Module 7
Video Transcripts

Video 1: Introduction to Matplotlib (6:29)

Welcome. Today we will be talking about getting started with Matplotlib. First, we'll talk about the importance of data visualization, then we'll talk about an introduction to Matplotlib. We'll go through where to go for some help if you want it with Matplotlib. We'll talk about the basic visualization concepts, which are figures and axes. We'll talk about one of the most important modules within the library called PyPlot. And then we'll talk about additional packages for data visualization within Python. So first of all, the importance of data visualization for data science. So, we know within data science we work with a lot of data. And so, in order to be able to work with that data, sometimes you want to look for things like patterns, trends, or correlations. Data visualization is an ideal way to be able to do that. And so, these visualization libraries allow us to be able to see our data and then visually look for things like patterns, outliers, or trends. So, that's a really important part of what we want to be able to do as a data scientist.

So, Matplotlib is just one of those libraries. So, sometimes it goes by the acronym MPL. And really, what the library's for is it allows interactive cross-platform control of figures and plots. Which we'll get to in a minute. It makes it easy to produce static raster or vector graphics without the need for GUIs. And if you've worked with GUIs before or programming languages, sometimes it can be very difficult. So, let's talk a little bit about where to go for help. So, where can you go? I included a number of different links. So, you can click on any one of these links if you want to look at more detail what Matplotlib is and does and explore some of the features that it has available. So, within Matplotlib, what are the core data visualization concepts? So, the first one is called the figure.

The figure is really the canvas on which everything is drawn within the Python library. So, within the figure, you can have what are called subplots or axes. They're basically the same terminology. It's basically the same thing. So, the axes and subplots, you can have multiple within a figure. So, within those axes or subplots, you can then associate with it X axis and Y axis. So, you can see here now how we can plot two-dimensional objects directly within the axes all within one single figure. It doesn't just limit us to two-dimensional drawing. We can actually go three-dimensional. We don't really talk about it here but there is a Z axis as well. So, we can go 3-D. We'll talk about it in a later session. One of the most frequently used modules within Matplotlib is called PyPlot.

PyPlot contains many of the functions that we want to use for data visualization around things like graphs and charts. So, we'll get to a few of those in a minute. So, first let's look at a simple example. So, in this example, I just want to be able to draw a series of points on a plot. And so those points will be relatively simplistic, and I just want to draw a straight line. And then I notice there's a couple different things. oops. So, here I want to first include my library. I want to define my plot. And then I want to give it a Y label. And then just show the diagram. Relatively simple. And that's what the diagram would look like. So here, if you remember earlier, we're using a lot of the different constructs. So, plot actually plots on a figure. It includes an axis.

And it gives us the ability to label our axes directly within the subplot of the axes. And then the show command is really what you want in order to be able to display. So, those are some of the basic things that you need to know in order to be able to plot within Matplotlib. So, let's move forward. So, if we wanted to look at some of the common usages of plot so the function plot takes in, as we saw earlier, X and Y values and then a number of different formatting strings. And we'll get to the formatting strings, because that's how we can make our diagrams look really interesting. So, let's take another example. So here I want to include NumPy, just to generate some data. And then I'm going to use my Matplotlib library with PyPlot. And then here what I want to do is basically just take and evenly sample set of data. And then what I want to do is I want to plot that. And so, here I'm starting to use some of the different formatting constructs. So, you'll notice here I'm using things that are called "color." So, the color objects are being defined by the text here that's included in quotes. So, basically, I'm using red circles, blue squares, and green up cerets.

Then I'm going to plot them on my axis, set my axis, and then I'm going to show what this would look like. So, let's take a look. So here, relatively simple again. But basically, I'm just taking a random set of data and then what I'm doing here is I'm creating a function to be able to display my X and Y coordinates. And then I'm changing the colors and the shapes of what I'm plotting. So, that's an interesting function as well. So, you can see how simple it is to use Matplotlib PyPlot libraries. So, next I want to talk about what some of the useful methods are that we want to be able to use within PyPlot. So, some of the useful methods are text, where we define text that we want to be able to put on our plot. The labels that we already saw.

The title that we may include on our graphs. And then there's also something called "clear," where we can remove all the plots from a particular axis. So, some of those functions will come in handy. So, make sure that you look through them. And also, you might want to go online to some of the links that I provided.

So, in wrapping up, there's lots of specialized functions that you can use. We only just touched on the beginning of it. But there's plenty that you could do. And if you check out the Matplotlib.org site, you can see plenty of examples that will give you some ideas for ways that you can create interesting graphics and plots. And one final note, there are a series of additional visualization packages other than Matplotlib and PyPlot. There are things like Pandas, Seaborn, Ggplot, Pygal, Plotly. So, I included a number of links here too so that you can check out some of those links if you want to learn more. Thank you for your time.

****

Video 2: Simple Line Plots (4:25)

Welcome. Today we will be talking about simple line plots in Python. From an outline perspective, we'll first go through a definition of what line plots are, then we'll talk about a data set that we're going to create that we could use for plotting some of our line plots. We'll through single series line plots, multi-series line plots, and then we'll wrap up. So, what is a line plot? A line plot is a type of chart that displays information as a series of points, which are called markers, and they're connected by straight line segments. Which you're all very familiar with what they are. The X axis requires a quantitative value. The Y axis must be numbers that can be interpreted or formatted as decimals, percentages, or currencies.

So, if we look at an example of this, what some of the basic components are, we have our Y axis, our X axis, with labels of course so we can determine what their values are. But when we plot our points, or each one of the points associated here, we connect them with line segments, and that gives us our line chart graphic. Next, we're going to define the data set for our line plot. So, here I want to define a series of X and Y values so that I can plot it. So, I'm going to create a simple sine curve with my X and Y values. And I'm using NumPy and the math function that allows me to be able to calculate that. So, Numpy arrange allows me to define my X values and then it also allows me to create a set of Y values based on those X values through a sine function. It's relatively simple, so let's just go ahead and run that. So, the next thing I want to do is I want to plot that. So, it should be relatively easy for us to go ahead and plot that. So, all I want to do is plot my X and Y functions. So, I'm including PyPlot from Matplotlib.

I'm defining a plot with X and Y and then I just want to show or display that particular graph. So, that's relatively easy. So, that's pretty simple. So, I want to actually add a little bit of window dressing to make it look a little bit nicer. So now what I can do is I can add in some labels. So, I can say my X label is equal to angle, my Y label is equal to sine, and give it a title and then go ahead and plot it again. So now I've got a little bit of dressing. I've put a couple labels on and then I've labeled the overall chart. Next, I went to talk about multiple series line plots. So, in this particular case, I can graph or plot multiple series on the same plot. So, how can I do that within Matplotlib? So, the way I'm going to do that is I'm going to define my X and Y axes. I'm going to define my values. And then here I'm going to use a cosine function and incorporate that along with my sine function so that I can actually plot both. So here I'm going to use similar type labels that we used before. And then I'm going to actually give this a legend, because now I want to be able to show both the fact that I'm plotting my sine and cosine functions. So, I want to be able to distinguish between the two of them. So, let's go ahead and take a look at what that would look like. On this potential case, I can see now that I've got a sine wave and a cosine wave and I've incorporated a legend. So, it's relatively simple for me now to also plot both, so I have a multi-series line plot with a legend and my labels. So, this is a pretty nice graph.

