Engineering at Alberta

Bokeh

Weichen Qiu

July 2020

Contents

Introduction

Bokeh is a powerful Python graphing library for interactive visualization and hosts a range of widgets.

Please note that Python is both **case** sensitive and **indent** sensitive.

The version of Bokeh used in this documentation is Bokeh 2.1.1.

Bokeh Documentation here.

This documentation will start with the basics of making a graph. It will first cover how to modify a figure, and the different types of plots available and how to use each one. Interactive widgets will then be introduced with some examples and a table of available widgets. Finally, how to deploy an app with widgets to Heroku's server will be covered in depth.

Installation

Bokeh Installation Documentation here.

Installs Required:

1. Python 3
    a. Bokeh requires **Python 3**, if not already installed, download here.
2. A code text editor (2 recommendations below)
    a. Sublime Text Editor 3
        i. Free software, small download size
    b. Visual Studio Community
        i. Free software, larger download size
3. Bokeh Library
    a. Once Python is installed, open a *command prompt* window
    b. Enter `pip install bokeh` into the command prompt to install Bokeh

The Basics

Quick Start Documentation here.

Open the code text editor and start a new **python** file (*must* end with **.py** extension), name it `example.py` and save. The file path will be needed later to run the code.

Use `print()` to print variables or strings to the command prompt, this will be useful for debugging.

1 - Figure

First declare a figure, which is the canvas where all plots are drawn onto and is where the title and axis labels are defined.

Import figure function from bokeh.plotting

```
from bokeh.plotting import figure
```

Next, declare the title and axis labels of the graph as figure parameters.

```
fig = figure(title='title here',
             x_axis_label='x label here',
             y_axis_label='y label here')
```

where `title`, `x_axis_label`, and `y_axis_label` correspond to the graph's title, x-label, and y-label, respectively.

2 – Plot graph

To plot graphs onto the figure, use `<figure variable>.<plot type>( x,y )`.

```
# first define some data
x = [1,2,3,4,5]
y = [4,6,3,7,9]
```

Where x and y must be the same length.

Now, plot the x and y data on a basic line graph using `fig.line()`.

```
lineGraph = fig.line(x,y)
```

All graphs can be assigned a legend label using `legend_label=''` and line graphs can specify line width with `line_width=''`.

```
lineGraph = fig.line( x,y, legend_label='line 1', line_width=2 )
```

3 – Show Plot

Finally, to view the graph using the bokeh server , import `curdoc` from `bokeh.plotting`.

```
from bokeh.plotting import curdoc
```

Pass the **figure** variable into the curdoc root, *not* the individual plots.

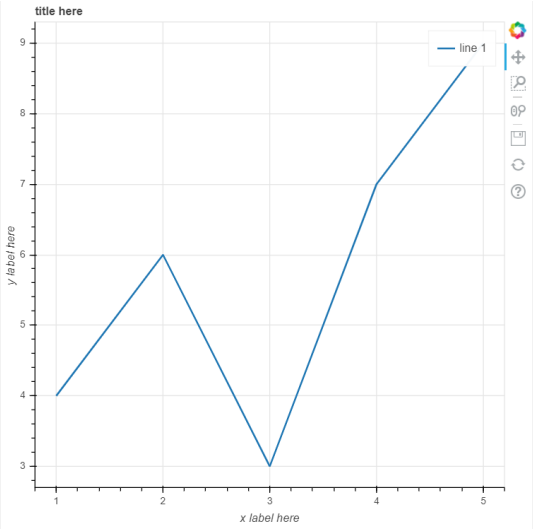
```
curdoc().add_root(fig)
```

Open the command prompt and navigate into the folder where the code file is located using `cd <folder>` to move *into* a folder, for instance, cd Documents, and `cd ../` to move *out* of a folder.

Once in the same folder as the .py file, run `bokeh serve --show example.py` to open the graph in `localhost`.

To exit the server, press `Ctrl + C` in command prompt. It may take a few seconds to shut down.

4 – Full Code




```
from bokeh.plotting import figure, curdoc
fig = figure(title='title here',
             x_axis_label='x label here',
             y_axis_label='y label here')
# define some data
x = [1,2,3,4,5]
y = [4,6,3,7,9]
# plot a line graph
lineGraph = fig.line( x,y, legend_label='line 1', line_width=2 )
# show figure
curdoc().add_root(fig)
```

Command Prompt Commands

Table 1 Command Prompt Shortcuts

Code	Command	Example
cd <*folder*>	Change Directory	cd Documents
cd ../	Exit current folder	
Ctrl + C	Exit Running Server/Program	
Up Arrow	Scroll through past commands	
TAB	Auto fill file/folder names	cd Doc + [TAB] = cd Documents

Export

Export Documentation here.

Bokeh Server

Bokeh Server Documentation here.

A Bokeh server must be used when working with interactive widgets.

In the python file, `curdoc` is used to specify the widgets and figures to display on the webpage.

`from bokeh.plotting import curdoc` to import curdoc.

To pass multiple figures or widgets into curdoc's root, a layout such as `row()` or `column()` must be used.

```
curdoc.add_root(row(<figure>,<widget>))
```

To run the code on your computer, open the command prompt and navigate to the file using `cd <folder>` and enter `bokeh serve --show <python file>.py`. A web browser with address `localhost:5006` will open.

This server is running locally on your computer and is only used for development. To exit the server, go back to the command prompt and press `Ctrl + C`.

To deploy the graph on the web for others to use, a web server must be used (see the **Heroku** section).

example.py

```
from bokeh.plotting import figure, curdoc
fig = figure()
x = [1,2,3,4,5]
y = [5,3,2,1,3]
fig.line(x,y)
curdoc().add_root(fig)
```

In command prompt: bokeh serve --show example.py

output_file()

For basic graphs that do *not* have interactive widgets, `output_file` should be used to export a static HTML file.

Import `output_file()` and `show()` functions from bokeh plotting:

```
from bokeh.plotting import output_file, show
```

`output_file(<filename>.html)` specifies the name of the HTML file to output the static graph.

`show(<figure>)` specifies the figure(s) to display on the HTML file.

To run the code, open the command prompt and navigate to the file using `cd <folder>` and enter `python <python file>.py`. A HTML file will appear in the same folder and is opened with a web browser.

example.py

```
from bokeh.plotting import figure, output_file, show
fig = figure()
x = [1,2,3,4,5]
y = [5,3,2,1,3]
fig.line(x,y)
output_file("export.html")          # exports to export.html file
show(fig)                           # show fig in html file
```

In Command Prompt: python example.py

Figures

A basic figure can be declared without any parameters as `figure()` but parameters allow customization. All figure parameters can be found here.

The following is a standard figure declaration with common parameters:

```
fig = figure(title='title here',                    # figure title
             x_axis_label='x label here',           # x-label
             y_axis_label='y label here',           # y-label
             width=500, height=500,                 # width and height
             x_range=[0,10], y_range=(-10,10))      # mins and maxes
```

Note that for `x_range` and `y_range` the use of `[]` or `()` yield the same results, so the above ranges can also be written as `x_range=[0,10],y_range=[-10,10]` or `x_range=(0,10),y_range=(-10,10).`

See the table below for figure parameters, where # refers to a number.

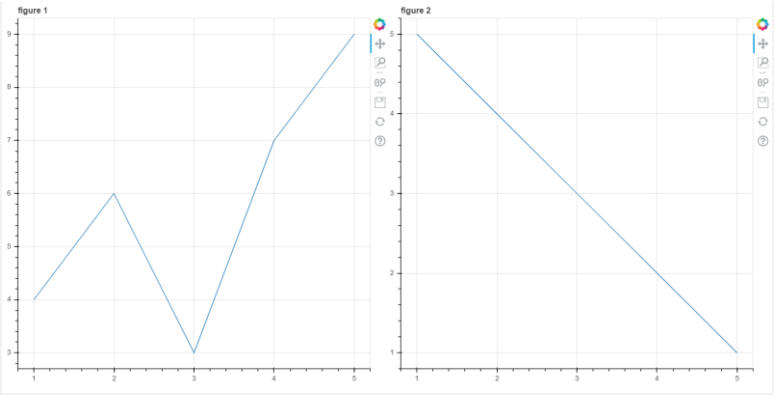
Table 2 Figure Parameters

Code	Parameter Name	Example
title = " "	Graph title	title = "my graph"
x_axis_**label** = " "	X axis label	x_axis_label = "x"
y_axis_**label** = " "	Y axis label	y_axis_label = "y"
width = #	Figure width	width = 500
height = #	Figure height	height = 500
x_**range** = [#,#] or	X axis range [x min, x max] or	x_range = [0,10] or

