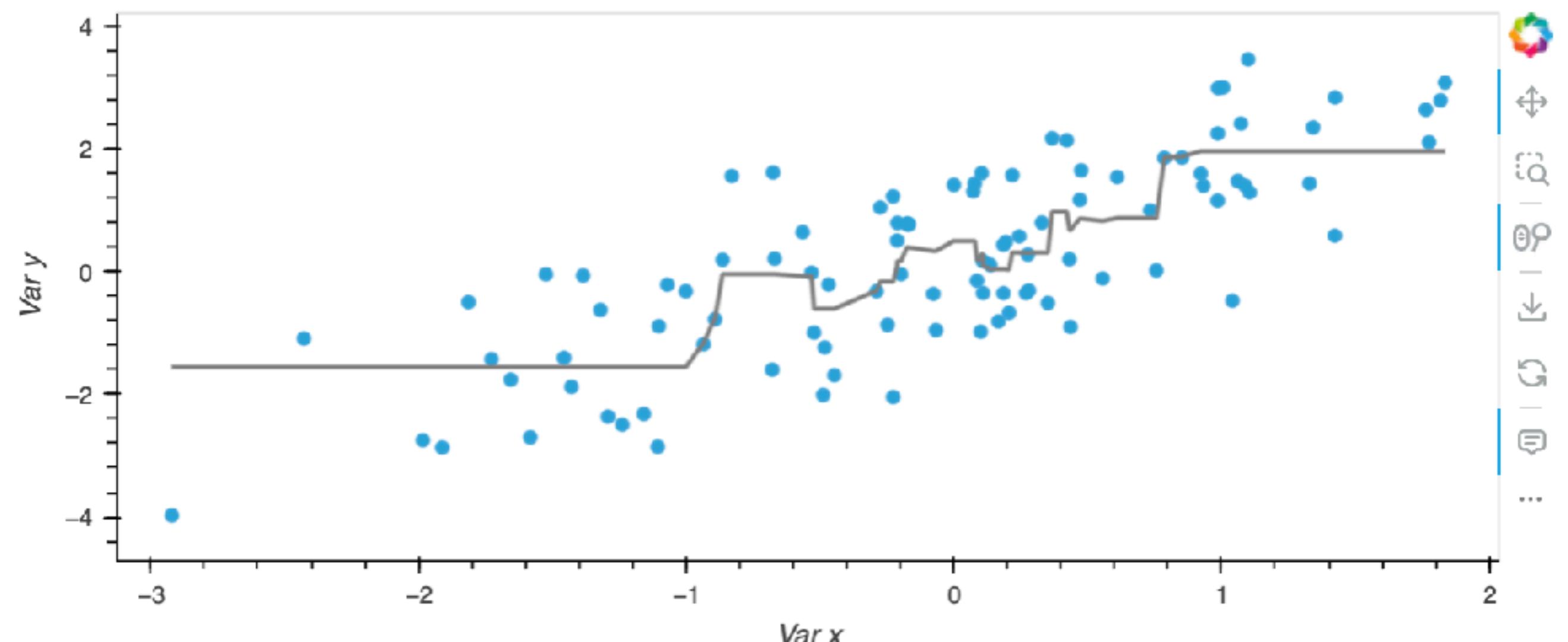
COMPOSITE PLOT

"scatter + s" would put them side by side

Overlay plots

```
# Fit a fancy model on the data
fancy_model = HistGradientBoostingRegressor().fit(df[["x"]], df[["y"]])
df[["pred"]] = fancy_model.predict(df[["x"]])
# Plot the predicted values (careful, this is in-sample)
s = df.hvplot.line(x='x',
                    y='pred',
                    color="gray"
                    )

# Show
scatter * s
```

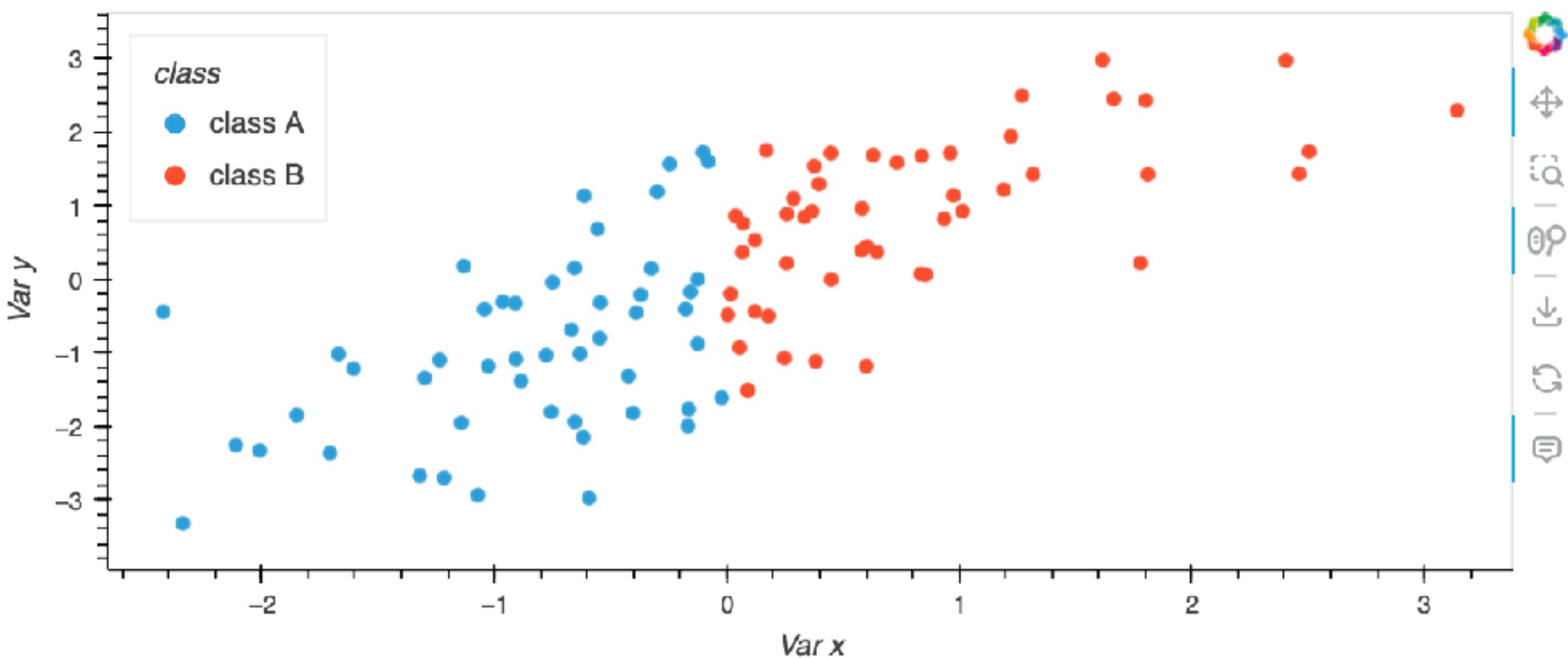


1. HV PLOT

COLOR BY CLASS

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             by="class",
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left")
```

```
# Show
scatter
```



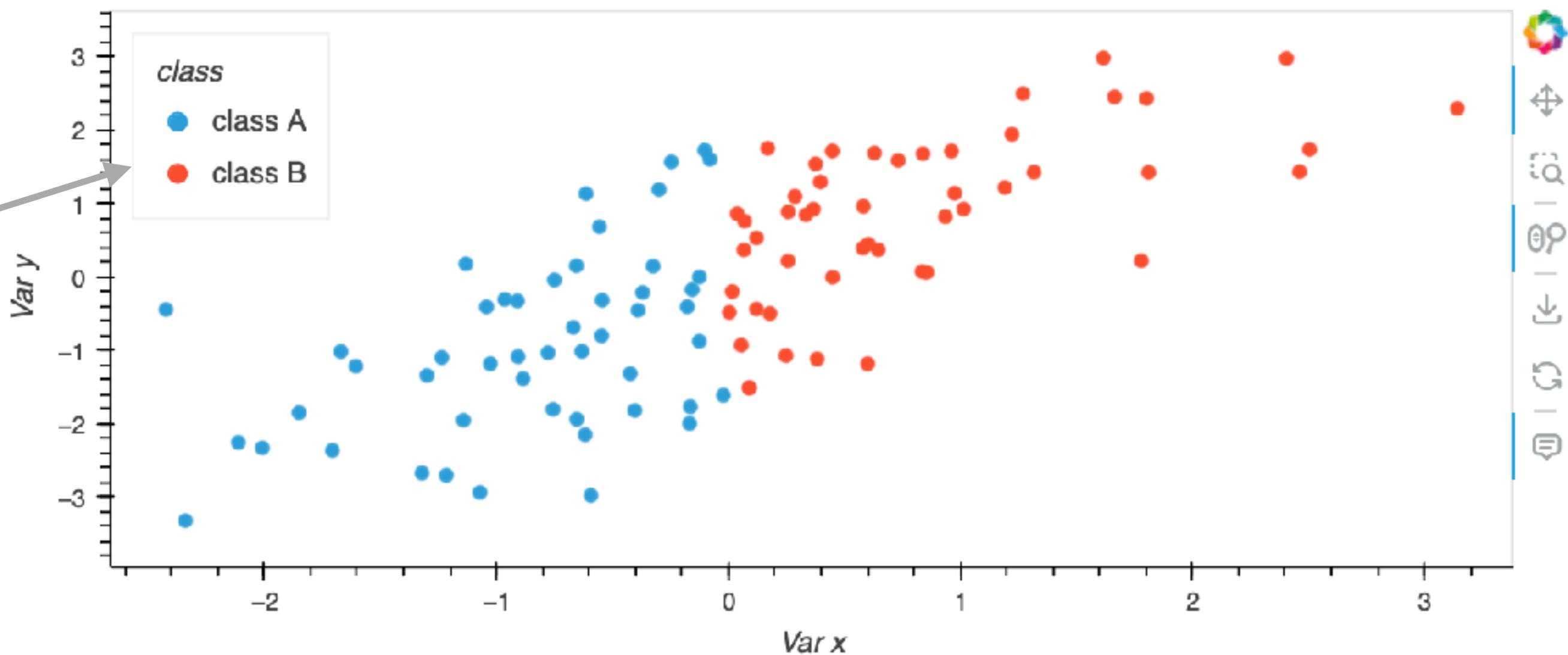
1. HV PLOT

COLOR BY CLASS

Legend interactive by default

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             by="class",
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left")
```

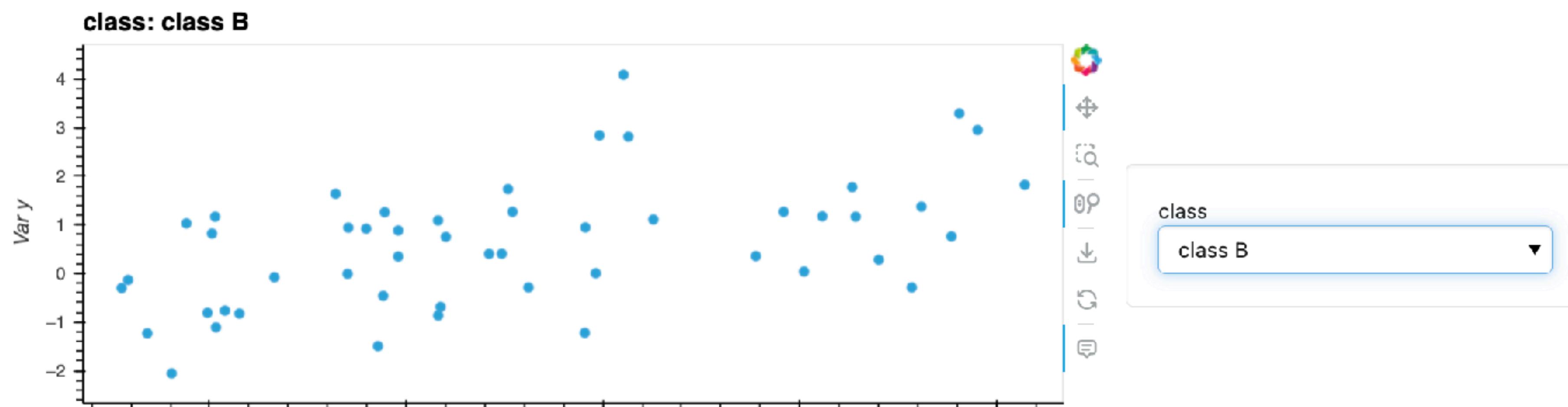
```
# Show
scatter
```



1. HV PLOT

FILTER BY CLASS

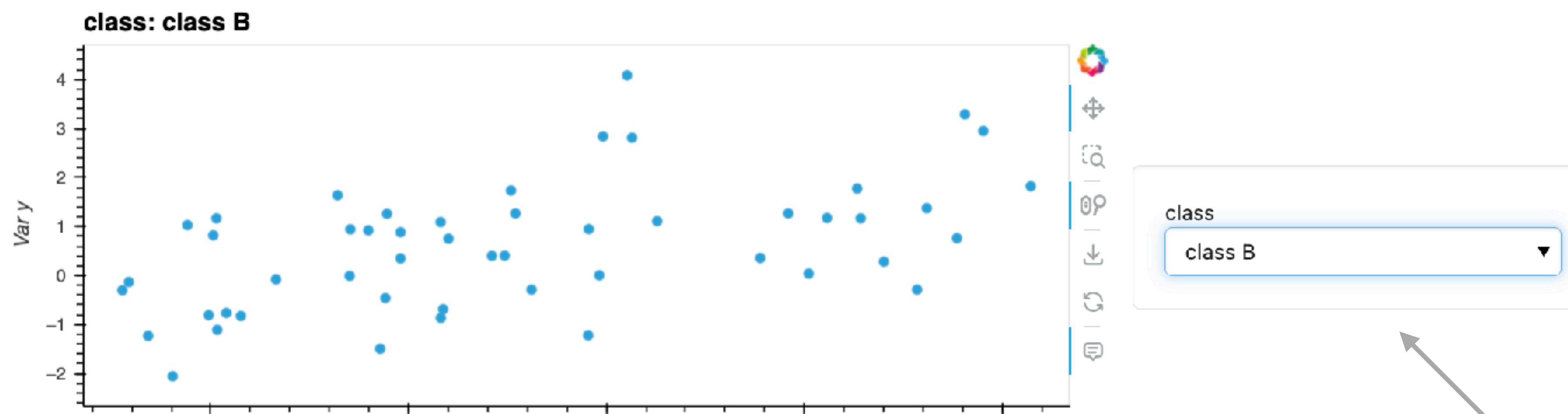
```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             groupby="class",
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left",
                             )
# Show
scatter
```



1. HV PLOT

FILTER BY CLASS

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             groupby="class",
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left",
                             )
# Show
scatter
```



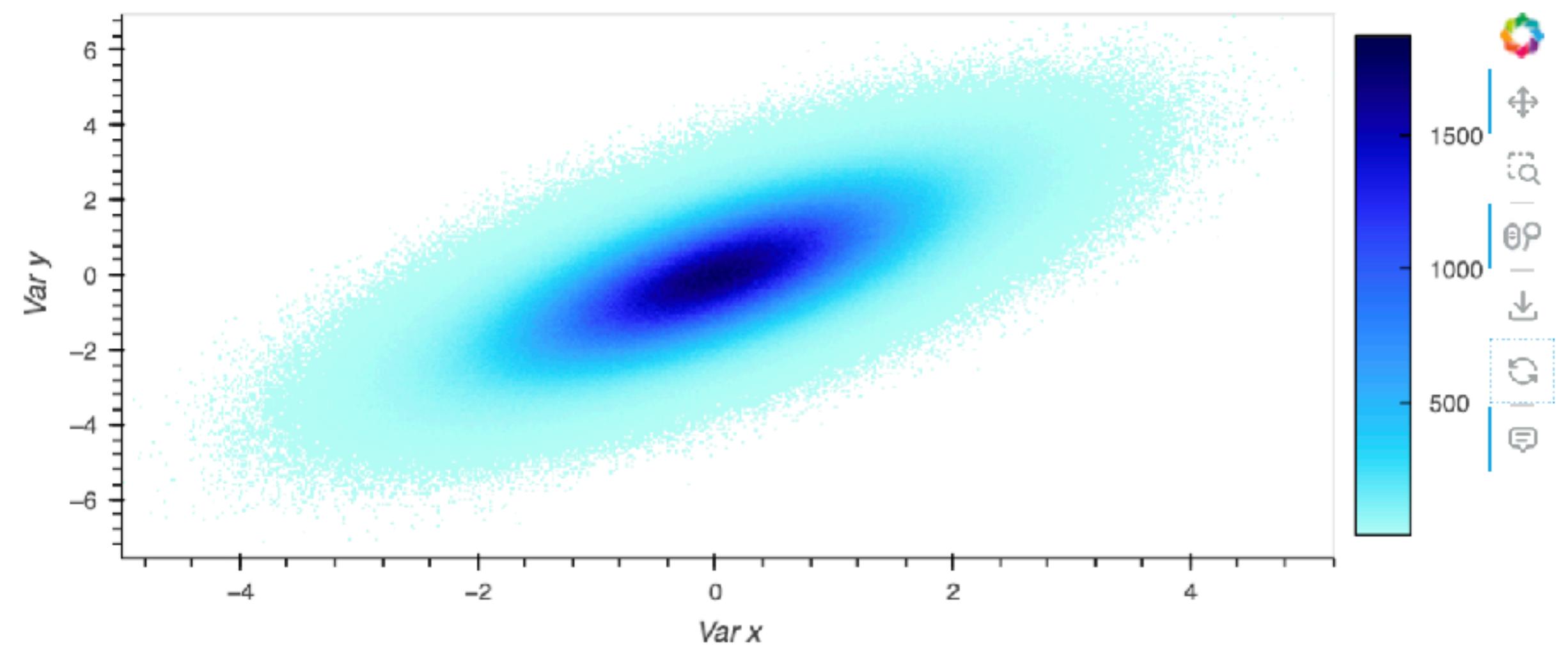
Filters by class

1. HVPLLOT

VISUALISE 10M POINTS

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             rasterize=True
                            )

# Show
scatter
```

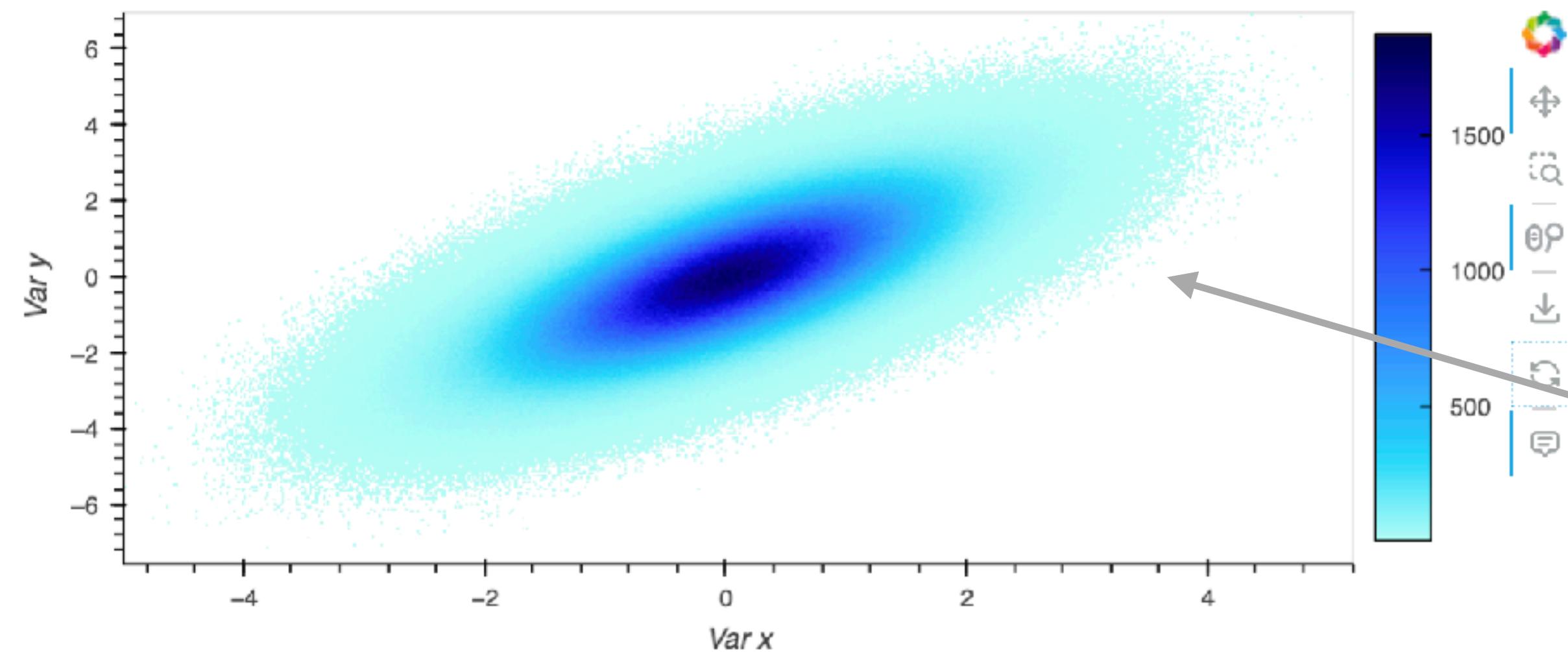


1. HVPLLOT

VISUALISE 10M POINTS

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             rasterize=True
                            )

# Show
scatter
```



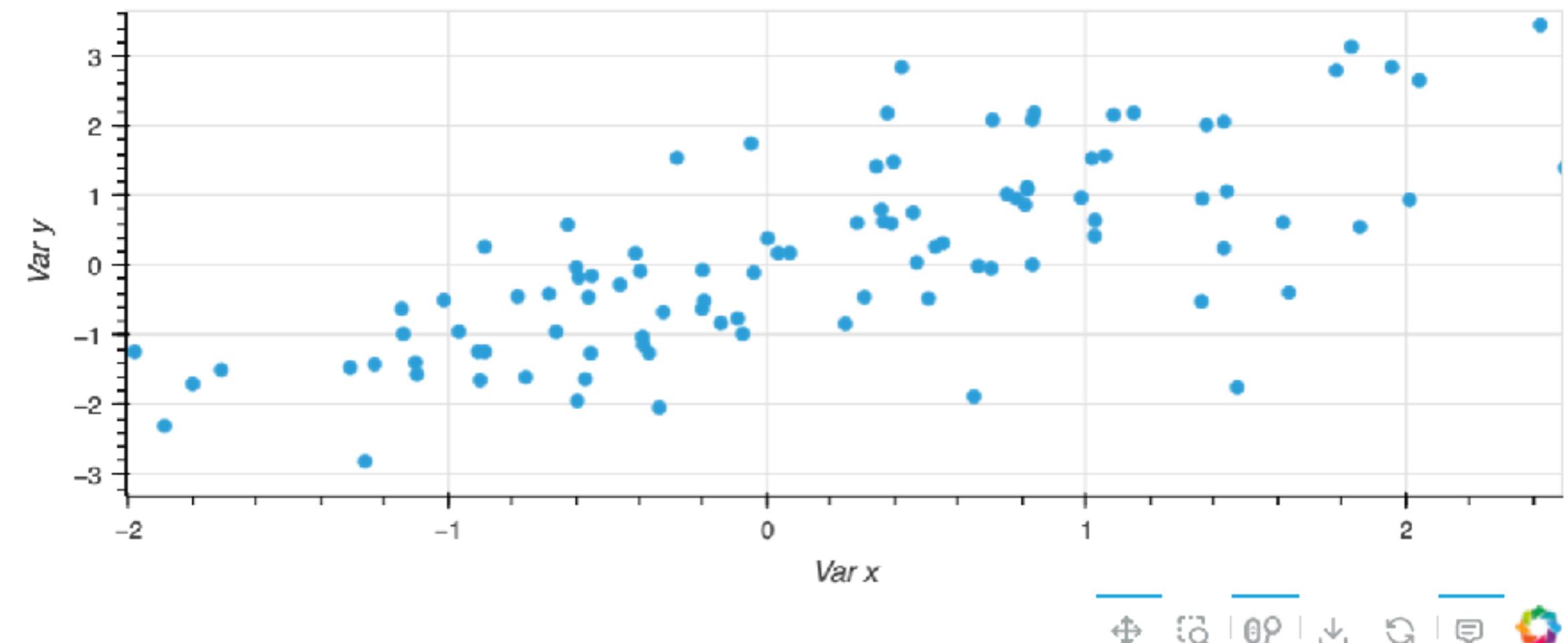
Plot updates when zoom in

1. HV PLOT

BOKEH PARAMETERS

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left",
                             )

# Show
scatter.opts(show_grid=True,
             toolbar="below")
```



