Building Interactive Visualizations Using Bokeh

by Janani Ravi

Start Course

Bookmark Add to Channel Download Course

Table of contents Description **Transcript** Exercise files Discussion Learnin

Course Overview

Course Overview

Hi. My name is Janani Ravi, and welcome to this course on Building Interactive Visualizations Using Bokeh. A little about myself. I have a Master's degree in Electrical Engineering from Stanford and have worked at companies such as Microsoft, Google, and Flipkart. At Google, I was one of the first engineers working on real-time collaborative editing in Google Docs, and I hold four patents for its underlying technologies. I currently work on my own startup, Loonycorn, a studio for high-quality video content. In this course, you will learn how to use Bokeh to build interactive visualizations on the web, which allows for easy exploration of data without in-depth coding knowledge. We start off by understanding the internals of how Bokeh works and the basic building blocks of Bokeh plots. We'll work with glyphs, plot tables, arbitrary shapes, and visual layouts. We'll then study some of the more specialized plots in Bokeh, such as plots which work with categorical data, network graphs, and geographical data. We'll build geo-plots using built-in Bokeh maps, as well as using the Google Maps API. We'll also use Bokeh's huge sample data set to prototype some interesting plots. We'll then cover some advanced features in Bokeh, such as integrating with Bokeh plot tools to enhance plot interactivity and working with the Bokeh server, which offers a model-view-controller paradigm to manipulate data in Python and view it using a browser. At the end of this course, you should be comfortable working with Bokeh plots and features to extract insights from your own data in the real world.

Plotting with Glyphs in Bokeh

Module Overview

Hi, and welcome to Building Interactive Visualizations Using Bokeh. As data becomes more important, exploring and analyzing data using some kind of visual tool becomes very necessary for data analysts and data scientists. Bokeh is a visualization package that focuses on offering interactive tools that allows you to visualize, interact with, zoom in, pan out, and work with your data. There are a variety of visualization packages out there and each of them have their own strengths and weaknesses. Bokeh's strength is the fact that it offers interactive visualizations on the browser. Bokeh offers browser-based interactivity over very large or even streaming data sets. Bokeh is declarative and not procedural, which means you set up a figure and plots within the figure, and you define attributes and properties which apply to the figure and to the plot. This makes Bokeh very approachable, even for programming newbies. If you want a visualization package that allows you to interactively explore a data before you perform machine learning or some other kind of data analysis, Bokeh is the one for you. It's very easy to get up and running and produce rich visualizations in the form of an HTML file or within your Jupyter Notebook.

Prerequisites and Course Outline

Before we get started with Bokeh itself, let's take a look at what prereqs you need in order to make the most of this course. This course assumes that you have a basic understanding of Python. If you're not very comfortable with Python, I'd recommend these courses on Pluralsight, Python: Getting Started and Python Fundamentals. All the demos in this course are built using Python 3, though we'll talk about JavaScript in the browser, no prior knowledge of JavaScript or HTML is required. You'll be working with Jupyter Notebooks and display your plots inline. You need to have some basic knowledge of statistics so that you understand some basic factors of how we set up our visualizations. Nothing beyond high school level stats. The basic building blocks of Bokeh plots are visual representations called glyphs. We start off by plotting glyphs in

Bokeh. We'll be introduced to the different APIs that Bokeh offers for plotting, and we'll work with markers, lines, shapes, layouts, and axes. In the next module, we'll see how we can build basic and intermediate plots with Bokeh. We'll work with categorical data. We'll build up a network graph. We'll work with geodata and annotations. Visualizations that you build using Bokeh are interactive by default; however, this interactive can be configured and added to. That's what the next module is all about. We'll see how to add interactions with legends, tooltips, toolbar. We'll also see how you can get a Bokeh server up and running. This allows you to interact with your plots in JavaScript and have the events be handled within Python.

Introducing Bokeh

Organizations have been collecting data for many years because they know that data is important, but often this data is in a raw form and sitting somewhere in a data warehouse. In order to see whether the data can be used for any kind of business decisions, you need to visualize the data so that you can absorb the information that it contains, and then use it for further analysis, such as with machine learning. An important precursor to any kind of exploration and analysis that you perform with data is visualization, which is why visualization techniques have become a very important part of data science, and there are entire packages built in Python, and other languages, expressly for this purpose. Visualization is what helps you to develop an intuition for the relationships that exist in data and is an important precursor to higher-level data analysis using techniques such as machine learning and deep learning. Let's say you're a business analyst generating reports or a data analysist looking to find insights in data. Adding interactivity to your visualizations helps with exploration and experimentation because it allows you to zoom in and see the portions of the data that might be important. Interactivity is an important functionality, which is lacking from many visualization tools and libraries out there. Interactivity enables exploration, which is why Bokeh is so important. Bokeh offers interactive visualizations, which dramatically increases your understanding of the underlying data. Here is a definition of what Bokeh is all about from the bokeh. pydata. org website. The key differentiator of Bokeh from other visualization packages is the fact that the visualizations that it offers are interactive by default, and also the fact that Bokeh is primarily meant for use from within web browsers. The output is an HTML file, along with CSS and JavaScript, that runs along with the BokehJS library. The Bokeh API is available in Python so you build up your plots and visualizations in Python, but it can also be used from R, as well as Scala. Amongst visualization packages, you might have worked with Matplotlib. This has been around for a long time and is very powerful; however, it's not that accessible. Bokeh emphasizes accessibility, ease of use, and interactivity. In today's world, Python is the programming language of choice for data scientists and data analysts, and there are a number of Python-based visualization packages out there. Matplotlib offers MATLABs, production-ready visualizations for the Python developer. If you want a higher-level API, then you have Seaborn. This is a high-level library, which is built on top of Matplotlib and allows you to build complex charts very easily. In recent times, we have other Python packages which do not depend on Matplotlib, and Bokeh is amongst those. Bokeh is focused on browser-based interactive visualizations and produces a JSON object that is interpreted by JavaScript on the browser, and the same goes for Plotly. Not built on Matplotlib either, this focuses on online collaborative visualizations. Let's quickly see how Bokeh stacks up against these other Python packages. Matplotlib is powerful, but complex to figure out and use. Bokeh is simple because it's declarative, not procedural. Bokeh focuses on interactive visualizations. Matplotlib focuses on providing powerful, low-level granular control over your plots. Because of its declarative nature, Bokeh's accessible even to developers with less experienced programming. Seaborn is a high-level wrapper over Matplotlib and can be complex as well. Bokeh focuses on interactivity. Seaborn's focus is on making Matplotlib's API more accessible and intuitive. Bokeh can be thought of as a more general-purpose visualization package, whereas Seaborn is focused on scientific and statistical visualizations. Bokeh and Plotly are both browser-based visualization tools and are not built on top of Matplotlib. Bokeh focuses on interactivity. Plotly focuses on online collaboration where you can share your plots with others. Bokeh's ease of use and intuitive API is great for exploratory data analysis. Plotly is great for production-ready visuals. Now Bokeh offers very simple general-purpose standard visuals; however, Plotly has some sophisticated visualizations such as funnel charts, Sankey diagrams, and so on. Bokeh is a product that has been developed by NumFOCUS, which is a nonprofit organization. Bokeh is open-source software and is part of the PyData Stack. Bokeh is part of the PyData Stack, which is an open data science stack built for Python developers. If you like working with Bokeh and you want to contribute, there are many ways in which you can do so. You can offer your engineering and programming expertise and make code contributions, or you can choose to make a donation.

Plotting Basics

If you haven't worked with visualizations before, let's get our basic terminology straight. This is the anatomy of any figure that you'll build up using a library such as Bokeh. A figure may be embedded within your browser window or within your Jupyter Notebook. A figure may have grid lines. Grid lines make it easier to view and interpret the data in the figure. A figure is also made up of axes. Axes can be the x-axis and the y-axis, and in the case of visualizations that support 3D, you can have a z-axis as well. Numeric values that are represented along these axes are generally represented by our ticks. Now ticks which are darker and represent significant points are major ticks. You also have minor ticks. Visuals can take on a variety of different forms. You might have a simple line that is drawn through your plot, and this line represents something. This is typically a line plot or just a line. You might have dots representing significant points. These are called markers. The title of any plot is usually an explanatory statement indicating what exactly this visualization represents that's important to convey additional information to the user. If you have multiple lines and markers within your plot, you might also choose to include an axis legend. These concepts form the basic building blocks of a figure. There are, of course, other concepts such as interactions, toolbars, tooltips. All of those we'll see when we actually do the demos.