So in wrapping up, we went through some of the examples which were associated with the ability to be able to not just define data in the beginning but also to use that data and plot it as a line plot with a single series. Then we talked through how you can use multi-series line plots and incorporate labels so that your graph looks really nice, but you're also able to distinguish between the different line plots that you may include on a multi-series line plot. Thanks for your attendance.

****

Video 3: Bar Plots (04:59)

Welcome. Today we're going to be talking about bar plots or graphs or charts. I'll be using the terms interchangeably. From an outline perspective, we'll be talking about the bar function and then we'll go through a couple examples. The first example will be a standard bar plot and the second will be a multi-series bar plot. So, let's take a quick look at this. If we look at what a graph is composed of, our bar graph is composed of, we can see that it's really just a series of bars. And the bars represent the data. So, the data has a height, which is our Y component which is going to be a magnitude on the Y axis.

And then on the X axis, we're going to have the series of the data or basically the list items that we're going to use in order to be able to show which are the different components that we want to be able to display. The simple format of a bar function looks like this. We have X, height, width, bottom, and alignment. And you can look in the library function definition for more details online. From a data perspective, we just have a very simple data set: two lists of data. The first is going to be of course names, which will eventually go on our X axis. And then students, which are our magnitude, which will go on the Y axis eventually. So, let me run this quickly so that we can define our data set. Next, we want to be able to actually create the bar plot. And in order to do that, we have to include our library, our pyplot library, which is matplotlib.pyplot. And then we create a figure. We want to add axes. And then we can call the bar function with courses, as we talked about, which is our X component, and students, which is our Y component.

Add some labels, and then show the diagram. So, let me run that quickly. And there we go, there's our first bar chart. And if we look at it, so now we've got student enrollment by course. Relatively simple. We could see our courses on the X axis. And then we could see the magnitude, or the students to have enrolled, on the Y axis. Now we'll take a slightly more complicated look at a bar chart by using a multi-series bar chart or bar plot. So, here what we're going to do is we're going to use a similar technique, but in this particular case now, instead of just using one set of data for our Y axis, we want to plot three different sets of data. So, here we can see students by campus. And I'm using campus as the series dimension. So now, I'm going to have three sets of data, which defines my Y data set. And just like before, I'm going to create my figure, add my axes. And then I'm going to define my X axis by using an arrange function, which is going to give me a bit of a distribution so that I can actually display these. So, my bar function now instead of just being X and Y axes, I want to include a couple other things in order to be able to make it distinguishable.

So, the first thing I want to do is I want to add an offset to the X axis, so that I can say that when I display each one of those particular X variables, that it's not on top of the other ones that I want to display. So, in order to have three, I have to have three different offsets. In this particular case, I'm using .25 as the distribution offset. So, the first one gets 0, .25, and then .5. And then I can use each one of my different array functions that I can see that I have students the first set of students by 0 campus 1 and 2. But the most important part of this one is that I'm going to assign each one a color. And so, I get blue, green, and red, so that I can actually distinguish them.

Now I want to include labels for each one of my axes, so that I can now start to include a label for the X and the Y axes. Which we've seen before. Add a title. And then I want to be able to add my tick marks, which allows me to be able to say, this is where my X axis will be. And I have to offset this as well by just a little bit in order to get it to display properly. And then also important is that I can add a legend, so that I know exactly which color represents which campus within my display. And then show, which allows me to display my graphic. So, let's just run this quickly. And here you go. So, relatively simple way for us to be able to display multiple series of data on a bar plot.

If you look at what we did today, so today we talked about bar plot and the bar function. And so, we learned about some of the formatting options you can use within the bar plot and bar function itself. We went through a standard bar plot, which we saw was relatively easy to do. And then we did a multi-series bar plot, which allowed us to add things like a legend and coloration but also to display multiple series at one time using a bar plot. If you want more examples of this, go to the matplotlib gallery and you'll see a number of different bar plotting examples up on that site and the code that goes with it, so you can learn more. Thank you.

****

Video 4: Scatter Plots (7:15)

Welcome. In this session, we're going to be talking about scatter plots. From an outline perspective, we'll talk about the definition of a scatter plot. We'll define some data for the scatter plot examples we're going to be providing. We'll talk about how to create a scatter plot, just a general scatter plot. And then we'll talk about generating a scatter plot, which is a little more complex, with a series of groups. So, first, what is a scatter plot? So, a scatter plot allows us to be able to determine the relationships between 2 different data dimensions or 2 different axes. In the 2-dimensional sense, we're really just comparing x-values against y-values. And so, the x-axis measures one dimension, and the y-axis measures the other in this simple example. The one thing to notice is that if both variables are increasing at the same time, they have a positive relationship, as in the example here. If one variable decreases while the other increases, they have a negative relationship. And sometimes, the variables don't follow any patterns at all, and they have no relationship. And that will be our first example, actually. So, we'll use a completely random set of data. If we look at the matplotlib function for plotting a scatter plot, it's called scatter. And the definition of the function is comprised of the first parameter being the x-coordinates. The second parameter being the y-coordinates. A shape parameter, colors, and an alpha. And the alpha really just determines the transparency. So, first, let's define data for our scatter plots. So, in the first case, we want to define a random set of data for x and y that we can use, which will allow us to be able to plot 500 points on a scatter plot. So, let's execute this code. And the one thing to notice here is we use NumPy, the rand function, which allows us to be able to create a random set of data.

The second data set's a little bit more complicated but not much. And this uses the same function. However, we're going to use some scalers to be able to change the values slightly so that that will be plotted in a different location. And we also want to create 3 different groups of data, which we'll eventually include in the single scatter plot. So, let's run that. And now we've defined our 3 sets of data. So, if we look at the simple scatter plot in the first example. So, we want to effectively just plot the first 500 points that we defined for data set 1. So, the first thing we're going to do is we're going to import our pyplot library, as we've done before. And now we want to invoke the scatter function with an x and a y variable. We want to give it a color. And the indicator here is g for green with an alpha value of 1, meaning that there is no transparency or it's fully opaque. Then we want to add a title, a label, and then we want to show our diagram. So, let's just run this. And you can see, relatively simple. And you can also see, because of the randomness of the data set, there doesn't appear to be any correlation between the data that's included here. So, one thing that was interesting about this was that, if you remember, when we typically draw matplotlib plots, we usually include a figure first. And then we include our axes. And then we draw to the axes.

In that particular case, all I did, if you look at it just a little bit closer. I just went ahead and called scatter. So, the figure and the axes were predefined in that particular case. So, what I wanted to show you next was just a simple example of how we can actually go through the same process that we've been going through in the past. Which is to define the figure first and then go ahead and then define the axes. Then call the scatter function and then the titles. And this effectively will give us the exact same plot. But we just went through it in a little bit longer way. But if you want to take the default in a simple example like this, then you don't actually have to define the figure and the subplots. So, it's an interesting approach. But something that you should also keep in mind. So, next we want to create a scatter plot with groups. So, if you remember, we defined 3 different groups of data or 3 sets of data. So, we want to be able to plot them. And then we want to be able to distinguish between the 3 groups of data. And we're going to do that by changing the color of the data. And then we want to also include a legend, which will allow us to be able to define what that data set is. So, the first thing we do in this case is we want to group that data together in a variable by creating a tuple of tuples. So, we're going to take data and assign it groups 1, 2, and 3.