x_**range** = (#,#)	(x min, x max)	x_range = (0,10)
y_**range** = [#,#] or y_**range** = (#,#)	Y axis range [y min, y max] or (y min, y max)	y_range = [0,10] or y_range = (0,10)
x_axis_**location** = “above”/”below”	Position x axis above or below	x_axis_location = “above”
y_axis_**location** = “left”/”right”	Position y axis left or right	y_axis_location = “right”
x_axis_**type** = “linear”/”log”/ “datetime”/”mercator”	Type of x axis	x_axis_type = “log”
y_axis_**type** = “linear”/”log”/ “datetime”/”mercator”	Type of y axis	y_axis_type = “datetime”
tools = “ , , , “	List of tools in toolbar	tools = “ pan,wheel_zoom, box_zoom,save,reset,help” List of tools here
toolbar_**location** = “above”/”below”/”left”/”right”	Position of toolbar	toolbar_location = “above”
x_minor_**ticks** = #	Number of small ticks between large ticks in x-axis	x_minor_ticks = 3
y_minor_**ticks** = #	Number of small ticks between large ticks in y-axis	y_minor_ticks = 3

Multiple Figures

To create multiple figures, define a different variable for each `figure()` and assign each plot to the corresponding figure by using `<corresponding figure variable>.<plot type>( x,y ).` Remember to pass all figures into the curdoc root at the end using a row/column layout (discussed later).




```
from bokeh.layouts import row           # layout
from bokeh.plotting import figure, curdoc
# declare all figures
fig1 = figure(title='figure 1')
fig2 = figure(title='figure 2')
```

```
# some data
x = [1,2,3,4,5]
y1 = [4,6,3,7,9]
y2 = [5,4,3,2,1]
# assign each plot to its figure
plot1 = fig1.line( x,y1 )
plot2 = fig2.line( x,y2 )
# add all figures to curdoc
curdoc().add_root( row(fig1,fig2 ) )
```

Plots

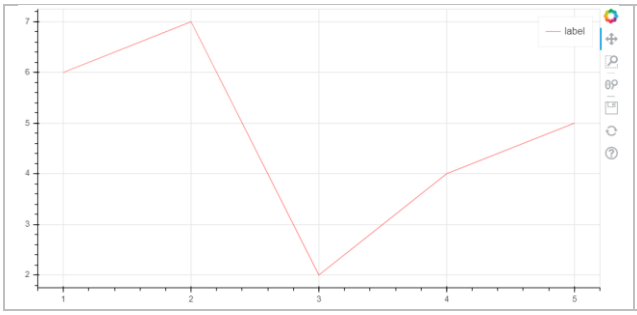
Plots are containers that hold all the various objects (renderers, guides, data, and tools) that comprise the final visualization that is presented to users [1]. Plots Documentation here.

Types of Plots

All plots have the following parameters:

Table 3 Parameters for all graphs

Parameter	Info
x =	X data (required)
y =	Y data (required)
legend_label = "*label*"	Name in the graph's legend
alpha = #, *where # is number between 0 and 1*	Opacity of plot, 0 is transparent and 1 is opaque
color = "*color name*" eg. color = "orange" or RGB **color** = (*red,blue,green*) *where red, blue, green are values between 0 to 255* eg. color = (255,0,100) RGBA **color** = (*red,blue,green,alpha*) *where alpha is opacity from 0 to 1* eg. color = (255,0,100,0.5)	Color of plot All colors here **Table 4 Common Color Names** black blue brown cyan green orange pink purple red white yellow

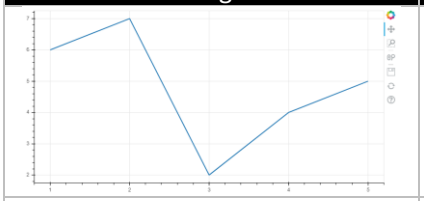
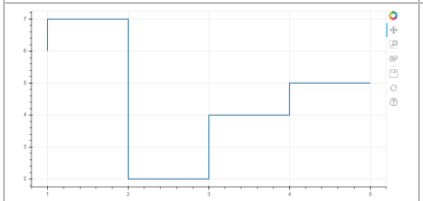
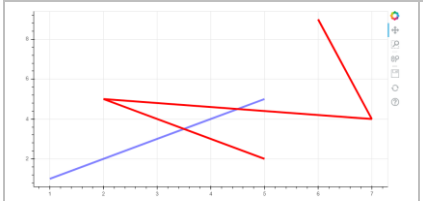
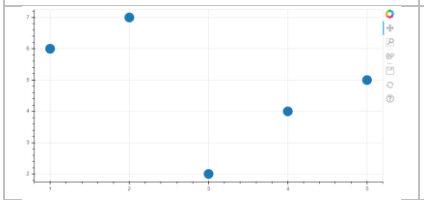
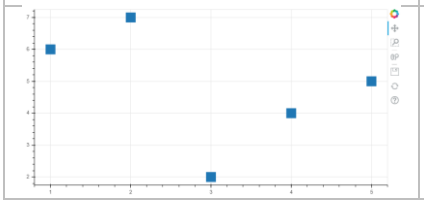


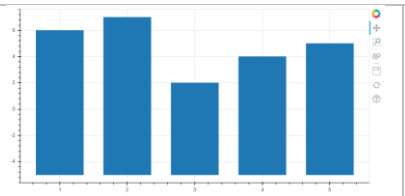
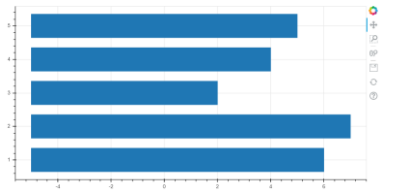
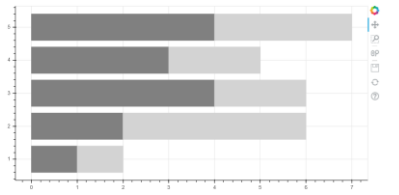
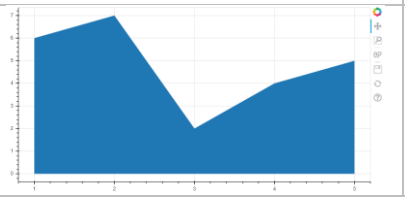
```
fig = figure(width=800, height=400)
x = [1,2,3,4,5]
y = [6,7,2,4,5]

fig.line(x,y,
legend_label="label",
alpha=0.5,
color="red")
```

Let `x = [1,2,3,4,5]`, `y = [6,7,2,4,5]` and `fig = figure(width=800, height=400)` in the following table:

Table 5 Type of Plots

Image	Plot Name	Code
	Line	fig.**line** (x,y, line_width=2)
	Step	fig.**step** (x,y, line_width=2, mode="center"/"after"/"before") ***mode:*** *step levels drawn before, after or center on x-coordinates*
	Multiple Lines	y1 = [9,4,5,3,2] fig.**multi_line**([x,y],[x,y1], line_width=4, color=["blue","red"], alpha=[0.5,1])
	Scatter - circles	fig.**circle**(x,y,size=20)
	Scatter - squares	fig.**square**(x,y,size=20)

	Vertical Bar	fig.**vbar** (x=x, top=y, bottom=-5, width=0.7) ***top**: y values ; **bottom**: min number*
	Horizontal Bar	fig.**hbar**(y=x, right=y, left=-5, height=0.7) ***y**: x values ; **right**: y values ; **left**: min number ; **height**: thickness of bar*
	Stacked Horizontal Bar	from bokeh.models import ColumnDataSource source = ColumnDataSource(data=dict(y=[1, 2, 3, 4, 5], x1=[1, 2, 4, 3, 4], x2=[1, 4, 2, 2, 3],)) fig.**hbar_stack**(['x1', 'x2'], y='y', height=0.8, color=("grey", "lightgrey"), source=source)
	Vertical Area	fig.**varea**(x,y)

Data

So far, the data for x and y have been given in simple lists (ie. `x=[1,2,3,4,5]`), however, there are more efficient ways using **for loops** and **numpy**.

For loops

`for i in range(10)` starts with the variable i equal to 0, and after each loop adds 1 to i until i equals 9.

Start by declaring empty lists for x and y, and use a for loop to append (add) data to x and y.

```
x = []
y = []
for i in range(10):
    x.append(i)
    y.append((i+1)/10)
```

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] and y = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

The shorthand for the y list is `y = [(i+1)/10 for i in range(10)]`

Numpy

Numpy is a python library with mathematical functions, that can be used to make data.