Bokeh Internals

This class is very hands-on and very focused on learning by actually building up these visualizations, but before we get started on the demos, let's understand a little bit of the Bokeh internals. Bokeh offers two categories of APIs that you can use to build up your visuals. Bokeh. plotting can be thought of as a mid-level to a high-level interface, which is centered around composing the visual elements that make up your graph. Bokeh. models is a much lower-level interface that allows much more fine-grained control over your visuals. This allows you to work directly with the Bokeh model, which is an object graph that encompasses all of the visual and data aspects of the scene that you build up in Bokeh. When we talk about the architecture of Bokeh, Bokeh actually is made up of two different library components. First is the BokehJS, which is the JavaScript library, and then we have Bokeh Python or Bokeh R or Scala. This is the programming language interface that offers APIs to developers. The Python library is what exposes bokeh. models, which allow you to work with low-level objects, and bokeh. plotting, the mid-level API. Now in this course, and generally if you are a data scientist, you won't work directly with BokehJS. BokehJS is a JavaScript library that runs on the browser. The Python library is the code that you write. It runs within your Jupyter Notebook or Python interpreter. If you're working in Python or Scala or R and you're using that language to build up your visuals, why is BokehJS even needed? This is because every plot that you produce using Bokeh is serialized as a JSON object, and BokehJS consumes these JSON objects in order to render your plot to the browser. When you're using the programming APIs in Python to build up your visualization, you're producing those JSON objects consumed by BokehJS. BokehJS renders JSON objects, Bokeh Python produces them. When you're actually working on the Python code to build up your plots in Bokeh, you'll find that you won't have to think about the BokehJS library at all, but you have to remember that your plots are finally JSON objects, which are consumed by the JS library. We'll be using the bokeh. plotting interface a lot, so let's see the basic steps in setting up a plot using bokeh. plotting. We first prepare data in the form of lists, arrays, or dataframes. Bokeh interacts very well with pandas, as well as NumPy. We then specify where this plot should be displayed, the output sink. This can be an HTML file or within your Jupyter Notebook. We then use a figure function in order to instantiate a figure. This takes in the title, the tools you want to display, and axes in our plot, labels in our plot, and then we move on to adding renderers. Renderers are responsible for colors, legends, widths of the plot, and the actual plot itself. Once we've configured the visual exactly how we want to, we can then call the show function, which then displays the plot in the browser. You can also choose to show it in a notebook. Once the plot has been displayed, you may also choose to save it to a file. Jupyter Notebooks, the browser-based interactive shell to Python, is widely used by the scientific and the analysist community. Bokeh integrates seamlessly with Jupyter Notebooks, and you can choose to display your plots within your notebook. If you want to render your visualizations on the browser, you'll simply use the output_file function. You can render the same visualizations within a Jupyter Notebook when you use the output_notebook function.

Plotting to File and within Jupyter Notebooks

Bokeh is very simple and easy to use, and it's best studied with hands-on practice. Let's get introduced to basic plotting. We start off within our Jupyter Notebook window. This is our current working directory. The datasets directory under our current working directory is where we'll have the CSV files for the data sets that we'll use in plotting later on in this course. For now, let's start up a new Python 3 Notebook. All the demos in this course are written using Python 3. This notebook is going to be our introduction to Bokeh, so I'm going to name it Introduction. You can get the Bokeh Python library on your local machine using a simple pip install. Make sure you specify the exclamation point before you call pip if you are running it from within your Jupyter Notebook. If the Python that you have installed on your local machine is the Anaconda Python Distribution and you have Conda set up, you can also use this command at your terminal prompt, conda install bokeh, in order to get Bokeh on your local machine. The current version of Bokeh is 0. 13. I'm going to import the Bokeh model into my local Python Notebook and print out the version. You can see that it is indeed 0. 13. If you have a later version, there might be a few subtle differences based on the improvements that the Bokeh engineers have made. We'll first randomly generate a few x and y data points. In the form of Python list, this is what we'll use to plot in our very first Bokeh figure. The figure is a model in the Bokeh plotting interface. This is Bokeh's high-level interface to generate plots. All Bokeh visualizations are within a figure called a figure function in order to instantiate a figure, p. We specify the plot_width and the plot_height of this figure, which is 600 x 300. Our first Bokeh plot will be to draw a line connecting these x, y coordinates that we had set up earlier, and we can do this very simply by calling the line function on the figure. The basic visual marks that Bokeh displays on screen are called glyphs. The line that we are going to generate here is a very basic glyph. Once the glyph has been instantiated, you can display this using the show function. You need to import it from the bokeh. io namespace, call show, pass in the figure, and by default, Bokeh will save this plot to an HTML file. That is the default setup. You can see here that our browser opens up a new tab with our Bokeh plot. This is a simple line plot, and you can see that we have a line connecting the five x, y coordinates that we had set up earlier. Here is our first point, at 1, 6, it's connected to the other points, you can see the second to last point here at 4, 6, and the last point at 5, 5. The plot that we generated using Python code was rendered on the browser, and the browser, you know, renders just JavaScript and CSS files. This means in order to generate plots, the BokehJS JavaScript library and the corresponding CSS files need to be loaded onto your browser in order to display these plots. The JavaScript and CSS needed for this visualization is automatically loaded into your browser by referencing it from the CDN at pydata. org. There is nothing additional that you need to do. Notice the URL in the browser window, Bokeh has generate a temporary HTML file which stores your plot. Bokeh's plots are interactive by default. The interactive controls are available to the side within the toolbar. You can hover over these interactive controls. Pan allows you to pan through your control. You can move your control around to zoom in to the bits of data that you're interested in. You can click on the Refresh button highlighted here to get back to your old representation. The toolbar offers a Zoom button by default. If you click on this, you can then select those portions of the plot that you want to zoom in to, and you'll find that it automatically zooms in. Click on the Refresh button in order to get back to your original representation. Let's head back to the Jupyter Notebook here. If you want your plots to be rendered inline within your Jupyter Notebook, you can call the output_notebook function, which will configure your plots to be displayed embedded within Jupyter. You'll simply invoke the output_notebook function as you see on screen. These functions are generally invoked at the beginning of an interactive session or at the beginning of your script files so all the plots that you generate follow this same output mode, either within a Jupyter Notebook or to an HTML file. Invoking the output_notebook function loads the BokehJS library within your Jupyter Notebook. Remember, this is the library that is used to display your plots. Now if you call show p, you will find that the same figure that we had set up earlier is now displayed inline in Jupyter. If you want to switch back to viewing your plots within an HTML file on your browser window, you can use the output_file function. The output_file function can also be used to export plots to a specific file by specifying that file name. Here, our plot will be exported to the line. html file within the current working directory. The show p function is what we use to display our figures, and you'll find that this opens up a browser window, and the URL of the browser window shows that this file has been saved into the line. html file. The plot is also displayed within your Jupyter Notebook. We had explicitly specified output_notebook earlier. If you want to go to the default Bokeh settings for how a plot output should be displayed, you can disable all settings by calling the reset_output function. We'll now call the output_file function. We want our plot saved out to line. html as before. This will cause the old plot to be rewritten, and so I'm instantiating the same figure once again, drawing the same line, and calling show. This will now be displayed on my browser and not within my Jupyter Notebook. The old settings have been reset.

Customizing Markers and Lines

In this demo, we'll see how to plot and customize simple visual glyphs such as markers and lines, set up the import statement for all the Bokeh modules that you need, and then specify that you want all of the plots to be embedded within Jupyter. The output_notebook function is what we'll call at the beginning of each of our files. Instantiate a figure that is 600 px by 300 px, and this time, let's plot a circle glyph. The circle glyph plots a scatterplot of your data, not a line plot. The show command will display a simple scatterplot with five data points. The first point is at 1, 6, the second point at 2, 7, and the last point at 5, 5. Let's instantiate a new figure, and this time we'll plot the same data points as before, but we'll customize our markers. We want our markers to be larger, size 20, and we want them to be in the maroon color with an opacity value of 0. 5. That means we want them to be slightly transparent. You can see that our scatterplot plots the same points, but looks different. You can see that we have a point at 1, 6 and 2, 7, as before. You can see that each of these points are larger, maroon, and slightly transparent. You don't have to instantiate a new figure for every plot. You can choose to add glyphs to your existing figure. Here, we are going to use the same plot to draw a line through the new data points. These data points are different from the original points that we set up. Now if we display the figure, you can see that the line gets added to the circle markers that were already present on this plot. When you're plotting with Bokeh, you're not limited to just using circles as your markers, you can have different marker types. For this, we'll instantiate a new figure. This time with the title for this figure called Markers. Bokeh supports different marker types for our different functions. If we want to plot the asterisk marker at the same data points, the size should be 15 and they should be green in color. We'll add more markers at different coordinates. This is the diamond marker, which we'll set up by calling p. diamond. Colors in Bokeh can be specified using their name, as well as hexadecimal notation. Our diamond markers are not vertically oriented. They are at an incline or an angle. To the same figure, we add in square markers as well. This time, we want the fill color and the boundary color of the square markers to be different. We specify the colors in RGB notation. We want the line that is drawn as the border of this square to be 2 px in width. Call show p on the figure, and here you can see our visual with different kinds of markers: the asterisk, the square, and the diamond. Notice that the asterisks drawn at the different locations are green in color, and the size is 15 px, and here are our diamonds drawn using the p. diamond function. They are of size 20 px from tip to toe, and they are at an angle. We also have the squares here drawn using p. square. The fill color and the boundary color of the squares are different, and the width of the boundary is 2 px. This was an example of how we could customize the markers that we plot in our graph. Let's now see an example of how we can customize the lines that we draw. Use the p. line function to draw a line connecting the five coordinates. This is something that we've seen before, but this time we've customized the line. The line width is 4 px, it's red in color, it's lightly transparent, and dashed. You can see how easy it is to customize your visuals using Bokeh. Everything is a property. Specify custom values for these properties in order to customize your display. If you have a series of data points, there are different ways you can have the data points connected. For example, you can use a step to connect the various coordinates. This draws discrete steps between your data points. The first point is at 1, 6 and instead of drawing a straight line in order to connect the second point at 2, 7, there is a step. This step has been drawn with a line_width of 3 and is in the color gold. There's also a property to configure your step levels. Step levels can be drawn before, after, or centered on the x-coordinates. We've chosen the mode, before, here. You can change the mode and see how the step changes. When we use the before mode, the step goes up at the same x-coordinate as the previous data point, and the next data point is connected by a horizontal line. When you're working with data in the real world, it's very common to have missing values in your data, and these missing values may be represented as nans. Bokeh is able to debug these missing values in an intelligent way. So let's say we have a line drawn through coordinates, and one of the data points is missing. When you display this line within your figure, Bokeh does not throw an error, it'll connect only those data points for which it has complete information. You can see that the points with missing data are not connected using our pink line. Bokeh simply draws a disconnected line. Bokeh also has functionality to draw multiple lines in one go using the p. multi_line function. In this case, you can see that both the x-, as well as the y-coordinates are specified as multidimensional lists. Here is the set of points for the first set of connected lines, and here is a second set of points for the second set of connected lines. Notice that the first set of lines are not connected to the second set. They are different lines.