Then we're going to assign colors. Which we're going to use in a minute when we actually display the scatter plot. And then we're going to give the groups names. And here it's just a simple set of names, which will be used in the legend. But it's called the crimson, green, and blue group. The next thing I'm going to do is define my figure. I'm going to create my axes, as we talked about before. And now I'm going to use a for loop, which allows me to be able to call the scatter function for each one of the groups. So, effectively I'm going to iterate to the groups in that I want to be able to plot them in their independent colors. And so, that's effectively what's going on here. So, I'm going through each one of the groups. And every time that I iterate through a group, I'm going to change its color. So, in this particular case, it goes from crimson to green to blue.

Then I want to include a title. Give it a legend. And because I included this label parameter here or the argument at the end of my scatter function, now I have available to me this location indicator or variables, which I can use to define the legend. So, this actually is a convenient way for me to create the legend. I give it my 2 axes labels. And then I can show the diagram. So, let's go ahead and run that. Relatively straightforward piece of code. So, now here you can see the scatter plot using multiple groups. I have different colors. And it's easy to distinguish which is which, because now I have a legend that I've included.

So, what did we talk about? So, we talked about what a scatter plot is. We defined a data set for the scatter plot in 2 different ways. So, we basically defined groups of data and a single set of data. Then we talked about how to create a scatter plot in the simple sense. And then we talked about how you can create a more complex version of a scatter plot by using groups. Hope you found this useful and thank you for your attention.

****

Video 5: Histograms (8:13)

Welcome. In this session, we're going to be talking about histograms. From an outline perspective, we'll be talking about what a histogram is from a definitional perspective. We'll define the data set for our histogram. We'll create a histogram using Matplotlib, and then we'll create a histogram using Matplotlib with a line of best fit, and then we'll wrap up. So, let's talk about what the definition of a histogram is. So, from a statistical perspective, we use rectangles to be able to show frequency of data, and then we want to be able to include that data in successive numerical intervals of equal size, which are sometimes referred to as bins. In the most common form, the independent variable is plotted along the horizontal axis, or the x-axis, and the dependent variable along the vertical, or y-axis. And so here, this figure describes what a histogram looks like. If we look at Matplotlib, the library, there is a pyplot.hist function, which allows us to be able to graph a histogram. If we look at the general format of the hist function, you can see that the first parameter is the set of x values. Then we have the number of bins, a color, and a transparency, which will include the transparency of the color that we're using for the graphing. So, let's talk about the data sets that we're going to use in our examples. In the first data set, we want to define a set of points, which we are going to break into five bins, and this will just be a standard list of points which we will type in or define. In the second case, we want to create a distribution, or a normal distribution of data, so we're going to use a mean value and a standard deviation and then use a NumPy function to allow us to be able to distribute this points in a way that's easily graphable as a standard normal distribution, and then we're going to separate that data into 20 bins.

So, the first thing we do in terms of defining the data, we want to include NumPy, as I mentioned, to be able to use the random function so that we can generate our points, and then we're going to go ahead and perform the multiplication in a minute. So for data set one, in the x variable, I'm going to assign a list of values, which I have predefined, and then I want to specify the number of bins as another variable, which I'll use, which are the five bins for the first data set. Then in the second data set, I want to calculate a normal distribution, so I'm going to use a mean value of 100, a sigma of 15, and then I'm going to perform the calculation. So, I'm going to take the mean plus the standard deviation times the randomized function for 10,000 points, and then I'm going to define the number of bins equal to 20, so I'm going to break those into 20 different bins. You'll see that in the distribution in a minute. So first, let's run this code so we can generate our data. And now we're going to go into creating the actual histogram. So, this is relatively simple now that the data's been defined. So. once we've defined the data, really, all we have to do is call the function. So, if we look at the function itself, we're going to call the hist function. We're going to pass in the x value, the number of bins, which is equal to 5, if you remember.

I'm going to give it a facecolor of blue and an alpha, which is the transparency level of .5. I'm giving it title, and x and a y label, and then I'm going to show the graph of the diagram. So, let me just run this. Okay, and then you can see, here we've got just a simple histogram using Matplotlib. So, it's relatively easy, and it's just a set of data that we've defined. The one thing that you'll notice in these diagrams is that they're different from a bar chart or a bar plot in that a histogram typically does not have any gap between the bins, which gives it the feel of a distribution, which is really what a histogram is all about. In this next section, we're going to draw a histogram with a line of best fit. And so in this particular case, we're going to draw if you remember, the second data set was a histogram, which was a normal distribution, and then we want to use a line to be able to show the line of best fit, which should also look like a normal distribution.

So, what we're going to see in this example is, we're going to return the values from the hist function, which we're going to use to be able to help us to plot the line of best fit, and then we're going to end a legend and put in our normal labels. So, let's take a look at what this would look like. So, the first thing we want to do is, we want to call our function, so we're calling our hist function again. We're using the second data set. If you remember correctly, the second number of bins, which is equal to 20 in this particular case. We're giving it a face color, and we're giving it an alpha, again, of .5 so we can display this. Then we're going to move on to the best fit line. So, if you remember correctly now, I've returned some values from the hist function, so I have my n value, the bins value, and patches. And we'll use patches in a minute, so let's just keep moving on. In order to determine the line of best fit, I want to calculate my y values, which are going to be equal to the normal distribution of the bins. And we've already defined the mu and sigma, so we already have those values, so we're going to reuse those. So now I've got my y values, which will go along with my x values, which are my bin values. So now I can actually plot that.

So the second piece is really just plotting the line, which would really represent my x and y values, and if you remember from when we were plotting lines, I have my x, my y, and then this particular case, this code indicates that I'm going to draw a red dashed line. So that's what we'll see in our graph, or in our diagram. The next thing we want to do is add a series of labels, so I added x and a y label and a title. Then I want to go ahead and create a legend on top of it because I want to indicate what this line really means. And in our particular case, we're going to call it a fit line, but now I just want to draw a line for the legend. So, I'm going to create a 2D line, which I'm going to assign to the variable red line, and then I'm going to place in the legend the red line itself, which is really nothing more than a marker of an underbar, which we can see here with a label called fit line and a color red. So pretty simple to add a legend indicator. You could add multiple as well, but in this particular case, we're just adding one. And then finally, I want to just adjust the subplot a little bit to the left, and then show the plot. So, let's run all this code and see what this looks like with our new data set. And it takes a little while to calculate. We're calling a series of functions, but you get a nice diagram here, a nice histogram which shows the line of best fit, which is really just the normal distribution, as you can tell, which you would expect. And we get our legend and the labels as expected. And then finally, we'll just run that last cell.

Finally, in wrapping up, I just wanted to talk a little bit about what we learned. So here, we learned what a histogram is. We talked about creating a histogram using Matplotlib in the simple case, and then we talked about creating a histogram with a line of best fit, which involved a little bit more code, but it gave us a pretty nice diagram in order to be able to determine what a normal distribution would look like and also include that line of best fit. If you want more examples on what histograms look like and some code, go to the Matplotlib library, and thanks for your attention.