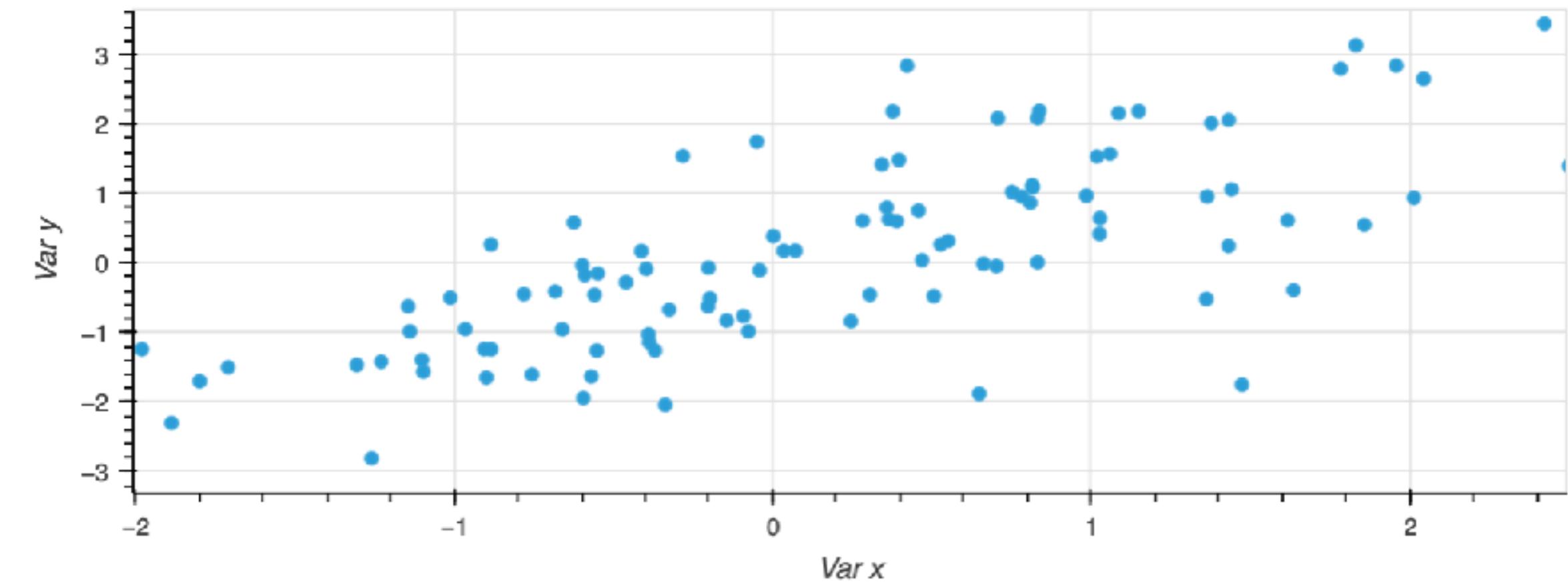
1. HV PLOT

BOKEH PARAMETERS

Default, work with any backend

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left",
                             )

# Show
scatter.opts(show_grid=True,
              toolbar="below")
```



1. HV PLOT

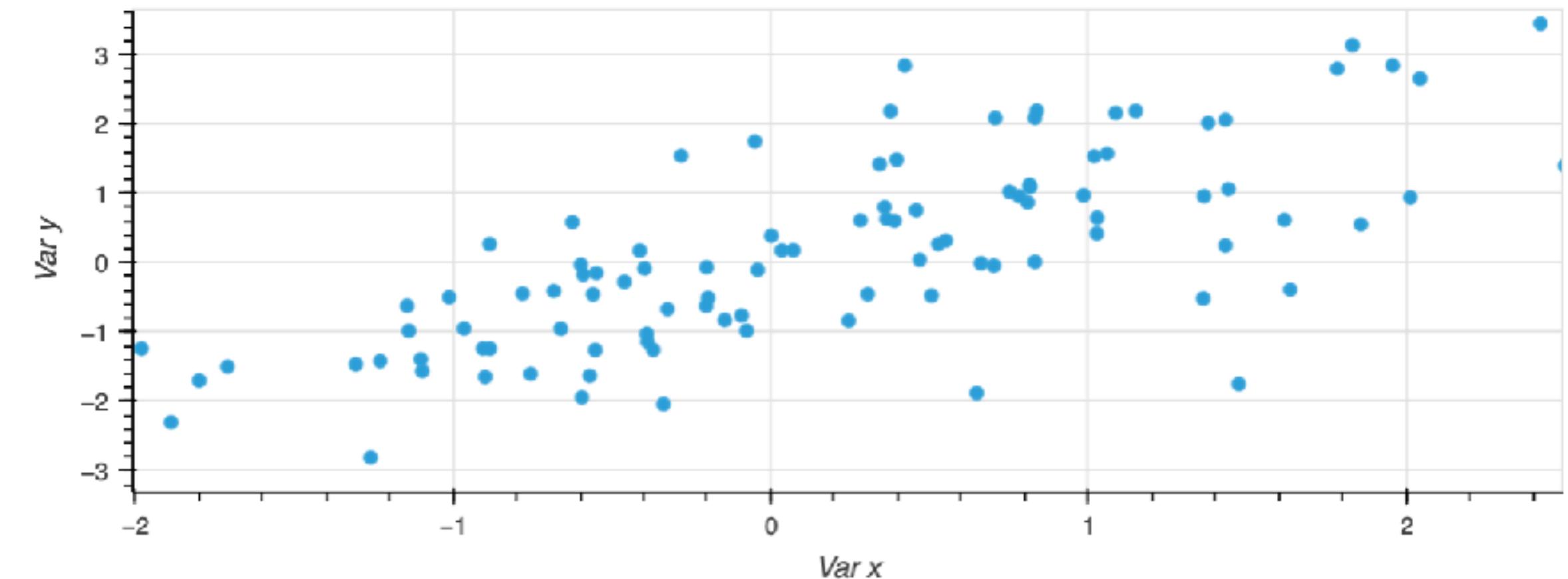
BOKEH PARAMETERS

Default, work with any backend

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left",
                             )
```

```
# Show
scatter.opts(show_grid=True,
             toolbar="below")
```

Get passed to the underlying library (bokeh)



1. HVPLOT

ONE OF THE MANY THINGS WHY I LOVE BOKEH: INFORMATIVE ERROR MESSAGES

```
# Plot
scatter = df.hvplot.scatter(x='x',
                             y='y',
                             xlabel="Var x",
                             ylabel="Var y",
                             legend="top_left",
                             )

# Show
scatter.opts(show_grid=True,
              toolbar="bottom")
---

~/miniforge3/envs/st/lib/python3.10/site-packages/param/parameterized.py in __set__(self, obj, val)
 1199         val = self.set_hook(obj, val)
 1200
-> 1201     self._validate(val)
 1202
 1203     _old = NotImplemented

~/miniforge3/envs/st/lib/python3.10/site-packages/param/__init__.py in _validate(self, val)
 1320         break
 1321     items = '[' + ', '.join(items) + limiter
-> 1322     raise ValueError("%s not in parameter%s's list of possible objects, "
 1323                      "valid options include %s" % (val, attrib_name, items))
 1324

ValueError: bottom not in parameter toolbar's list of possible objects, valid options include [above, below, left, right, disable, None]
```

1. HVPLLOT

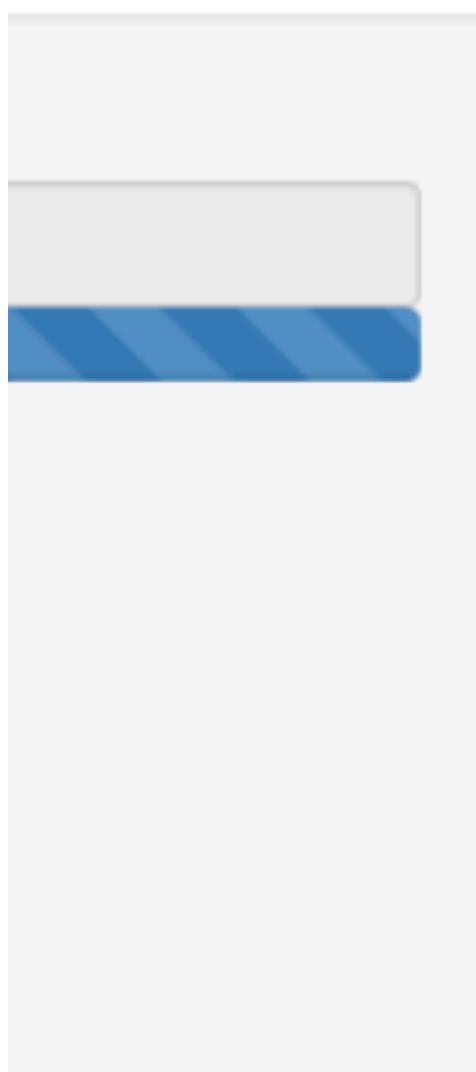
SHINY

Validation App

Data set

- mtcars
- mtifful
- iris

Error: invalid first argument
Error: invalid first argument



About file Data Summary

Error: An error has occurred. Check your logs or contact the app author for clarification.

An error has occurred

The application failed to start (exited with code 1).

```
— Attaching packages ————— tidyverse 1.3.0 —  
✓ ggplot2 3.3.0    ✓ purrr   0.3.4  
✓ tibble  3.0.1    ✓ dplyr   1.0.1  
✓ tidyr   1.1.0    ✓ stringr 1.4.0  
✓ readr   1.3.1    ✓ forcats 0.5.0  
— Conflicts ————— tidyverse_conflicts() —  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag()   masks stats::lag()  
Error in value[[3L]](cond) : app.R did not return a shiny.appobj object.  
Calls: local ... tryCatch -> tryCatchList -> tryCatchOne -> <Anonymous>  
Execution halted
```

 R Session Aborted

R encountered a fatal error.
The session was terminated.

[Start New Session](#)

2. PANEL

2. PANEL

WHY WE NEED PANEL

Creating interactive web apps, allowing to:

- easily arrange visual components (layout)
- make them respond to user input (reactivity)

2. PANEL

PANES

Blocks to add to your app:

- * Visualizations
- * Tables
- * Images/Videos/PDF
- * Markdown/HTML
- * Alerts
- * Streaming data (Perspective)

2. PANEL

PANES

Blocks to add to your app:

- * Visualizations
- * Tables
- * Images/Videos/PDF
- * Markdown/HTML
- * Alerts
- * Streaming data (Perspective)

```
png_pane = pn.pane.Image('https://assets.holoviz.org/panel/samples/png_sample.png', width=400)  
png_pane
```



2. PANEL

PANES

Blocks to add to your app:

- * Visualizations
- * Tables
- * Images/Videos/PDF
- * Markdown/HTML
- * Alerts
- * Streaming data (Perspective)

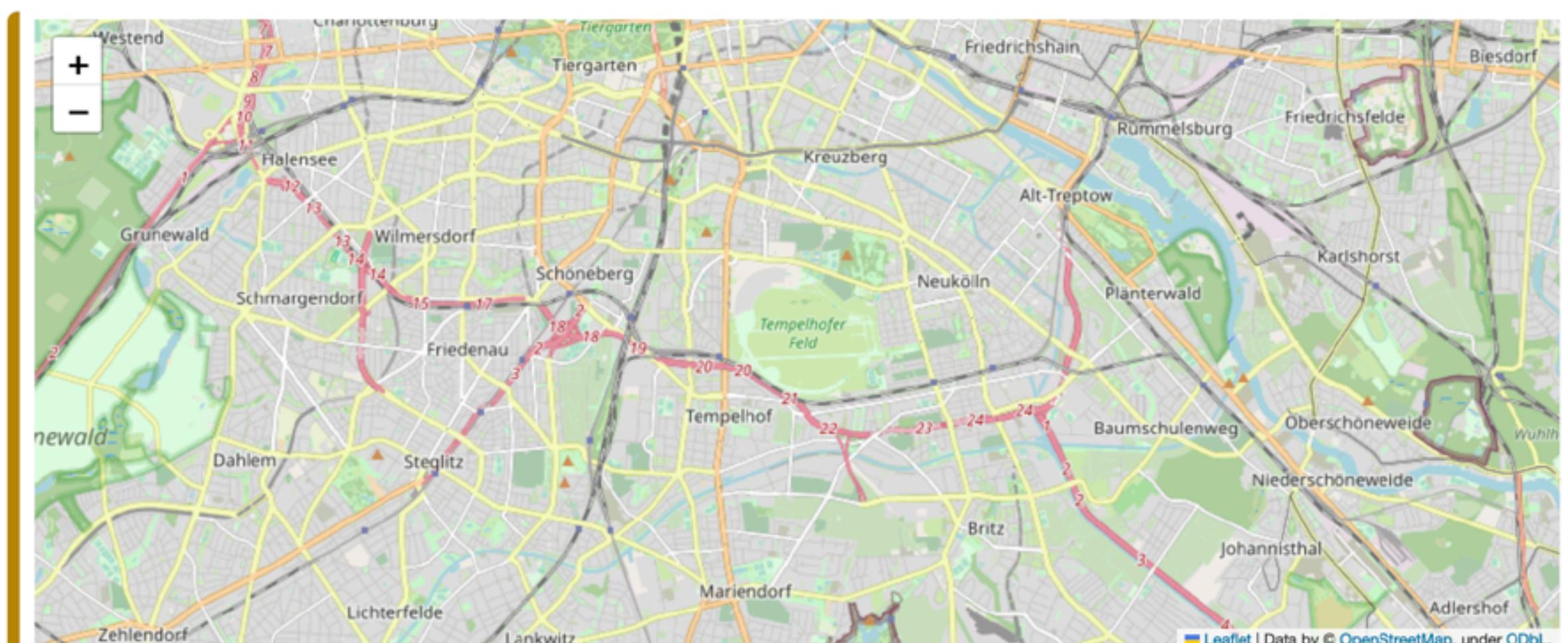
<https://panel.holoviz.org/reference/index.html#panes>

```
png_pane = pn.pane.Image('https://assets.holoviz.org/panel/samples/png_sample.png', width=400)
```

```
png_pane
```



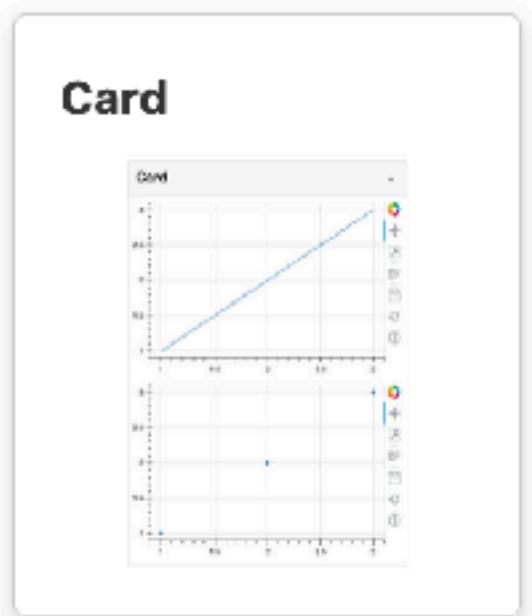
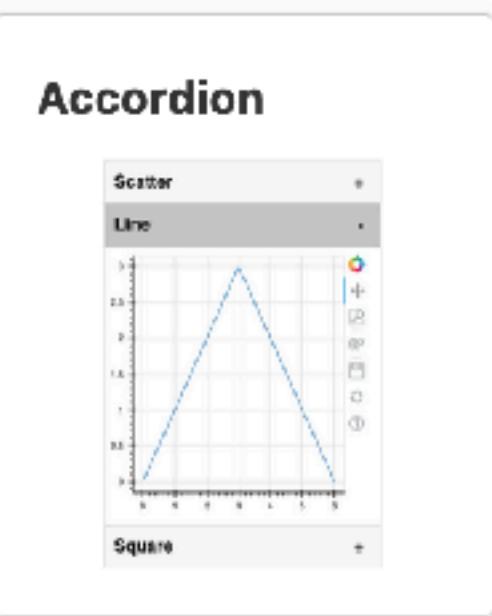
```
m = folium.Map(location=[52.51, 13.39], zoom_start=12)  
folium_pane = pn.pane.plot.Folium(m, height=400)  
folium_pane
```



2. PANEL

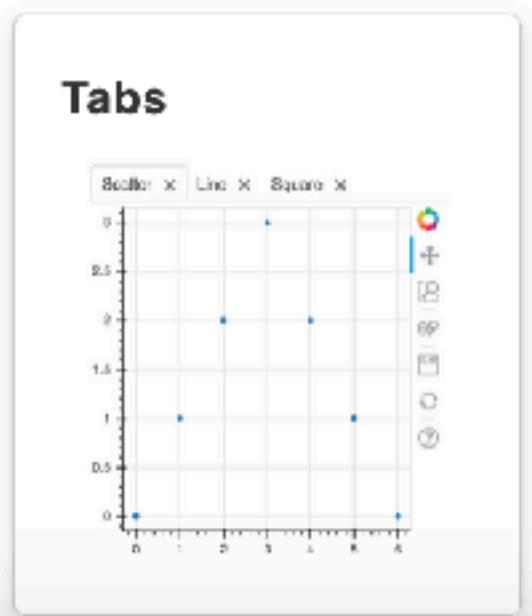
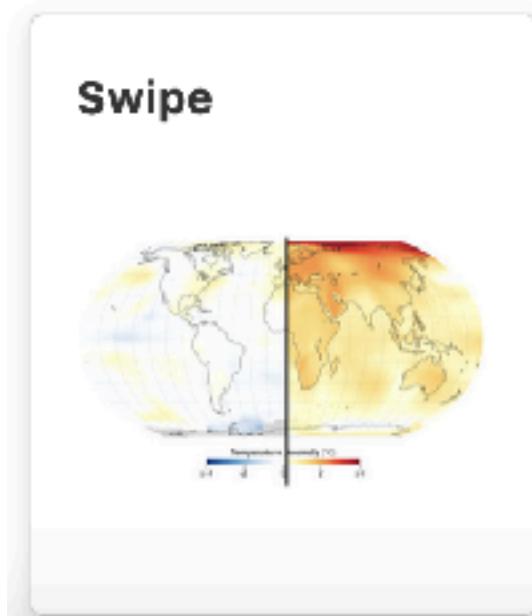
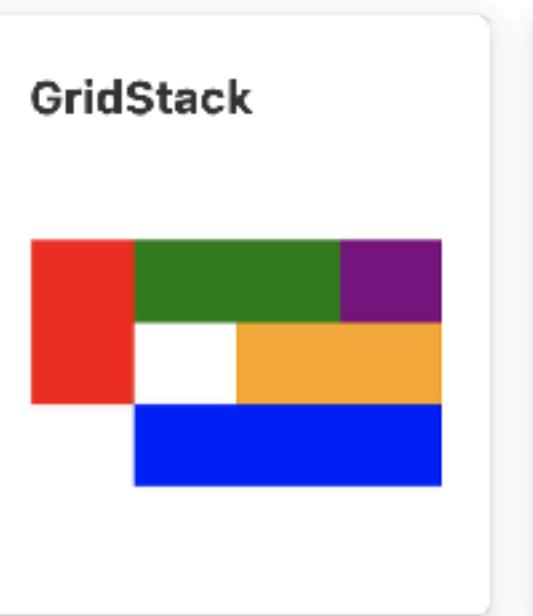
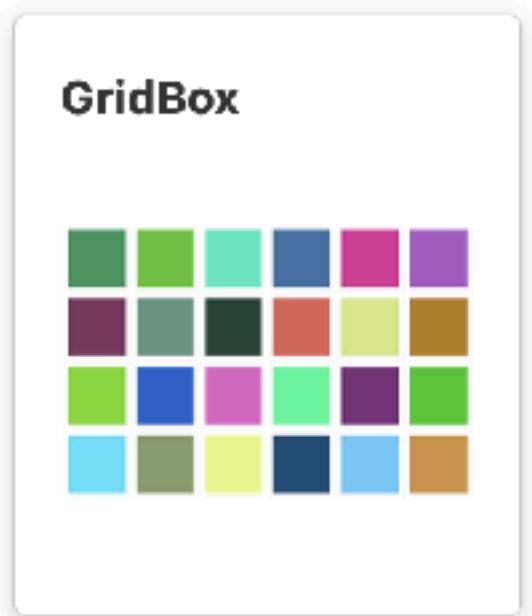
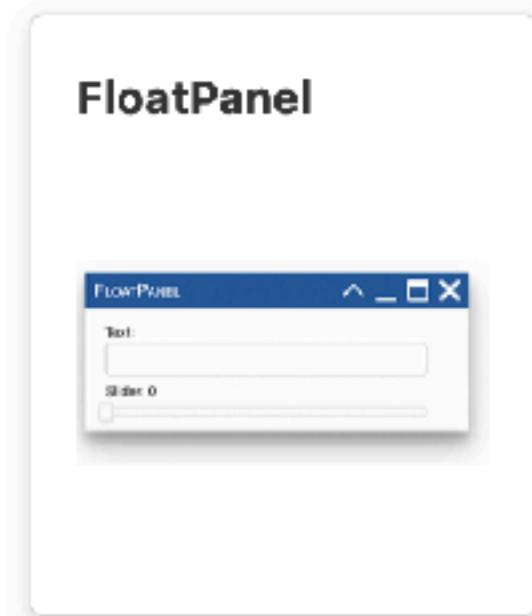
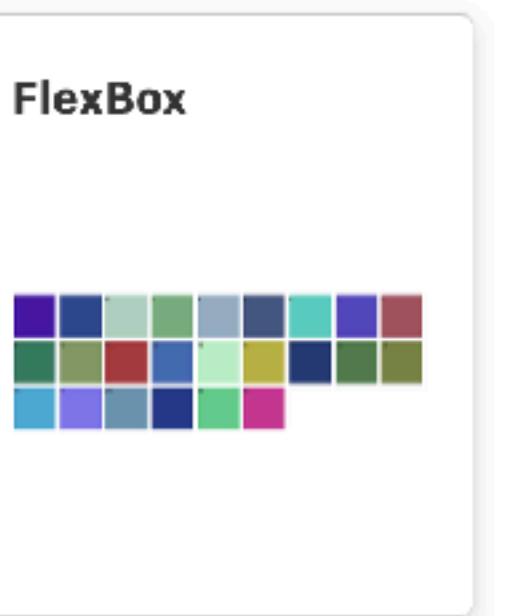
LAYOUT