Bokeh also allows you to specify customizations for individual lines. Let's draw the same two lines as before using p. multi_line, but this time, we want the color of the first line to be green, the second line to be orange, and you can see that you can specify this within a list. You can specify the colors for the individual lines as items in a list. The opacity and the line widths of the individual lines can also be specified in the list format. What cannot be specified as a list though is how that line is displayed. We cannot pass a list of values for the line_dash property.

Drawing Shapes

Bokeh figures offer you a bunch of built-in functions allowing you to specify shapes of different types. Let's see how we can draw shapes with Bokeh. Once again, we'll be working within our Jupyter Notebook. All plots will be displayed inline. We start off with a figure as usual, assign it to the variable p, and we want to draw a quadratic figure. We can do so very easily be specifying the four edges of the figure. We've used the p. quad function to draw a four-sided figure or a quadrilateral. This is the edge at the bottom at coordinate 1, the edge at the top is at coordinate 7, the edge on the left is at coordinate 1 once again, and on the right is at coordinate 2. If you observe the structure of this quadrilateral and the x- and the y-axis carefully, you'll find that this quadrilateral is distorted. Its width is just 2, and its height is 6, but it seems to be wider than it's taller. We can fix this visual distortion in our figure by specifying something called the match_aspect. We have the height of the plot. Match_aspect will fix the width of the plot so that the shape that we draw is positioned at the center of the plot, and the axes are adjusted accordingly. Let's draw the same quadrilateral as before, and when we display it, we see that the axes have been adjusted to position the shape at the center of the plot. The axes have been extended. Just like Bokeh allows plotting of multiple lines in one go, you can plot multiple quadrilaterals in one go by specifying all of the edges in the form of a list. Bottom, top, left, and right are all lists. The first element in each of these lists refers to the first shape that we draw, and the second element refers to the second shape. One is the bottom edge of the first shape, and 3 is the bottom edge of the second quadrilateral. Similarly, 7 is the top edge of the first shape, and 6 is the top edge of the second shape. Notice that the fill_color is also specified as a list. The first shape is cyan, and the second shape is light yellow. You can also use p. rect function in order to draw rectangles. These take in a slightly different set of coordinates. Notice this rectangle on screen here, the x- and the y-coordinate refers to the center of this rectangle. Once you have the center, you can specify the width and the height of the rectangle separately. We can customize how this rectangle looks by specifying the fill_color, which is lightblue here; the color of the border, that is navy; and the line_width of the border is 2 px. This rectangle is also drawn at an angle. The edges of the rectangle are not parallel to the x- and y-axis. The ranges, specified by the x- and y-axis, can be defined explicitly. The range object defines the data-space bounds of your plot. If you want your x- and y-axis to have custom limits, you can specify them using the Range1d class. P. x_range and p. y_range allow you to configure the range of your figure. P refers to the same figure where we draw our rectangle. The x-axis range is from -2 to 10, and the y-axis range is from -1 to 7. The rectangle has been positioned accordingly, and you can now see that it's not out of bounds of this plot. The plot dimensions need not be specified, only when you instantiate the figure. You can specify them later on as well, along with the ranges. Bokeh has a bunch of built-in functionality to draw shapes. Let's set up a figure as before. This time, we'll draw a number of different shapes within the same figure, starting with circles. The x- and the y-coordinates refer to the center of the circle, and the list indicates that there are two circles here. The p. hex function can be used to draw hexagons, the x- and y-coordinates specify the center of the hexagon, and the p. ellipse function can be used to draw ovals. As you can see, a number of different shape types are supported. If you're interested in a different shape, such as a triangle, you should just look up the functionality available. It's probably already built in. Let's take a look at the two circles that we've drawn using p. circle. We've specified two centers here in the x- and y-coordinates. We've also customized the look and feel of the circle. One size is smaller than the other; they're both drawn transparently; and we have specified the fill color, line color, and the line widths for both of these circles. Some customization, such as the size, is on a per-circle basis, that are meaning apply to both circles. And on the left here is the hexagon that we've drawn using the p. hex function. There is just a single hexagon. We've specified the center of the hexagon here. We've also specified additional customizations for this hexagon. And finally, here are the two ellipses that we've drawn using p. ellipse. We've specified two x, y coordinates corresponding to the centers of the ellipse, and we've customized the widths of each ellipse individually. The height of both of the ellipses are exactly the same. The color property applies to both of these ellipses. They are both tomato in color. Now, it's not just a standard shape that you can plot using Bokeh. In fact, you can use Bokeh to plot any arbitrary shape by calling the patch function on the figure. The p. patch function takes in any arbitrary set of x, y coordinates and basically produces a closed figure connecting these coordinates. You can see the various coordinates that we've specified. All of these coordinates together are connected, and then you can fill this figure with the color of your choice and produce an arbitrary figure. The patch function ensures that you're not limited to the shapes that Bokeh offers. You can hand draw shapes of your choice.

Working with Layouts

Let's say you have a number of different visualizations showing different pieces of information, and you want to lay them out neatly. Bokeh allows you to format your figures in a grid format. Your figures can be displayed as cells in this grid. You can also customize this grid. In this demo, we'll instantiate three separate figures. We'll call them p1, p2, and p3. This is our first plot where we've drawn circles. This is our scatterplot. We then instantiate a second figure called p2. This is a simple line plot. The line is blue in color and is dashed. The third figure stored in the variable p3 holds an ellipse, which is tomato in color. Bokeh allows you to customize the rows and columns within your grid display, import the row and column libraries. Let's show all of our three figures in the same row. Call the show function and pass in p1, p2, and p3 to the row. The row function allows you to display your figures in a single row. You can see that we have the first plot, the second plot, and the third plot. All of these are displayed one after the other in the form of a grid. Just like with the row layout, you can use the column function to display all of these three figures in a single column. P1, p2, p3 are now stacked 1 on top of another. Rows and columns make up a grid. If you want to display your plots in a matrix format, you can import the gridplot library. The gridplot allows you to customize the display of your figures within a grid. Notice here that we have specified the figures within our grid as a list of lists. The first list contains 2 figures, p1 and p2, and the second list contains just p3. And this is what the resulting grid looks like. The three figures that you have specified in this grid have been laid out in a matrix format. Now how many rows and columns exist in this matrix? That depends on the size of your inner list. Our first inner list specified two figures. Both of these are displayed on the same row, the first row. The second list specified just a single figure, p3. This has been moved to the next row. If you want more fine-grained control over which grid cell displays your figure, you can use the None keyword. Here we have p1 and p2 displayed in the first row. This is exactly the same as in the previous example, but we want the figure p3 to be displayed in the fourth grid. So in the third position, we specify None, and that results in an empty display. If you want more fine-grained control over your grid layout, you can import the layout model, which allows you to specify the sizing_mode. Here, we are laying out the same three figures that we have instantiated earlier. We have specified sizing_mode to be scale_width. The sizing_mode determines how the items that we have specified within a layout resize to fill in the available space. A sizing_mode of scale_width indicates that your figures will be stretched or scaled to fit the width of your grid. Other options are scale height, scale both, stretch both, or fixed. The default sizing_mode used so far is the fixed sizing_mode. And here are the same three figures in a grid representation. The first two figures are in the first row. The third figure is in the second row, but it has been scaled to span the two columns occupied by the first two figures.