****

Video 6: Customizing Graphs (7:43)

Welcome. Today, we'll be talking about customizing graphs using Python. First, we'll start with an introduction. Then we'll go through plot customizations. We'll talk about legend, color codes, marker codes, line styles. We'll talk about setting limits. We'll talk about grids. And then we'll wrap up. So, first, from an introduction perspective, what are customizations? So, within matplotlib, we have the ability to set aesthetic features that make our graphs and charts more appealing for our audiences. So, there are a number of ways to do that within matplotlib and pyplot. And we'll talk about a number of them. So, the first thing we'll talk about are plot customizations. So, plot customizations allows us to do things like adding legends, adding colors, and setting marker and line styles. So, I've included here a series of what they would look like. So, you can look at these in more detail. But basically, you can actually look at legend placement, different criteria, where you can set different values for locations for your legends that you may be adding to your plots.

Additionally, you can look at different colors and marker codes and line styles that you might want to use as you're plotting your lines or setting the colors of your plots. So, here are a list, as well, of the number of different types of color codes, different marker codes that you might use, and line styles. So, let's go through an example that will use several of those customizations that we defined above. So, let's add some code here. So, first, we want to include pyplot, which is a plotting library. And we can define some data that we want to be able to use. So, we're going to just set a simple data set. We're going to create our x and y values. In this particular case, we just want to set a single y data set and use 2 sets of x values so we can get a little bit of difference in distinction. So, we create our figure. The figure is going to be, as you remember, in Canvas, which allows us to be able to draw. We're going to add an axis. And then we're going to go ahead and define the plot. So, we're going to create 2 different lines.

The first is going to be red color with square markers. The second line's going to be blue color with circles. And then we're going to create a legend. What I'm going to use for that legend, we're going to just give it some legend labels. Let's just say iPhones and Samsung. Different types of phones, just as an example. And then we're going to give a title to our overall figure. And then we're going to set some x and y labels. And then we're going to show what that plot would look like. So, let's go ahead and see what it looks like. So, as you can see in this particular case, the use of several of those different aesthetic capabilities that we talked about earlier. So, we've set labels. We set colors. We set line styles. And we set marker types. So, that's a really unique way to be able to add some color and aesthetic value to the way that we are plotting. So, one thing that you'll notice here is that we didn't really set the x and y tick marks, if you will.

Basically, it gives us the ranges of the x and y values. What we can do, is we can set the limits which defines the data set of what we plotted on our axes or within our figure. Matplotlib automatically arrives at min and max values for our x and y labels. So, as you can see on the chart. So, notice the ranges that are already set. So, this is automatic. So, we want to set it explicitly, so I want to go through an example of how you can set that explicitly. So, the first example, we're just going to show how to set it all the limits, which is really the default. But I just wanted to give you a feel for what that would look like. So, the first thing we want to do is to plan our figure, create our x-values and our y-values, and then go ahead and plot the x and y values. In this particular case, we're just using an exponent. We set our title. And then we want to just show the values. So, this is an example of what it would look like by the auto limit setting. Okay, so, here's our example. Auto limits is the label. And it actually fits very well. Let's just say that in a particular case, we wanted to set the limits specifically. So, let's just say at the top I wanted to include something like a legend, and I want it to allow a little more space. So, what I would do, is I would set a very particular limit. Which would give me a little bit more space within my subplot or within my figure. That allows me to be able to see that. So, I'm going to do the same thing. However, this time now I want to create the limits on my own. So, in this particular case, I'm going to change my title slightly to say that it's explicitly setting the limits.

And then I want to set my limit values explicitly. So, let's take a look and see what that would look like. So, now you can see that, because I set my limits explicitly, the ranges, they start at 0 still. Which is fine, which is what I wanted. But now my maxes, both in x and y are different. So, it does give me a little bit of space at the top. So, that's a convenient function that'd be useful, especially for trying to do things aesthetically. Like include legends and including a location. So, next I want to talk about grids. So, grid function allows us to be able to add grids to our canvas or on our plot as we draw it. So, we can include major and minor ticks on the grid. And we can include things like color, line style, and line width. So, that's a convenient function, as well.

Just to make the plots that you're creating a little bit more aesthetically pleasing. So, in this particular case, we're going to go through a similar process. So, we're going to create a figure and a set of subplots. In this case, we're going to use the subplots, which are going to allow us to be able to see the different styles that we want to be able to use. So, this is a good feature of the figure and subplot model that we can explain by using 3 different charts for the particular grid functions that we're going to be using. So, now we go ahead and start to create some of the aesthetics of what this would look like. So, the first one, I want to plot a line in a subplot 1. And this is with the grid turned on. The second one, I'm going to plot a line in the second subplot. And this time, it's a custom grid that I'm turning on. And so, the custom grid is going to effectively use the grid, which is set to a color, which is blue. And I'm using a different line style. Notice here is the line style setting. And I can set the line weight, as well, which is the other thing I'm setting. And then the last one I'm setting is no grid. And so, let's take a look at what this would look like. So, notice, I have the 3 different subplots or axes that are included here. I have my default grid. I have my custom grid, which is light blue with a light line style. And then I have no grid.

So, in wrapping up, we covered how we can customize the different matplotlib plots by using some of the feature that are made available to us. And so, we just covered a small subset of them. There are many more. But we covered legends, color codes, markers, setting limits, and grids. And so, if you want some more examples, you can actually go up to some examples here in the link that I've included. Thank you so much for your attention.

****

Video 7: Line of Best Fit (7:59)

Welcome. Today we're going to be talking about the Line of Best Fit. And we're going to be talking about how can draw the line of best fit on a scatter plot using Matplotlib. From an outline perspective, we'll define what the line of best fit is and then we'll go through a couple examples. The first example will be using the "least sqaures" method to determine the line of best fit. And then we're going to use the linear regression module library from Sklearn, which is a Python library, to be able to calculate that line of best fit. So, first of all, what is the line of best fit? So, the line of best fit is really a trend line in its linear equation, which shows the correlation between two variables. In our particular case it will be between our x and y variables. But not only does it use the values that are there to be able to determine what the trend line would be, but it also allows us to then predict future events based off of the slope of that line. The other thing that it will do is it helps us to define whether or not there is a correlation between the values. And so, if you look at this particular example the tightness of fit of the line to the points allows me to be able to determine that this is a pretty good grouping of points and that the line is a pretty good indicator of whether or not there's a relationship between, in this case, temperature and sales. And so, we'll go through that in more detail. But, if we look at the equation, which I won't read you in detail, but basically these are the steps to be able to apply the least squares method to determine what the line will be.

And so, we would go through the calculation and we'll do that in the next step. But fundamentally, where we end up is we want a calculation of the line itself. And we're calculating the line using the equation y = mx + b. So, let's go on to the actual code. The first thing we want to do is we want to include "Numpy" the numpy library. And we're going to generate some points for x and y. In this particular case we're using 500 points. And so, that's going to give us a randomly generated set of points that we can use so that we can draw our trend line. We'll calculate a denominator, which we're going to use in the equation, so refer back to the equation, we'll calculate the m and b variables and we'll use that for the line calculation and we'll assign the line values to ybf1 as the final step. So, let's go ahead and execute that. And now we will have our line that's been calculated so we can go ahead and display the plot using Matplotlib. So, the first thing we're going to do here is we're going to include the library of pi plot. We import the library as we've done before. We're going to associate some colors. We're going to define an area. And then we're going to go ahead and create the scatter plot itself. So, we're going to create the scatter plot, give it a title and then here the plot is really the key thing to look at. So, this is where we're going to use the actual y coordinates that we've created, which were the line itself. And so, here we're using our x coordinates, which were from the original data set, and then we're going to use the y coordinates that were calculated for the line itself. And so, that's what we'll see here and we're going to use the color indicator of b, which is blue, so that we can actually see the line.