To install numpy, go to the command prompt and enter: `pip install numpy.`

To use numpy, import it into the code: `import numpy as np` .

Numpy's **linspace** and **arange** functions are commonly used to generate x values:

1. Arange – returns evenly spaced values with a *step size*
   a. `np.arange (start, stop)`
      i. `np.arange (0,10)` is `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
   b. `np.arange (start, stop, stepsize)`
      i. `np.arange (0,10,2)` is `[0, 2, 4, 6, 8]`
2. Linspace – returns evenly spaced *number of values*
   a. `np.linspace (start, stop)`
      i. `np.linspace (0,10)` is `[0.,0.20408163, … ,9.79591837,10.]` (default is 50 values returned between start and stop)
   b. `np.linspace (start, stop, numberOfValues)`
      i. `np.linspace (0,10,2)` is `[0.,10.]`

For example, x can be generated using `arange`, and `y=sin(x)` can be generated using `np.sin` and a `for loop`:

```
import numpy as np
x = np.arange(0,10,1)
y = []
for i in range(10):
    y.append(np.sin(i))
```

The for loop can be skipped by passing x directly into `np.sin`, use this method moving forward. This can be done with most numpy math functions:

```
import numpy as np
x = np.arange(0,10,1)
y = np.sin(x)
```

Table 6 Common numpy functions

Code	Example Input	Example Output
np.**arange** (*start, stop, stepsize*)	np.arange (0,5,2)	[0, 2, 4]
np.**linspace** (*start, stop, num*)	np.linspace (0,5,2)	[0., 5.]
np.**pi**	np.pi	3.141592653589793
np.**exp**(#) is $e^{\#}$	np.exp(1)	2.718281828459045
np.**random.random** ()	np.random.random ( )	0.5922195266392477
np.**random.randint** (*max num*)	np.random.randint (5)	2
np.**sin** ( )	np.sin([1,2,3])	[0.84147098, 0.90929743, 0.14112001]
np.**cos** ( )	np.cos ([1,2,3])	[0.54030231,-0.41614684,-0.9899925]

np.**log** (#) is $\ln(\#)$	np.log (5)	1.6094379124341003
np.**log10** (#) is $\log(\#)$	np.log10 (5)	0.6989700043360189
np.**sqrt** (#)	np.sqrt (4)	2
len (*list*) *length of a list*	len ([1,2,3])	3
a ** b is a^b	2**3	8
np.**abs** (#)	np.abs (-10)	10
np.**round** (#)	np.round (10.1)	10.0

Modifying Plot Data

Plot data can be modified by first assigning the plot to a variable.

```
plot1 = fig.line(x,y)
```

To access the x and y data, use **<*plot*>.data_source.data** to get a dictionary of x and y:

```
data = plot1.data_source.data
data['x']         # x values of plot1
data['y']         # y values of plot1
```

Now to change the values, set `data['x']` and/or `data['y']` to a new list.

```
data['x'] = [1,2,3]
data['y'] = [9,4,6]
```

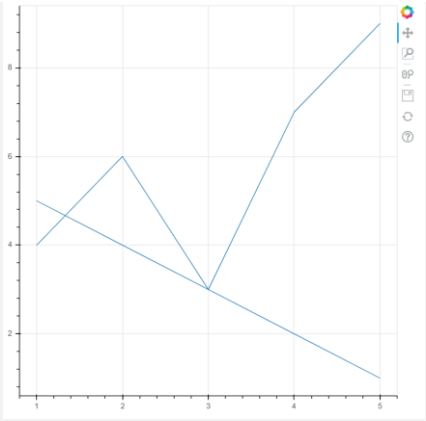
Full code:

```
from bokeh.plotting import figure, curdoc
import numpy as np
fig = figure()
x = np.arange(0,10)
y = np.sin(x)
plot1 = fig.line(x,y)
data = plot1.data_source.data
data['x'] = [1,2,3]
data['y'] = [9,4,6]
curdoc().add_root(fig)
```

Note that to plot data from an excel file, use Python's Pandas library.

Multiple Plots

To create multiple plots on a single figure, assign the figure() a variable and use the *same* figure for all plots.



```
from bokeh.plotting import figure, curdoc
fig = figure()
# some data
x = [1,2,3,4,5]
y1 = [4,6,3,7,9]
y2 = [5,4,3,2,1]
# assign all plots to fig
plot1 = fig.line( x,y1 )
plot2 = fig.line( x,y2 )
curdoc().add_root( fig )
```

Layouts

Layouts Documentation here.

The 3 layouts control how figures and widgets are organized on the page.

These layouts are passed as arguments into `curdoc().add_root()` or `show()`.

Ex. `curdoc().add_root(row( … ))` or `show(row( … ))`

row()

All arguments inside the row function will be organized in a horizontal row

```
from bokeh.layouts import row
curdoc().add_root(row(fig1,fig2,widget1,widget2))
```

column()

All arguments inside the column function will be organized in a vertical column

```
from bokeh.layouts import column
curdoc().add_root(column(fig1,fig2,widget1,widget2))
```

gridplot()

Gridplot organizes its arguments in a grid-like fashion

```
from bokeh.layouts import gridplot
```

The first way to use gridplot is as follows:

```
curdoc().add_root(gridplot([fig1,fig2],[widget1,widget2]))
```

or

```
curdoc().add_root(gridplot([fig1,fig2],[None,widget2])) # None is empty
```

The second way is as follows:

```
curdoc().add_root(gridplot([fig1,fig2,widget1,widget2],ncols=2))
```

Widgets

Widgets Documentation here.

Bokeh widgets can be divided into 2 groups, `.on_click` widgets (ie. button, dropdown, checkbox, radio) that require a callback function *without* any parameters: `def update()`.

And `.on_change` widgets, which require a callback function with 3 parameters: `def update(attr, old, new)`, where `attr` refers to the changed attribute's name, while `old` and `new` refer to the old and new values of the changed attribute.

To create a widget, first declare the callback function which is called when the user changes a value or clicks the widget. In the following, this function will be called update, but it can take any non-keyword name.

```
def update():            OR          def update(attr, old, new):
    pass                                    pass
```

Then import the desired widget from **`bokeh.models`**, and declare a new widget assigned to a variable.

```
from bokeh.models import <widget>       # import desired widget
my_widget = <widget>( … )               # create a new widget, my_widget
```

Now link the widget to the corresponding callback function using `.on_click` or `.on_change`.

For on_change, the attribute to listen to or that will change is the first argument, then the function.

```
my_widget.on_click(update)     OR     my_widget.on_change("<attr>",update)
```

Finally, add the widget to the curdoc root.

```
curdoc().add_root( my_widget )
```

.on_click Example:

```
from bokeh.models import Button
from bokeh.plotting import curdoc
def update():
    print("Clicked!")
button = Button(label="foo",button_type="success")
button.on_click(update)
curdoc().add_root(button)
```

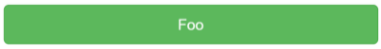
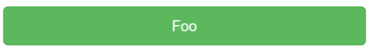
.on_change Example:

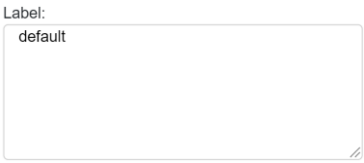
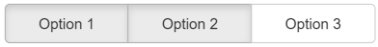
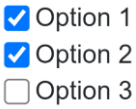
```
from bokeh.plotting import curdoc
from bokeh.models import Slider
def update(attr, old, new):
    print(attr)  # "value"
    print(new)
    print(old)
slider = Slider(start=0, end=10, value=1, step=.1, title="Slider")
slider.on_change("value",update)
curdoc().add_root(slider)
```

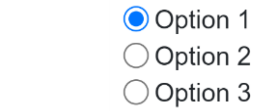
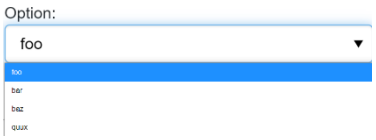
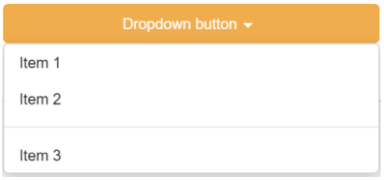
Note that widget names are capitalized, not doing so will result in an error.