Configuring Multiple Axes

When you use Bokeh, it's very easy to set up different visualizations on the same figure, even if they use different axis representation. Here in this example, let's see how we can work with multiple axes. Instead of working with randomly generated data, let's read in some data from a file. This file is called AAPL. csv, and it's present under the datasets folder, and it contains stock price information for the Apple stock over a period of 2 months in the year 2017. I'm using this read_csv function from the pandas library in order to read in this data set. This will load in a data frame, and the head function will display the first few rows of data. A data frame is nothing but a row and column representation, a tabular representation, of your data. For Apple stock, we have the open, high, low, close, adjusted close, and volume information across a series of dates. When we read in from a CSV file, the Date data is available in the form of a string. Let's change this to be off the datetime format in Python. We use the pd. to_datetime to perform this conversion. We'll also adjust the data in our Volume column to be expressed in terms of millions of shares, rather than the huge numbers present at this point in time. Running the describe function on a pandas DataFrame will give you quick statistical information about the data that you've loaded. Information such as the number of records, mean, standard deviation, and the various percentiles. You can see here that we have stock price information for around 63 days, or 3 months. When you're working with real-world data for visualizations, Bokeh figures accept a data source from which to draw information that it needs to plot. Now there are different kinds of data sources that you can set up when you're working with pandas DataFrame. The ColumnDataSource is one that you'll use over and over again. The ColumnDataSource allows you to map the column names and the values stored in a column to sequences or arrays, and these sequences or arrays can be used as the source of data for our Bokeh figures. Let's instantiate a ColumnDataSource and pass in that aapl data frame to it. This is the data source for our plots. We'll instantiate a figure by calling the figure function, as usual. The one new property that we have specified here is the x_axis_type set to datetime. Bokeh recognizes datetime as a special type, and it'll adjust its axis accordingly when you specify this property. We can also specify the limits of our y-axis using Range1d. We use the p. line function in order to draw a line plot. The x- and y-coordinates of the line come from our data_source, from the Date column and the AdjClose column from our data_source. Notice that the x and the y values are not actual coordinates, but they are actually keys from our data_source, and the data_source is the pandas DataFrame that we set up to be a

ColumnDataSource, and that's it. This is how Bokeh displays a plot from real-world data. Notice that the x_axis_type is of type datetime because it automatically adjusted and formatted the x-axis tick labels, and you can see the range limits that we've specified on the y-axis. Let's configure our plot even further. We want labels for both the x-, as well as the y-axis. The x-axis should say Date, and the y-axis should say Adj Close. We also want additional configuration for the axis labels, as well as the major ticks. And this is what the resulting plot looks like. You can see that we have an x axis_label that says Date. The y axis_label says Adj Close USD. P. yaxis. axis_line_width makes our y-axis thicker. It's 2 px in width now. The yaxis. axis_label_text_color makes our text blue in color. The yaxis. axis_line_color changes the color of the axis. And the major_label_text_color changes the labels associated with our text. This same Bokeh plot can be displayed with multiple axes as well, especially if you are displaying different pieces of data which have different ranges. The LinearAxis library allows us to define a new axis on the same plot. We want a new y-axis in our example. We use the p. extra_y_ranges attribute in order to specify this axis. We give this axis a specific name. We call it the VolumeAxis, and also specify the range for this y-axis. Now when we set up a line, we specify what axis this line should look at. The y_range_name property, when we call p. line, allows us to indicate what axis this line refers to. This is the VolumeAxis that we just set up. The data that we want to display is the volume of the created Apple stock, so the x-axis remains the same. It is the Date from our data_source. The y-axis is the Volume from our data_source. We now need to explicitly add this new axis to our plot, and we do this using the p. add_layout function. We instantiate a new LinearAxis, and we give it a y_range_name. This is the VolumeAxis name that we had set up earlier. The y_range_name property indicates that it's a new y-axis that we've added, and we want it to display to the right of our plot. Earlier when we had accessed the y-axis property on our figure, we'd assumed that we have a single y-axis. Now that we have one more y-axis, we need to access it in the form of a list. Yaxis of 1 refers to the y-axis at index position 1, which is the second y-axis that we just added, the VolumeAxis. The properties available to configure on this newly added LinearAxis are the same as any other y-axis, and here is how the resulting multi-axes display looks. You can see that we have a new y-axis to the right of our plot. You can take a look at the code that we used to configure this newly added y-axis, and you can see that the axis is displayed based on the configuration settings that we had specified. And this brings us to the very end of our introductory module on Bokeh. We saw that Bokeh was a visualization package, which focused on interactive browser-based visualizations. Bokeh is declarative, which means we specify properties to configure our plots, which makes Bokeh accessible to people with less programming experience. Unlike other data visualization tools, Bokeh does not rely on Matplotlib. It's, in fact, built to be browser based. It renders your plots to JSON, and this JSON is visualized using the BokehJS library. This module focused on the building blocks of Bokeh. In the next module, we'll use Bokeh to build basic, as well as intermediate-level plots.

Building Basic and Intermediate Plots with Bokeh

Module Overview

Hi, and welcome to this module where we'll build basic and intermediate plots with Bokeh. In this module, we'll be very hands-on. In fact, this module is all demos. We'll start off by seeing how we can visualize categorical data using Bokeh where data is present in discrete categories. It might be the case that your data is present in the form of a network. You have nodes and edges which connect those nodes. Bokeh has built-in support that allows you to visualize these network graphs. Bokeh can also work with geographic data. You can use the preconfigured geo tiles available in Bokeh or use Google Maps for your geographical visualizations. In this module, we'll also study annotations in Bokeh, specifically text, as well as other visual annotations.

Visualizing Categorical Data

Bokeh's support for categorical data is very intuitive and easy to use. Let's see how we can visualize categorical data using bar charts, as well as stacked bar charts. The data set that we are going to use for this particular demo refers to the sale of wine. Imagine that you are a liquor store and you have wines from these different countries, and then you have different types of wines as well, such as sparkling, white, as well as red wines. The Quantity column refers to how many liters of these different kinds of wine you sell per year. As you can see from this data set, the Origin and the Type of wine are both categories. This is categorical data. Let's get these categories in the form of an array where every tuple in our list contains an origin and a type of wine from that origin. Categories is a list of tuples where the first field in the tuple refers to the country of origin, and the second field is the type of wine. Bokeh gives us built-in support to visualize data in terms of the categories that we just set up. Let's instantiate a figure here and assign it to the variable p. The x_range of the figure are the categories that we set up. The categories that we have in our data set are not just simple categories. We have an outer level, which refers to the country of origin, that is the grouping of our categories, and then we have an inner level, which refers to the type of wine. The FactorRange function takes all of this into account to generate a range of values for our categorical data. We've also specified a new configuration property here, toolbar_location. Setting this to None will not generate the interactive toolbar for this plot. We want our wine sales to be displayed in the form of vertical bars. The x-axis comprised of the categories that we set up. We have two levels of categories. Make sure you remember that. The height of each of our bars represents the amount of that wine that we've sold, which comes from the Quantity column in our data frame. And here is what our resulting bar graph looks like. The x-axis refers to the categories that we have. The categories are grouped by the outer level, which is the place of origin of the wine, and the inner level is the type of wine. We have one bar for each place and type combination. The y-axis tells us how much of each type of wine we have sold for each place of origin. The width of each of our bars is 0. 9, and the bottom here starts at 0. Let's say you don't like how this figure looks, you can choose to customize it further. The x_range. range_padding property leaves a little gap between the very beginning and the end along your x ranges so that your graph is not flush against either end. Setting the grid_line_color for your xgrid to None removes all of the vertical grid lines. The category labels that we had on the x-axis weren't really displayed clearly. They tended to overlap with one another. We can change this by changing the major_label_orientation of our xaxis to 1. This is the enum value for vertical orientation. The outer grouping by origin is the group. The group_text_color can be configured separately as well. Bokeh offers a number of different colors palettes that you can use to customize how your graph looks. Here we are going to use the Spectral3 palette. This has three different colors. Here are its RGB values. We'll set up a figure to display the same bar chart as before, but this time we want colors on our vertical bars, and this we specify using the color property. The factor_cmap is the function that we use to apply color mapping to categorial data. The categorical data is along the x-axis as specified by the field_name equal to x. We want the different spectral colors to apply to the different types of wine that we display. Not to the countries of origin, but to the types. And you can see from the resulting graph that it now looks a lot more vibrant with the spectral colors. You can see that we applied the colors based on the values of the x-axis, and the palette that we use is the Spectral3 palette, which has 3 different colors. The one thing that we have yet to explain here is the start and end property. This is what applies the colors to the second level of data. That is the type of wine in our data set. Start and end are the slicing indices. We want colors to apply only to the second column, the second category of grouping, which is the type of wine.

Visualizing Categorical Data as Stacked Bars

Let's now get our same wine categorical data to display in the form of vertically stacked bars. For this, we need to structure a new data source. Let's get a list of all the countries from where we have wine available. That is our origin_list. We'll also get the sales of each type of wine from each of these origin countries also present as elements in a list. If you print out the values in the red_sales, whites_sales, and sparkling_sales list, you can see these are a list of quantities of wine sold from each of the origin countries. We'll now create a data source that our Bokeh plot can use. This is a simple dictionary of key value pairs. The keys are Origin, Red, White, and Sparkling, the country of origin and the different types of the wines, and the values are the numeric arrays or list that we set up earlier. Because we want the different types of wines to be displayed in the form of vertically stacked bars, the categories that we want to display on the x-axis are only the country of origin from the origin_list. A vertically stacked bar can be drawn using the vbar_stack function. Let's see what the resulting stack bar representation looks like before we understand what each of these properties configure. The stackers refer to the information that we want to represent in the stacked bar. The stackers are the different types of wine that is sold, red, white, and sparkling. The x-axis refers to the origin of the wine. X is equal to Origin refers to the data_source that we've assigned to the source property. It's a key in our data_source. The values corresponding to that key are the list of origin countries. The remaining properties should be very familiar to you. The width refers to the width of our stacked bars. The source refers to the data_source where we get information about the different types of wine, the sales information, and the colors refers to the colors of our stacked bars, the Spectral3 colors once again. Let's configure this stacked bar graph of categorical data a little more. Instead of using the Spectral3 colors, we want to use our own colors, which we specify as a list of hex values. We also want our plot to display a legend. In order to specify the details for that legend, we need a valuespec. The valuespec is just a list of dictionaries where the key in every dictionary is value, and the corresponding value for that key is the different types of wine, Red, White, and Sparkling in our case. These different types of wine are what we want to represent in our legend. The figure we instantiate is exactly the same way as before. The one change that we've made here is that we have specified some range limits for the y-axis. The x-axis still refers to the list of countries, the origin_list. The configuration settings that we specify for our stacked vertical bars are almost exactly the same as what we've seen before. There are just a few differences here. The data_source continues to the be the data_source that we created earlier. The color is customized. We want to use the wine_colors list that we had set up, and the legend takes in our value specification. We can specify additional properties to configure our legend further. We want the legend to be oriented vertically and located on the top right, and here is our resulting stacked vertical bars. These use our own colors, and we have a legend corresponding to each of our wine types.