And then we're going to give it a label for the x and y. And then we're going to show it so that we can actually see the figure that's been generated. So, let's go ahead and run that. And then here you could see that even though it's a number of randomly distributed points, there's probably not a huge correlation between this as we could see from the original definition that we went through, but you do have a line actually, which does represent the least squares method. And so, the line looks like it should probably be going through the center of the scattered points and that seems reasonable in terms of how we would use the least squared method. Next, we'll look at how we can create a line of best fit by using linear regression. In this case we're going to use the Python Library Sklearn, because the model for linear regression is included in that libary. So, here you can see that the first thing we would do is create our imports. The first thing that I'll point out is that in order to create a random dataset, we're going to use a dataset function within SKlearn called "make blogs," which will allow us to create three different datasets, which you will see in a minute. Then we're going to define the model, which is a linear regression model. And then, of course we're going to use pi plot for our Matplotlib library drawing techniques. And then Pandas as a data frame, which is going to be used to hold our data. So, the first thing we're going to do is create our datasets using make blobs. We're going to assign the values to a data frame, which we're going to use. We're going to define a set of colors, which will be used in the plotting of the scatter plot values. Then we're going to go ahead and define our figure and our axes using pi plot. We're going to go ahead and then plot the actual clusters, so the three different clusters, which we'll see below in a second. And then we can go ahead and start to create the line of best fit.

So, we're going to start to take the values, the x and y values then from the data set and call the linear regression model, first we define it, and assign to the linear regressor variable. But then recall the linear regressor.fit with the x and y variables, which allows the model to then execute against the x and y coordinates to determine what is the line of best fit. So, where we're going to assign that the prediction to the y pred, which is going to hold all of the y values that we want to be able to display as the line of best fit, which basically fitting to the linear regression model. OK, so we're going to go ahead and plot that. And I'm going to use the color orange for that particular line. And in the rest, we've seen before. We're going to include a title and then we're going to use some labels and then show the plot. So, let me go ahead and execute that and there you go. So, we've generated our data set, we've generated or red, blue, and green data sets. And then here you can see that the line of best fit has been drawn. But in this particular case we're using linear regression in order to determine what that line should be. The one thing that I will mention to you is because this is a random set of data generation, if you run this multiple times you will get different sets of data and then you'll also get a different line of best fit. So, if I ran this again, you can see that I'm changing the data set and I'm changing the line of best fit. And in this particular case, even though it's random, this one looks like it actually is a pretty good correlation between the variables because of the way that the cluster have been created. There are some others where if we run it say numerous times, we may see that it's a little bit less correlated and even in this particular case it looks more like a negative correlation. So, you can play with this a little bit and you have the code. Just to kind of give yourself some practice for how to create a line of best fit.

But let's just talk about what we covered. So, we talked about what a line of best fit is and we went through a couple examples. The first one we used the least squares method and in the last one that we just talked about was using linear regression. So, hopefully you've learned a lot in this session with respect to how you create a line of best fit. And thank you for your time.

****

Video 8: Box Plots (5:20)

Welcome. Today, we're going to be talking about box plots. From an outline perspective, the first thing that we'll talk about is the definition of a box plot. We'll create a data set that we can use to display our box plots. We'll create a box plot and then talk about the customization of a box plot. So, from a definition perspective, a box plot is sometimes called a whisker plot. But it really allows us to be able to show a set of statistical value of a data set. It allows us to be able to see our minimum value. It allows us to be able to see the first quartile value, the median value, as well as the third quartile value. And then, finally, the maximum value. So, it provides a lot of valuable information to us very simply so we can see how a data set is distributed based on the statistical calculations.

So, let's talk about some data that we might use to be able to create our box plot. In this case, we're going to create some random data. And we're going to use NumPy to be able to do that. So, here, I am going to create 4 sets of data, which would allow me to be able to see 4 different box plots to give us some variation in terms of how we're going to graph. So, let's do that first. So, let's calculate our data set. So, here now, we've consolidated all of our data into 1 particular variable called data to plot. And that's the data that we're going to be passing into our function. So, from the perspective of creating the box plot, the first thing you need to do is define our figure. And if you remember, the figure construct is going to be our canvas. And then we add our axes. And then we call the box plot function. And we call the box plot function with the set of data that we want to be able to graph. And if you remember before, up here, we created 4 different sets of data.

So, let's just go through this process. As I said before, the first thing we want to do is, well, clearly, is to include our library. And then run the different functions that we described. So, if we go through that process, you can see that just a simple box plot would look like this, with 4 sets of data. And then, if you remember earlier, the min and the max functions generated these top and bottom whiskers, if you will, in the box plot itself. And then the bar represents the first and third quartile. And then the line in the middle represents the median value. So, this is a very powerful graphic capability that allows us to be able to see a lot of information about a data set very quickly. But there aren't many things that are customized on this, so it can be difficult to read. But that was just a simple version of the box plot. So, let's customize it just a little bit in the next set of exercises that we do. So, in this next section, we're going to customize the box plot a little bit. And we're going to use some of the basic capabilities of the box plot function to be able to do that. So, the first thing we're going to do is we're going to change their orientations. We're going to create a horizontal box plot, which we might change for more readability.

But I just wanted to make sure that you understood how you would do that. There is something called the vert parameter, which allows us to be able to change from vertical to horizontal. In this particular case, if we set it to 0, we need to display the box plot in a horizontal form. The next thing we want to do is set a patch artist. And what that does is allows us to be able to change the configuration parameters, which allow us to be able to set color. So, in this particular case, we would set patch artist to true so that it can actually color the boxes which represent the first quartile, third quartile, and median, if you remember from the original diagram. Then we add some labels. And we set the face color, which we said before allows us to be able to set color. So, the first thing we want to do Is we want to be able to create the box plot itself. So, we're going to actually use the same set of data. But what you can see here is that we're really just changing some of the parameters. So, we're adding in the patch artist equal to true, as I said before. We're setting the vert variable to 0, which means horizontal.

We're adding labels. Then we add colors. And we set those colors using the base color or set base color function. And so, if we run this, you'll see that we end up with the following diagram. Which allows us to be able to add a little bit of dressing on top of the classic box plot. Which allows us to draw attention to the users by subtle things like color, adding labels. And then also we could change some of the other characteristics if we wanted to.

From a wrap-up perspective, what we did cover today was what a box plot was, how to create a box plot. And then some of the customizations that you can perform on a box plot in order to make it more readable for your end users. There are plenty of examples in the gallery, so make sure you check out the gallery for some additional features that you could possibly set. As well as learning more about how to create a box plot. Thank you.

****

Video 9: Pair Plots (8:23)

Welcome, today, we're going to be talking about pair plots. From an outline perspective, first we'll talk about the definition of a pair plot. Then we'll talk about creating a pair plot with Matplotlib, and then, we'll talk about creating a pair plot with Matplotlib in Pandas. And the reason we'll be using Pandas is to allow us to be able to do this in a much more efficient way, which we'll see in a minute. So, first of all, what is a pair plot? So, a pair plot, as you can see from the diagram, really, is an end-by-end matrix.