Organize your blocks



Column

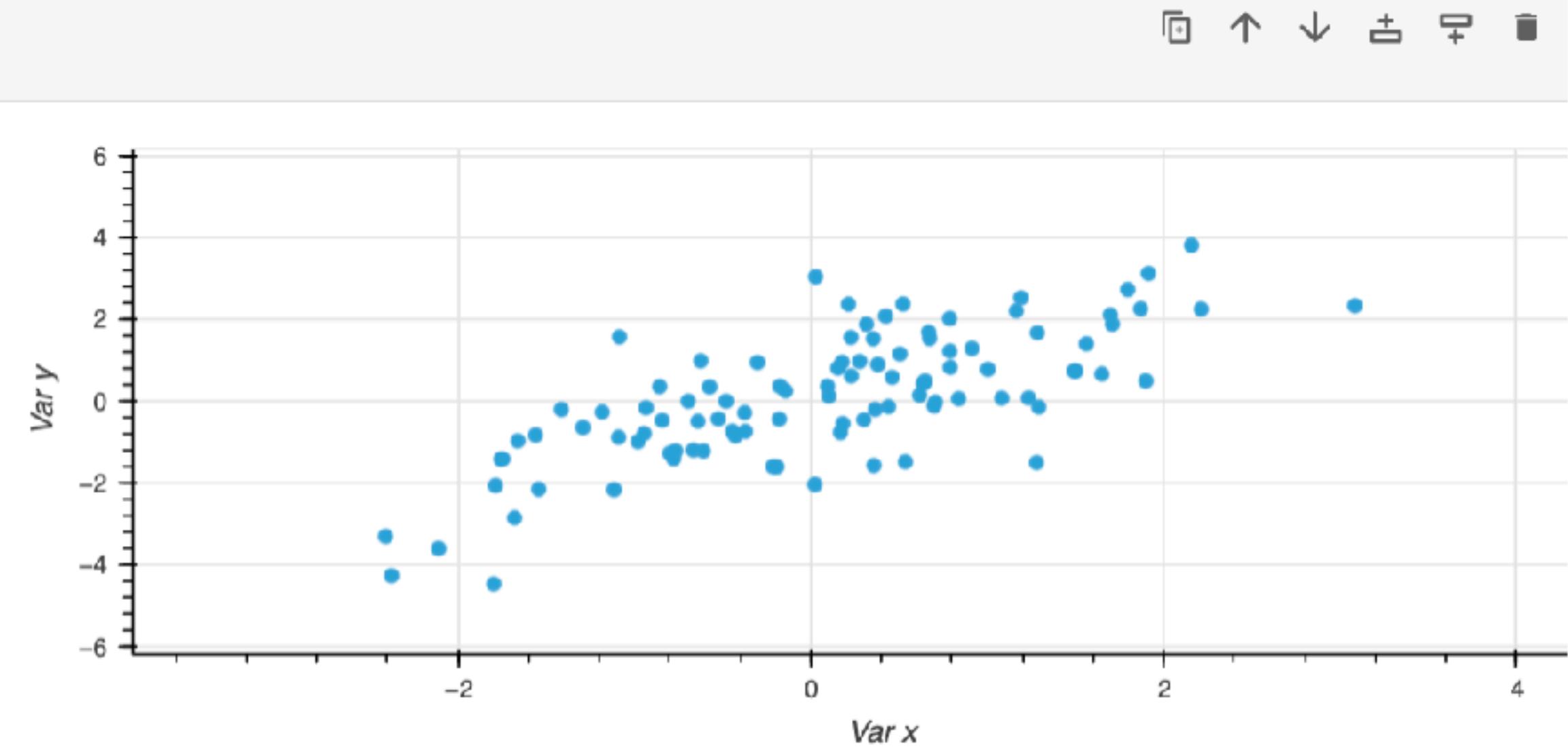
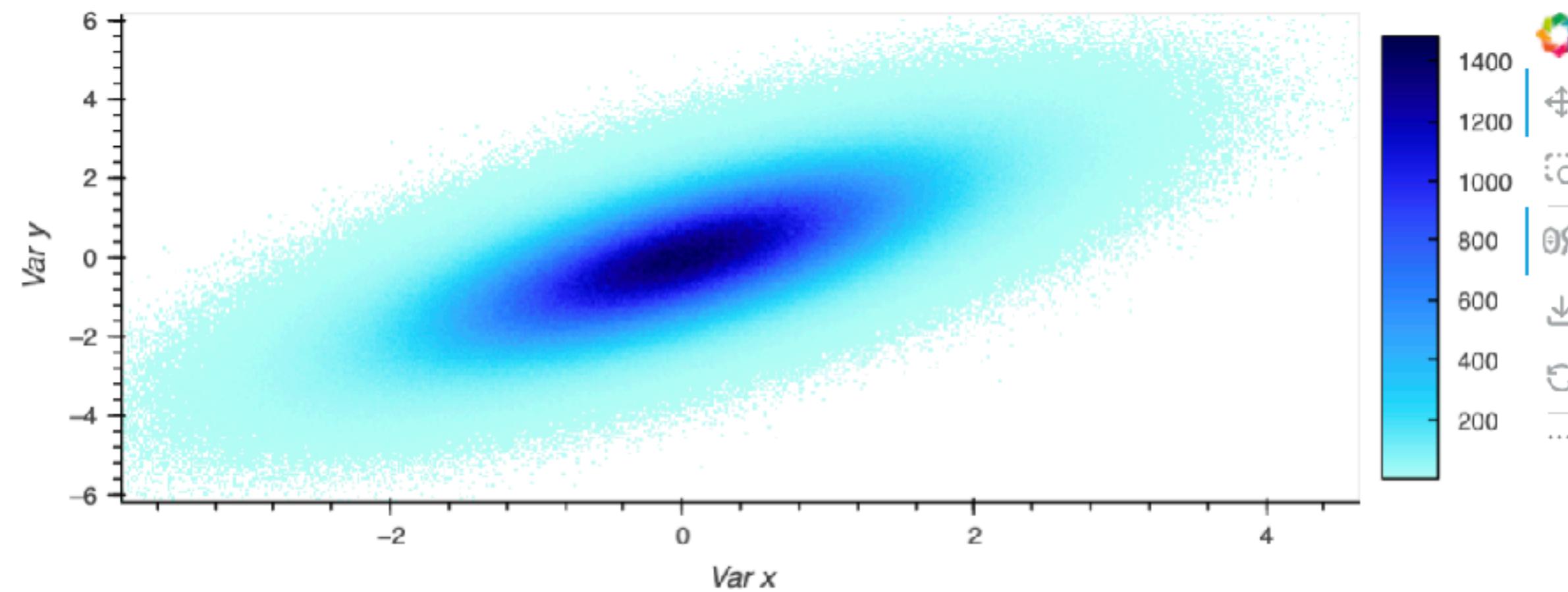
Divider



WidgetBox

2. PANEL

```
# Combine plots Linked by default!!  
pn.Row(large_scatter, scatter)
```



TEMPLATES

We often don't need full control, just want some defaults

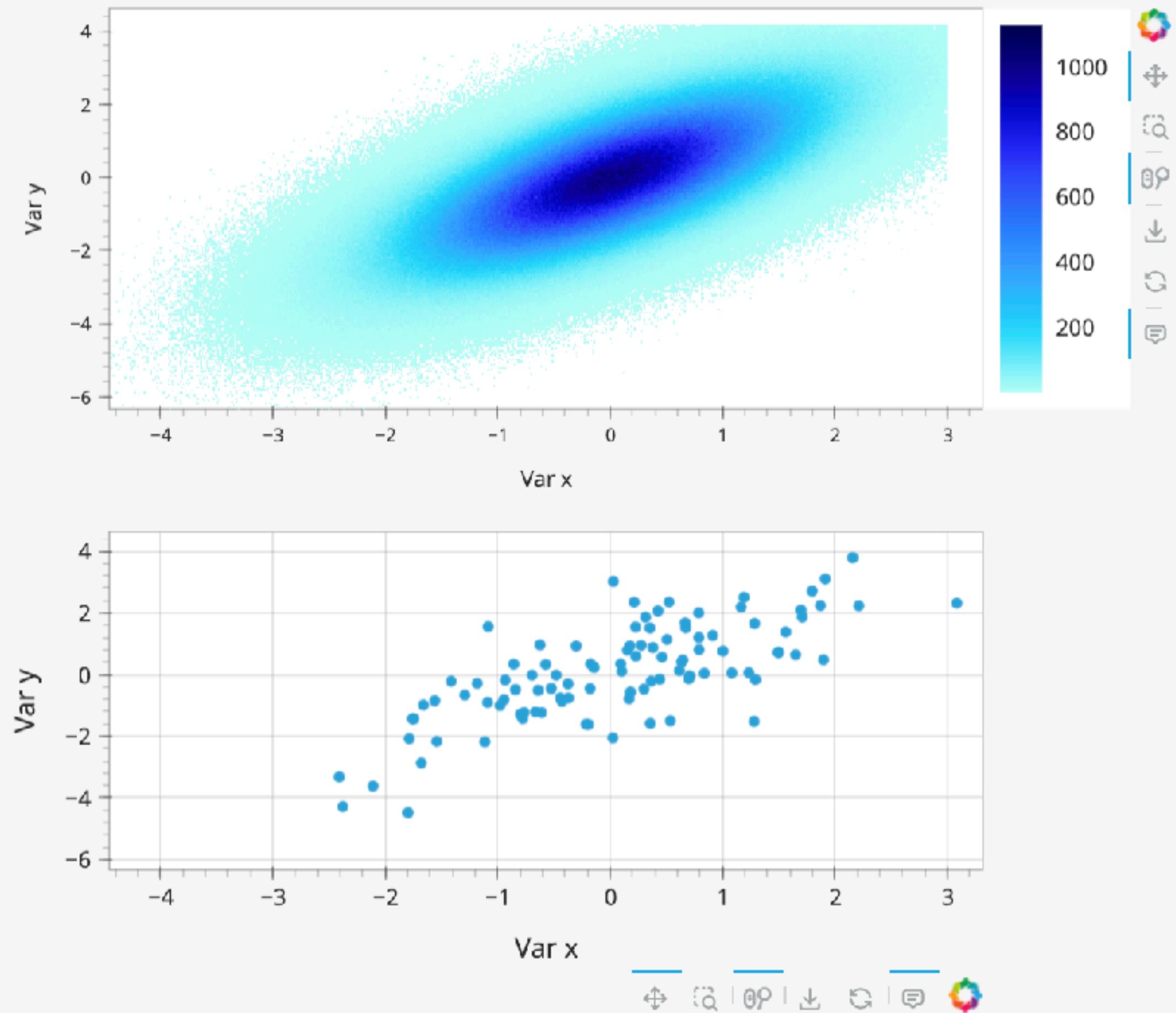
Don't work well on notebooks (use Jupyter lab, or template.show())

```
# Instantiate the template with widgets displayed in the sidebar
template = pn.template.BootstrapTemplate(
    sidebar="Here we would usually have some controls",
    main=[pn.Column(large_scatter, scatter, sizing_mode="scale_width")]
)
```

```
template.show()
```

≡ Panel Application

Here we would usually have some controls



```

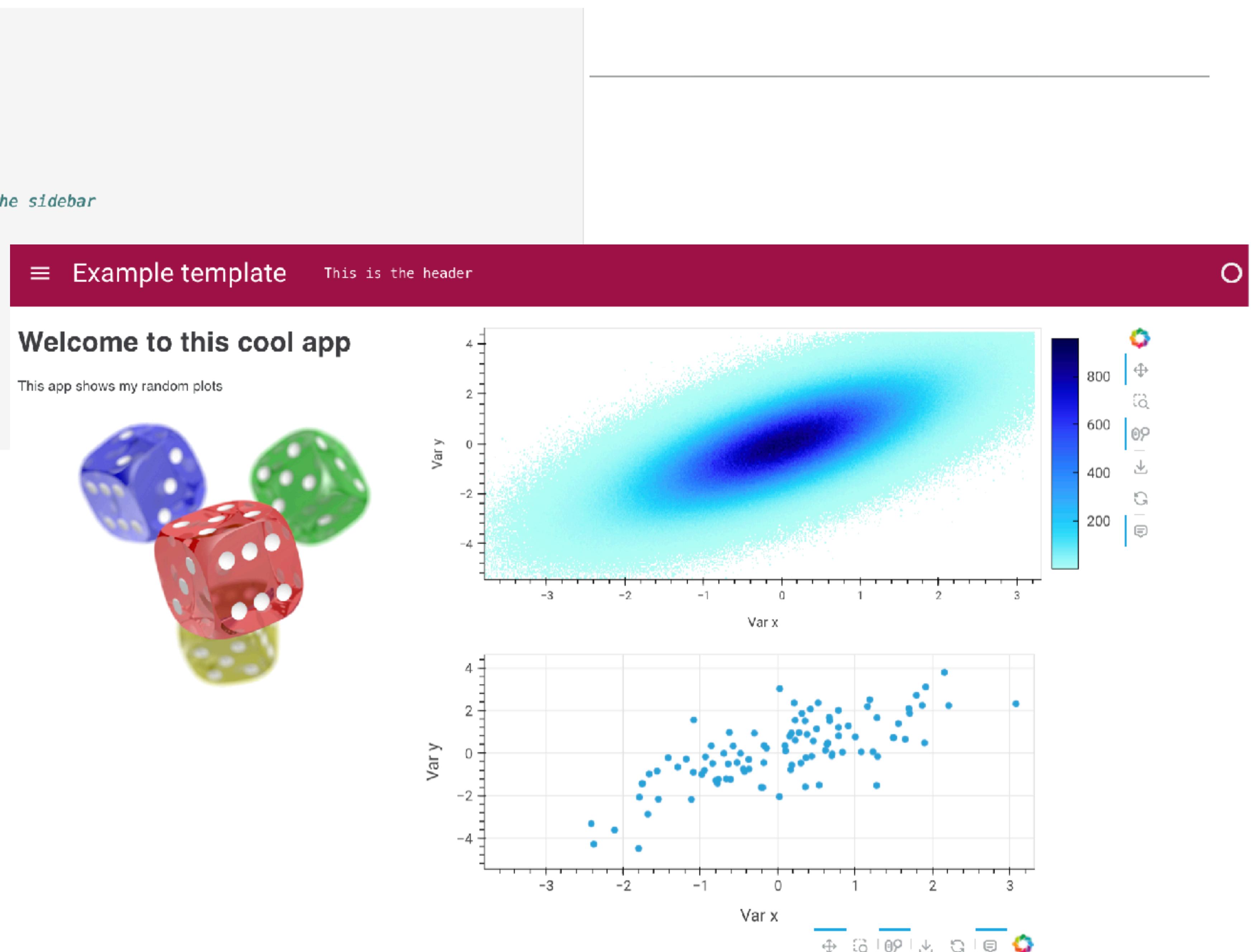
explanation = """
# Welcome to this cool app

This app shows my random plots
"""

explanation = pn.pane.Markdown(explanation)

# Instantiate the template with widgets displayed in the sidebar
template = pn.template.MaterialTemplate(
    title='Example template',
    sidebar=[explanation, png_panel],
    main=[pn.Column(large_scatter, scatter)],
    header=pn.pane.Str("This is the header"),
    header_background="#A01346",
)
template.show()

```



SAVING APP/VISUALIZATIONS (I)

Interactive by default!

The data gets attached to the HTML/JS code

No server needed (for anything we've seen so far)

```
# Data gets embedded  
template.save("apps/lecture_app.html")
```

EXERCISE

Build a panel app with no interactivity.