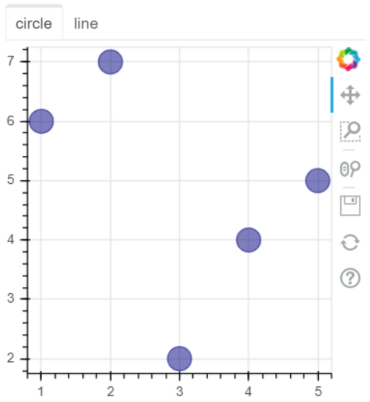
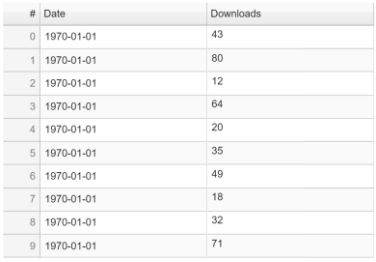
For the following table, let update be `def update():` for on_click widgets and `def update(attr, old, new):` for on_change widgets.

Table 7 Widgets table

Widget	Code	Parameters
	from bokeh.models import Button button = **Button** (label="Foo", button_type="success") button.on_click (update)	**Label:** text shown on button **Button_type:** color and style of button (default, primary, success, warning, danger)
	from bokeh.models import Toggle toggle = **Toggle** (label="Foo", button_type="success") toggle.on_change ("active",update)	**Active**: True/False, if in pressed mode or not **Label:** text shown on button **Button_type:** color and style of button (default, primary, success, warning, danger)

	from bokeh.models import Slider slider = **Slider** (start=0, end=10, value=1, step=.1, title="Stuff") slider.on_change ("value",update)	**Start:** min slider value **End:** max slider value **Value:** current slider value **Step:** step size when sliding **Title:** text above slider
	from bokeh.models import RangeSlider range_slider = **RangeSlider** (start=0, end=10, value=(1,9), step=.1, title="Stuff") range_slider.on_change ("value",update)	**Start:** min slider value **End:** max slider value **Value:** current slider value as a tuple: (#,#) **Step:** step size when sliding **Title:** text above slider
	from bokeh.models import TextInput text_input = **TextInput** (value="default", title="Label:") text_input.on_change ("value",update)	**Value:** text currently in box **Title:** text above box
	from bokeh.models import TextAreaInput text_input = **TextAreaInput** (value="default", rows=6, title="Label:") text_input.on_change ("value",update)	**Value:** text currently in box **Rows:** max number of rows, determines height of box **Title:** text above box
	from bokeh.models import CheckboxButtonGroup checkbox_button_group = **CheckboxButtonGroup** (labels=["Option 1", "Option 2", "Option 3"], active=[0, 1]) checkbox_button_group.on_change ("active",update)	**Labels:** list of the names of options **Active:** which labels in labels list is currently selected
	from bokeh.models import RadioButtonGroup radio_button_group = **RadioButtonGroup** (labels=["Option 1", "Option 2", "Option 3"], active=0) radio_button_group.on_change ("active",update)	**Labels:** list of the names of options **Active:** which *one* option is currently selected
	from bokeh.models import CheckboxGroup checkbox_group = **CheckboxGroup** (labels=["Option 1", "Option 2", "Option 3"], active=[0, 1]) checkbox_group.on_change ("active",update)	**Labels:** list of the names of options **Active:** which label in labels list is currently selected

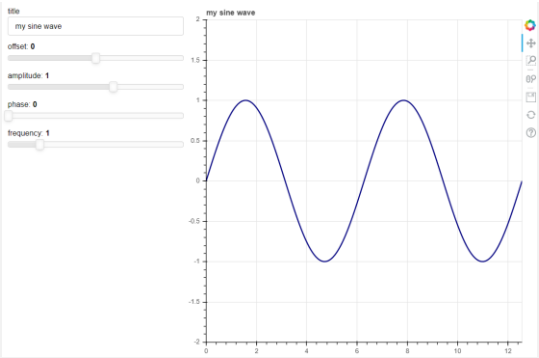
	from bokeh.models import RadioGroup radio_group = **RadioGroup** (labels=["Option 1", "Option 2", "Option 3"], active=0) radio_group.on_change ("active",update)	**Labels:** list of the names of options **Active:** which *one* option is currently selected
	from bokeh.models import MultiSelect multi_select = **MultiSelect** (title="Option:", value=["foo", "quux"], options=[("foo", "Foo"), ("bar", "BAR"), ("baz", "bAz"), ("quux", "quux")]) multi_select.on_change ("value",update)	**Title:** text above selection box **Value:** options current selected **Options:** list of all options, where [("to sent to function","text to display")]
	from bokeh.models import Select select = **Select** (title="Option:", value="foo", options=["foo", "bar", "baz", "quux"]) select.on_change ("value",update)	**Title:** text above select box **Value:** selected option **Options:** list of options
	from bokeh.models import Dropdown def update(event): print(event.item) # selected item menu = [("Item 1", "item_1"), ("Item 2", "item_2"), None, ("Item 3", "item_3")] dropdown = **Dropdown** (label="Dropdown button", button_type="warning", menu=menu) dropdown.on_click (update)	**Label:** text shown on button **Button_type:** color and style of button (default, primary, success, warning, danger) **Menu:** items in the dropdown menu where [("text to display","to pass into function")]
	from bokeh.models import ColorPicker color_picker = **ColorPicker** (color="#ff4466", title="Choose color:", width=200) color_picker.on_change ("color",update)	**Color:** currently color selected in hex **Title:** text displayed above widget
	from bokeh.models import FileInput file_input = **FileInput** () file_input.on_change ("value",update)	Once passed into update file, the file or picture will need to be decoded

Your HTML-supported text is initialized with the **text** argument. The remaining div arguments are **width** and **height**. For this example, those values are *200* and *100* respectively.	from bokeh.models import Div div = Div (text="""Your HTML-supported text is initialized with the text argument. The remaining div arguments are width and height. For this example, those values are 200 and 100 respectively.""", width=200, height=100)	**Text**: html text or plain text **Width:** width of div **Height:** height of div
	from bokeh.models import Panel, Tabs from bokeh.plotting import figure p1 = figure(plot_width=300, plot_height=300) p1.circle([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=20, color="navy", alpha=0.5) tab1 = Panel(child=p1, title="circle") p2 = figure(plot_width=300, plot_height=300) p2.line([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], line_width=3, color="navy", alpha=0.5) tab2 = Panel(child=p2, title="line") tabs = Tabs(tabs=[tab1, tab2])	
	from bokeh.models import ColumnDataSource, DataTable, DateFormatter, TableColumn from datetime import date from random import randint data = dict(dates=[date(2014, 3, i+1) for i in range(10)], downloads=[randint(0, 100) for i in range(10)],) source = ColumnDataSource(data) columns = [TableColumn(field="dates", title="Date", formatter=DateFormatter()), TableColumn(field="downloads", title="Downloads"),] data_table = DataTable(source=source, columns=columns, width=400, height=280)	

Examples

All bokeh examples here.

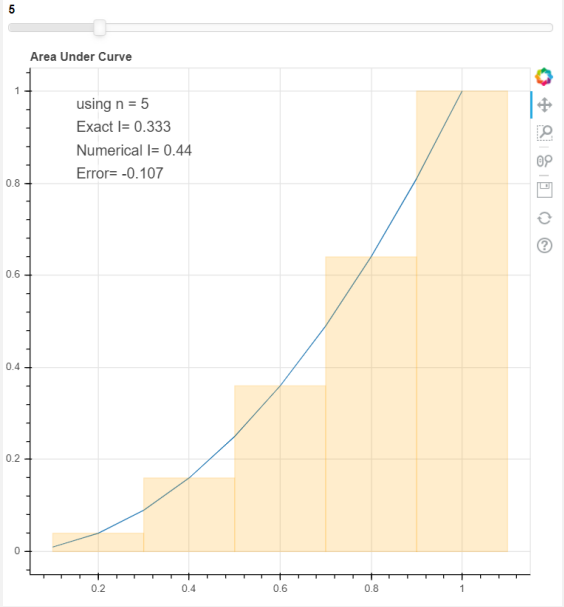
Sine Graph Example




```
import numpy as np
from bokeh.layouts import column, row
from bokeh.models import Slider, TextInput
from bokeh.plotting import figure, curdoc
# create figure
fig = figure(title="my sine wave",x_range=[0,4*np.pi],y_range=[-2,2])
# data
x = np.linspace(0,4*np.pi,200)
y = np.sin(x)
# plot sin
plot = fig.line(x,y,line_width=2,color="navy")
# widgets
text = TextInput(title="title", value='my sine wave')
offset = Slider(title="offset", value=0.0, start=-5.0, end=5.0, step=0.1)
amplitude = Slider(title="amplitude", value=1.0, start=-5.0, end=5.0,
                   step=0.1)
phase = Slider(title="phase", value=0.0, start=0.0, end=2*np.pi)
freq = Slider(title="frequency", value=1.0, start=0.1, end=5.1, step=0.1)
# callback functions and link widget to function
def text_update(attr,old,new):
    fig.title.text = new
text.on_change("value",text_update)
```

```
def data_update(attr,old,new):
    a = amplitude.value
    b = offset.value
    w = phase.value
    k = freq.value
    # Generate the new curve
    y = a*np.sin(k*x + w) + b
    # update
    data = plot.data_source.data
    data['y'] = y
for widget in [offset, amplitude, phase, freq]:
    widget.on_change('value', data_update)
curdoc().add_root(row( column(text,offset,amplitude,phase,freq),fig ))
```