Creating Network Graphs

In this demo, we'll use Bokeh to plot a network of nodes and edges. The specification for our nodes and edges are in different files. All our node specifications are in this file called nodes. csv. We have a very simple network here. We have four different nodes, A, B, C, and D, and the corresponding x, y coordinates for these nodes. In order to use this information for our network graph, let's set it up in a dictionary format. Nodes is a dictionary where the keys of the dictionary are the node names, and the values are tuples of the x, y coordinates of the nodes. Here is the nodes dictionary with the keys and the corresponding values. We'll import a bunch of libraries from Bokeh in order to lay out our network graph and to color it the way we wanted to. We'll understand each of these are we use them. Let's instantiate a figure first. There's nothing new here. Bokeh has a special renderer called the GraphRenderer that we can use to visualize our networks. We can simply specify the nodes and the corresponding edges, and the GraphRenderer will take care of the rest of the configuration. The GraphRenderer has to work off our data_source. We can add the list of nodes that we have set up in our nodes dictionary. The keys of the nodes dictionary gives us the names of our nodes, which is just A, B, C, and D. These are now associated with the index key of the data_source. Bokeh layout providers are what renderers use to lay out the visuals. The layout provider that is used to specify the node coordinates in space is the StaticLayoutProvider. At this point in time, the GraphRenderer just works with the StaticLayoutProvider. Pass in the nodes dictionary to the StaticLayoutProvider. The StaticLayoutProvider requires the x, y coordinates of the nodes to be specified in the form of a tuple. Every figure in Bokeh has renderers associated with it. You simply append the GraphRenderer to your figure, and once you display the figure, there you have the points on your network graph. That network graph was kind of boring. It just had the nodes in there. Let's customize these nodes a little bit. We'll add a color property to our data source, and we'll use the Viridis4 palette. And instead of having those simple dots represent the nodes of our network, we'll represent them using ovals. The Oval class is a visual glyph in Bokeh, and we can specify the node_renderer_glyph to be ovals. These ovals can be customized as well. The fill_color for our ovals uses the color property that we have specified in our data_source. Let's append this newly updated GraphRenderer to our figure once again. We'll not display our figure just yet. Let's go ahead and add labels to each of our network nodes. The labels for each of our nodes are available in our nodes data frame. Let's set it up as a ColumnDataSource. Node labels in our network graph can be thought of text annotations on our graph. The LabelSet class can be used to set up text elements, which we use to annotate the regions in our plot. The text that we want to display in our network graph comes from the ColumnDataSource that we had set up earlier. The ColumnDataSource references our nodes data frame, and we want to access the Node column values to set up the text. The x- and y- coordinates refer to the position of where the node text will be located. This is also picked up from our ColumnDataSource. The x and y columns contain the coordinates of our nodes. Add the text annotations that we just set up as another layout to our figure, and we are now ready to display the nodes in our network graph. You can see that the nodes are ovals. Each node is in a different color, and they also have associated labels. The x_offset and the y_offset property in the LabelSet defines the distance of the name nodes from the actual points. This is to prevent our text annotations from overlapping our plot visuals. We are now ready to add edge data to our network graph. We'll read in the edges from the edges. csv file. Every edge is defined as a source destination pair. The GraphRenderer that we had instantiated earlier and that we've used so far, in addition to a node_renderer, has an edge_renderer, which also accepts a data_source. This data_source, it's in the form of a dictionary. The keys of the dictionary are start and end. Start is a list of Source nodes, and end is a list of Destination nodes. And this is all that we need to set up to specify the edges. Add the graph to the renderers on our figure, and go ahead and display our network graph. We now have the nodes, as well as the edges which connect these nodes.

Working with Geographical Maps

Bokeh has built-in tiles for geographical maps. It can also work with Google Maps as well. Let's see how we can map GeoData in Bokeh. Bokeh comes preconfigured with two specific kinds of geographic tiles, the CARTODBPOSITRON and the STAMEN_TERRAIN, and you can import both of these from the bokeh. tile_providers namespace. Geographic data are displayed within a Bokeh figure, as usual. The x_range and the y_range though, refer to latitude and longitude. Positive x values refer to meters East of the 0-degree longitude. Negative values refer to meters West of 0 degrees. Positive values in the y_range refer to meters North of the equator. Negative values refer to meters South of the equator. Let's first work with the CARTODBPOSITRON maps that Bokeh offers built in. You can simply call show on the figure, and that will display a geographical map, as easy as that. You can see that the range positions that we had specified as our x- and y-coordinates cover most of Europe. Let's plot a point at the city of Berlin. We first need to specify the latitude and longitude of the city of Berlin, which is what we see here. For users and GPS systems, the more natural way to specify coordinates is the LAT and the LONG; however, web maps often use a different coordinate system called the Mercator system. The Mercator coordinates are different in that they preserve angles and shapes across regions. Converting from regular LAT/LONG coordinates to Mercator coordinates requires a mathematical computation, and this is what you see here on screen. It's not really necessary for us to understand this conversion. We can simply perform it and print out the Mercator coordinates for Berlin to screen. You can install a library that will allow you to perform this conversion automatically, such as the PyProj. You don't need to perform the mathematical computation yourself. When you use this PyProj library, you simply specify the projections that is your input and the corresponding output that you're interested in. The projections are expressed in the terms of their standards specifications, such as epsg:4326 is for GPS systems. Those are regular LAT/LONG coordinates. We want to convert regular LAT/LONG to Web Mercator, which is the epsg:3857 standard. I'll form this conversion within a simple utility function by calling the transform function within pyproj. Let's convert our Berlin coordinates using this utility function, and we've eliminated all of the complex math that we saw earlier. In our geographic map, let's plot the position of Berlin using the Web Mercator coordinates. Notice how we can use the p. circle glyph on the geographical plot as well. Display the figure, and there is Berlin denoted as a pink circle. The x and y ranges for your geographic plots are, by default, expressed in terms of Mercator coordinates. These may not be very intuitive to the users who view this. Let's instantiate a new figure which displays the x- and y-coordinates in the GPS system. We simply specify x_axis_type and y_axis_type to be mercator. Bokeh will convert these Mercator coordinates to the GPS system. This time, we want to display a different kind of map. We'll use the STAMEN_TERRAIN. We'll, once again, use a deep pink circle to represent the position of Berlin here on this map, and this is what a Stamen Terrain map looks like. This is a map that we use to display the physical features of the geography, such as where water is, where there are mountains, and so on. Observe that the x, y ranges on the map use the more intuitive GPS coordinate system. Bokeh also offers a very easy integration with Google Maps, so if you want to use Google Maps within your plots, you can do so in a very simple and straightforward way. The GMapOptions class is what we use to configure how we want the Google Map to be displayed. The lat/lng property that we pass in to GMapOptions indicate the coordinates of the map center. Google has several map options of which we'll display the roadmap. Other options here are satellite, terrain, and hybrid. Google Maps allow you to configure the zoom levels at which you can view the map. Here, you'll need a little tweaking. Start off with a certain zoom level, see what the map looks like, and then tweak your zoom level to suit your needs. If you want to use Google Maps from within Bokeh, you need to get an API key from Google. This is available using the GCP, or the Google Cloud

Platform, APIs for which you need to set up a GCP account. Creating this account is absolutely free and all you need is a Gmail ID. If you do a Google search for get api key, one of the first links that you'll see is how to get an API key for the Maps JavaScript API. That's exactly what we want. Click through here, notice the URL. You can directly go to this URL in order to get your API key. Click on the GET STARTED button here, that is prominently displayed, and this will take you to the Maps console on the Google Cloud Platform. Now this will walk you through all of the steps that you need to get your Maps API key. I'm going to get the API key for Google Maps, Routes, as well as Places. Now the steps involved here might be a little different based on whether you're logged in, whether you already have a GCP account. Google will walk you through these steps. I'm already logged in to my GCP account here, and I'm asked to choose a project with which to associate my API key. When you work on the Google Cloud Platform, all the resources that you instantiate are within a project, and the same is true for the Maps API key. I already have some projects preconfigured. If you don't, Google will walk you through these steps, so you can either create a new project or associate your key with an existing project. That's what I choose to do here. Because we're not really planning to make a huge number of requests to Google Maps, you should not really need to have to pay anything in order to use the Google Maps API as specified in this demo. Once your API key has been created, you can simply copy it over, and use it from within your Jupyter Notebook. In order to use Google Maps, we instantiate a figure using the gmap function, as you see on screen. This takes in the google_api_key as an option. This also accepts the GMapOptions that you had set up earlier as further configuration of the Google Maps to display within Bokeh. Within the Google Map that we display here, we'll highlight three different cities, Berlin, Frankfurt, and Munich, and we're also using GPS coordinates. And here is the resulting Google map within Bokeh with our three cities highlighted in deep pink.