→ Check B_ex0_hvplot

→ Work on B_ex1_intro

3. INTERACTIVITY/ REACTIVITY

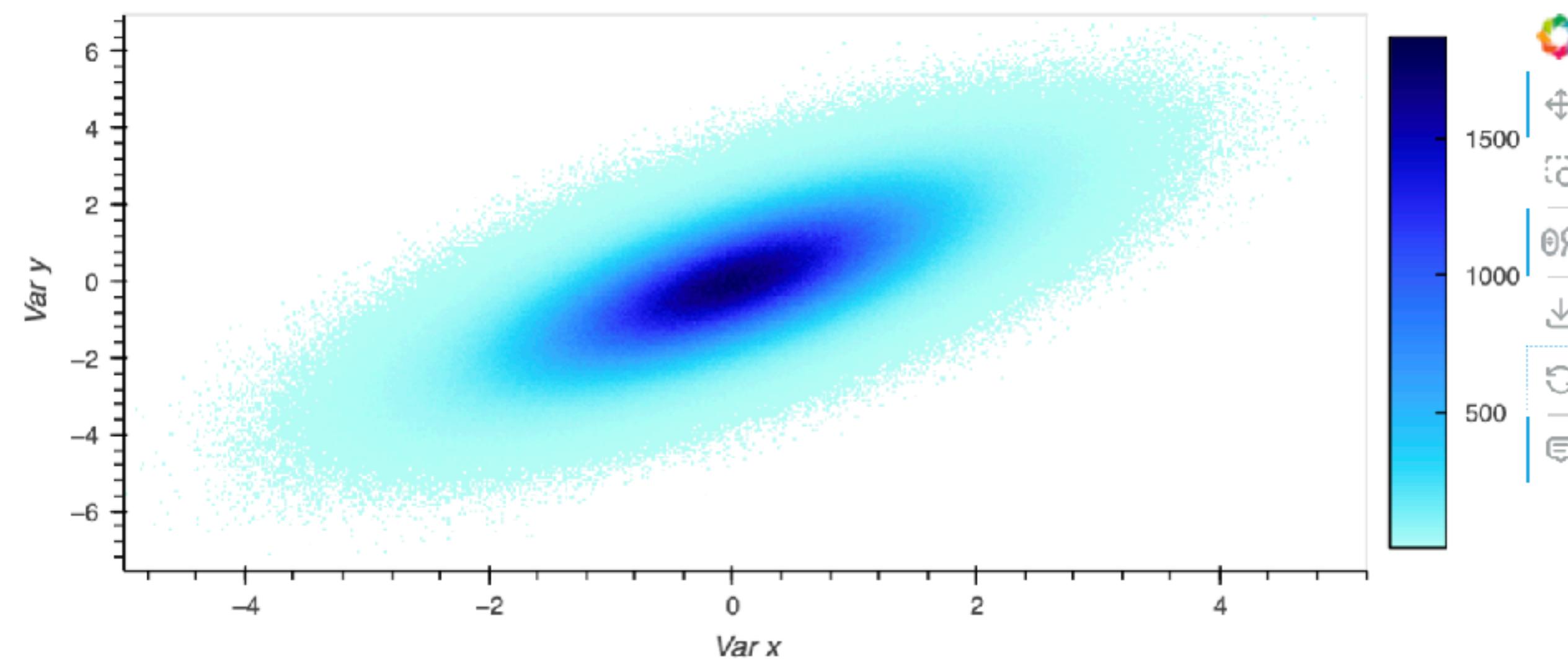
3. INTERACTIVITY/REACTIVITY

INTERACTING WITH THE VISUALIZATION

Bokeh provides some interactivity, such as zooming in
Rasterize aggregates points in a smart way (using datashader)

```
# Plot
scatter = df.hvplot.scatter(x='x',
                            y='y',
                            xlabel="Var x",
                            ylabel="Var y",
                            rasterize=True
                            )

# Show
scatter
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY / WIDGETS

We want to interact with the data:

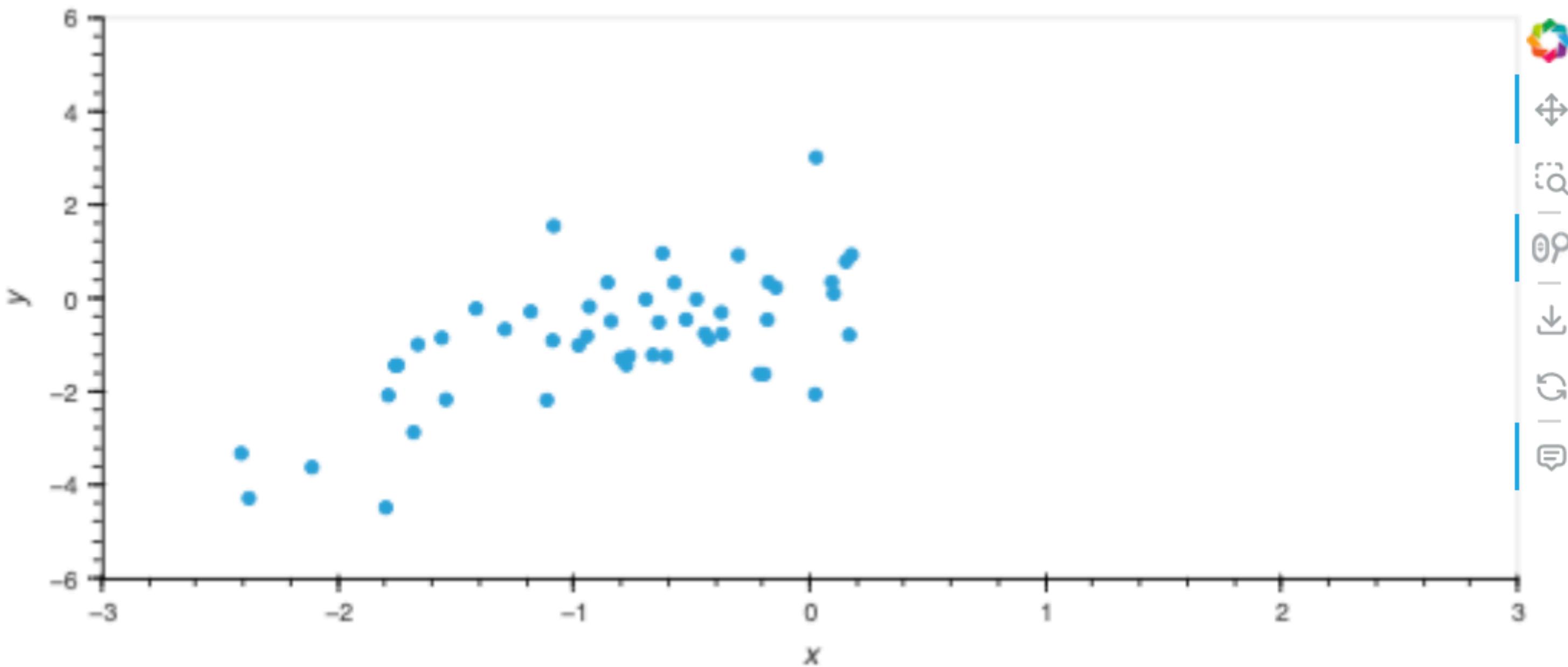
- * Change **visualisations** depending on different inputs (plot by class, select a range of values, rolling mean...)
- * Allowing **app** users to download files, record sounds, write passwords, etc

Widgets provide controls that can trigger events
(e.g. when a number is selected → update viz)

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

(df
    .loc[df["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```

class A ▾

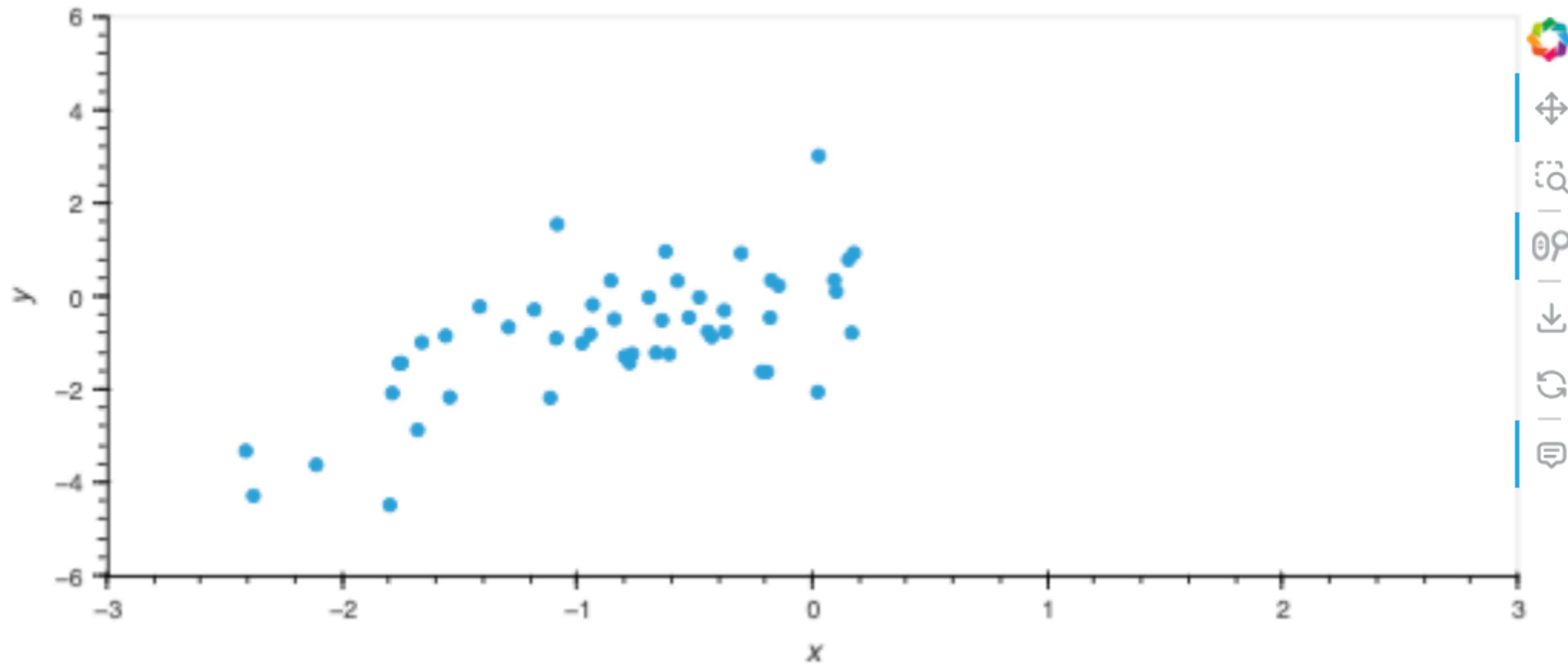


Widget, provides value

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

(df
    .loc[df["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```

class A ▾



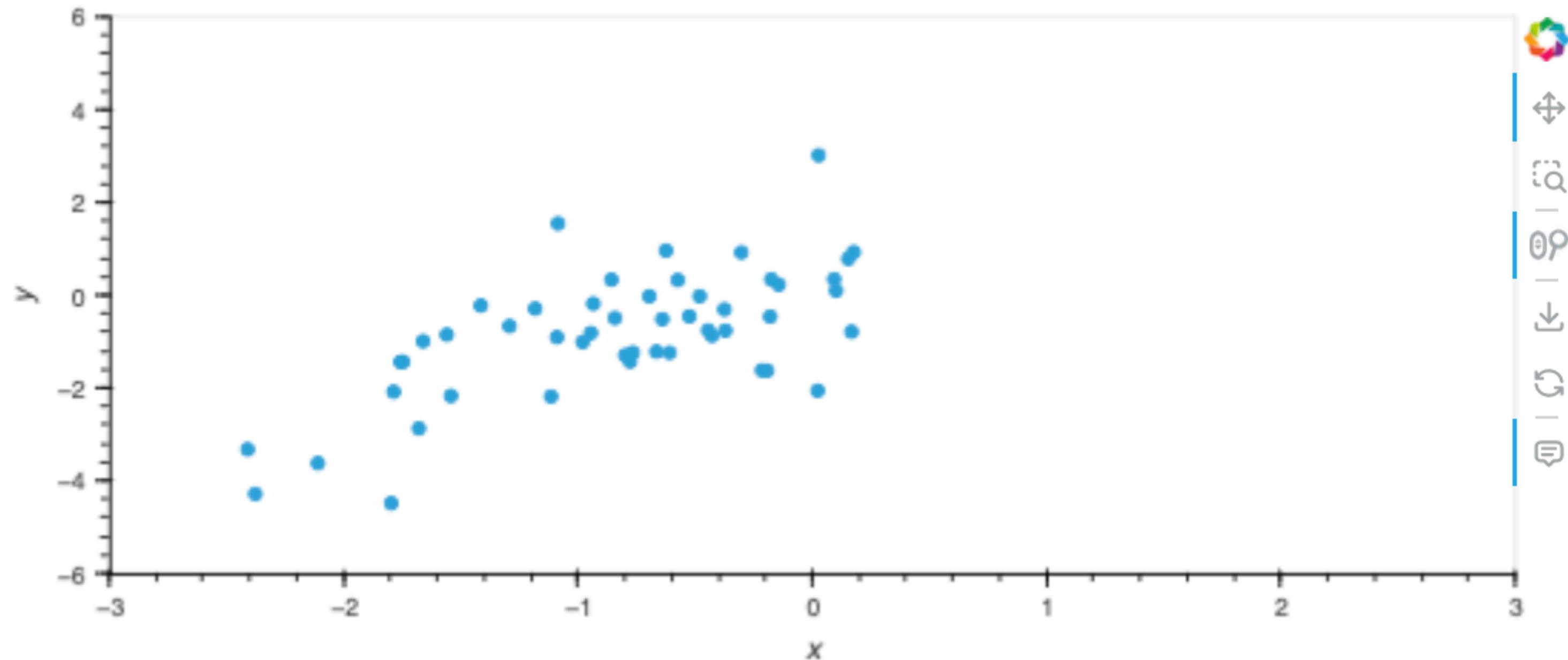
Widget, provides value

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

(df
    .loc[df["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```

Value is used to filter data

class A ▾



3. INTERACTIVITY/REACTIVITY

REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

3. INTERACTIVITY/REACTIVITY

REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

Select

Select
Biology

Select
 Kinology
Chemistry
Physics

RadioBoxGroup

Biology Chemistry Physics

Biology Chemistry Physics

Biology
 Chemistry
 Physics

MultiChoice

Apple | Pear |

Banana Press to select

Strawberry

TextInput

Text Input
A string

Text Input
Enter a string here...

Text Input
Enter a string here...

DateRangeSlider

Date Range Slider: 25 Mar 2017 .. 16 Aug 2018

Date Range Slider: 15 Sep 2017 .. 24 May 2018

Date Range Slider: 13 May 2017 .. 01 Jan 2019

FloatInput

FloatInput:
5.1

FloatSlider

Float Slider: 1.57

Float Slider: 2.31

Float Slider: 0.80

3. INTERACTIVITY/REACTIVITY

REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

Select

Select
Biology

Select
 Kinology
Chemistry
Physics

RadioBoxGroup

Biology Chemistry Physics

Biology
Chemistry
Physics

MultiChoice

Apple | X Pear | X

Banana Press to select

Strawberry

TextInput

Text Input
A string

Text Input
Enter a string here...

Text Input
Enter a string here...

DateRangeSlider

Date Range Slider: 25 Mar 2017 .. 16 Aug 2018

Date Range Slider: 15 Sep 2017 .. 24 May 2018

Date Range Slider: 13 May 2017 .. 01 Jan 2019

FloatInput

FloatInput:
5.1

FloatSlider

Float Slider: 1.57

Float Slider: 2.31

Float Slider: 0.80

Button



3. INTERACTIVITY/REACTIVITY

REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

Select

Select
Biology

Select
 Kinology
Chemistry
Physics

RadioBoxGroup

Biology Chemistry Physics

Biology
Chemistry
Physics

MultiChoice

Apple | X Pear | X

Banana Press to select

Strawberry

TextInput

Text Input
A string

Text Input
Enter a string here...

Text Input
Enter a string here...

DateRangeSlider

Date Range Slider: 25 Mar 2017 .. 16 Aug 2018

Date Range Slider: 15 Sep 2017 .. 24 May 2018

Date Range Slider: 13 May 2017 .. 01 Jan 2019

FloatInput

FloatInput:
5.1

FloatSlider

Float Slider: 1.57

Float Slider: 2.31

Float Slider: 0.80

Switch



Button

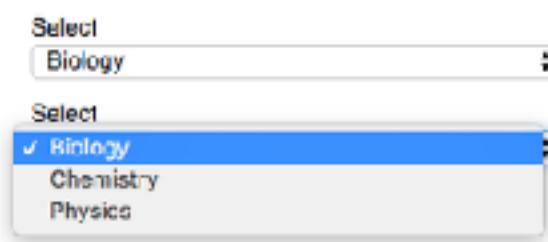


3. INTERACTIVITY/REACTIVITY

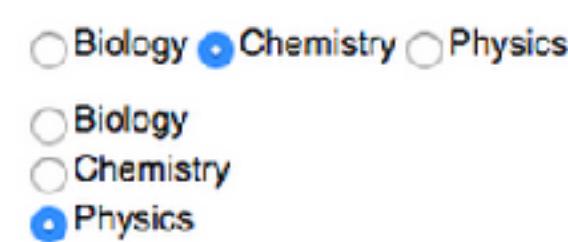
REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

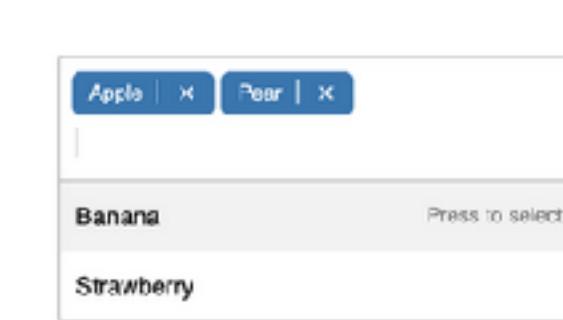
Select



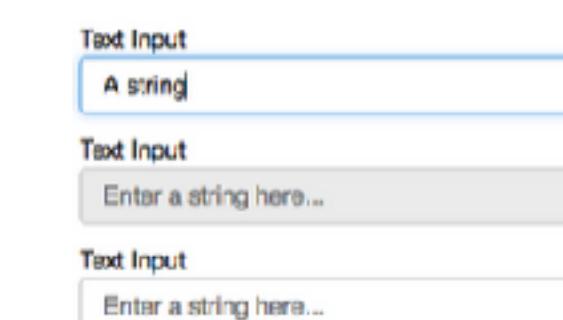
RadioBoxGroup



MultiChoice



TextInput



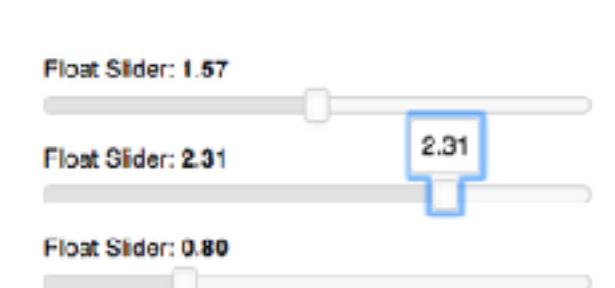
DateRangeSlider



FloatInput



FloatSlider



Switch



Button



FileDownload

Right-click to download using 'Save as' dialog
[Download FileDownload.pynb](#)

FileInput

[Choose File](#) Vega.html
[Choose File](#) No file chosen

FileSelector

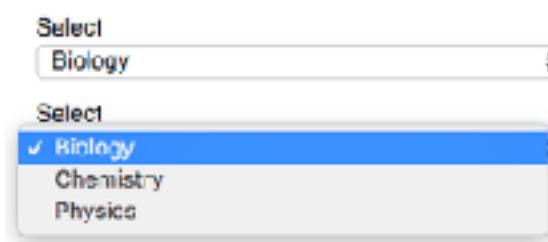


3. INTERACTIVITY/REACTIVITY

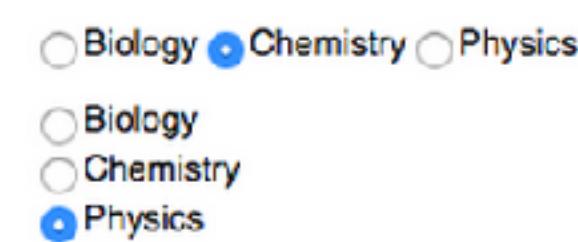
REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

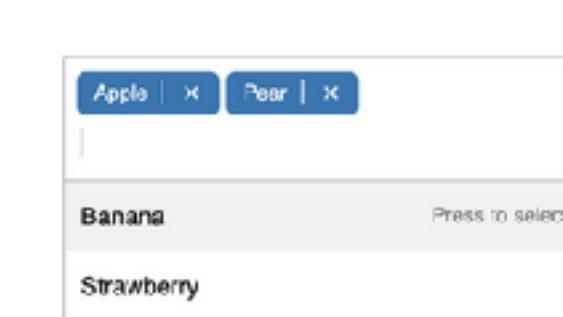
Select



RadioBoxGroup



MultiChoice



TextInput



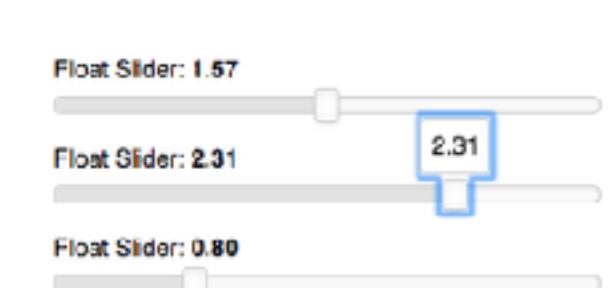
DateRangeSlider



FloatInput



FloatSlider



Switch



Button



FileDownload

Right-click to download using 'Save as' dialog
[Download FileDownload.pyrb](#)

FileInput

[Choose File](#) Vega.html
[Choose File](#) No file chosen

FileSelector



DiscretePlayer

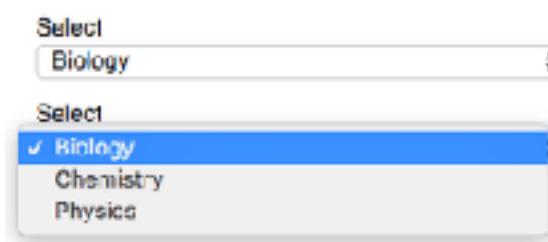


3. INTERACTIVITY/REACTIVITY

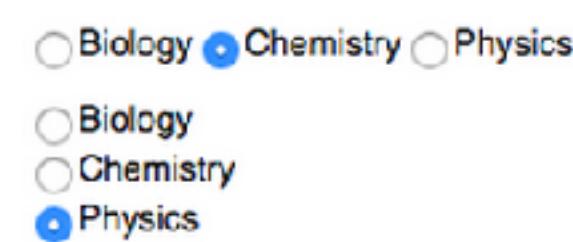
REACTIVITY / WIDGETS

<https://panel.holoviz.org/reference/index.html#widgets>

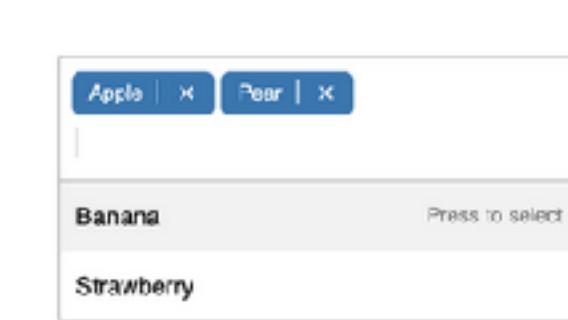
Select



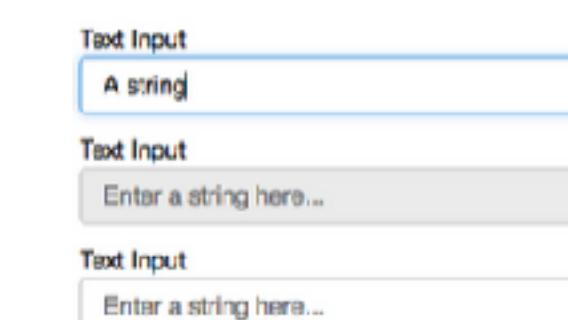
RadioBoxGroup



MultiChoice



TextInput



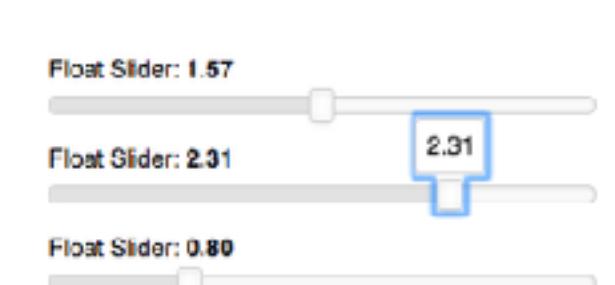
DateRangeSlider



FloatInput



FloatSlider



Switch



Button



FileDownload

Right-click to download using 'Save as' dialog
Download FileDownload.pyrb

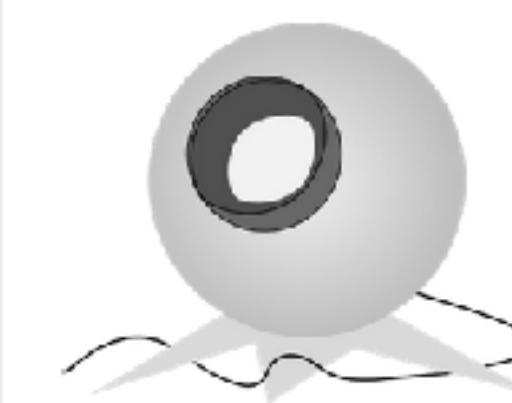
FileInput

Choose File Vega.html
Choose File No file chosen

FileSelector



VideoStream



DiscretePlayer



3. INTERACTIVITY/REACTIVITY

REACTIVITY

What is the value of y?

x = 1

y = x+1

x = 5

3. INTERACTIVITY/REACTIVITY

REACTIVITY

What is the value of y?

x = 1

y = x+1

x = 5

Usually, it would be 2

In reactive programming, x and y are reactive, updating
the value of x updates the value of y

This creates a graph of dependencies

3. INTERACTIVITY/REACTIVITY

REACTIVITY

What is the value of y?

$x = 1$

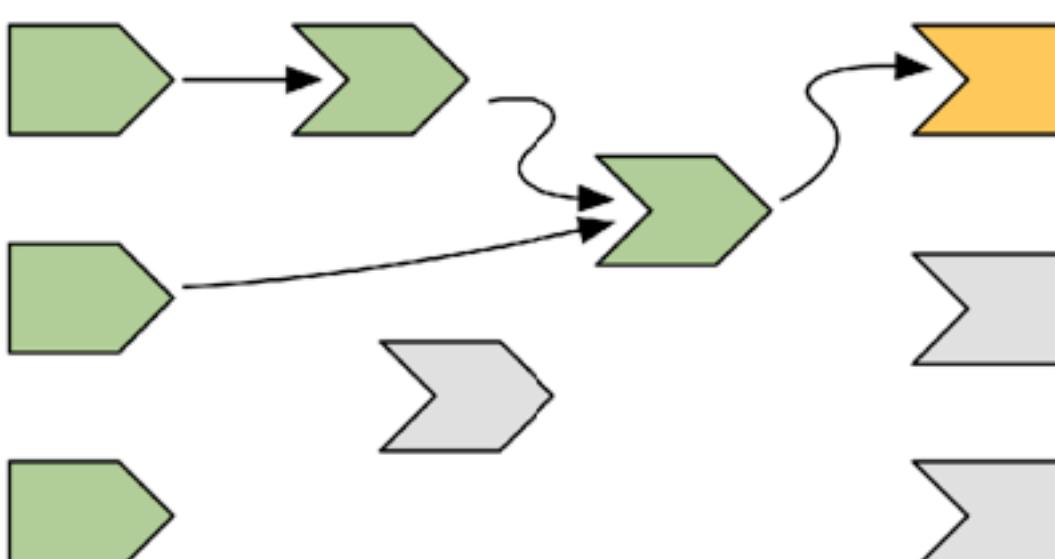
$y = x+1$

$x = 5$

Usually, it would be 2

In reactive programming, x and y are reactive, updating the value of x updates the value of y

This creates a graph of dependencies



Source: Mastering Shiny

3. INTERACTIVITY/REACTIVITY

REACTIVITY

class A

What is the value of y?

x = 1

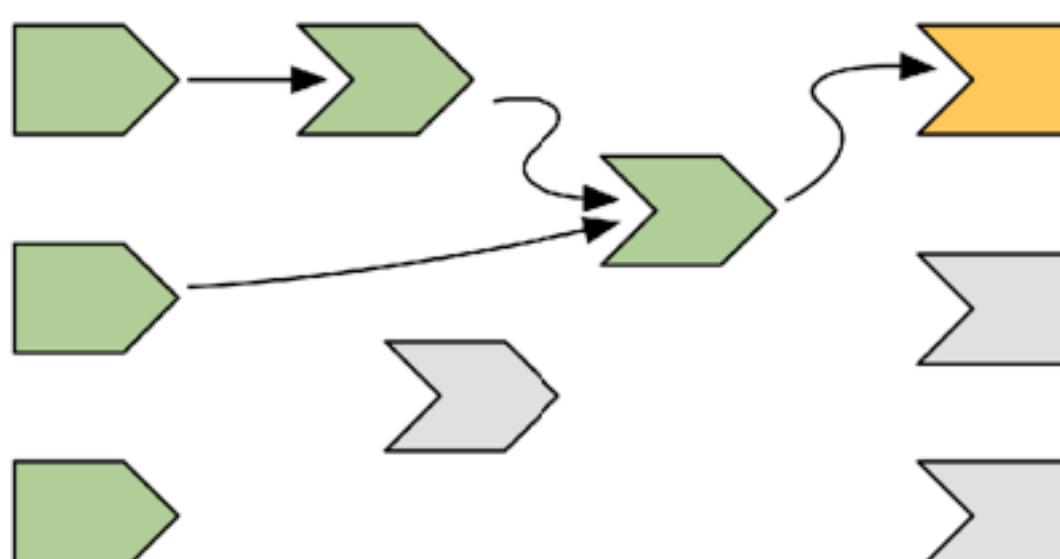
y = x+1

x = 5

Usually, it would be 2

In reactive programming, x and y are reactive, updating
the value of x updates the value of y

This creates a graph of dependencies



Source: Mastering Shiny

3. INTERACTIVITY/REACTIVITY

REACTIVITY

What is the value of y?

x = 1

y = x+1

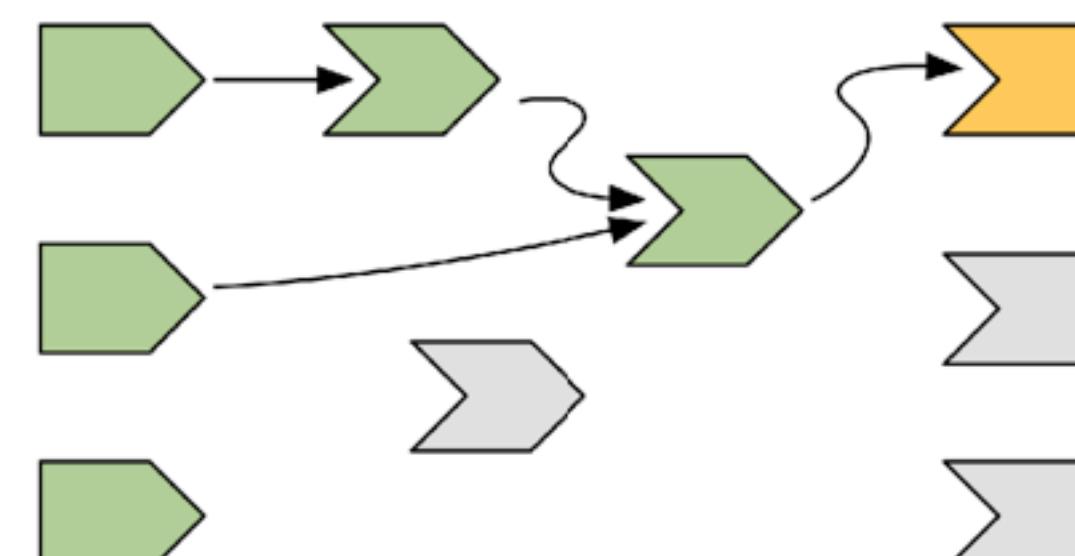
x = 5

Usually, it would be 2

In reactive programming, x and y are reactive, updating
the value of x updates the value of y

This creates a graph of dependencies