Area Under Curve Example

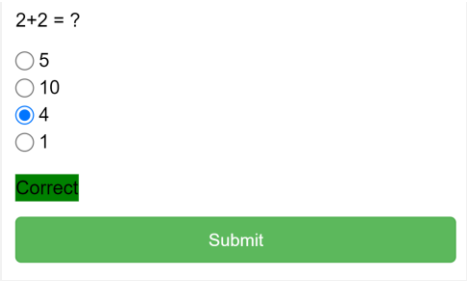



```
from bokeh.layouts import column
from bokeh.models import Slider, Label
from bokeh.plotting import curdoc, figure
# draw the starting graph figure (empty canvas)
fig = figure(title="Area Under Curve")
# data for graph
```

```
n = 5
width = 1/n
L1 = [(i+1)/n for i in range(n)]
L2 = [(i+1)/10 for i in range(10)]
ybar = [L1[i]**2 for i in range(n)]
ycurve = [L2[i]**2 for i in range(10)]
# add line graph to figure
fig.line(L2, ycurve)
# add bar graph to figure
barGraph = fig.vbar(x=L1, top=ybar,alpha=0.2,color='orange',width=[1/n]*n)
# numbers to display
TrueArea = 0.333
NumArea = round(sum(ybar)*width,3)
Error = round(TrueArea-NumArea,3)
# create labels for all text to be displayed
text1 = Label(x=50, y=500, x_units='screen', y_units='screen',
              text=r'using n = '+str(n),
              background_fill_color='white', background_fill_alpha=.6)
text2 = Label(x=50, y=475, x_units='screen', y_units='screen',
              text=r'Exact I= '+str(TrueArea),
              background_fill_color='white', background_fill_alpha=.6)
text3 = Label(x=50, y=450, x_units='screen', y_units='screen',
              text=r'Numerical I= '+str(NumArea),
              background_fill_color='white', background_fill_alpha=.6)
text4 = Label(x=50, y=425, x_units='screen', y_units='screen',
              text=r'Error= '+str(Error),
              background_fill_color='white', background_fill_alpha=.6)
# add all text to figure
fig.add_layout(text1)
fig.add_layout(text2)
fig.add_layout(text3)
fig.add_layout(text4)
```

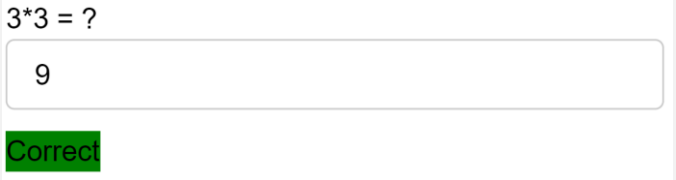
```
# update called when slider is moved
def update(attr, old, new):
      # new graph data based on slider number
      n = new # new is number on slider
      L1 = [(i+1)/n for i in range(n)]
      ybar = [L1[i]**2 for i in range(n)]
      # apply the changes to the barGraph's data
      barGraph.data_source.data['x'] = L1
      barGraph.data_source.data['top'] = ybar
      # all data must be samelength
      barGraph.data_source.data['width'] = [1/n]*n
      # change the text
      width = 1/n
      TrueArea = 0.333
      NumArea = round(sum(ybar)*width,3)
      Error = round(TrueArea-NumArea,3)
      text1.text = r'using n = '+str(n)
      text2.text = r'Exact I= '+str(TrueArea)
      text3.text = r'Numerical I= '+str(NumArea)
      text4.text = r'Error= '+str(Error)
# Slider
slider = Slider(start=2, end=20, value=5, step=1)
# when value changes, call update function
slider.on_change("value",update)
# send to server
curdoc().add_root(column(slider,fig))
```

Multiple Choice Example




```
from bokeh.layouts import column
from bokeh.models import Div, RadioGroup, Button
from bokeh.plotting import curdoc
question = Div(text="2+2 = ?")
# Radiogroup, multiple choice selection
multiple_choice = RadioGroup(labels=["5", "10", "4", "1"], active=0)
check_button = Button(label="Submit", button_type="success")
checked = Div(text="")
# mc_check function called when check_button clicked
def mc_check():
    if multiple_choice.labels[multiple_choice.active] == "4":
        checked.text = "<div style='background-color:green'>Correct</div>"
    else:
        checked.text = "<div style='background-color:red'>Incorrect</div>"
check_button.on_click(mc_check)
# send to server
curdoc().add_root(column(question, multiple_choice,checked,check_button))
```

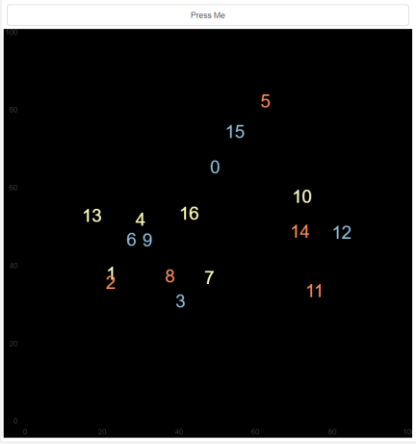
Written Response Example




```
from bokeh.layouts import column
from bokeh.models import Div, TextInput
from bokeh.plotting import curdoc
```

```
# text input, get user's answers
text_input = TextInput(value="",title="3*3 = ?",placeholder="type answer
                       here")
checked2 = Div(text="")
# text_check called when text input is changed
def text_check(attr, old, new):
    if new == "9":
        checked2.text = "<div style='background-color:green'>Correct</div>"
    else:
        checked2.text = "<div style='background-color:red'>Incorrect</div>"
text_input.on_change("value",text_check)
# send to server
curdoc().add_root(column(text_input, checked2))
```

Random Numbers on Figure Example




```
from random import random
from bokeh.layouts import column
from bokeh.models import Button
from bokeh.palettes import RdYlBu3
from bokeh.plotting import figure, curdoc
# create a plot and style its properties
p = figure(x_range=(0, 100), y_range=(0, 100), toolbar_location=None)
p.border_fill_color = 'black'
p.background_fill_color = 'black'
```

```
p.outline_line_color = None
p.grid.grid_line_color = None
# add a text renderer to our plot (no data yet)
r = p.text(x=[], y=[], text=[], text_color=[], text_font_size="26px",
           text_baseline="middle", text_align="center")
i = 0
ds = r.data_source
# create a callback that will add a number in a random location
def callback():
    global i
    # BEST PRACTICE --- update .data in one step with a new dict
    new_data = dict()
    new_data['x'] = ds.data['x'] + [random()*70 + 15]
    new_data['y'] = ds.data['y'] + [random()*70 + 15]
    new_data['text_color'] = ds.data['text_color'] + [RdYlBu3[i%3]]
    new_data['text'] = ds.data['text'] + [str(i)]
    ds.data = new_data
    i = i + 1
# add a button widget and configure with the call back
button = Button(label="Press Me")
button.on_click(callback)
# put the button and plot in a layout and add to the document
curdoc().add_root(column(button, p))
```

Miscellaneous

Styling Documentation here.

Styling Visualizing network graphs here.

Maps in Bokeh here.

Annotations here.

Heroku

www.heroku.com is a server hosting platform and should be used to deploy interactive applications that use widgets. The code must use Bokeh Server (curdoc()) and *not* output_file()). For static applications without widgets that use output_file(), Heroku is not needed; Github Pages provides a free service for hosting these static HTML/Javascript files.