Using Bokeh's Sample Dataset

If you are a student learning Bokeh, it's often useful to have sample data which you can use within your visualizations. Now, you've spent a lot of time hunting around on the net for sample data, but Bokeh has sample data available for you from within Bokeh itself. You can simply download it with a command. Let's see how. Sample data in Bokeh can be downloaded using a Python module, the bokeh. sampledata, which is what we are going to use here. Simply call the download function on sampledata, and this will begin downloading the data to some local folder on your machine. You need to be a little patient as the download process completes. Make a note of the directory where the downloaded data is stored by default. This is within the. bokeh/data directory, and here is a sample of the data that's available. There is stock data for tech stock such as Google, IBM, Microsoft. There is fertility data, unemployment data, US counties data, and airports data. The airports data is what we'll work with. We'll go ahead and import the other modules from Bokeh that we need, and we'll use output_notebook to plot within our Jupyter Notebook. We'll work with the same geographical tiles that we have gotten familiar with now. We've already seen this in last demo. There's no explanation needed here. The CARTODBPOSITRON tile is what we'll use. I'm going to copy over this airports. csv file and load it into the datasets folder under my current working directory. That's where I have the rest of the data sets that I've used in this particular course. Remember if you're working on a Mac or a Linux machine, you can run shell commands using the exclamation or the bang at the beginning of your command. Let me quickly check what's there within the datasets folder. Airports. csv has been successfully copied over. I'm now ready to read in this data set, which I'll do using the pandas. read_csv function. This will store this data in a pandas DataFrame, and I can explore what's within here by calling the head function on this df, and you can see the columns of data that we have available. The airports. csv file contains the airport locations of all the airports within the United States, and this is what I want to plot within my geographical map. So I'm going to extract only the values for column Name, Latitude, and Longitude. My airports_df data frame now only contains the data that I'm interested in, the name of the airport, latitude, and longitude. Instantiate a figure and specify the x_range and y_range of your geographical map. This is what will tell Bokeh what portion of the map you want to display by default. I have roughly chosen the area over the United States. Add the CARTODBPOSITRON tile, and go ahead and display the figure. This is our map. We haven't plotted any points on it yet. That's what we'll do now. The lat/lon data in our data set is expressed in GPS coordinates. We need to convert it to Web Mercator so that we can display it on our Bokeh map, and we do this exactly like we did in the last demo using a function called toWebMerc. I'm going to add a new column to my airports data frame called Mercator, which holds these Mercator coordinates. I use the df. apply function and pass in a lambda to this function, which will call the toWebMerc function on the Latitude and the Longitude, and then create this new column called Mercator. At this point, we have the coordinates in the right form in our pandas DataFrame; however, both of the Mercator coordinates are in the same column. Let's extract these to individual columns. This will contain the Mercator_x coordinates and the Mercator_y coordinates. Extracting the x-, as well as the y-coordinates to individual columns in our pandas DataFrame makes it easy for us to feed in this data when we plot our geographical map. We'll display all of the airports in the United States using circles. Pass in the x and y Mercator coordinates. We specify that the size of the circle is 10, and it is green in color, and it's slightly transparent. Go ahead and display the map, and here it is. All of the airports plotted on our map of the United States. I can't really see clearly here, so I'm going to zoom out, and you can see the airports in Hawaii, Alaska, and the continental US. If you want to play with data sets beyond what is available in this course, the Bokeh sample data set is a great resource.

Creating and Customizing Annotations

When you're setting up a visualization, it's often useful to add additional annotations on your graph explaining what exactly is going on. Here we'll see examples of text, line, as well as box annotations. We'll work with a slightly different data set here. The corn_prices. csv file contains dates and the price of corn at that particular day on a weekly basis. You can see that the dates are all one week apart. The Date that we read in from a CSV file is in the form of a string. We can convert this to a Python datetime format by calling pd. to_datetime on this column. We now have a data frame where the Date is represented in the form of a string, as well as a Python ______. We'll run the describe command on the pandas DataFrame in order to get a quick sense of the data, specifically the minimum and maximum prices represented. In order to use this data set as a data_source for our Bokeh plots, we instantiate it as a ColumnDataSource. We'll first see how we can set up tooltips as an interactive plot element. Tooltips are specified in a list format. The list gives you all of the information that we want to display within our tooltip. Here we want to display the date, as well as the price of corn. Each piece of information is represented as a tuple of field_title and a reference to the datasource. Every record in a pandas DataFrame is associated with an index. Let's find the indices of those records for which the price of corn is at its maximum and minimum, and we'll print this out to screen. We'll now instantiate the figure in order to view how corn prices change. On the x-axis, we want to specify dates, and we can specify a range of dates using the dt function. We want to view corn prices for dates between 1 January 2015 and 30 November 2017. The x-axis displays dates. The x_axis_type is datetime. Assign the tooltips property to the figure. The information displayed in the tooltips will be picked up from the data_source for the actual plot. Use the p. line function to plot the line representing corn prices. The source of the data is the ColumnDataSource that we had set up earlier. The x values contain the FormattedDate, and the y values, the price of corn. The actual values used to display this yellow line come from the ColumnDataSource. If you hover over the line, you'll find that Bokeh automatically displays tooltips. The format of the tooltip is the configuration that we had set up earlier, and the actual price and date information is picked up from the data_source. We'll now make this graph more user-friendly by adding annotations. Let's find the dates at which the price of corn was at its minimum and maximum. We'll print these out to screen. In addition to the dates at which the prices of corn were at its minimum and maximum, let's find the actual minimum and maxprice and print those out to screen. We can then use a circle annotation to annotate those regions of the plot at which the corn prices were at its minimum and maximum. Text annotations on your Bokeh plots can be set up using a LabelSet. Label sets can be configured with a data_source. Here, we want to label the minimum and maximum corn price using text labels. We set up our data source dictionary with the x-coordinate, the y-coordinate, and the text that we want displayed in the label. We'll instantiate this dictionary as a ColumnDataSource and use this data source to provide the text annotations for our plot. We can now set up our LabelSet. The x and y positions of the labels come from the x and y columns from the ColumnDataSource, and the text comes from the text column of our label_source. Add the LabelSet to our figure layout, and go ahead and display our plot. You can see the minimum and maximum prices of corn now have text annotations on our graph. Line annotations on our Bokeh plots can be set up using the Span class. Spans can span the entire width or the height of the plot. Let's have a line, which is a span, indicate the mean price of corn across all of the days for which we have information. We need to calculate this average corn price first, which we do by calling the np. mean function. Let's configure and customize a horizontal Span to represent this mean price information. Just like the Span class, you can use the Slope class if you want a slanting line. Spans are lines which have just a single dimension, either the width or height. Specifying dimensions equal to width sets up a horizontal line. The location of this horizontal line represents the average price of corn. This span can be added as a renderer to your figure, and here is what the resulting visualization looks like. The horizontal green dashed line at the center of our plot represents the average price of corn. Just like the line annotation, you can have a box annotation as well. Let's say in addition to the average corn price, you also want to represent the standard deviation of corn prices. You can use a box annotation for that. We calculate the standard deviation first using np. std from the NumPy library. You can instantiate the BoxAnnotation class to represent the standard deviation in the form of a box. You need to specify the bottom, as well as the top edges of your box here. The bottom shows average minus standard deviation, and the top shows average plus standard deviation. Add the BoxAnnotation as a layout to the figure that you already set up, and now when you call show p, we have the standard deviation represented in this plot as well. The top, as well as the bottom edges of this box are one standard deviation away from the mean. The dotted green line represents the mean as it did before. The box itself can be customized. It is transparent and filled in the green color. And this brings us to the very end of this module where we built both basic, as well as intermediate plots in Bokeh. We saw how we could visualize categorical data, and we also saw the setup of a network graph in Bokeh. We saw that Bokeh has built-in functionality for geographical plots. In addition, Bokeh also integrates with the Google Maps API. We also saw how we could make our visualizations more interactive and more interesting by adding annotations. These annotations can be in the form of text, lines, and boxes. In the next module, we'll focus on how we can add interactivity to our Bokeh plots. We can configure the toolbar and tooltips, and also make the legend interactive. We'll also see how we can run the Bokeh web server, which allows you to display interactive plots on the web browser.

Adding Interactive Features to Bokeh Apps

Module Overview

Hi, and welcome to this module where we'll see how we can add interactive features to our Bokeh applications. Now, we've seen that visualizations in Bokeh, by default, already come configured with some interactive features. Here, we'll see how we can set up interactive legends to work with your visuals. We'll integrate tooltips and customize the toolbar that we see with every Bokeh plot. We'll also use plot tools in visualizations. And finally, we'll see how we can get a Bokeh server up and running. A Bokeh server allows you to view your plot in a browser window and interact with it using UI widgets. This browser object is kept in sync with the Python model at the back end.