```
class A ▾  
    dfi = df.interactive()  
  
(dfi  
    .loc[dfi["class"] == w_select_class]
```



Source: Mastering Shiny

3. INTERACTIVITY/REACTIVITY

REACTIVITY

What is the value of y?

$x = 1$

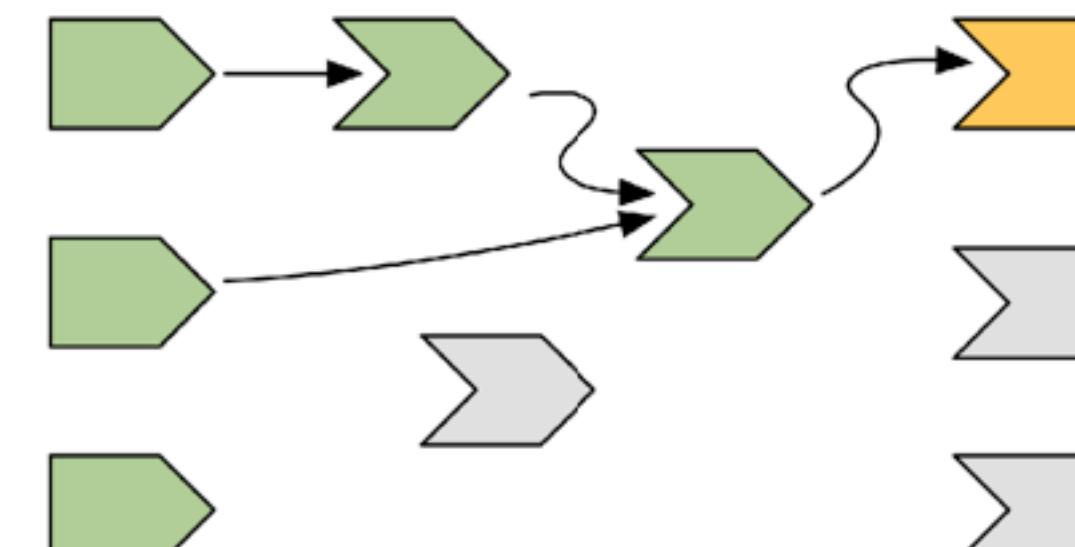
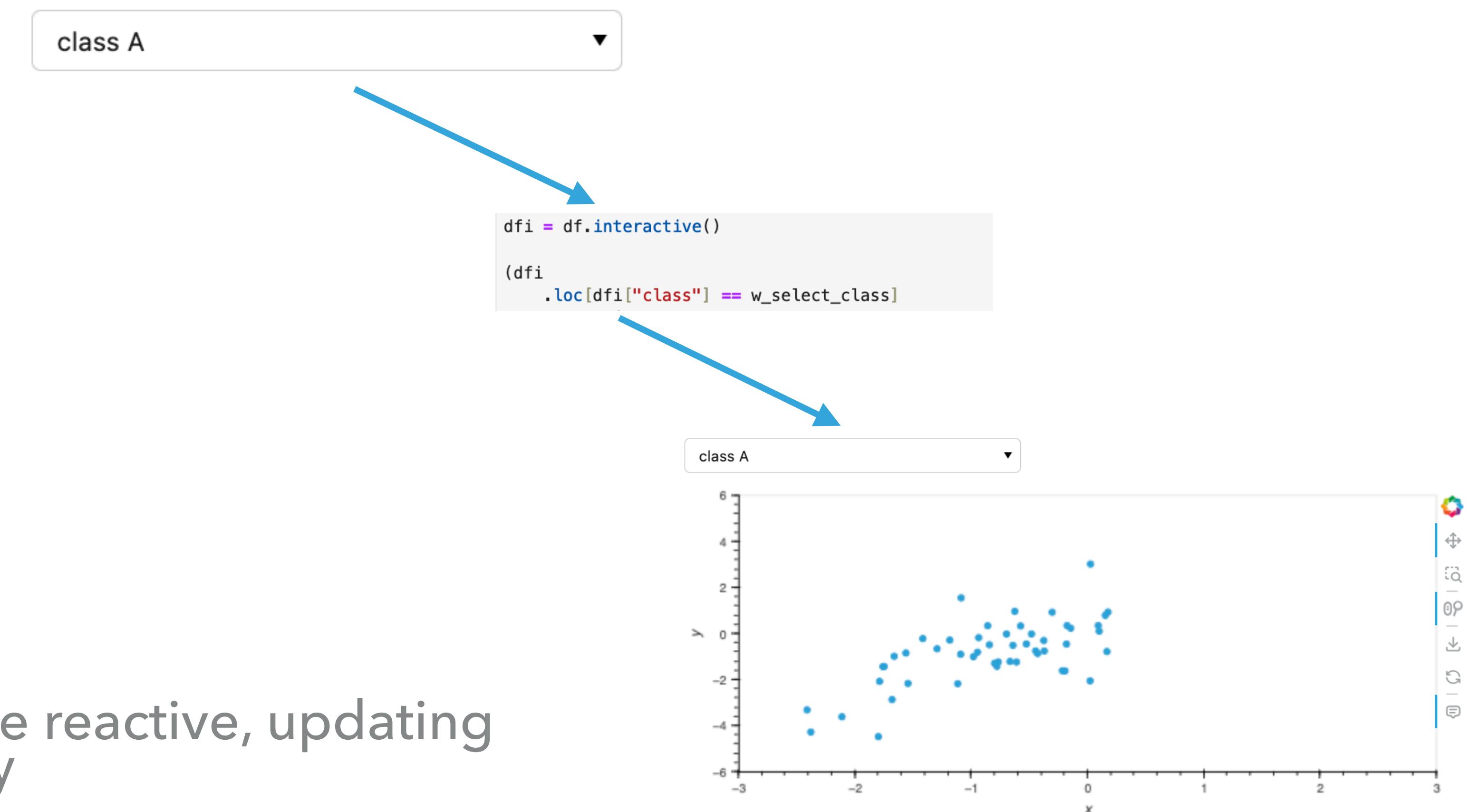
$y = x+1$

$x = 5$

Usually, it would be 2

In reactive programming, x and y are reactive, updating the value of x updates the value of y

This creates a graph of dependencies



Source: Mastering Shiny

3. INTERACTIVITY/REACTIVITY

REACTIVITY, SUMMARY

df.interactive()

pn.bind(function, widget)

widget.link(pane, widget_attribute="pane_attribute")

widget.param.watch(function, "widget_attribute")

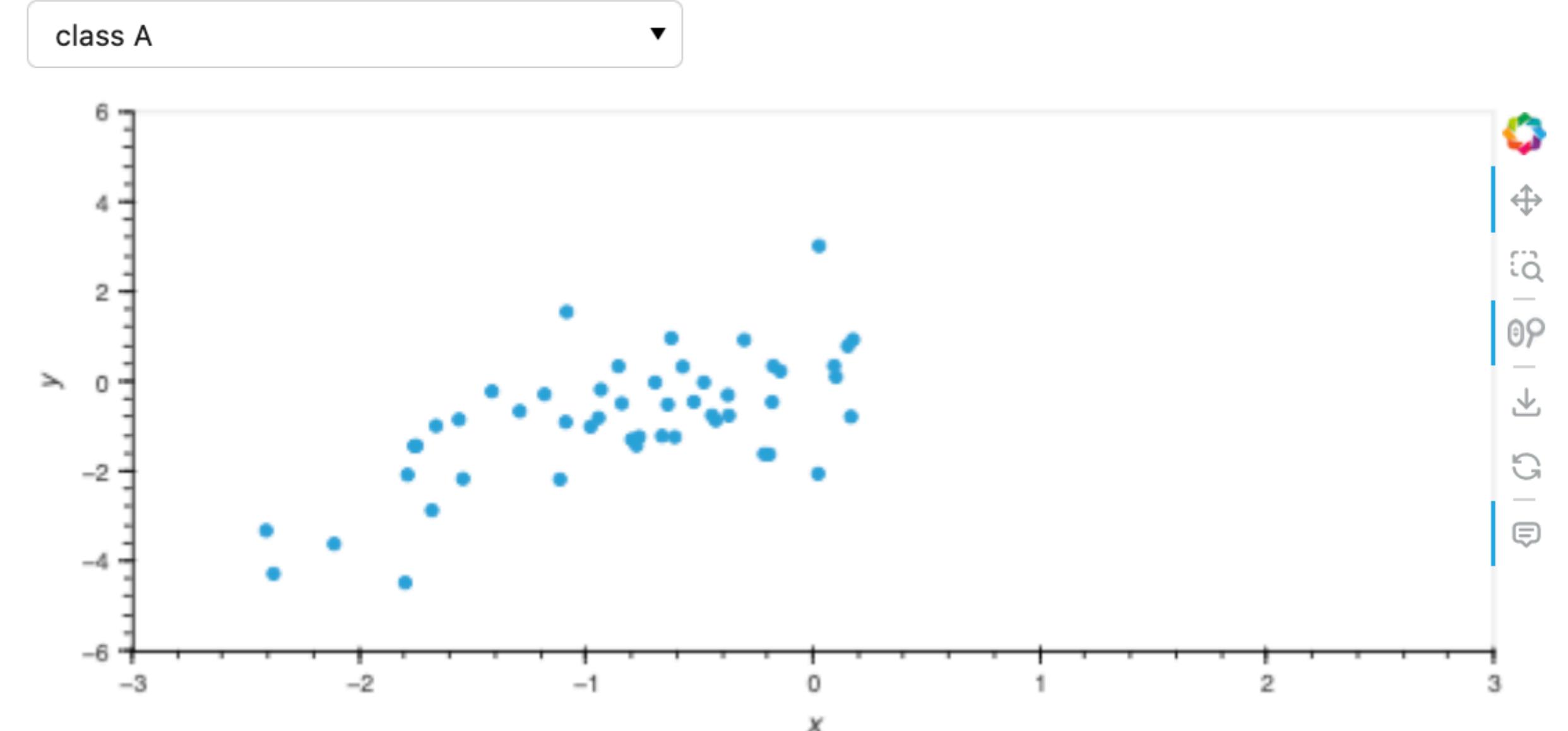
3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 1: INTERACTIVE DATA

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

# Interactive data
dfi = df.interactive()

(dfi
    .loc[dfi["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 1: INTERACTIVE DATA

Create widget

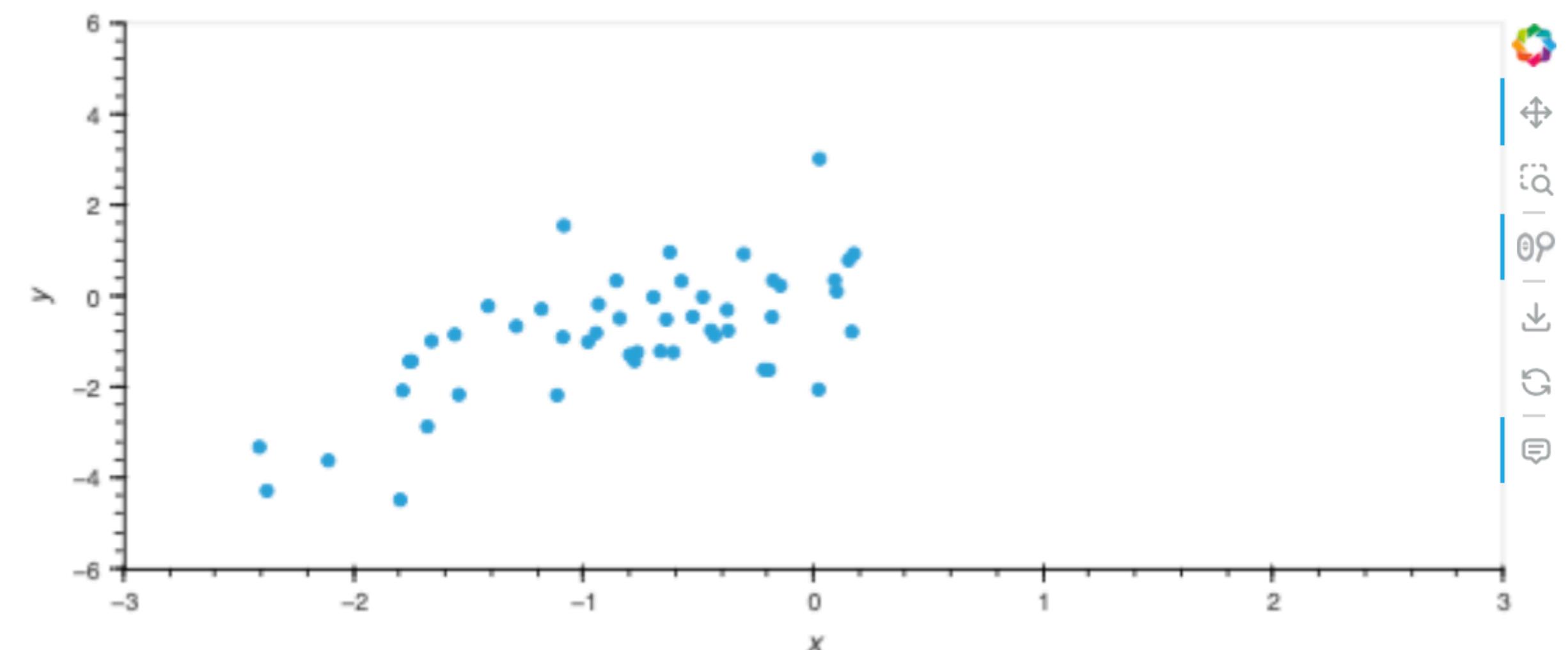


```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

# Interactive data
dfi = df.interactive()

(dfi
    .loc[dfi["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```

class A ▾



3. INTERACTIVITY/REACTIVITY

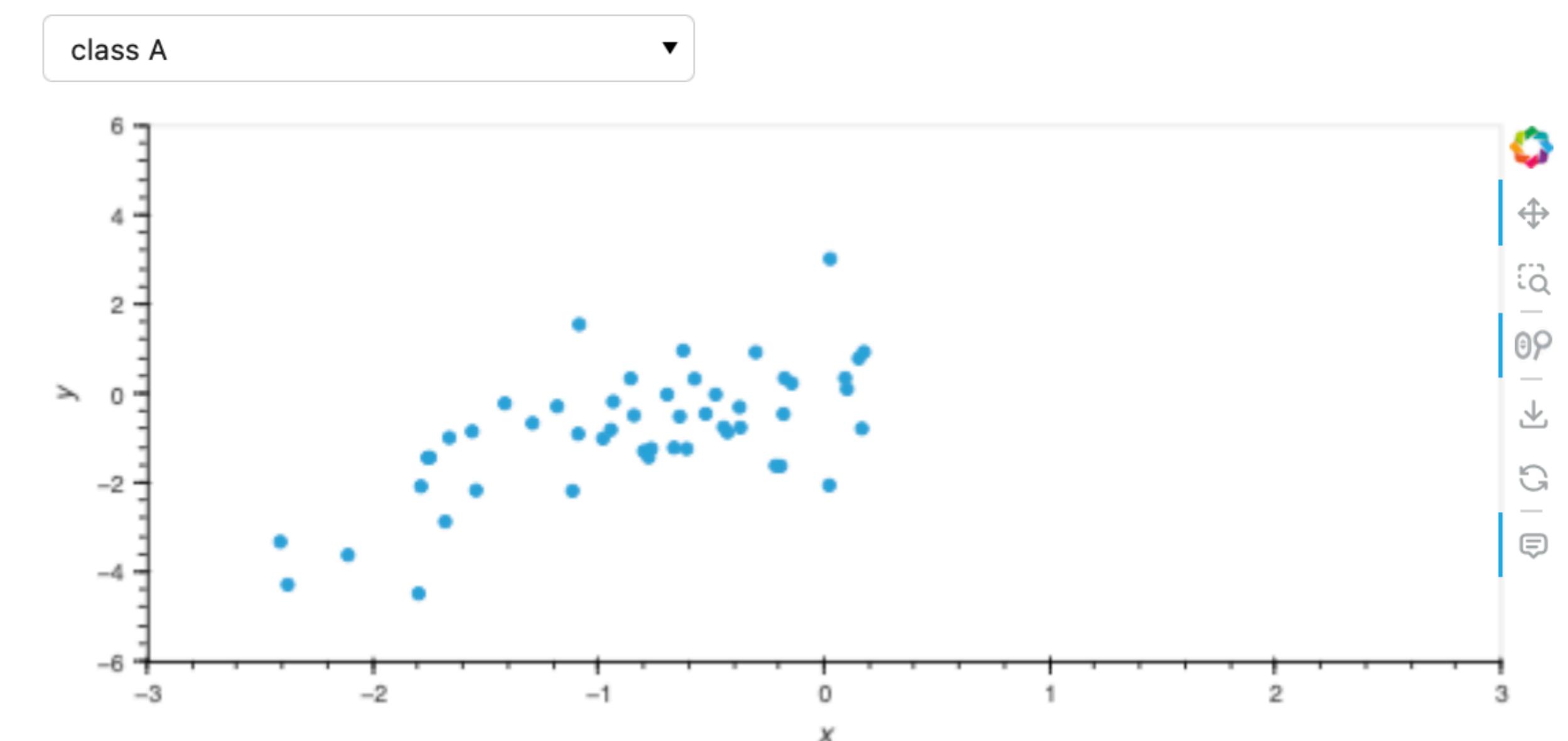
REACTIVITY, WAY 1: INTERACTIVE DATA

Create widget →
Make your data interactive →

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

# Interactive data
dfi = df.interactive()

(dfi
    .loc[dfi["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 1: INTERACTIVE DATA

Create widget →

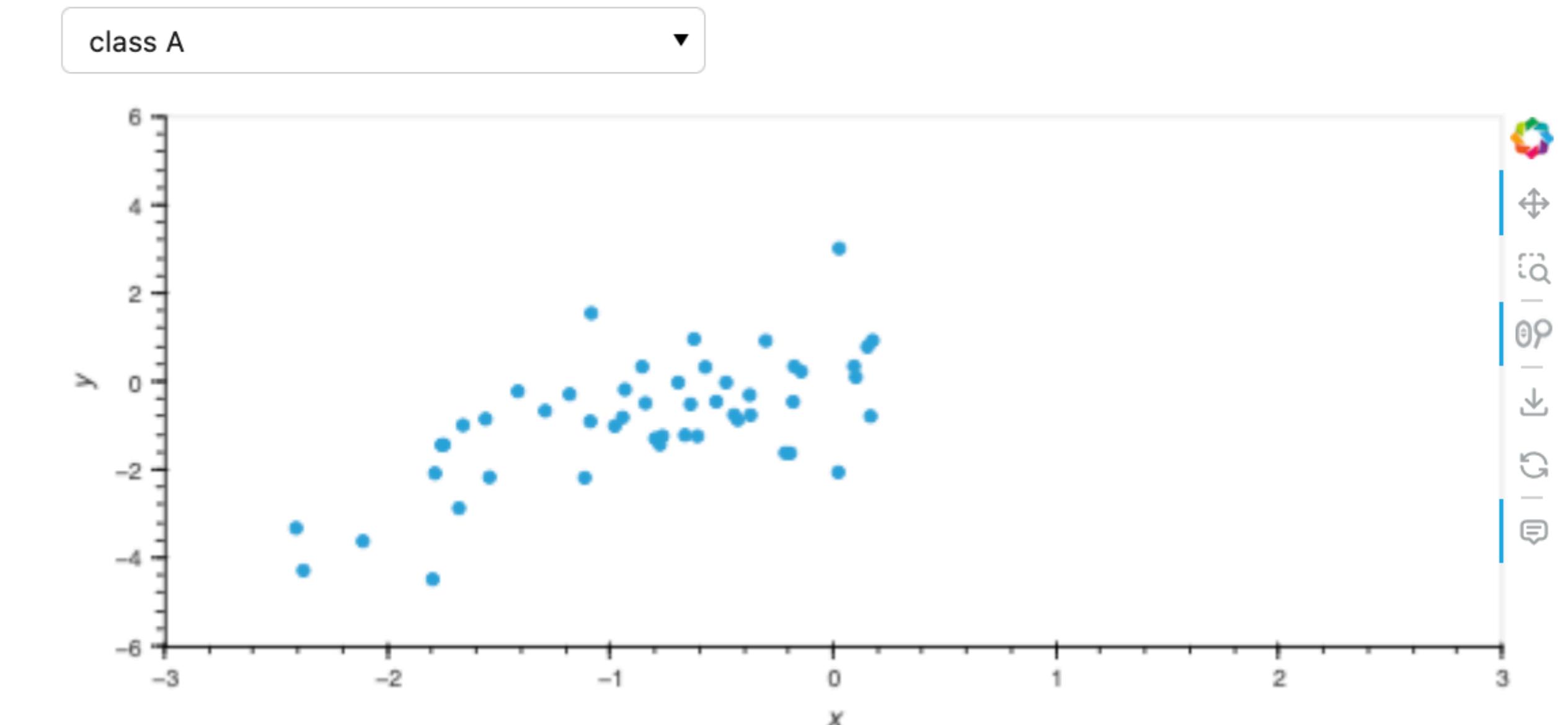
```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
```

Make your data interactive →

```
# Interactive data
dfi = df.interactive()
```

Filter data by widget
(combinations are possible) →

```
(dfi
    .loc[dfi["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 1: INTERACTIVE DATA

Create widget →

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
```