And so, what this allows us to be able to do is it allows us to be able to see multiple scatterplots, including was called a scatter matrix. And so, it really allows us to be able to see a series of different scatterplots, which are two by twos in a convenient way, and allows us to be able to compare a number of axes against each other in a very quick way. So, this four by four, for example, gives us 16 two by twos, which we can use for quick analysis and allows us to be able to see in a very quick way whether or not there may be correlations between two of the variables, and this could be very, very useful, if you wanted to do some more exploration into a particular area, because you may identify a particular correlation of interest. So, let's look at how you would do this with Matplotlib. So, the one thing I'll mention, at least in the start of this, is that Matplotlib does not have a particular function which allows you to be able to do the matrix itself, although it does have a scatterplot object, which allows us to be able to do the two by two.

So, the first thing I'm going to show you is how can you create a function which would allow you be able to create the matrix itself and to give you the same kind of matrix that you would expect in a pair plot? And so, here I go through and I use the series of functions of Matplotlib to define what that may be. So, the first thing I would do is to create a figure and axes, which allow me to be able to create a number of subplots, which would allow me to be able to do exactly what I require in terms of being able to create my matrix. The next thing I'll do is I want to be able to then, say, set the axis variables, and then I want to assign the tick marks to each one of the two by twos, which would allow me to be able to then create an axis for each one, as I'm going to display them.

Next, I plot the data or assign the data to each one of the two by twos, and then, I can create labels that I want to be able to use, in order to be able to make this look like a matrix, as I would expect. So, I can tell which particular two by twos I'm really looking at any particular time. And then finally, what I want to do is I want to make sure that I turn on the proper x and y axes ticks, so that I can see exactly what's required, and then return the figure. So, if you look at this, it's a relatively simplistic, in the grand scheme of things, function, which allows me to be able to my end-by-end grid, and it allows me to be able to see, then, the matrix, or the pair plot, directly in Matplotlib, without a specialized function. So, let me run this quickly, which defines the function, and then, the next thing I want to do is I want to define the data set and then call my function with that data set to see what this may look like. So, here I'm creating my data set using the random function from NumPy, and then, here you can see that I'm going to call my particular scatterplot function. I'm going pass it my data. I'm going to pass it my variable, and I'm going to specify some of the characteristics that I want to be able to see. Of importance here is the fact that I'm allowed to use circle markers.

The outlines of those markers will be blue, and then, the face color, or the internal color, of each one of those markers will be red. Then I can go ahead. I can call it a simple scatterplot matrix, and then, I can go ahead and show it. So, let's run that, and you can see now I have in my end-by-end matrix, or in this particular case, it's a four by four matrix, and it's showing me each one of the two-by-two scatter plots that I would be interested in, and in this particular case, because I use random data, there's not really a lot of interesting things to look at. But you get the idea. So, next I wanted to cover how to create a pair plot with Matplotlib and Pandas. And so, in the case of Matplotlib, I had to actually create the function, which allowed me to be create a matrix scatterplots. However, in Pandas, there already is the scatterplot matrix function, which we want to be able to take advantage of, and then, still show it within our Matplotlib Canvas or Matplotlib figure.

So, this is a relatively easy thing for us to do. So, if you look at this, the only thing we really have to do is import Pandas, and then, from Pandas, we want to import the function called scatter_matrix, and that's the function which would allow us to be able to display our particular graphic object. So, let's take a look at this. Because of using Pandas, usually everything in Pandas is a data frame, but in this particular case, the first thing I want to do is I'm going to create a data frame with a random set of numbers, and I'm going to give it a set of columns, because if you remember correctly, the whole idea of a pair plot is it allows us to be able to do an end-by-end comparison where end is defined by these columns. So, the next step, once I have my data frame, is to be able to call the scatter_matrix function directly within Pandas with my data frame and then specify some of the variables that I want to be able to do, in terms of display. So, the alpha value is going to be the amount of translucency that I would have, or transparency that I would have.

Create the figure size, which is really, the figure itself in pyplot, and then, what I want to put on the diagonal, and in this particular case, it's a kernel density estimation that's going to, and then, here, you can see, basically, it's a Gaussian or a relatively Gaussian distribution, which will be included on the axis, and then, finally, I want to be able to show it. So, let's go ahead and run that, and if I run it, you can see that it's pretty simple to run, and it gives me a nice display of a similar kind of thing, and again, just like we did with the Matplotlib scatterplot that we created by ourselves, or by hand, it's a random set of data. So, we're not getting too much interesting here, but you can see that as compared with this kind of a scatterplot representation, you can quickly then be able to look through the dataset and see if there's any patterns that might of interest to you, that you could use for further investigation.

So, what did we learn? So, what we learned today was that we went through what a pair plot was, the definition of what a pair plot was, and then we learned how to actually create one using Matplotlib. But it did require us to actually do a little bit of coding in order to be able to get the matrix, in order to be able to look like exactly what we expected. But we also found that we could do it relatively easily by using Pandas and Matplotlib together to be able to allow us to be able to do the end-by-end matrix with a scatterplot matrix, a relatively simple way. There are definitely more examples you can look at in the Matplotlib library as well as the Pandas library, and then look in the function in a little bit more detail to give you some more features and capabilities that may be associated with the scatter_matrix function. Thank you.

****

Video 10: Time Series Plots (8:11)

Welcome. In today's session, we'll be talking about time series plots. From an outline perspective, we'll cover the definition of a time series plot, and then we'll go through a couple examples. The first example will be a time series plot using Pandas, and the next will be a forecasting time series plot using a seasonal ARIMA model. So, from a definition perspective, a time series plot is really nothing more than a line chart, and the line chart has one distinction, and that distinction is that the x-axis is time. The y-axis is going to be the variable that we want to be able to measure. So, if we look at this simple example, we can see here that the x-axis is our time component, and it has years, and then the y-axis is temperature. And so here, the line chart represents the comparison of temperature across time, and it's a very powerful diagram if you wanted to be able to measure things like trends, which is really what time series are good for. So, let's take a look at creating a time series plot using Pandas. and the reason we're using Pandas in this particular case is that there are some distinct advantages to being able to use a data frame for our data, and I'll go through those in a second. So, the first thing I want to do is I want to import the Pandas library, then I want to include a parser, which allows me to be able to parse my dates.

So if you remember, we want to be able to use, on the x-axis, time components, and so my data will have dates, and so I want to be able to parse them out and put them into a format that I can display on my x-axis. The next thing I'm going to do is I'm going to read a CSV file, which is available on the web, and this includes passenger data. And then finally, I want to create an index, which is going to be the dates, which are my primary index within my data frame. And then the last thing I'm going to do here is just take a quick look at the data. So, let's run this code and see what we get. So here, you can see the top five records within our data set, and there are plenty more, but it really just shows us the dates, which are monthly dates, followed by the number of passengers. And so, now this is really a powerful technique, and so within the data frame, Pandas actually includes the ability to plot data using Matplotlib libraries. And so, what I can do now is, within the data frame, I can use the data frame dot plot function and actually see that data as a line chart. So, let's go ahead and do that. And here, you can see this is a time series plot because I have my dates across the x-axis, and then I have the magnitude or number of passengers on the y-axis. So, very powerful ability to be able to do this directly within a Pandas data frame.