Configuring Interactive Legends

In the past, we've displayed Bokeh plots with legends. In this demo, we'll see how we can make this legend interactive. For this demo, we'll work with a data set that we are familiar with, Apple stock price information, which we get from the AAPL. csv file. We read it in as a pandas DataFrame, and when you view this data frame, you see that we have date, open, high, low, close, adjusted close, and volume information for Apple stock across a period of 3 months. The date information that we read in from the CSV file is in the form of strings. We convert it to the datetime Python format so that we can represent it on the x-axis of our visualization. We'll now instantiate a figure that's going to hold 2 different kinds of information, Apple high and low prices for the period of these 3 months. The x-axis is a datetime axis. We'll use the p. line function to track the closing prices of Apple stock. The x-axis carries the date information. The y-axis tracks the high prices for Apple stock. The legend associated with this line plot will have the label High. Similarly, we set up a separate line plot tracking the low prices of Apple stock. The legend associated with this plot will have the label Low. Display the figure, and here is what the resulting visualization looks like. We have specified different colors for each of these lines. High prices are in green, low prices are in pink, and we have a legend to go with each of these. Making the legend associated with the plot interactive is just a matter of setting the right properties on that legend. We can set the location of the legend. We want it to be displayed on the bottom right. We also set the click_policy to be hide. The legend. click_policy will allow us to click on the individual elements of the legend in order to show and hide the data corresponding to that part of the legend. Let's display the figure, once again, using show p. You can see that the location of the legend has changed. It's now at the bottom right of the plot. The legend also has a click_policy set. We can now try this out. Now, hover over the legend, and click on the High label. Clicking on High will hide the data that corresponds to that part of the legend. You can see the line tracking Apple's high price of the day has disappeared. If you click on this once again, the line will reappear. Try the same thing with the Low label on the legend. Click on the Low price and the corresponding line will disappear from our graph. If you click on the Low price again, the corresponding line will reappear. Our legend is interactive. Let's change how this legend is interactive. Instead of hiding the line when we click on it, we want to simply mute the line, or rather show it in a muted color. Here is the line tracking Apple's high prices. We've added two additional properties here, muted_color and muted_alpha. And here is the second line tracking Apple's low prices. This too has a muted_color and muted_alpha. Both of these lines have corresponding legends. Once we have specified these muted properties, you can have the legend. click_policy set to mute. Muting a line obscures that line so that other lines in the plot become more prominent. Display the plot, and let's see how the mute policy works for the legend. Hover over High, and then click on High. Instead of the line disappearing entirely, you can see that the line is more translucent. Its opacity has changed. It has been muted. Click on High once again, and that will restore that line. Now go ahead and click on the Low part of the legend, and you can see that the Low line is now muted. Another click here will restore the line to its original color.

Introducing Plot Tools

The Bokeh library comes with a number of interactive plot tools that can be used to report information to change plot parameters in an interactive manner, and also to add, edit, or delete glyphs from plots. Bokeh plot tools serve to enhance the basic interactive functionality offered by default on all Bokeh plots. Bokeh plot tools can be grouped into four basic categories. The first of these are gesture-based tools that respond to single gestures, such as a pan movement. You can have action-based tools, which are immediate or modal operations. These are only activated when the corresponding button on the toolbar is pressed, such as hover. We'll see an example of hover in just a bit. Inspector tools are those that are used for reporting information. Tooltips are an example of inspector tools, or the crosshair tool is another example. Edit tools are far more sophisticated, and they can add, delete, or modify the glyphs that are present on a plot. We'll see one example of this in our demo. Different tools for different purposes. If you're interested in supporting gestures within your visualization, you might explore the Pan/Drag, Click/Tap, or the Scroll/Pinch tools that are available. Examples of action tools are tools that can be used to undo or redo your operation or reset your graph to the previous state. Tooltips are examples of inspector tools used to report information. Another example is the crosshair tool, which draws a crosshair annotation over the plot where your mouse is centered. Bokeh also offers a huge variety of edit tools, which are very sophisticated and interesting to work with. The PointDrawTool is an example. A PolyDrawTool allows you to draw shapes on your graph. Edit tools actually update the information that is displayed in your plot.

Tooltips, Toolbar, and the Point Draw Tool

In this demo, we'll add custom tooltips and customize our toolbar. We'll also work with the PointDraw plot tool. The data set that we'll use in this demo is the automobile price prediction data set, which is typically a machine learning data set that you use with regression models. This data set is available as a CSV file in imports-85. data. The records in this data set contains information about different cars, different makes, models. It has the body style of the car, the engine location, the wheelbase, the engine size, horsepower, and so on. At the very last column, you can see that the car price is available as well. This data set contains several records with missing information. We'll use the dropna function on the pandas DataFrame in order to drop those records. We want to use the price and horsepower information in our Bokeh plot. These have been read in, in the string format. We'll convert these to numeric values. This is the data frame that contains the source of the data that we want to plot instantiated as a ColumnDataSource, which we can use within Bokeh. We'll display some tooltip information on our plot. When we hover a particular car, we display the make, the price, and the horsepower. Tooltip information is specified as a list of tuples where the first field in the tuple is the label, and the second field is the data that we want to fill in and display. Tooltip data is picked up from the data source associated with our Bokeh plot. Instantiate a simple figure and specify the tooltips property pointing to the tooltips structure that we set up. We'll set up a simple scatter plot, which has circles. We'll plot the price of the car on the x-axis and the horsepower on the y-axis. The source of the data is the ColumnDataSource that we had set up earlier. And here is our resulting scatter plot visualization, horsepower versus price. Now if you click on the Hover button in our toolbar, this can be used to toggle whether tooltips will be displayed or not. Now if you hover over the individual data points, you'll get to see the make, price, and the horsepower of that particular car. If you don't want tooltips displayed, you can click on the Hover button to turn off tooltips. Now if you hover over the individual data points, no tooltips will be displayed. We can make the display of our tooltips nicer by formatting our tooltips. We are displaying price information here. Let's say we want price to be displayed in a specific format. We want the dollar sign before every price. Include the dollar sign before the at the rate sign, which indicates a reference into our data source. The format of the price is specified within curly braces. We want 2 decimal places, and we want commas after every 3 digits. Now that we have specified a new structure and formatting for our tooltip, instantiate a new figure with the tooltips set to the structure and draw circles to represent cars, mapping their price versus their horsepower. Click on the Hover button so you turn tooltips on and hover over a specific data point. You'll now see that the price is now nicely formatted with commas and a dollar sign. The toolbar with buttons is, by default, displayed to the right of your plot. We can configure its location. We can set it to be below the plot, as you see on screen. Bokeh shows some buttons by default within your toolbar. You can configure the toolbar to show only the buttons which is of interest to you. Instantiate a new figure with tooltips and the toolbar_location below the plot. Toolbar_sticky is set to False moves the toolbar to live outside the plot dimensions. By default, the space occupied by the toolbar is within your plot. If you want your plot to expand a little bit, locate your toolbar outside by setting toolbar_sticky to False. Configure your toolbar by specifying the buttons you want displayed within it. Point it to the toolbox list that you created earlier. Plot the same scatter plot of price versus horsepower. This is the same thing that we've before, but notice that the toolbox is now different. The order and the buttons are the ones that we have specified in our toolbox list. In this next example, we'll see how we can use the PointDrawTool. This is one of the edit plot tools offered by Bokeh. This allows us to update our plots on the fly. We'll instantiate the figure with the same configuration parameters that we saw earlier. We have the toolbar at the bottom, we have tooltips, and we draw circles to represent the cars. We plot the price versus the horsepower of the cars. All of the circle glyphs that we draw are stored in the circles variable. We assign p. circle to the circles variable here. Instantiate the PointDrawTool. The PointDrawTool takes in renderers, and the renderers are the circles that we created in our scatter plot by calling p. circle. The PointDrawTool makes the plot points interactive. You can add and delete plot points, and you can move them around. Use the add_tools function on your figure in order to add in this PointDrawTool and make it part of your visualization. The visualization looks the same as before, but notice the toolbox at the bottom. We have this extra element here associated with the PointDrawTool. Go ahead and click on this element, and you can now add points to this plot. Just click on any point within your visualization, and the new circle will be added at that point. You've added a new point, but there isn't really a car associated with that point, so if you hover over that point, you see that the Make and Style values are question marks. The only values that Bokeh knows about are the values represented on the x- and y-axis, and only those are populated here. In order to delete a point, just click on that point and hit Delete or Backspace. The point will be deleted. With the PointDrawTool, you can also drag an existing point to change its location on the graph if you want to change its price or horsepower. I'm going to move this to another location. The price and horsepower details have been updated.

Introducing the Bokeh Server

Bokeh offers this very interesting application called the Bokeh server that allows you to keep your browser interactions with your visualizations in sync with Python code. Now when we had spoken about Bokeh earlier, we had spoken about how Bokeh Python produces these visualizations, which are then serialized as JSON. This JSON is then read by BokehJS and rendered on the browser. This decoupled design allows Bokeh to work with different languages, such as R and Scala, in addition to Python. The Bokeh server gives you a way to keep the model objects in Python in sync with the visualization that you see on the browser by keeping the JavaScript code associated with the visuals in sync with the Python back end. Separating the model from the view in this way, allows Bokeh to work with very large data sets and even streaming data. The main purpose of the Bokeh server is to synchronize the data that is manipulated in Python with the visualization that is specified within a browser. If you've ever built any user interface before, you know that the best way to have a robust, maintainable code is to follow the MVC pattern. The Python code here is the model, which contains the data. The browser forms the view, and the Bokeh server is the controller in that MVC pattern. All the manipulation to the data source, which updates the visualization, is performed in Python. The browser simply renders the view and provides interactive controls. The advantage of setting up such an MVC pattern as provided by the Bokeh server is that it allows your visualization to respond to UI and tool events generated within a browser, and allows you to respond to browser widgets, such as sliders and selectors. If there are updates to your data on the server side, let's say new streaming data comes in, Python can handle this data, update the data source, and these updates will be reflected within the browser. Keeping the browser and Python code in sync in this manner allows us to use asynchronous callbacks and also work with streaming data. Let's quickly understand the basic architectural setup of the Bokeh server. We have the Bokeh server running on your machine. This is what will host the Python code, which you'll use to build up your visualization. The visualizations generated by this Python code are Bokeh models. There is a single back end to the Bokeh server that hosts the Python code, which holds the models. There can be multiple views of these models. You can have multiple views, one in Client Browser A and another in Client Browser B. The Python code generates models, which are called documents, and each of these documents is kept in sync with the client browser that displays these documents. The Bokeh server's job is to ensure that the models, as seen by the Python code and your client browser, are always in sync.