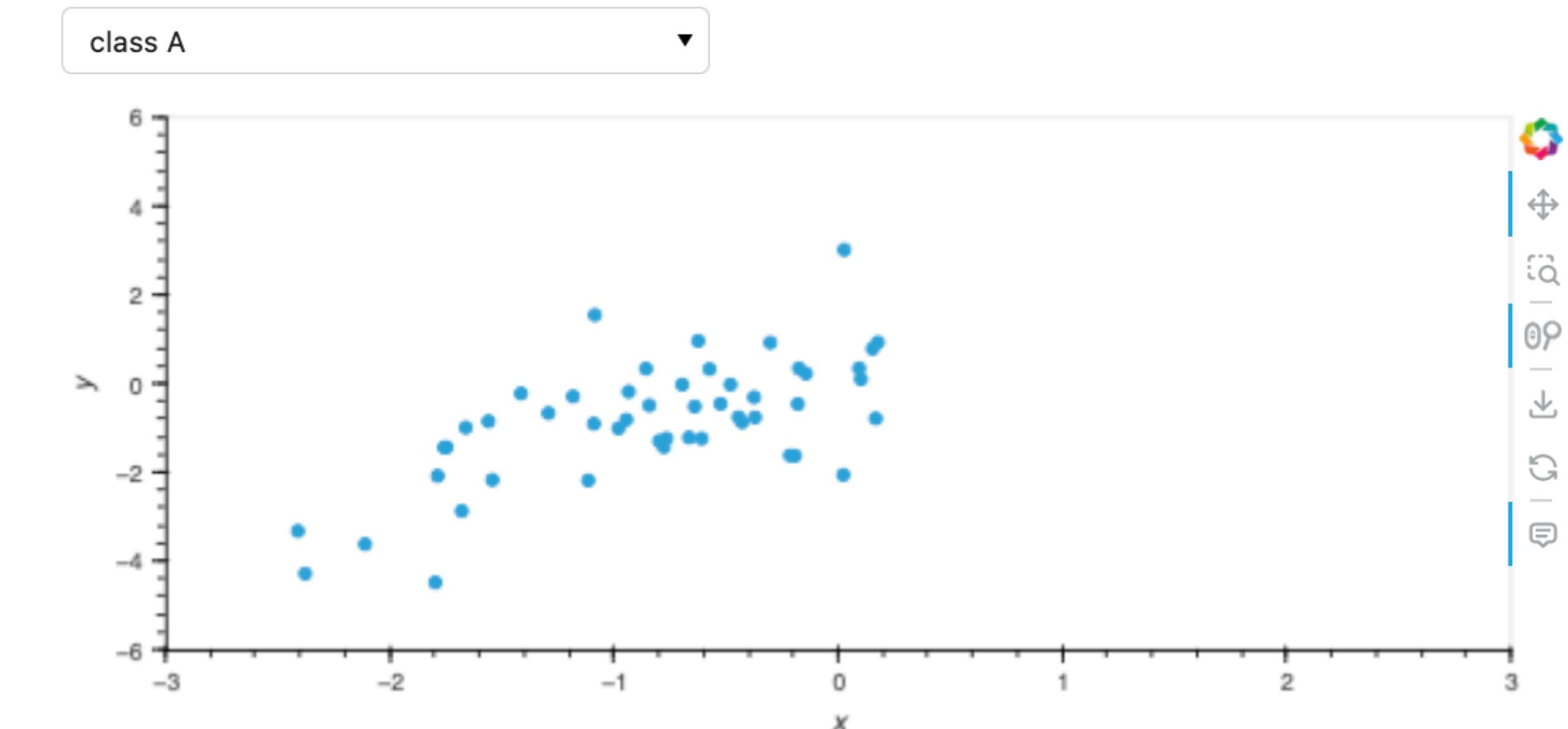
Make your data interactive →

```
# Interactive data
dfi = df.interactive()
```

Filter data by widget
(combinations are possible) →

```
(dfi
    .loc[dfi["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```

Plot or show data →



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 1: INTERACTIVE DATA

Create widget →

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
```

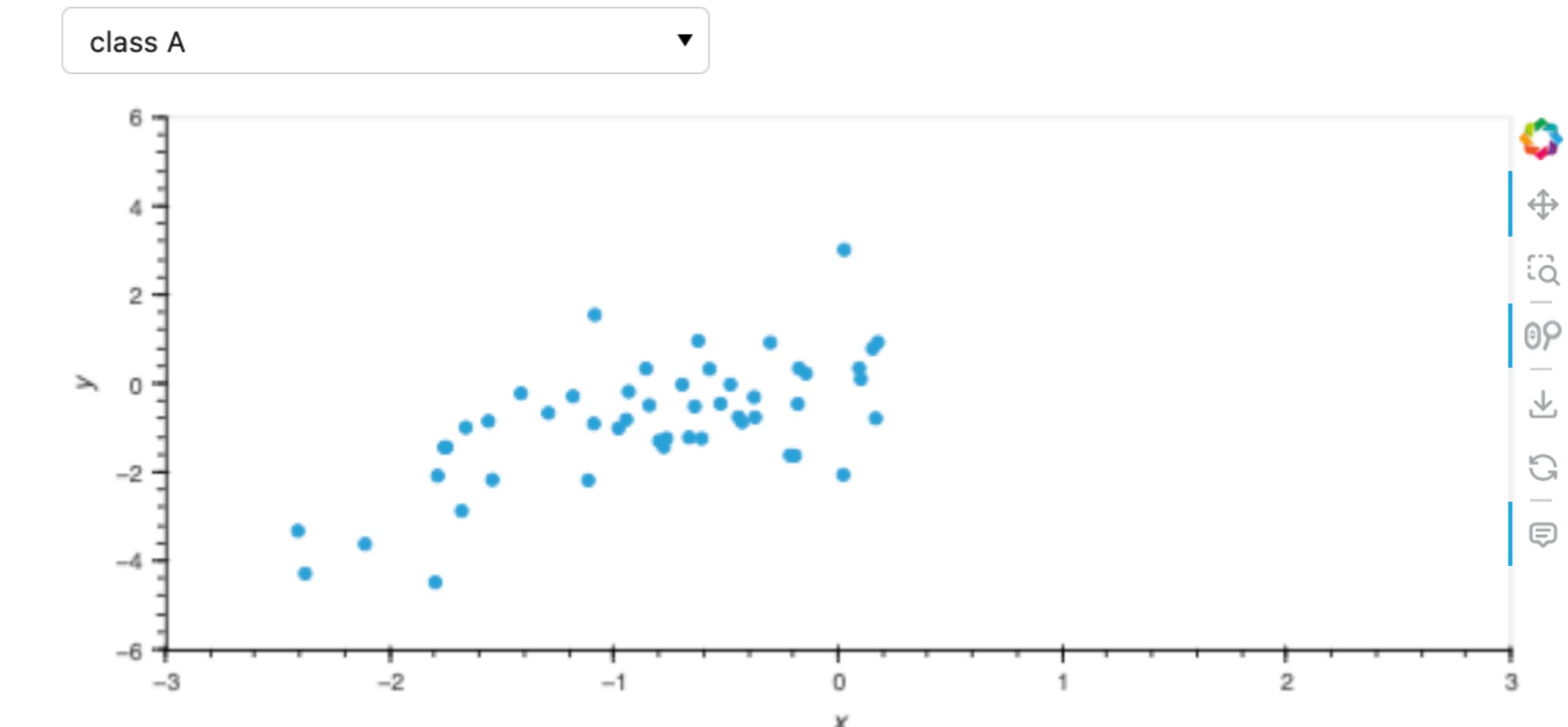
Make your data interactive →

```
# Interactive data
dfi = df.interactive()
```

Filter data by widget
(combinations are possible) →

```
(dfi
    .loc[dfi["class"] == w_select_class]
    .hvplot("x",
            "y",
            kind="scatter",
            xlim=(-3,3),
            ylim=(-6,6)
    )
)
```

Plot or show data →



Main disadvantage: introduced in 2020

3. INTERACTIVITY/REACTIVITY

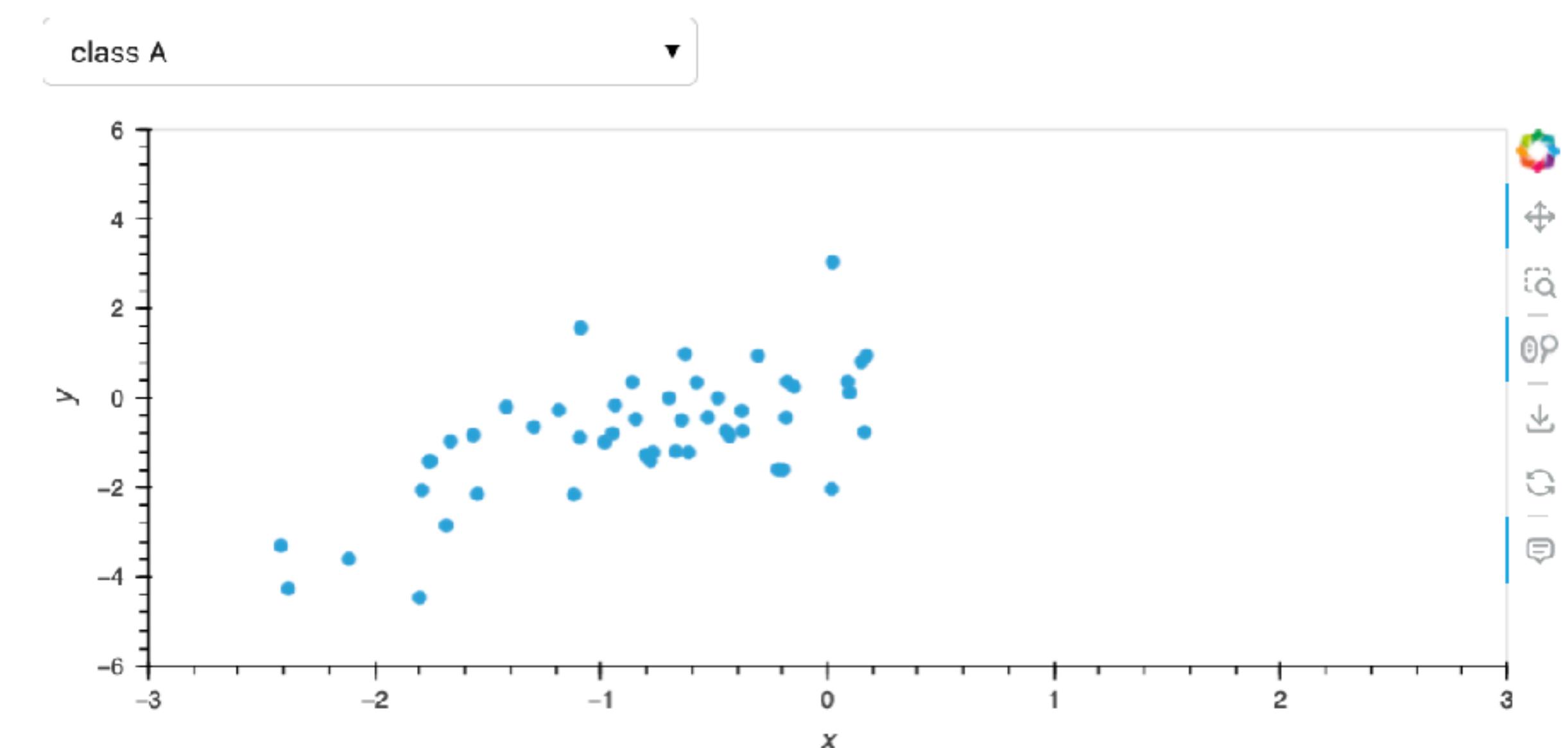
REACTIVITY, WAY 2: BINDING WIDGETS AND FUNCTIONS

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

def interactive_plot(class_subset):
    return (
        df
        .loc[df["class"] == class_subset]
        .hvplot("x",
                "y",
                kind="scatter",
                xlim=(-3,3),
                ylim=(-6,6)
        )
    )

interactive_plot2 = pn.bind(interactive_plot, class_subset=w_select_class)

pn.Column(w_select_class, interactive_plot2)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 2: BINDING WIDGETS AND FUNCTIONS

Create a function that affects an element of the layout, depending on the value of a widget.

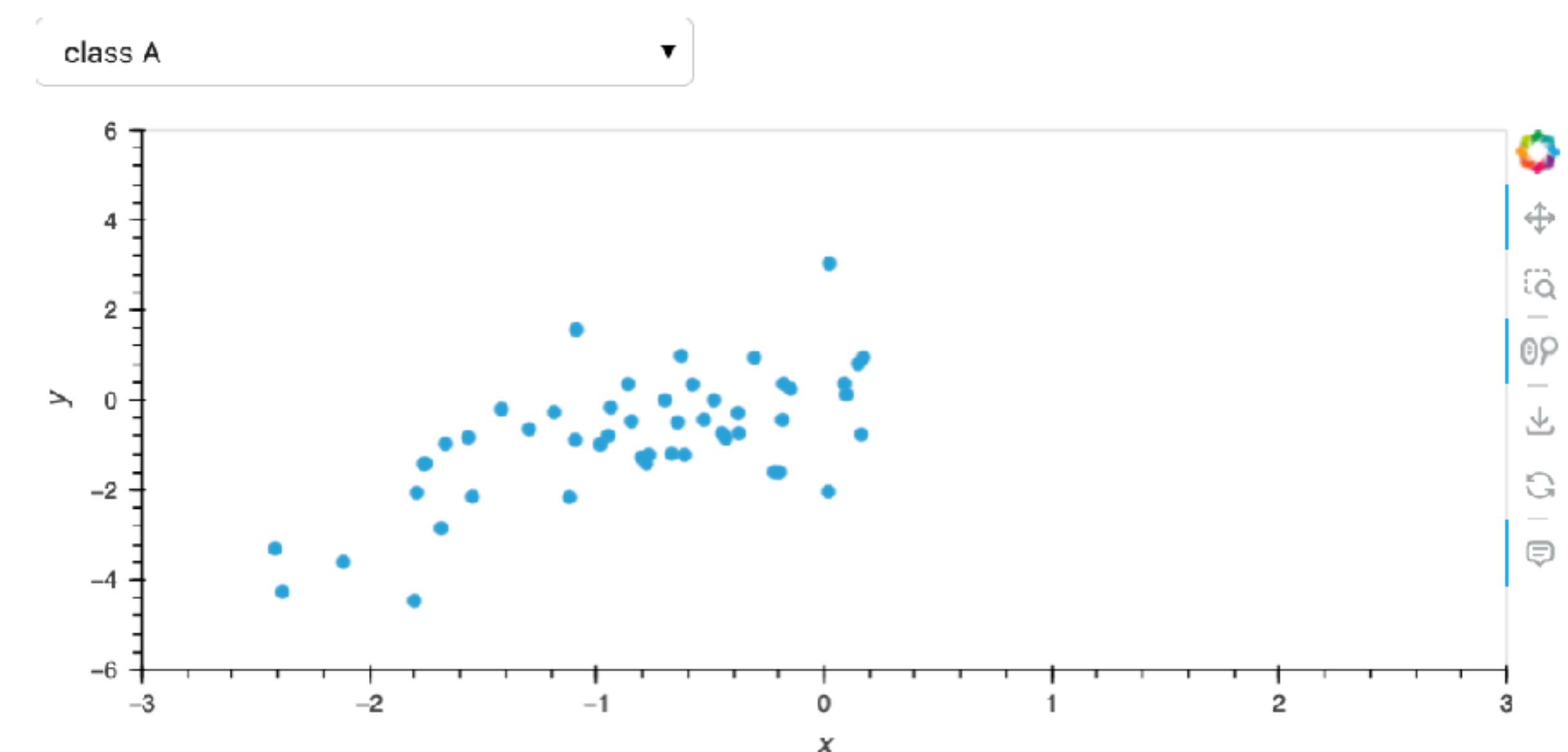


```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

def interactive_plot(class_subset):
    return (
        df
        .loc[df["class"] == class_subset]
        .hvplot("x",
                "y",
                kind="scatter",
                xlim=(-3,3),
                ylim=(-6,6)
        )
    )

interactive_plot2 = pn.bind(interactive_plot, class_subset=w_select_class)

pn.Column(w_select_class, interactive_plot2)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 2: BINDING WIDGETS AND FUNCTIONS

Create a function that affects an element of the layout, depending on the value of a widget.



```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

def interactive_plot(class_subset):
    return (
        df
        .loc[df["class"] == class_subset]
        .hvplot("x",
                "y",
                kind="scatter",
                xlim=(-3,3),
                ylim=(-6,6)
        )
    )

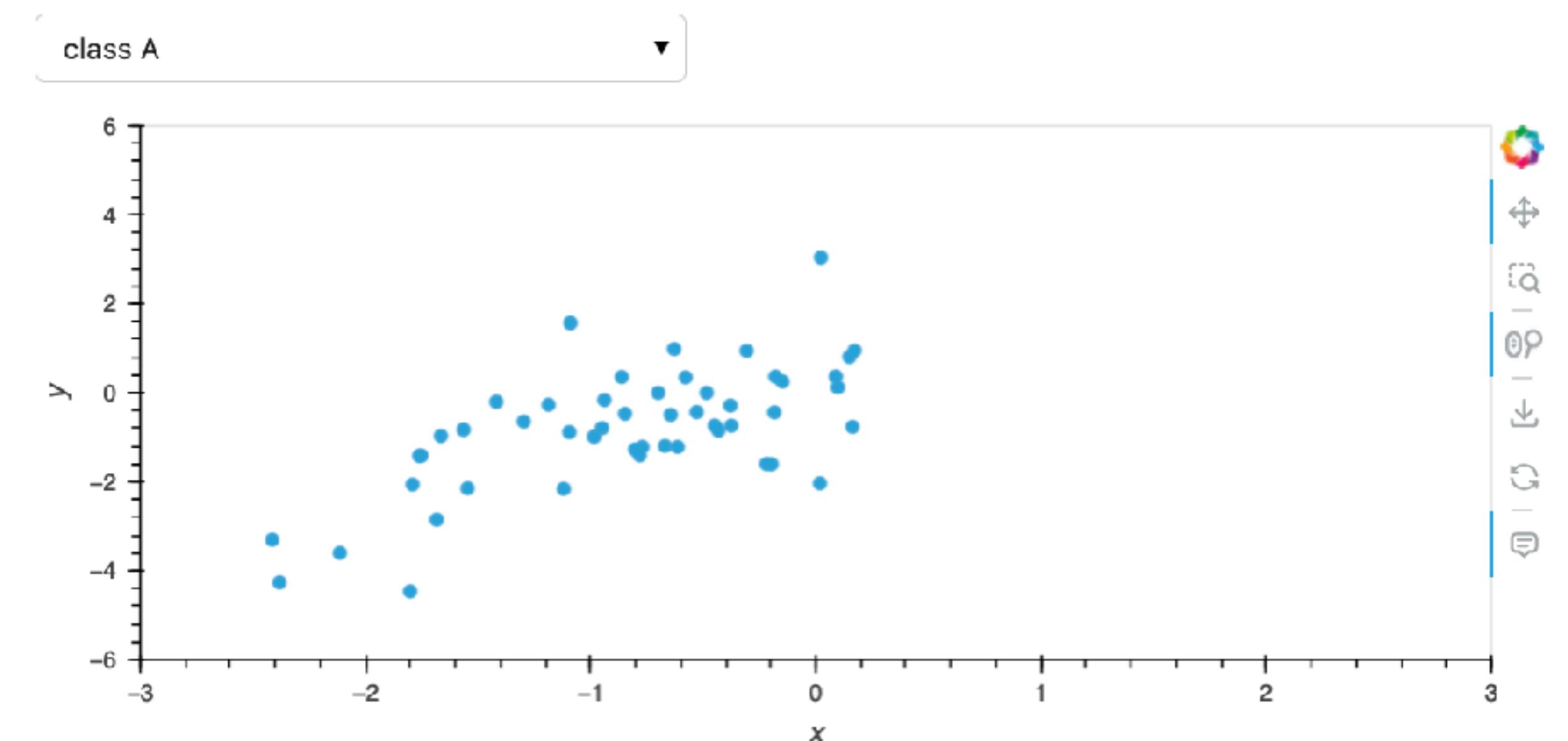
interactive_plot2 = pn.bind(interactive_plot, class_subset=w_select_class)