In this next example, I'm going to be talking about a forecasting time series plot using a seasonal ARIMA model, which sounds like a mouthful. And we're not going to get into details of the ARIMA model itself but suffice it to say that this is a forecasting model that uses historical data to be able to predict the future values. And so what we want to as part of this, is we want to be able to first fit our model to the data that we have available, and then eventually we're going to predict the future values of this data based off of the fitted data from the Legacy historical data. So, I'm going to include a few libraries here. First, I'm going to include pyplot, which allows me to be able to do my graphing, and then I want to include some statsmodel packages, which is going to give me my ARIMA model. I'm not going to go through the configuration of this model specifically, but suffice it to say that first, I'm going to fit my model, and then I'm going to plot it. So, what I'm doing here is I'm actually invoking the model with my data frame.

You can see the ts here. So basically, what's happening is, it's using the existing data set, and it's going to create a seasonal fitted model directly against that data. And then I have the results here, which are stored in results_ARIMA, and now I can go ahead and plot this. And so, the next series of steps are to plot. First, I'm going to plot the original data, and then I'm going plot the fitted data, and I want to compare the two directly within my plot. So, you'll see two different line graphs with a legend, which'll show us the fitted and the original data. So, let's go ahead and run that. And here you can see, and it's a little bit tough to see, but you can see the blue behind it, but you can see the red being the fitted data and then the blue being the original data. So, what's the value in this? So, it looks like there's a pretty good fit. The ARIMA model did a pretty good job of fitting to our actual data. So, now we want to go ahead and use that data for prediction purposes, which is really the value of the ARIMA model. So, now I'm going to go ahead and do effectively the same thing, is I want to plot the original data, but now I want to use the Predict function to be able to predict what would the data look like at some point in the future.

And in this particular case, what I'm going to do is I'm going to create a forecast variable, and I'm going to add it to the data frame. And so, I'm going to use the Predict function, which is part of my model, to be able to add, say, predictions for this date range. So, this is from 1957 to 1960, and I'm going to add that back into my data frame. And then I have the ability to just basically use my Plot function all over again and then be able to plot the two different lines or the two different time series by using the Pandas Plot function. So, let's go ahead and do that and well, actually, we also add a title to it, and then we show the plot overall. Let's just run this. And here, you can see, with the legend, you can see that we can see the two lines. So, first we have the blue line, which is the original data, and then what's in orange, or yellow I guess it's orange is the predicted data, which gives us now a good approximation of what the data may look like. But it's really nice because, in terms of the time series now, you can see how we can actually include both. So normally, with time series, you'd want to do some form of prediction or forecasting that usually comes with a time series. And so, with Matplotlib and with Pandas, you have the ability to do that very easily and simply.

So, in wrapping up, what we talked about was the definition of a time series plot and what it was, and then we talked about how to create a time series plot using Pandas. And then we even went so far as to include some seasonal ARIMA model capabilities to be able to show what a forecast would look like, and then how you would plot a forecast. If you want some more information, I included a couple links here where you can take a look at what some of the different features would be with respect to being able to create time series plots. And then just for fun, I actually went and created a small prediction, which would look at that same data set based off of the fitted model we already had. And it said, what would the data look like from 1960, which was the end of our data set, up until the present time, or near the present time? And so, let's just run this, and you can see that, you know, clearly, it looks more linear because there isn't a lot of fluctuation in the data set. But you can see that, in 2021, the prediction is that there would be a relatively large number of passengers according to this dataset. So, that's just for fun. So, thank you for your time, and I hope you learned something.

****

Video 11: Introduction to 3D Plotting (9:05)

Welcome. Today we'll be talking about 3-D plotting in Matplotlib. By way of an outline, we'll talk about an introduction to the library that's responsible for 3-D plotting in Matplotlib. We'll talk about creating a scatterplot in 3-D. And we'll talk about creating a bar chart in 3-D. So first of all, what are 3-D plots in Matplotlib? So, we've done two-dimensional plots and now we want to look at how to create that third dimension or add the Z axis. Well, in Matplotlib, it's relatively simple because we include the mplot3d toolkit, which will allow us to be able to invoke our functions and use what's called a "projection" in order to get the third dimension or the Z axis. Additionally, there are some 3-D commands which are provided by the toolkit which will allow us to do things like a bar chart in 3-D. And we'll go through that example a little bit later. So, let's first discuss creating a scatterplot in 3-D. In order to create a scatterplot in 3-D, the first thing we want to do is make sure that we include the toolkit mplot3d. And so, we're going to use our classic import statement. So, from mpl toolkits.mplot3d, we'll import axes3D, which will give us the projection capability which I'll talk about in a second. Then we want to include pyplot, as usual, which we use for plotting. We'll use numpy and pandas for the data definition.

So, from a data set perspective, this is going to be fairly easy. We're going to create a data frame which will be comprised of our XY and Z values. So, our X values will be defined by a range from 1 to 201. The Y values will be a random set of values multiplied by 15 by our same range from 1 to 201. And then the Z axis will be something similar. And I'll just scroll over here just a little bit. We'll do the same function and multiply it by 2, just to give us a random set of data and something that will look good in our plot. The next thing we want to do is use our plotting functions. So, first we want to create a figure. Then we want to create a subplot. And now we can invoke the scatter function and we can pass it our XY and now Z parameter. Which wouldn't be accepted before but now because we're using the toolkit, it will use the 3-D function. And then the next thing we want to do is include a color, just to give it some coloration. And then we want to give it a shape size of 60. So, let's go ahead and plot that. So, if we execute it and let me just scroll back down here you can see now, I have a nice three-dimensional plot of the points that we defined in our data frame. And so, this is relatively easy and really just a matter of including that new toolkit which allowed us to be able to define the Z dimension. Which I will just refer to back here. So, the third parameter here, which is our Z dimension. And it's really just because we've leveraged the mplot3d toolkit and we've included axes3D function into our Python program. So, let's go on and we're looking at creating a 3-D bar chart. So, if we think about a 3-D bar chart, similar kinds of things we would look at. So, we want to make sure that we calculate our Z value as part of it.

And then we want to make sure that we are able to plot it. So, first we have to generate a data set. And then using a little bit of a different technique this time for generating a data set, I'm going to use something called a "meshgrid." And the mesh grid will create a Cartesian indexed data set, which I'll get to in a second. So, first we want to import numpy, then pyplot, as usual, and then we're going to include our 3-D toolkit, which is called mplot3d again. And axes3D is the module we want to include from that particular package. So, the first thing we're going to do is we're going to set up our figures and axes. And in this particular case, I'm creating a 2 x 2 matrix or a 2 x 2 set of subplots, because I want to show four different figures. Two of them I want to be three-dimensional figures and two of them I want to be two-dimensional figures. And you'll see that in a second. So, that's really the indication here. So, once I've created those, then I want to define some fake data. And basically, all I'm doing here is I'm taking simple ndarray data sets, then I'm going to call this meshgrid function. Which is going to do nothing more than a Cartesian indexing of the values in order to give us a matrix. And then I'm going to flatten it back out again so that I actually have X and Y values, so I have something that I can plot. The next thing I want to do is calculate the top of my or the Z axis of each one of the points. And so, I'm going to create a top variable, which is going to be an addition of X and Y, just to give us some different values with respect to Z. And then I'm going to create a bottom and a width. Because I'm in three dimensions now, I actually have to draw the width of the bars as I'm looking at them as well as the bottom location of the bar. And so, it's going to start at zero in this particular case. Then I go ahead, and I want to now invoke the bar3d command.