Setting up the Python Code

In this demo, we'll see how we can get a Bokeh server up and running and build an interactive application that is rendered on the browser. We start off in our homepage here. This our current working directory. I have a webapp folder under here. This is what contains the HTML file where I'm going to render my plot. My HTML file already contains some data within it. I'm going to add my plot and visualization and other control widgets to this file. The plot that you're going to visualization is going to display birthrate versus GDP information for all countries in the world. This demo has been inspired by the sample demo present on Bokeh's GitHub repo. Let's switch over to our Jupyter Notebook called main. This is where we'll write the Python code that will be hosted by our Bokeh server application. The location of this Python Notebook is a little different. It is in the webapp directory, so in order to read in the CSV file which contains the GDP and birthrate information of the countries around the world, I need to navigate up one folder using the dot dot to go to the directory that we've been working on so far, and then into datasets folder in that directory. If you print out the contents of this file, you'll see that it contains a lot of information of all countries around the world. Of all of these columns, we are only interested in the per capita GDP and the Birthrate. Running the describe function on this data frame will give us quick information on this data. You can see that there are a total of 227 records. Some of these columns have information missing, which is why the counts might be a little different. Let's clean up our data a little bit. The Country and the Region column have some whitespaces, trailing and leading whitespaces, which we strip out. We'll now extract information about all of the regions that are available within our data set into a separate regions list. We also append to this list a special region called ALL, denoting all of the regions together. Here's what the resulting list of regions looks like. We'll set up a ColumnDataSource, which will hold the data that we want to visualize within our interactive plot. We'll initially set it up to be empty. We'll populate it later. In fact, the data that we populate within this ColumnDataSource will vary based on the interactions that the user has with the UI widgets on the browser. Any time you're setting up a plot, it's always useful to have tooltips, which give additional information to the user. Here these tooltips will show us Country, GDP per capita, and Birthrate information for every country plotted. Create a figure as you would normally in order to plot the birthrate information versus the GDP per capita for all of the countries for which we have information. We'll use a scatter plot representation with circles representing different countries. The x values come from the x column. Y values come from the y column of our source. The source here is the ColumnDataSource that we had instantiated before. Remember that at this point it is completely empty. It hasn't been populated yet. The next step is for us to set up a helper function. This helper function called select_countries will allow us to choose the countries that we want to display based on certain values that the user has selected using the browser UI. This is the function that will be called each time the user manipulates the UI widget to update the countries that he wants to view. We first set the selection to all of the countries that are available in our data frame. Countries is our data frame. We then select the region and the minimum GDP value as chosen by the user. These values are available in the UI widgets that can be manipulated by the user. The region_selector is a selection widget, which is our drop-down, and the gdp_slider is a slider that allows you to specify the minimum GDP value. Based on the values that the user has specified, we can now filter out the records in our data frame. If the region selected is not equal to ALL, only select those countries which fall within that specific region. Once we have the records for the countries in that region, we perform a second filter operation. Only select those records which have countries above the minimum GDP value specified by the user. If you want to debug this function, you can print this information out to screen. We don't really need that here, and then return the list of selected countries. We now define a second helper function called update. This is the function that is called to update the data that is displayed in your plot. Each time the user interacts with a UI widget, update is called, which then selects the correct countries and updates your data source. The update function calls the select_countries function that we just wrote. This will give us the records of all the countries that the user is interested in filtered by region and minimum GDP value. We then use the records of the selected countries in order to populate the ColumnDataSource that we had set up earlier. We update the source. data field to only hold those values that have been selected by the user. All of the fields in our ColumnDataSource have to be updated using the selected records.

Configuring UI Widgets and Change Listeners

With our helper function set up, we are now ready to set up the widgets that the user interacts with. We are going to use the Slider, as well as the Select widgets. These are interactive components, which provide a front-end user interface to a visualization rendering that on the browser. We want to display our visualization in the homepage. html file that we saw earlier. It already had some content. We can extract all of the content that was present within homepage. html into a div. So whatever the current HTML present in this file, which is outside of the plot that we are going to set up, has been extracted using this command and has been assigned as a div to the homepage variable. Let's set up the first of our control widgets, the region_selector. This will display as a drop-down within our browser window. This allows us to select a region, and we'll display only those countries which fall in that region within our plot. The list of options available in this drop-down is from the regions list, which we sort before we apply here, and the default value is set to ALL. We want to be able to handle events on this widget. We set up an on_change handler, which will call the update function to update the data source whenever the region selector is updated. The next interactive widget is the Slider, which represents the minimum GDP value. Only those countries with GDPs above this minimum value will be displayed in our plot. The start and end properties of the Slider gives us the range. The range is from 0 to 56000 US dollars. The value gives us the current value, and the step shows us the granularity of the Slider. The Slider represents the Minimum GDP per capita. That's the title. We add an on_change event to the gdp_slider as well. We call the update helper function that we had set up before each time the user manipulates the slider. We now have all of the building blocks ready. It's now time for us to layout our plot, as well as the widgets used to control that plot. UI widgets always go within a widgetbox, so instantiate a widgetbox and pass in the region_selector, as well as the gdp_slider. We're now ready to use the layout function to lay out our plot. Our plot will first contain the homepage, then the UI widgets, and then the actual figure. We call the update function in order to populate our ColumnDataSource for our visualization. The initial set up is set. We have all countries displayed on our plot. If you want to perform a quick check to see whether your plot layout looks okay, you can display the plot_layout here, right here within your Jupyter Notebook. The plot interactions will not work here though, and that's the warning that you see here on screen. This is how the HTML page will look when rendered on your browser. We first have the data that was originally present in our HTML file. We then have the UI widgets, and then our plot. Remember the event handlers haven't been connected up. When you display the plot within your Jupyter Notebook, this is meant to be run on a Bokeh server, so you can try changing the regions and try moving the GDP sliders, and you'll find that your visualization is not updated. Nothing changes. Our changes are not being propagated to the back end. In order to display our visualization and everything else that we've set up so far, the glyphs, the data sources, etc., we need to add our model to a document. The current document can be accessed using the curdoc function. This is required in order to display the plot on a web page.

Running the Bokeh Server

In order to run this visualization within a Bokeh server, we need a Python file. Click on File here and export this Python Notebook as a. py file. This will save the Python code that you have written within this Jupyter Notebook into a main. py file. This is currently in my Downloads folder. I'm going to switch over to my terminal window. I am under the webapp directory. If you run l-s here, you'll see that we have the homepage. html and the main. ipynb. I'm going to copy over the main. py file, which is currently in my Downloads folder. Now that I have my Python code within the current working directory, I'm ready to run my Bokeh server. You can simply call bokeh serve --show and specify the current working directory by the dot, and run your server right here. The Bokeh server will automatically open up a new browser window with the plot that you had set up. You can hover over your visualization and see that the tooltips that you customized are available. The UI widgets, the selector, and the slider are present as well. Let's use this drop-down to filter the countries by region. I'm going to choose EASTERN EUROPE here, and you can see, immediately my plot has been updated to display only countries from within Eastern Europe. You can filter by other regions as well. These UI widgets that you have set up, have JavaScript handlers, which talk to your Python back end and update the data source in order to reflect your filters. You can try and use the slider as well. The GDP slider can be moved in order to filter the countries that you view based on their GDP. The updates that we make to our UI, update the data source in the Python code that we've written. We've built an interactive web application and ran it using the Bokeh server.

Summary and Further Study

And with this last demo, we come to the very end of this module and to the end of this course on Bokeh. In this module, we focused on the interactive features that Bokeh has to offer. We saw how we could make our legends interactive and clickable. We also saw how we can integrate tooltips within our plot. We learned to customize the toolbar, and we also used the PointDraw plot tool within our visualizations. We rounded this module off by running the Bokeh server to display an interactive plot. Updates that we made via UI widgets on our browser were reflected in our back end source. If you're interested in data analysis and visualization, here are number of courses for you to explore on Pluralsight: Building Data Visualizations Using Matplotlib, Visualizing Statistical Data Using Seaborn, and Building Data Visualization Using Plotly. We'll expose you to the other Python packages available. And that's it from me here today. Thank you very much for listening.

Course author

Janani Ravi

Janani has a Masters degree from Stanford and worked for 7+ years at Google. She was one of the original engineers on Google Docs and holds 4 patents for its real-time collaborative editing...

Course info

Level	Beginner
Rating	★★★★⯪ (11)
My rating	★★★★★
Duration	1h 47m
Released	11 Sep 2018

Share course