pn.Column(w_select_class, interactive_plot2)
```

Bind the function and the widget:



`pn.bind(function, arg_function=widget)`



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 2: BINDING WIDGETS AND FUNCTIONS

Create a function that affects an element of the layout, depending on the value of a widget.

Bind the function and the widget:

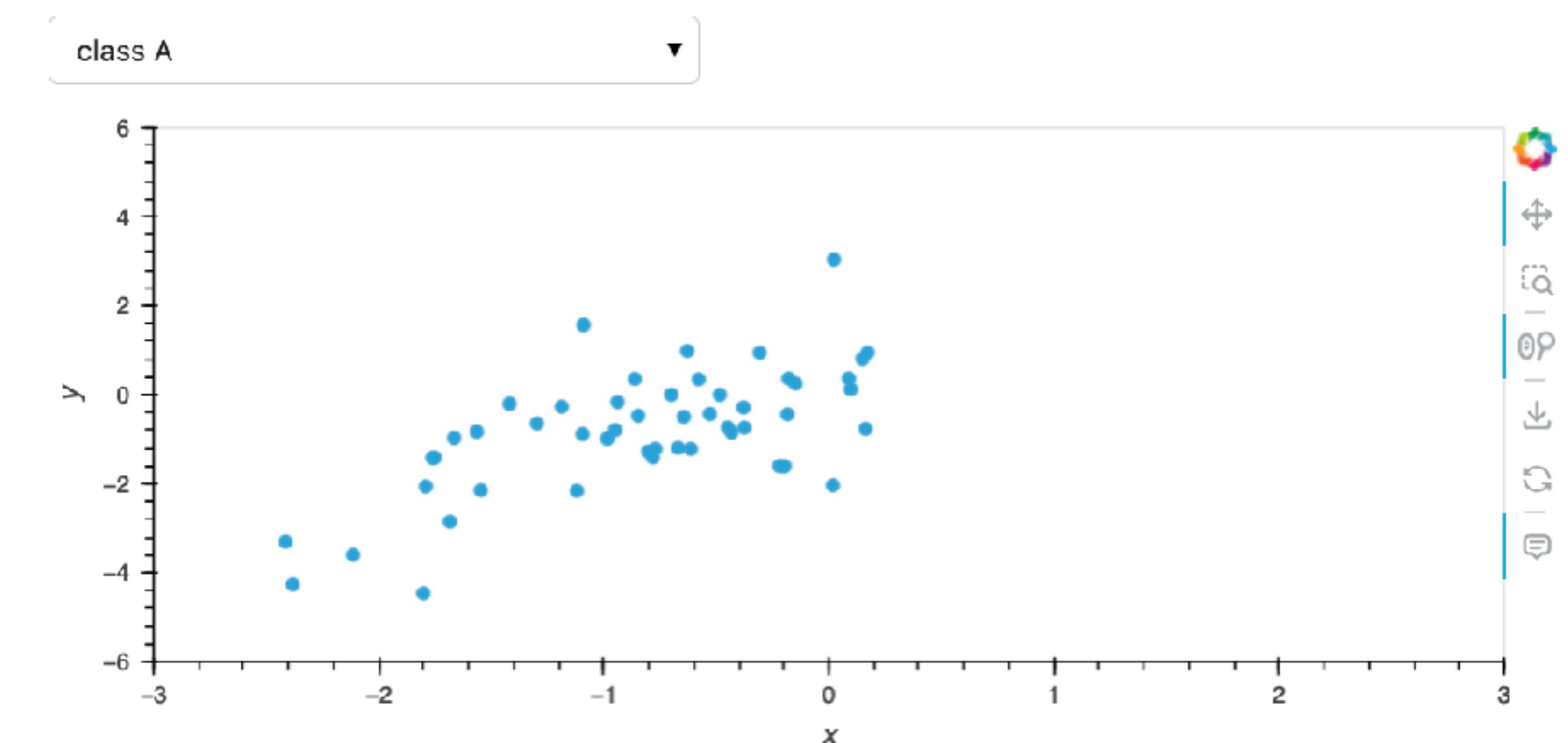
`pn.bind(function, arg_function=widget)`

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

def interactive_plot(class_subset):
    return (
        df
        .loc[df["class"] == class_subset]
        .hvplot("x",
                "y",
                kind="scatter",
                xlim=(-3,3),
                ylim=(-6,6)
        )
    )

interactive_plot2 = pn.bind(interactive_plot, class_subset=w_select_class)

pn.Column(w_select_class, interactive_plot2)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 2: BINDING WIDGETS AND FUNCTIONS

Create a function that affects an element of the layout, depending on the value of a widget.

Bind the function and the widget:

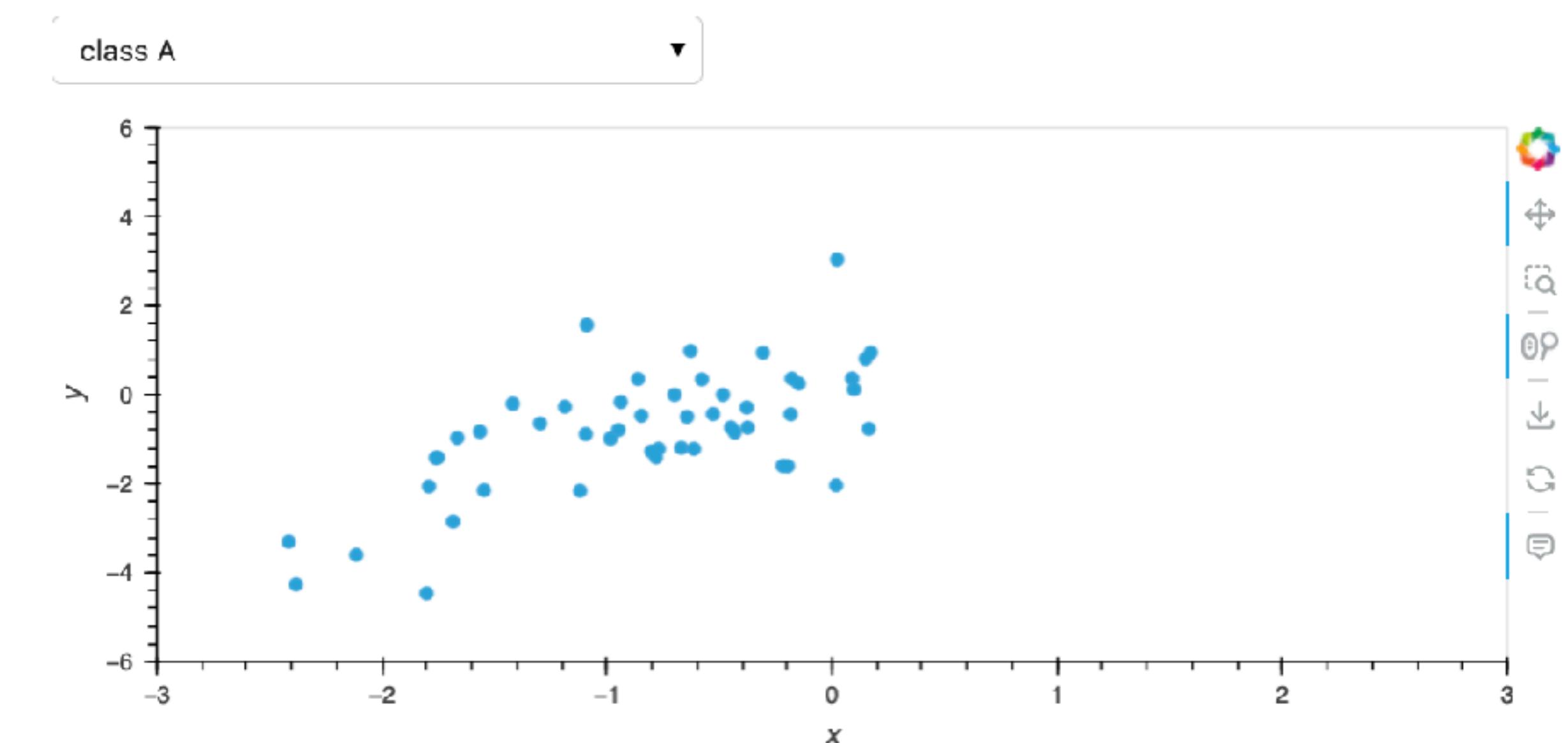
`pn.bind(function, arg_function=widget)`

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

def interactive_plot(class_subset):
    return (
        df
        .loc[df["class"] == class_subset]
        .hvplot("x",
                "y",
                kind="scatter",
                xlim=(-3,3),
                ylim=(-6,6)
        )
    )

interactive_plot2 = pn.bind(interactive_plot, class_subset=w_select_class)

pn.Column(w_select_class, interactive_plot2)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 2: BINDING WIDGETS AND FUNCTIONS

Create a function that affects an element of the layout,
depending on the value of a widget.

Bind the function and the widget:

`pn.bind(function, arg_function=widget)`

Advantage: More flexible/powerful

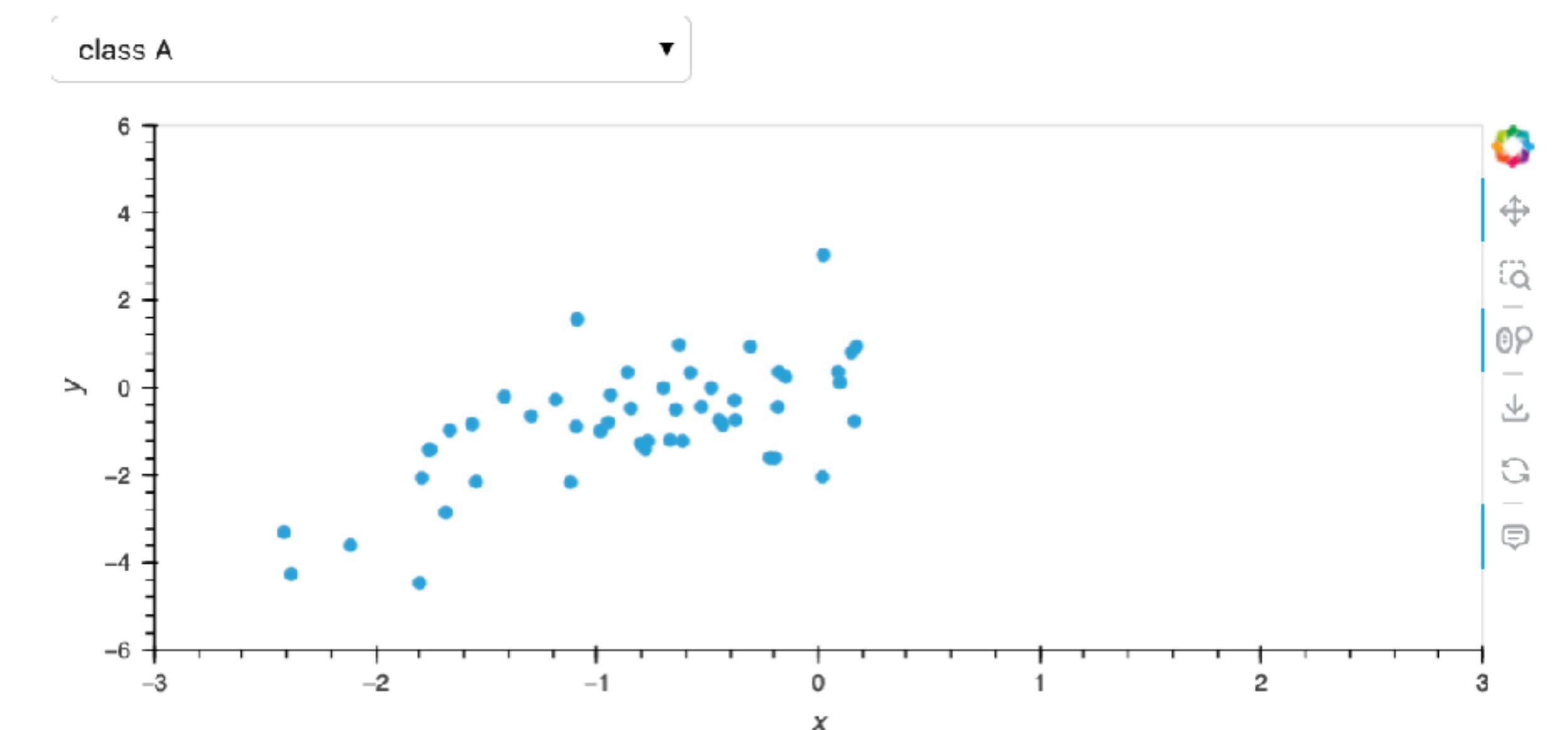
Not restricted to data operations!

```
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

def interactive_plot(class_subset):
    return (
        df
        .loc[df["class"] == class_subset]
        .hvplot("x",
                "y",
                kind="scatter",
                xlim=(-3,3),
                ylim=(-6,6)
        )
    )

interactive_plot2 = pn.bind(interactive_plot, class_subset=w_select_class)

pn.Column(w_select_class, interactive_plot2)
```



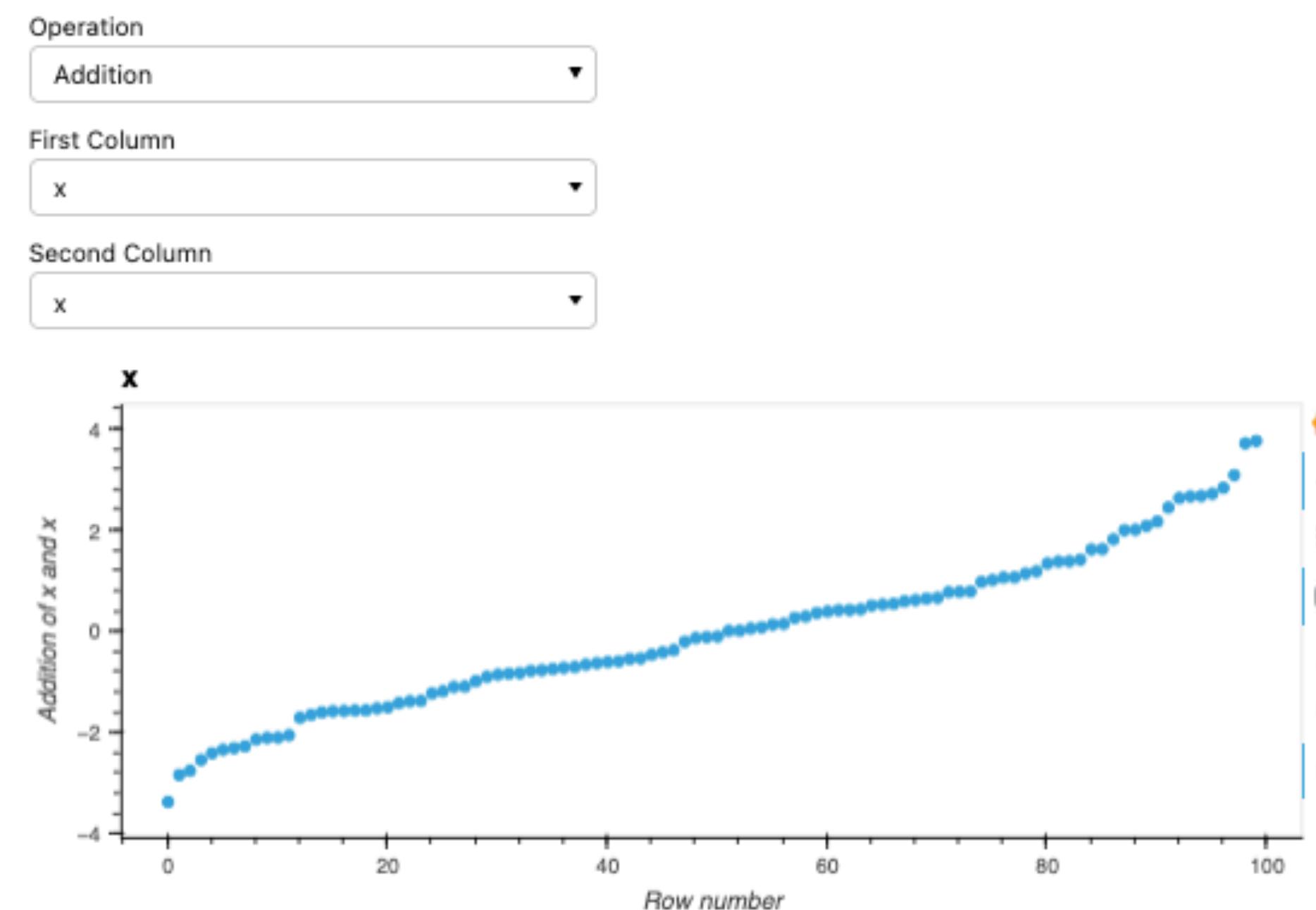
3. INTERACTIVITY/REACTIVITY

```
# Widgets
operation_selector = pn.widgets.Select(name="Operation", options=['Addition', 'Subtraction', 'Multiplication', 'Division'])
column_selector1 = pn.widgets.Select(name="First Column", options=["x", "y"])
column_selector2 = pn.widgets.Select(name="Second Column", options=["x", "y"])

def plot_data(operation, col1, col2):
    if operation == 'Addition':
        result = df[col1] + df[col2]
    elif operation == 'Subtraction':
        result = df[col1] - df[col2]
    elif operation == 'Multiplication':
        result = df[col1] * df[col2]
    elif operation == 'Division':
        result = data[col1] / data[col2]
    return result.hvplot(kind="scatter", xlabel="Row number", ylabel=f"{operation} of {col1} and {col2}")

# Bind the function to widget values
interactive_plot = pn.bind(plot_data, operation=operation_selector, col1=column_selector1, col2=column_selector2)

# Display
dashboard = pn.Column(operation_selector, column_selector1, column_selector2, interactive_plot)
dashboard.servable()
```



3. INTERACTIVITY/REACTIVITY

Function creates some non-trivial data transformation



```
# Widgets
operation_selector = pn.widgets.Select(name="Operation", options=['Addition', 'Subtraction', 'Multiplication', 'Division'])
column_selector1 = pn.widgets.Select(name="First Column", options=["x", "y"])
column_selector2 = pn.widgets.Select(name="Second Column", options=["x", "y"])

def plot_data(operation, col1, col2):
    if operation == 'Addition':
        result = df[col1] + df[col2]
    elif operation == 'Subtraction':
        result = df[col1] - df[col2]
    elif operation == 'Multiplication':
        result = df[col1] * df[col2]
    elif operation == 'Division':
        result = df[col1] / df[col2]
    return result.hvplot(kind="scatter", xlabel="Row number", ylabel=f"{operation} of {col1} and {col2}")

# Bind the function to widget values
interactive_plot = pn.bind(plot_data, operation=operation_selector, col1=column_selector1, col2=column_selector2)

# Display
dashboard = pn.Column(operation_selector, column_selector1, column_selector2, interactive_plot)
dashboard.servable()
```

Bind the function and several widgets:

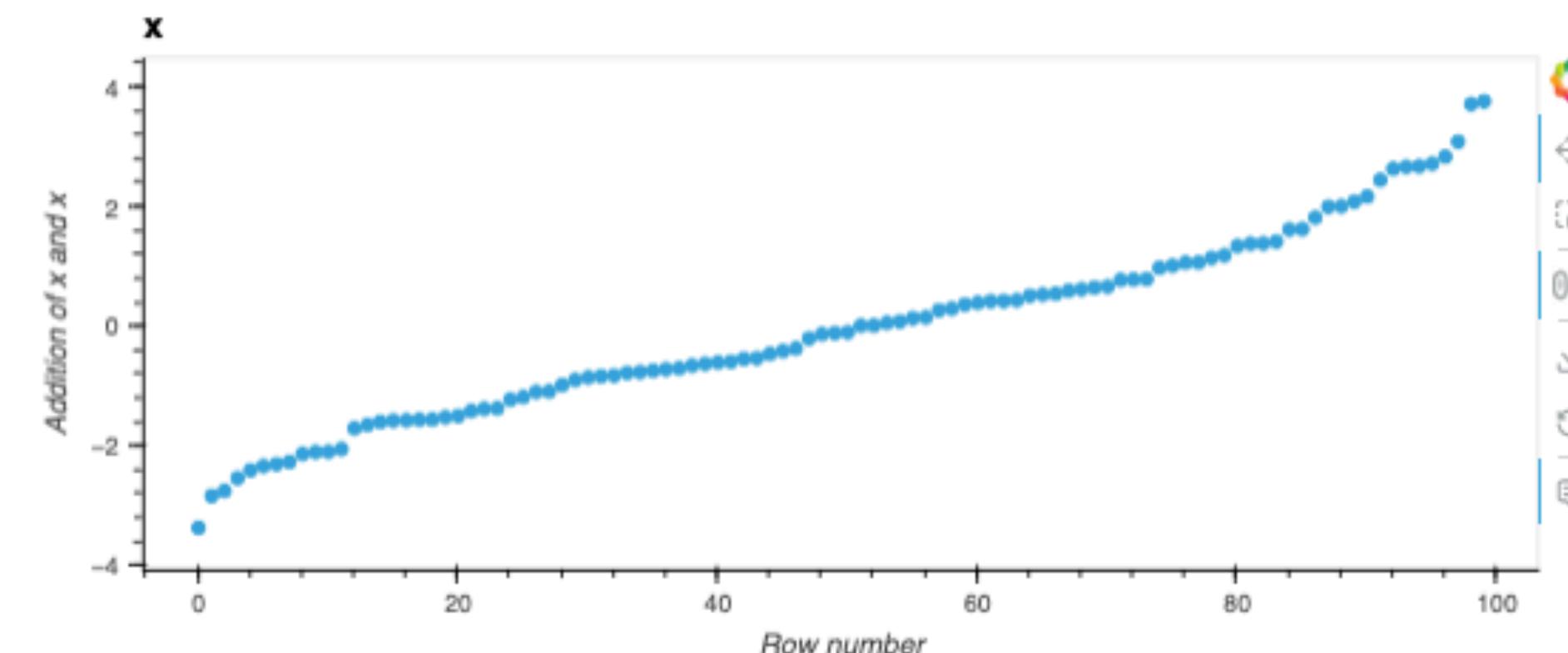


`pn.bind(function, arg_function=widget)`

Operation
Addition ▾

First Column
x ▾

Second Column
x ▾



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

```
w_button = pn.widgets.Button(name='Click me', button_type='primary')
selections = pn.pane.Markdown(object='You have not clicked the button')

def print_selected(event):
    selections.object = f"You have clicked the button {event.new} times"

w_button.on_click(print_selected)

pn.Column(w_button, selections)
```



You have clicked the button 5 times

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

```
w_button = pn.widgets.Button(name='Click me', button_type='primary')
selections = pn.pane.Markdown(object='You have not clicked the button')

def print_selected(event):
    selections.object = f"You have clicked the button {event.new} times"

w_button.on_click(print_selected)
pn.Column(w_button, selections)
```

Buttons have an `.on_click()` function



You have clicked the button 5 times

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

All Panel objects store their main component on their *object* parameter.

Buttons have an *.on_click()* function

```
w_button = pn.widgets.Button(name='Click me', button_type='primary')
selections = pn.pane.Markdown(object='You have not clicked the button')

def print_selected(event):
    selections.object = f"You have clicked the button {event.new} times"

w_button.on_click(print_selected)

pn.Column(w_button, selections)
```



You have clicked the button 5 times

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.link() links a widget to a pane/widget, making them have exactly the same value

```
# Linking two widgets

toggle = pn.widgets.Switch(name='Show Textbox', value=True)
textbox = pn.widgets.TextInput(name='My Textbox', value='Hello, Panel!')

toggle.link(textbox, value='visible')

pn.Column(toggle, textbox)
```



```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')
```

```
w_select_class.link(selections, value="object")

pn.Column(w_select_class, selections)
```

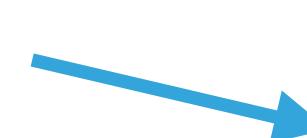


3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.link() links a widget to a pane/widget, making them have exactly the same value

Create widgets



```
# Linking two widgets
toggle = pn.widgets.Switch(name='Show Textbox', value=True)
textbox = pn.widgets.TextInput(name='My Textbox', value='Hello, Panel!')

toggle.link(textbox, value='visible')

pn.Column(toggle, textbox)
```



My Textbox

Hello, Panel!