If you remember as I mentioned earlier, the bar3d command now provides me with some additional parameters. Not only does it give me my X and Y coordinates, but it also allows me to define, as we talked about before, what is the bottom location of the bar, the width of the bar, the depth of the bar because we're in three dimensions now and the top, which is actually the Z. But it also gives me one additional parameter which allows me to include shading, or not. So, in two of the subplots, we'll actually show a shaded version and a nonshaded version. And then just for completeness, I wanted to also show you what it looked like to use the two-dimensional versions of this, just so that we could compare some of the data. So, now I could take the X set of coordinates in the Z or the top coordinates and then display them in a two-dimensional bar chart in red. And then for the last subplot, I wanted to show the Y values matched up against the Z coordinates, and this time in green. And give them all titles and then show them all. So, let's run this code quickly. And you can see, we have the four different graphs.

This is the shaded version, which gives me a nice representation of my bar charts in three dimensions. In this particular version, I turned the shading off, and so it's a little bit more difficult to actually determine what is going on here. So, if you're going to look for distinction, you may want to include the shading parameter, because it seems to be very helpful in terms of being able to tell exactly which bars are which. And then the two-dimensional we've seen before, but the two-dimensional ones are relatively simple, but they just give you a little bit of a distinction between the 2-D and the 3-D invocations. And so, it looks pretty nice from that perspective.

So, in wrapping up, we covered an introduction to 3-D plotting in Matplotlib and we talked about the use of the mplot3d. It's an API but actually it's a toolkit. So, I'll refer to it in a second as the API because that's where you'll see the function references. We also talked about creating a scatterplot in 3-D. And then we talked about creating a bar plot finally in 3-D. And if you want to see more, look in the Matplotlib gallery and there are plenty of 3-D plot examples there. As I said before, you can look at the API reference for the toolkit which will give you the parameter definitions and the invocations that you can use in your Python code. Thank you for listening and thanks for your attention.

****

Video 12 Exporting Figures (7:44)

Welcome. In this session, we'll be talking about exporting matplotlib figures. From an outline perspective, we'll give you an introduction to the function called savefig. We'll talk about exporting a PNG file, exporting a JPEG file, and then we'll take a look at our exports. So, let's talk a little bit about the function savefig. So, this is included as part of the pyplot module. And so, when you include it, we'll have access to this function. Which we'll get to in a minute. But the function can be called right above the plt.show. So, right before we actually show our graph, we want to make sure that we call it. There are some idiosyncrasies associated with how you're exporting your image before you actually are rendering it in something like Jupyter. So, the best practice is to actually call the savefig function after all of your components have been added to your figure but before you actually call your final plt.show function to be able to render it within your Jupyter file. So, what is the standard command look like? The standard command has as the first argument the name of the file that you want to create. So, you give it a name something like plot.png. One thing to notice here is that the extension is going to be used by the function to be able to determine the type of file that needs to be saved.

So, for example, if I call it a PNG file, it's going to be rendering or exporting a PNG file format. If I use the file extension of JPG, it's going to export a JPEG file. So, it allows me to be able to control the type of file that's going to be exported just by using the file extension. Which is really a nice feature. The next argument is called DPI, which is dots per inch. This controls the resolution of the file that's going to be exported. Usually for the types of things that you want to be able to have at high fidelity, you're going to use probably on average 300 DPIs. But you can control how many dots per inch the image will be. The higher is going to be a little bit larger in terms of size. The lower is going to be less. But also, make sure that you take into consideration the quality of the image that's required. And then the last thing that you're going to include is called the bbox inches. Which is really a bounding box that can be used to be able to identify how much space is going to be kept around the outside of the image when the image is exported. Typically, you'll use the command, sorry, you'll use the option tight. Which is going to be basically minimal amount of space, which will be included around the image itself. Let's take a look at exporting a PNG image. So, as I mentioned earlier, in file extension or the name of the file extension, is going to be what controls the type of image we're going to export or that's going to be exported. So, just as I would normally go through the process of creating an image or graph, I'm going to go through the same process here. So, I create some x and y coordinates. I create some labels. I define my plot. I draw a line. But then notice, I'm going to evoke or call the savefig function. And I'm going to give it a plot.png as my first argument, which is my file name. So, here I want to export a PNG file. I want to control the dots per inch to 300 dots per inch, which is my resolution. And then I want to say that the bounding box is going to be tight, a tight image. And then as is the recommendation and best practice, the last thing I'm going to do is then show the image so I can see it within Jupyter.

So, let's go ahead and execute that. And you can see that this is then, first it's going to generate the file. And then it's going to display my image directly within Jupyter. Now, I'm going to go over to my operating system here. And I'm going to take a quick look at this. Just so that we can see what this might look like. So, now, I've just exported plot.png, so I'm going to double click on this. And then open up the image. And if I take a look at that image, you can see what the export would look like. So, this is exactly what I would expect. So, relatively high fidelity. And it's actually a nice image. I could include that in my presentations. Or I could include it into a document as I see fit. So, that's a convenient way to do an export. So, now let's say I wanted to export a JPG, which is a JPEG image. So, now from here, I can actually, all I really have to do is change the name. I'm going to go back to exporting a JPEG image. So, next I want to look at exporting a JPEG image. So, here, if you remember from what I was talking about before, all I really have to do is change the file extension. If I create a file extension called JPG, it's going to export my image as a JPEG file. And here, I'm going to play with the parameters just a little bit.

I'm going to say that this is a little bit higher resolution at 400 DPI. And I'm going to turn off the bounding box in terms of inches. So, I'm not going to use a bounding box in this particular case to see what the impact may be. I think what we'll find is that there really is no impact when we actually render the image. So, let's do that. I'm going to go ahead and run it. And just like I did before, the first thing I'm going to do is export my graph. I'm going to save that figure using savefig. And then I want to show it in Jupyter so that I can have the ability to be able to take a look at it interactively. So, now let me go back over here. Same thing. I'm going to take a look at the file as it was generated to the operating system. So, here now I can see plot that JPG, which is the latest that was generated. I'm going to double click on it. And now I can take a look at that image. And you'll notice that it really looks exactly the same as the prior one. However, the dots per inch is a little bit higher than the 300 that we have it for. So, either way, you can generate these images, or you can export those images so that you can use them in files as file content or in presentations or in documents. So, it's a really convenient way to be able to export images directly out of matplotlib.

So, in wrapping up, we covered the basic functions, sort of the basic parameters associated with the savefig function as well as the best practices. So, don't forget about the best practices. Don't call the savefig function prior to calling the show function. Especially within Jupyter. And then how to control, which is the other important point, how to control the type of image that's exported. It's really based on the file extension. And then pretty simply, you can actually then take a look at your exports. Because we just saved out to the operating system, you may also want to look into the function. If you wanted to store it in a different location, take a look at the function parameters. And then you can figure out how you can solve it to a different location. So, that's it for this session, and thanks for your time.

****

---**END**---