Installation

Two applications need to be downloaded for deploying to Heroku:

1. Heroku CLI
2. Git

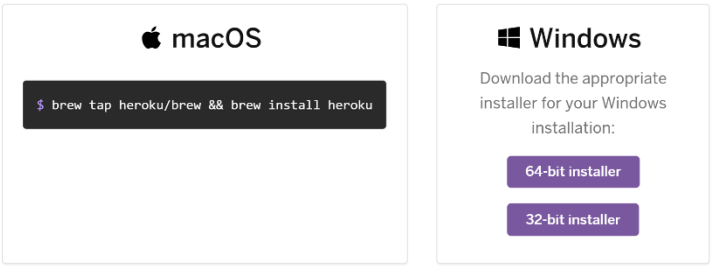
1 – Heroku CLI

Go to https://devcenter.heroku.com/articles/heroku-cli.

For Mac: type `brew tap heroku/brew && brew install Heroku` into the terminal

For Windows, download the 32-bit or 64-bit installer at https://devcenter.heroku.com/articles/heroku-cli.

Figure 1 Download Heroku CLI



Run the downloaded .exe file and press **Next** on *Choose Components* window

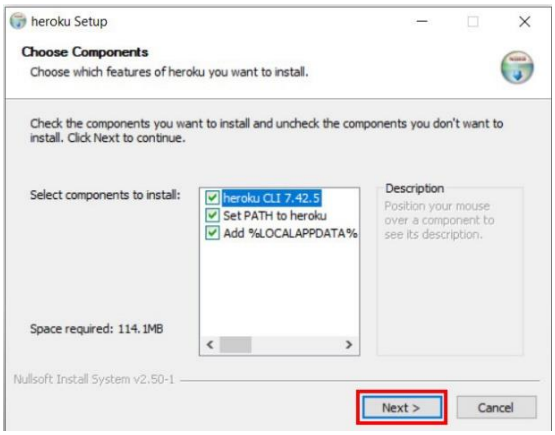


Figure 2 Press Next

Press **Install** on *Choose Install Location* window

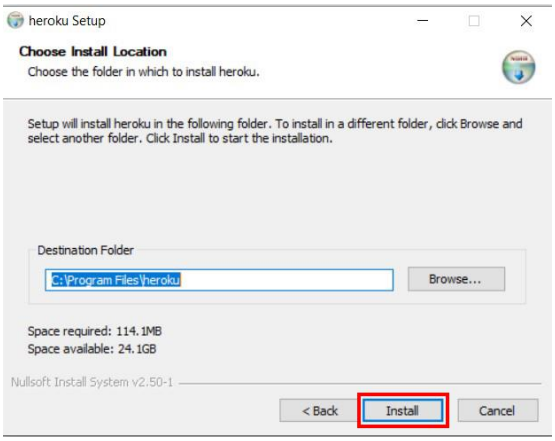


Figure 3 Press Install

2 – Git

Go to https://git-scm.com/downloads and download Git for your system

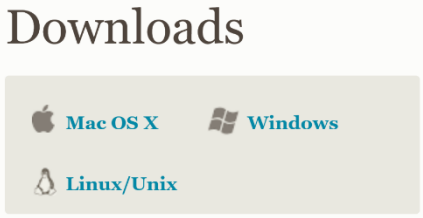


Figure 4 Download Git

Run the download .exe file and click **Next** on GNU General Public License, then Press **Next** on *Select Destination Location* window.

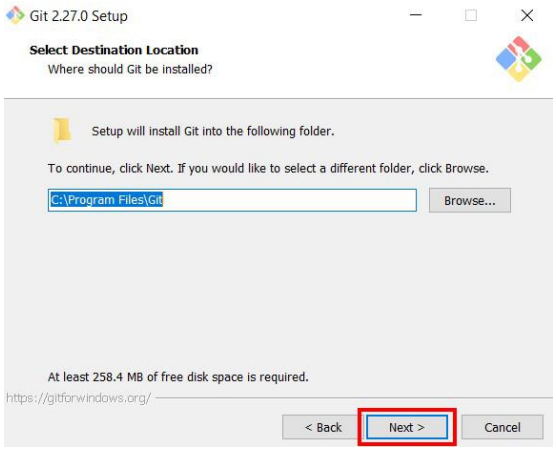


Figure 5 Press Next

Press **Next** on *Select Components* window

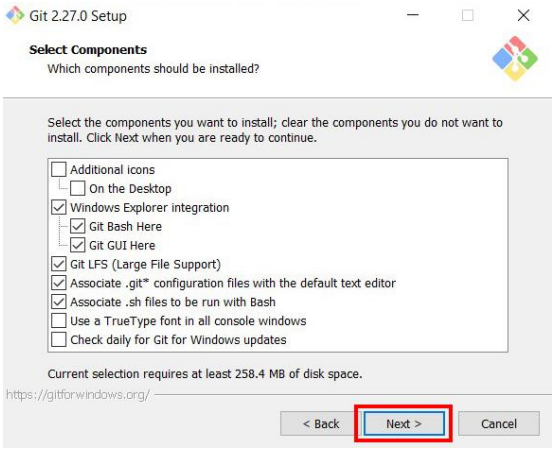


Figure 6 Press Next

Press **Next** on *Select Start Menu Folder* window

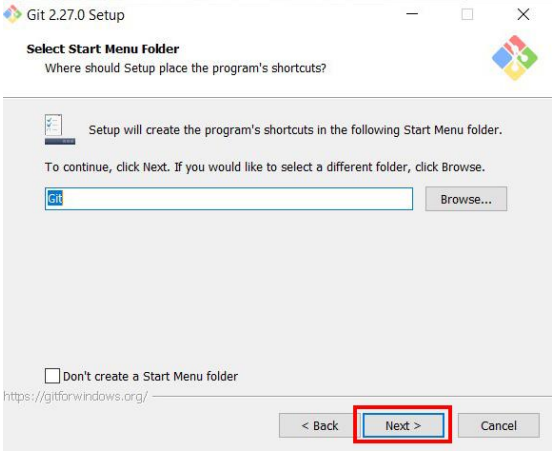


Figure 7 Press Next

On *Choose the default editor used by Git,* select your default text editor from the dropdown menu and then press **Next**