```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')
```

```
w_select_class.link(selections, value="object")
```

```
pn.Column(w_select_class, selections)
```

class B ▾

class B

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.link() links a widget to a pane/widget, making them have exactly the same value

Create widgets

Link the value of the toggle to the attribute "visible" of the textbook

```
# Linking two widgets
toggle = pn.widgets.Switch(name='Show Textbox', value=True)
textbox = pn.widgets.TextInput(name='My Textbox', value='Hello, Panel!')

toggle.link(textbox, value='visible')

pn.Column(toggle, textbox)
```



```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')
```

```
w_select_class.link(selections, value="object")
```

```
pn.Column(w_select_class, selections)
```



3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.link() links a widget to a pane/widget, making them have exactly the same value

Create widgets

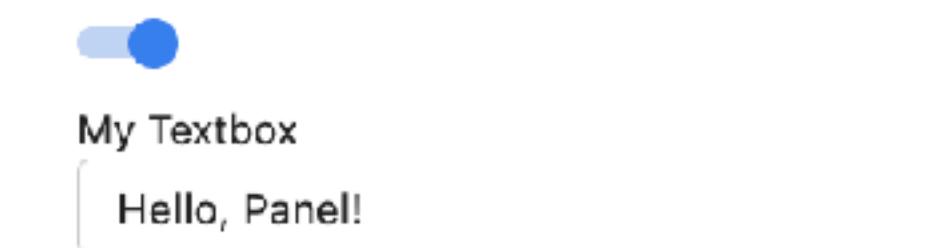
Link the value of the toggle to the attribute "visible" of the textbook

If toggle becomes "False" (i.e. turned off) → the argument visible of the textbox is set to False

```
# Linking two widgets
toggle = pn.widgets.Switch(name='Show Textbox', value=True)
textbox = pn.widgets.TextInput(name='My Textbox', value='Hello, Panel!')

toggle.link(textbox, value='visible')

pn.Column(toggle, textbox)
```



```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')
```

```
w_select_class.link(selections, value="object")

pn.Column(w_select_class, selections)
```



class B

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.link() links a widget to a pane/widget, making them have exactly the same value

Create widgets

Link the value of the toggle to the attribute "visible" of the textbook

If toggle becomes "False" (i.e. turned off) → the argument visible of the textbox is set to False

```
# Linking two widgets
toggle = pn.widgets.Switch(name='Show Textbox', value=True)
textbox = pn.widgets.TextInput(name='My Textbox', value='Hello, Panel!')
toggle.link(textbox, value='visible')
pn.Column(toggle, textbox)
```



Link the value of w_select_class to the attribute "object" of the markdown

```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')
```

```
w_select_class.link(selections, value="object")
pn.Column(w_select_class, selections)
```



class B

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.watch() watches for changes in parameter and runs a function

works through events (event.new, event.old, event.name)

```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')

def selected_class(event):
    selections.object = f"The selected class is {event.new}"

w_select_class.param.watch(selected_class, "value")

pn.Column(w_select_class, selections)
```

class B ▾

The selected class is class B

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.watch() watches for changes in parameter and runs a function

works through events (event.new, event.old, event.name)

Watch the Select widget, and on change of its value run the function



```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')

def selected_class(event):
    selections.object = f"The selected class is {event.new}"

w_select_class.param.watch(selected_class, "value")

pn.Column(w_select_class, selections)
```

class B ▾

The selected class is class B

3. INTERACTIVITY/REACTIVITY

REACTIVITY, WAY 3: LINKING WIDGETS

.watch() watches for changes in parameter and runs a function

The same code using pn.bind

```
# Linking a pane to a widget
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())
selections = pn.pane.Markdown(object=f'Select a class')

def selected_class(event):
    selections.object = f"The selected class is {event.new}"

w_select_class.param.watch(selected_class, "value")

pn.Column(w_select_class, selections)
```

The selected class is class B

```
# You can also do it with pn.bind
w_select_class = pn.widgets.Select(options=df["class"].unique().tolist())

# Define a function that returns the Markdown text based on the selected class
def update_selection(selected):
    return f"The selected class is {selected}"

# Bind the function to the widget value
selections = pn.pane.Markdown(object=fn.bind(update_selection, w_select_class))

pn.Column(w_select_class, selections)
```

The selected class is class B

3. INTERACTIVITY/REACTIVITY

REACTIVITY, SUMMARY

df.interactive()

Directly links DataFrames to widgets

Example: Filtering data based on widgets

pn.bind(function, widget)

Binding functions to widgets

Example: Doing more complex operations depending on the value of the widget

widget.link(pane, widget_attribute="pane_attribute")

Create a direct link between widget properties

Example: Area becomes visible is toggle is switched on

widget.param.watch(function, "widget_attribute")

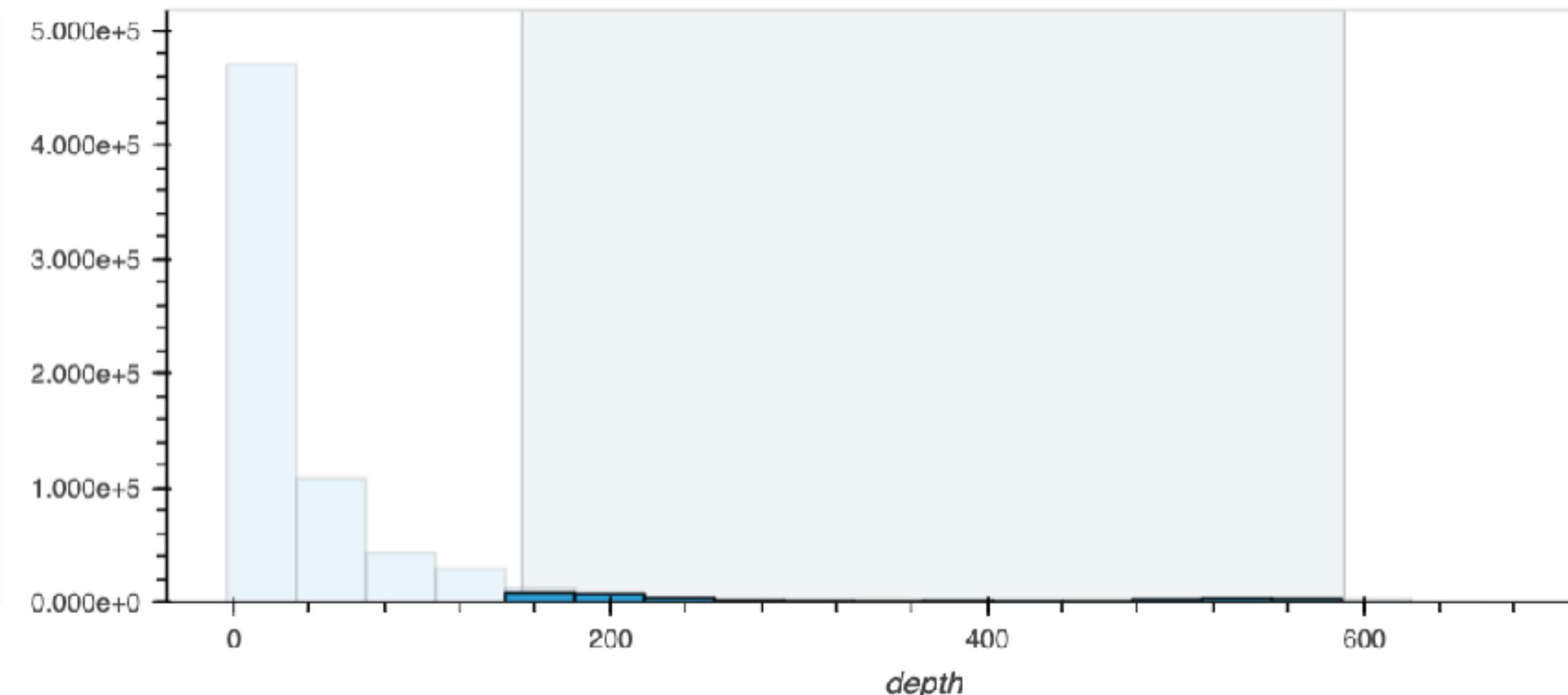
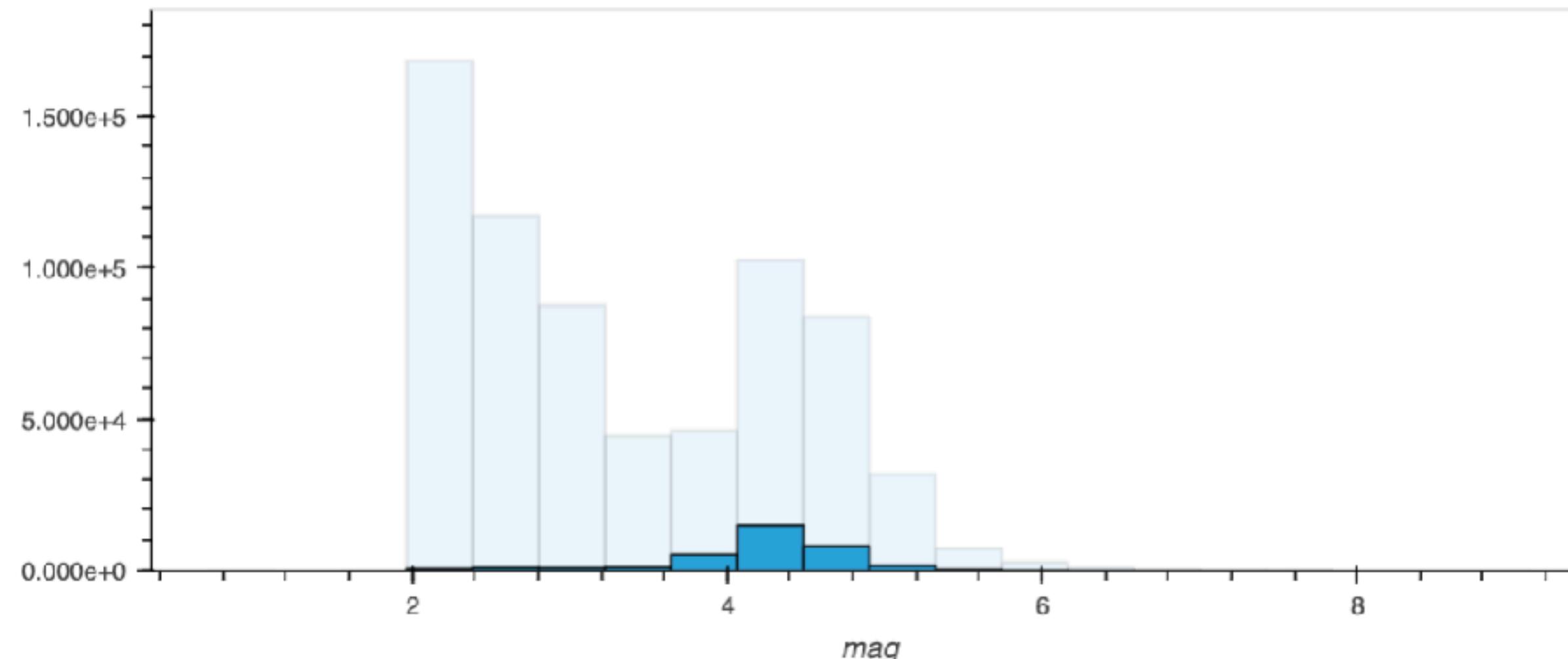
Setting up callbacks (functions) for custom logic

Example: Running a series of custom operations when a value changes

3. INTERACTIVITY/REACTIVITY

LINKING PLOTS (BRUSHING)

```
mag_hist = df.hvplot(  
    y='mag', kind='hist', min_height=250)  
  
depth_hist = df.hvplot(  
    y='depth', kind='hist', min_height=250)  
  
# Linked brushing  
import holoviews as hv  
hv.link_selections(mag_hist + depth_hist)
```

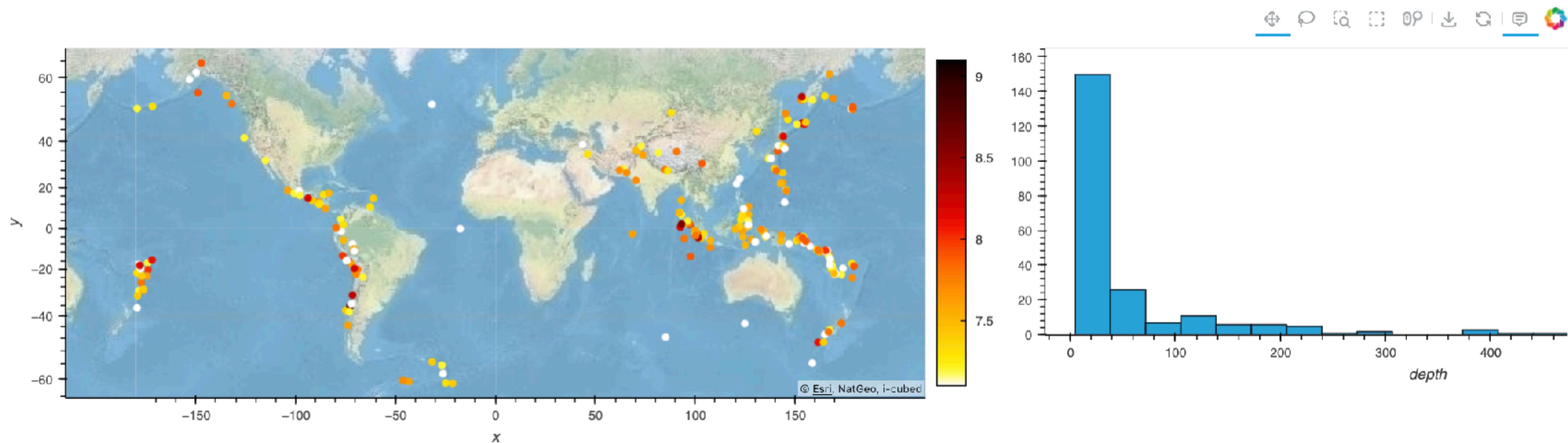


3. INTERACTIVITY/REACTIVITY

LINKING PLOTS (BRUSHING)

```
df_sev = df.loc[df["mag"]>7]
geo = df_sev.hvplot.points('longitude', 'latitude', geo=True, color='mag', cmap="fire_r",
                           tiles='EsriUSATopo', min_height=250)
hist = df_sev.hvplot.hist('depth', min_height=250)

hv.link_selections(geo + hist)
```



3. INTERACTIVITY/REACTIVITY

LINKING PLOTS (INTERACTIVE DATA)

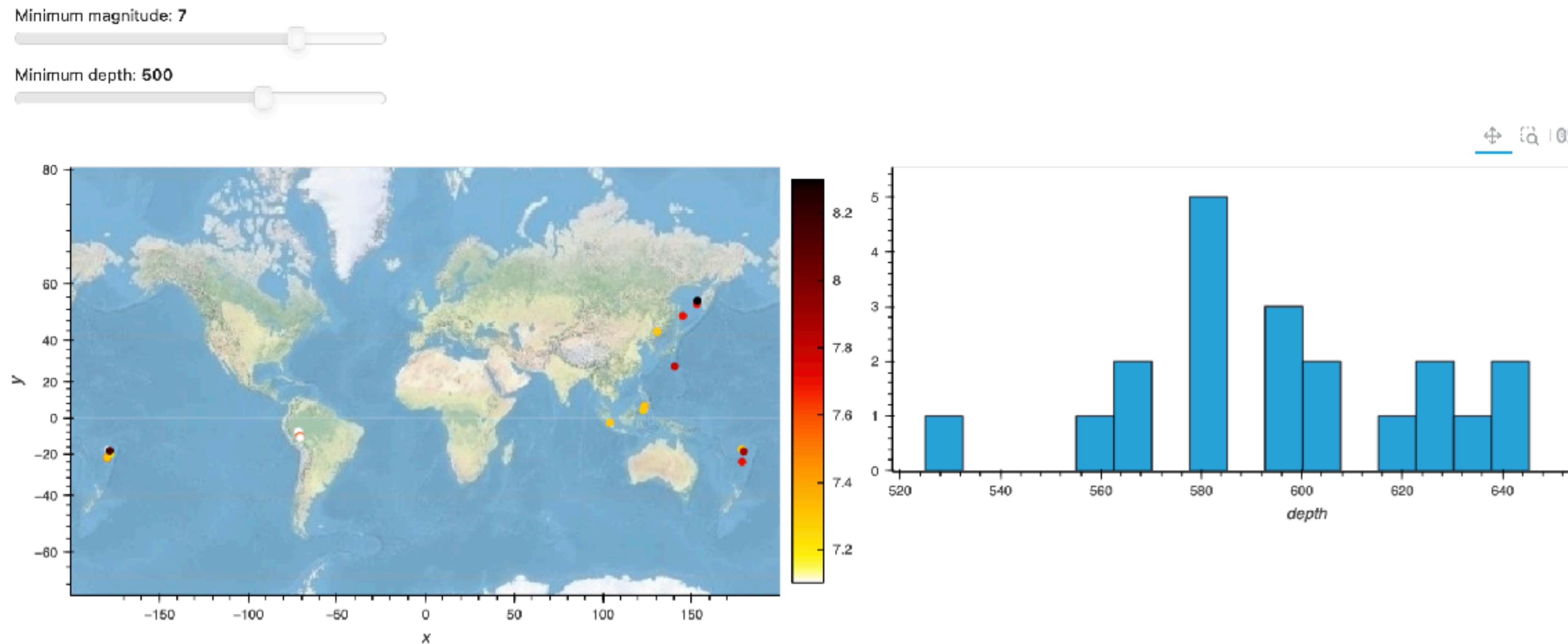
```
# Earthquake data
df = pd.read_parquet('data/earthquakes.parq', 'fastparquet').interactive()

# Two widgets
mag_select = pn.widgets.FloatSlider(name = "Minimum magnitude", end=9.2, value=7, step=0.1)
depth_select = pn.widgets.FloatSlider(name = "Minimum depth", start=-10, end=750, value=500, step=10)

df_subset = df.loc[(df["mag"] > mag_select) &
                   (df["depth"] > depth_select)]

geo = df_subset.hvplot.points('longitude', 'latitude', geo=True, color='mag', cmap="fire_r",
                               tiles='EsriUSATopo', responsive=True, height=400, width=700).opts(toolbar="below")
hist = df_subset.hvplot.hist(y='depth', min_height=250)

pn.panel(geo + hist)
```



EXERCISE

Play with interactivity

→ Check section 3: B_z_lecture_examples.ipynb

→ Work on: B_ex2_reactivity

4 MAIN ADVANTAGES OF PYTHON/PANEL/HVPLOT/BOKEH

1. Easy apps/viz do not need :
 - * a complicated setup
 - * a server
2. Iterative app building, testing each part independently
3. Informative error messages (bokeh <3)
4. Apps can run on the browser (privacy, easiness)

4. EXPORTING APPS/ SERVERS

4. EXPORTING APPS/SERVERS

Apps with widgets may require a server for interactivity

But if there are not many options you can embed all possible states of the visualization

```
#Export the app  
layout.save('apps/simple_app_template.html', embed=True)
```

4. EXPORTING APPS/SERVERS

Apps running python code require a server to run.

Options to run them:

- Locally (e.g. on jupyter notebook/lab)
- Locally (*panel serve your_app.py*)
- On a server (e.g. AWS)
- **Using the browser as a server**

##Convert to WASM app

panel convert bokeh_app.py --to pyodide-worker --out ./app --pwa

#Launch server to test locally (or upload to GitHub pages)

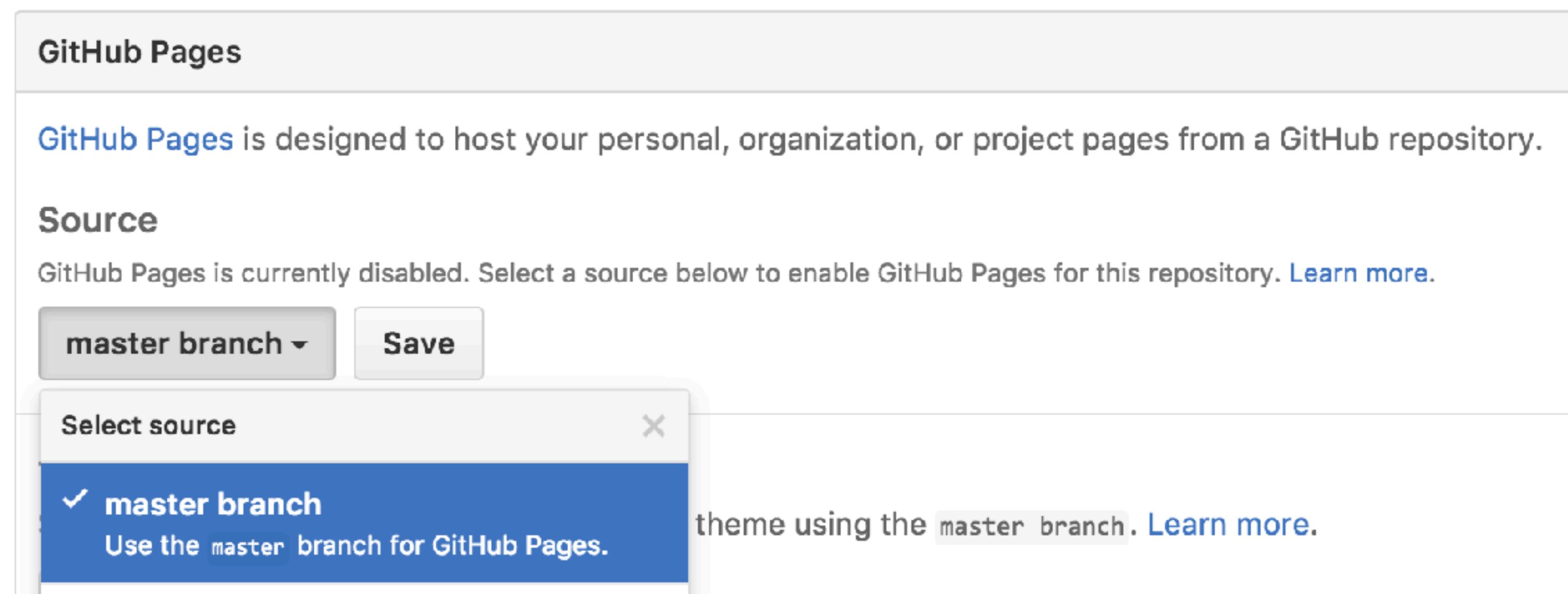
#python3 -m http.server

Upload your site to Github Pages (username.github.io/repository)

1. Create a personal GitHub page: <https://pages.github.com/>
2. Create a new repository
3. Upload the files created after `panel convert bokeh_app.py --to pyodide-worker --out ./app --pwa`
4. Do this step

Click on the Settings tab and scroll down to the GitHub Pages section.

Then select the **main branch** source and click on the **Save** button.



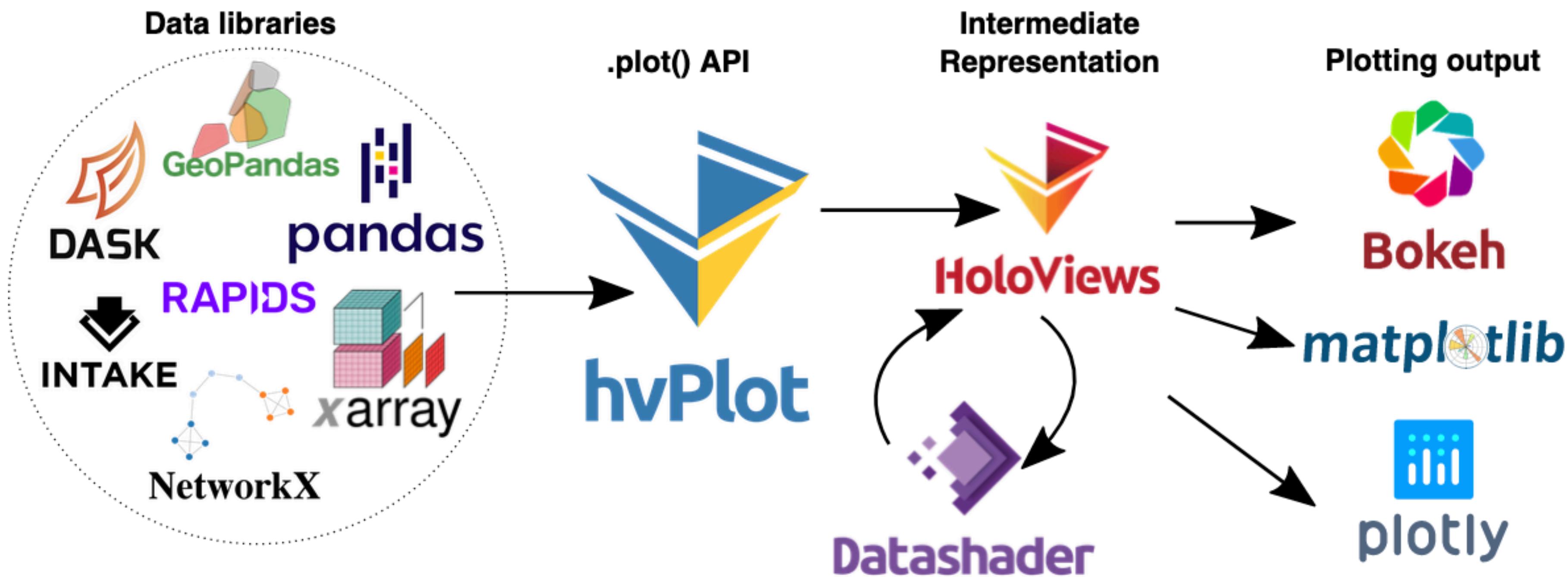
WHERE TO GO FROM HERE

Documentation of hvPlot

Custom visualizations? → holoViews

More on apps/dashboards? → panel

Use apps to label/collect data: https://holoviews.org/user_guide/Annotators.html



FINAL

EXERCISE

Work on your own app