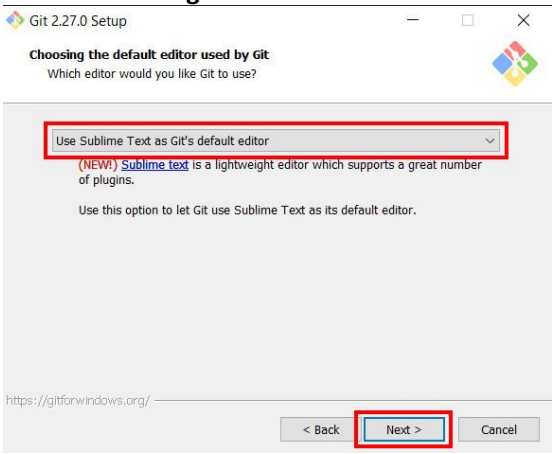


Figure 8 Select Text Editor, then press Next

Press **Next** on the *Adjusting your PATH environment* window

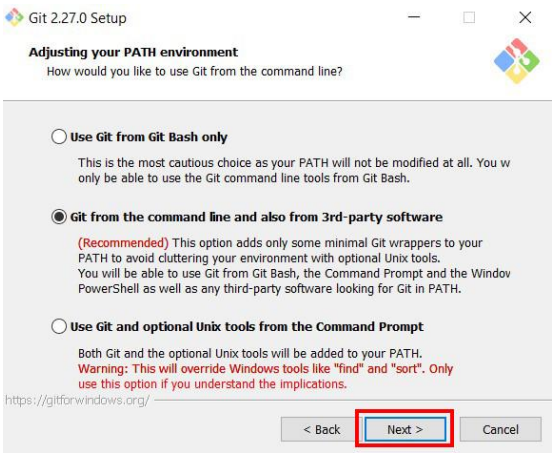


Figure 9 Press Next

Press **Next** on *Choosing HTTPS transport backend* window

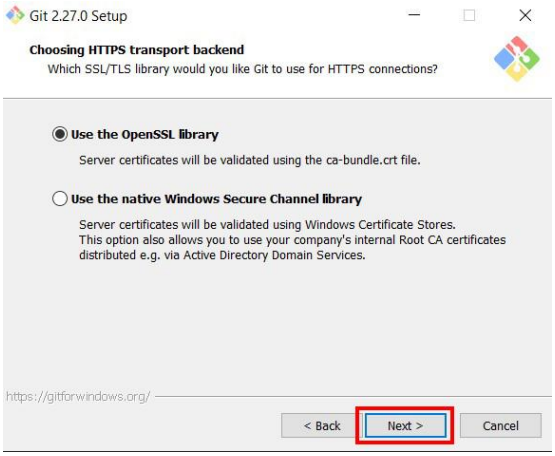


Figure 10 Press Next

Press **Next** on *Configuring the line ending conversions* window

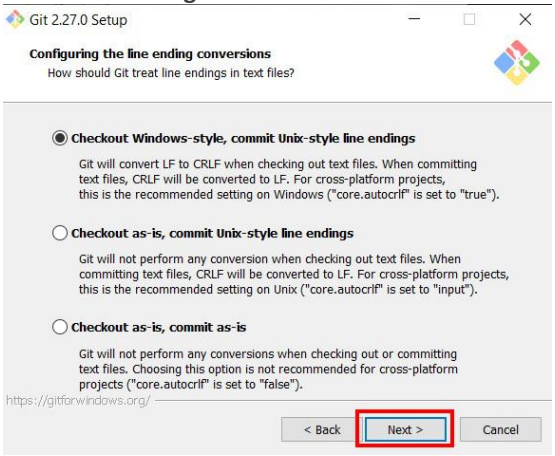


Figure 11 Press Next

Press **Next** on *Configuring the terminal emulator to use with Git Bash* window

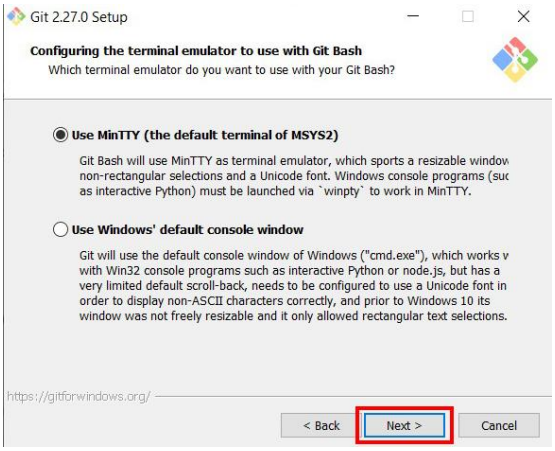


Figure 12 Press Next

Press **Next** in the *Choose the default behavior of 'git pull'* window

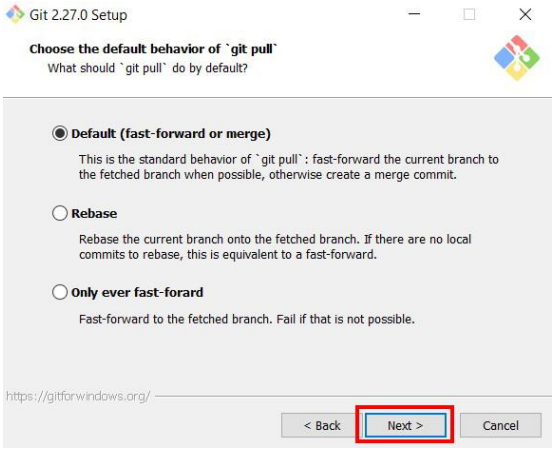


Figure 13 Press Next

On the *Configuring extra options* window, check "Enable symbolic links" and press **Next**

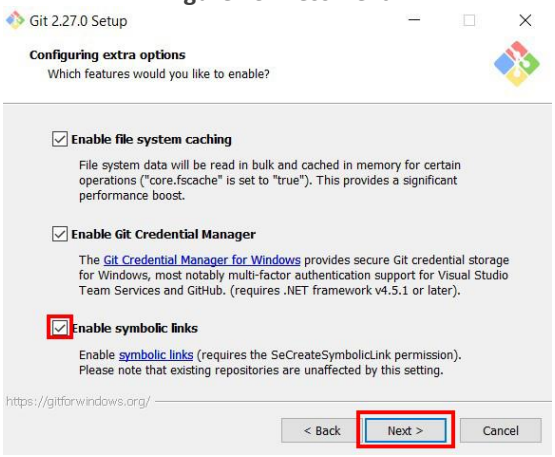


Figure 14 Check a box, then press Next

Finally, press **Install** on *Configuring experimental options* window

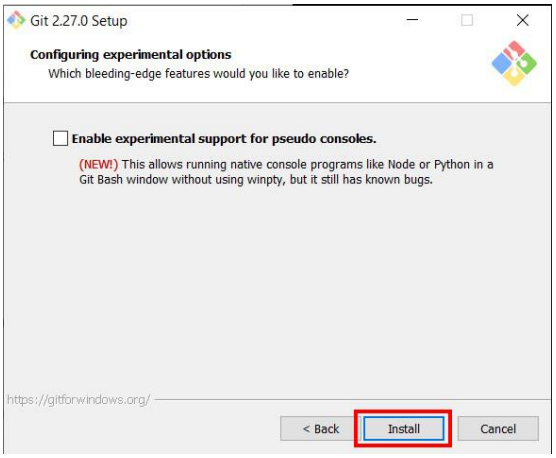


Figure 15 Press Install

Close and reopen all command prompt windows for Git to start working.

Heroku CLI and Git Commands

Table 8 Command Prompt Commands

Heroku CLI		Git	
heroku **login**	Login to Heroku account	git **add** .	Add all files to git
heroku **create** <*name*>	Creates a new app with the given <*name*>	git **commit** -am "message"	Commit all files. "message" can be any text
heroku **open**	Open website of app	**git push heroku master**	Push to Heroku website
heroku **apps**	List of your apps		

Login

Go to https://id.heroku.com/login to login to Heroku

Creating a New Heroku App

Move the Python (.py) file with the Bokeh code into a new folder and rename it to **app.py**

Open the text editor and create the following 2 essential files, the spelling and capitalization are important:

1. **Procfile**
   a. Note: capital **P** and **no** file extension, just "Procfile"
2. **requirements.txt**
   a. Note: lower case **r** and **.txt** file extension

```
└──example
       app.py
       Procfile
       requirements.txt
```

Figure 16 Files

Warning: do *NOT* name your .py file bokeh.py, this will cause issues.

Copy and paste the following into each respective file:

Procfile

The Procfile tells Heroku that the app is a *web* app and the command to run (bokeh serve app.py).

This file does *not* need to be changed unless you don't want the .py file to be named app.py. To rename the .py to anything other than app.py, go into Procfile and replace "app.py" with the name of .py file.

```
web: bokeh serve --port=$PORT --allow-websocket-origin=* --address=0.0.0.0 --use-xheaders app.py
```

requirements.txt

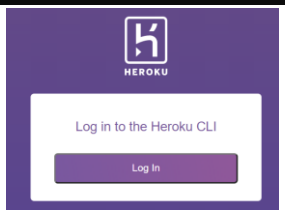
The requirements file lists all the python libraries needed to run the code.

- Add any additional libraries used in the code to this file in the format: `<library name>==<version number>`.
- If using a newer version of a library, update the version number in the requirements.txt file.

```
bokeh==2.1.1
Jinja2==2.11.2
numpy==1.19.0
packaging==20.4
pillow==7.2.0
python-dateutil==2.8.1
PyYAML==5.3.1
scipy==1.5.1
tornado==6.0.4
typing_extensions==3.7.4.2
```

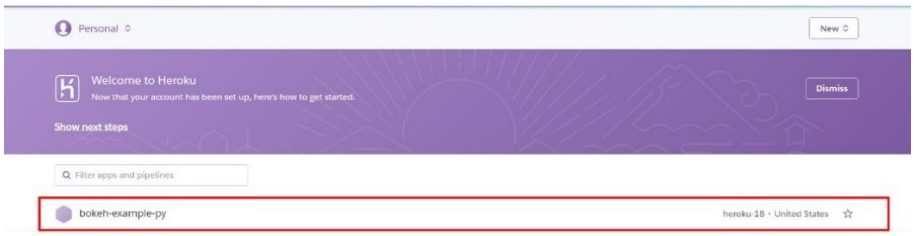
Command Line

Open Command prompt window and navigate to new folder using cd command.

1. Enter `heroku login`
   a.
   ```
   C:\Users\weich\Documents\example>heroku login
   heroku: Press any key to open up the browser to login or q to exit:
   ```
   b. A webpage will open. Login to Heroku with your login details.
   

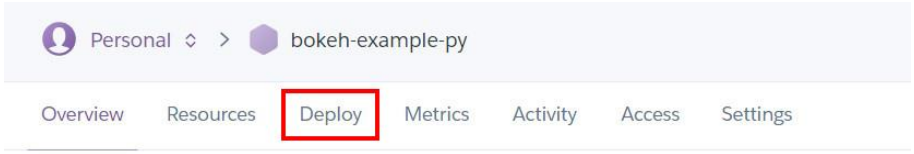
   c.
   ```
   Logging in... done
   Logged in as adeeb@ualberta.ca
   ```
2. Type `heroku create <app name>` to create a new app, note some names may be unavailable.
   a.
   ```
   C:\Users\weich\Documents\example>heroku create bokeh-example-py
   Creating ⬀ bokeh-example-py... done
   https://bokeh-example-py.herokuapp.com/ | https://git.heroku.com/bokeh-example-py.git
   ```
3. Open the dashboard in heroku.com: https://dashboard.heroku.com/, the new app is now shown

a.

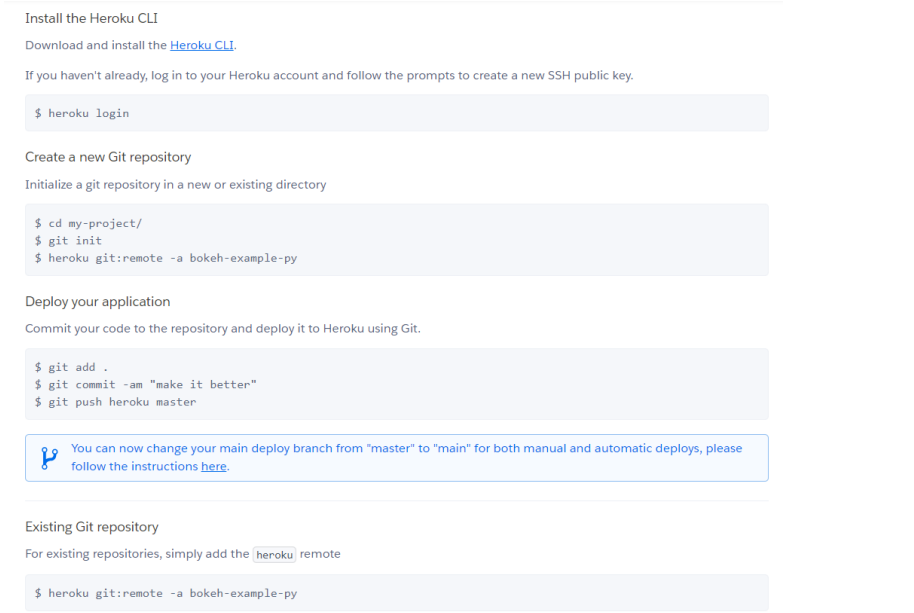


4. Click the app and click the **Deploy** tab to see the deployment command line codes

a.



b.



5. Back to the command line, enter `git init` to initialize the git repo

a.

```
C:\Users\weich\Documents\example>git init
Initialized empty Git repository in C:/Users/weich/Documents/example/.git/
```

6. Enter `heroku git:remote -a <app-name>` to link to Heroku app

a.

```
C:\Users\weich\Documents\example>heroku git:remote -a bokeh-example-py
set git remote heroku to https://git.heroku.com/bokeh-example-py.git
```

7. Enter `git add .` to add all files to git repo
8. Enter `git commit -am "make it better"` to commit the files

a.
```
C:\Users\weich\Documents\example>git commit -am "make it better"
[master (root-commit) 17ef8f5] make it better
 3 files changed, 46 insertions(+)
 create mode 100644 Procfile
 create mode 100644 app.py
 create mode 100644 requirements.txt
```

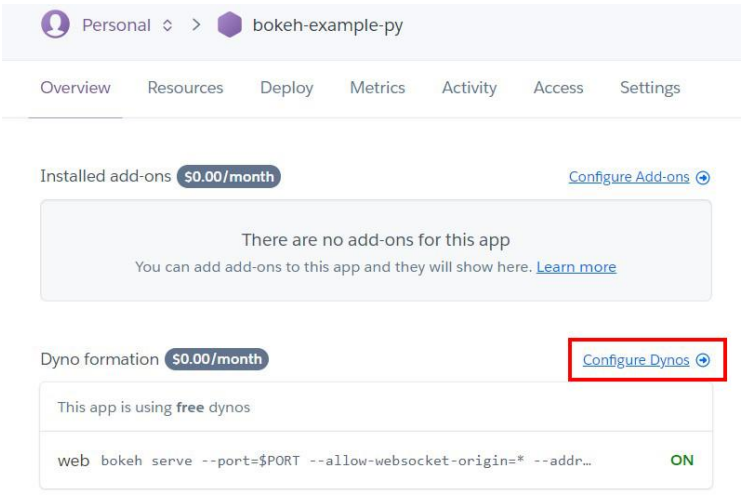
9. Enter `git push heroku master` to finally push all files to Heroku

a.
```
C:\Users\weich\Documents\example>git push heroku master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 1.05 KiB | 1.05 MiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Python app detected
remote: -----> Installing python-3.6.11
remote: -----> Installing pip
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote:        Collecting bokeh==2.1.1
```

10. Note that git commands *must* be in the order of git **add**, git **commit**, and then git **push**.
11. To open the site, enter `heroku open`.
12. Change the **Dyno Type**: the app is currently running on "free dynos", this should only be used for testing purposes. "Free Dynos" are limited by a set amount of runtime hours and long load times because it goes to "sleep" after 30 minutes of inactivity.

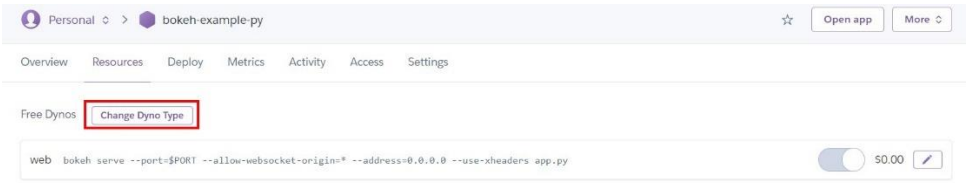
    Change dyno type to "**hobby"** when the app is finalized.

    a. Go back to the app in Heroku account and click **Configure Dynos**

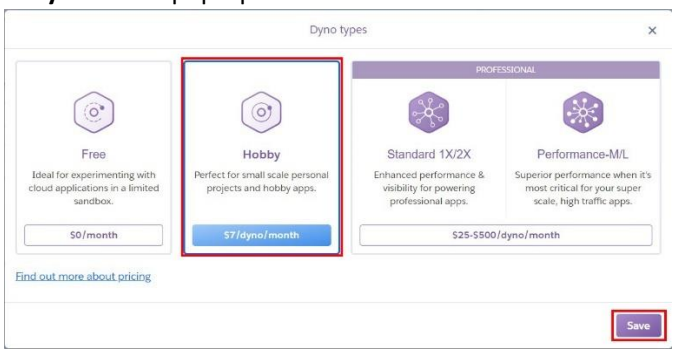


i.

b. Press **Change Dyno Type** button

i. 

c. Select **Hobby** from the pop-up window and hit Save

i. 

d. Done.

Modifying Deployed App

To modify the app after deploying it, see detailed instructions in Deploy tab. Here `git:clone` is used to copy the folder stored on the server to your computer where it can be modified. Once changes are made in the cloned folder, use git add, git commit, and git push to update the app.

In the commands below, "$" is not apart of the code and is rather to indicate that this code is to be executed in the command prompt.

```
$ heroku login
```

```
$ heroku git:clone -a bokeh-example-py
$ cd bokeh-example-py
```

```
$ git add .
$ git commit -am "make it better"
$ git push heroku master
```

References

[1] Bokeh, "Bokeh Documentation," [Online]. Available: https://docs.bokeh.org/en/latest/docs/user_guide.html#userguide. [Accessed July 2020].

[2] Numpy, "Numpy," [Online]. Available: https://numpy.org/doc/stable/. [Accessed July 2020].

[3] H. Rosenblum, "Having Fun with Bokeh Server App," [Online]. Available: https://medium.com/analytics-vidhya/having-fun-with-bokeh-server-app-part-iv-tutorial-af4b3f2b1ef8. [Accessed July 2020].

[4] J. Dorning, "How to deploy a Bokeh app on Heroku," [Online]. Available: https://medium.com/@jodorning/how-to-deploy-a-bokeh-app-on-heroku-486d7db28299. [Accessed July 2020].