Data Science Fundamentals in Pandas (Nonvisual Data Science Workshop #2)

(0:00 - 0:34)

So let's go ahead and get started. Welcome all. I'm really excited to have you here for the second workshop in our non-visual data science workshop series.

I am Patrick Smyth, and I'll say a little about myself in a minute, but once we get some of the procedural stuff out of the way, this is the second workshop in the series. It is a workshop on an intro to data science fundamentals with pandas, and we'll talk about what pandas is. We'll even talk a little about what data science is, though that's a little bit of an unsatisfying answer sometimes.

(0:35 - 0:44)

I'd also like to introduce our helpers who are going to be in the chat. We have with us Sarah Kane. Maybe, Sarah, you could say hi.

(0:45 - 2:01)

Hi, my name is Sarah. I'll be one of the helpers today. I'll be teaching some of the tutorials later on.

Thank you, my co-leader. Elizabeth, do you want to introduce yourself? Yeah, hi everyone. My name is Elizabeth, and I am a postdoc at the Institute of Astronomy in Cambridge.

Today, I will try to answer your questions in the chat as quick as I can. Excellent. I don't know if Alex is here yet.

I am here. Excellent. Let me give you permission.

You want to go ahead and introduce yourself? Hi, I'm Alex. I'm a PhD student at the Institute of Astronomy with Sarah and Elizabeth. I'm a Python man.

I will help you with all Python things in the chat. Thank you, Alex. We may also have joining us Stephen Zweibel, who is a Digital Scholarship Librarian at the Graduate Center, City University of New York here in New York.

Hello. Steve, you're online. Yes, I'm here.

Sorry. It's a snow day here. Yeah, your children are home, so I understand how this is, but thank you.

(2:03 - 6:03)

It's also possible we may have Paul Alexander Bloom and Monika Thieu, who are two psychologists and also teachers of R and Python joining us. You may see them in the chat as well. Basically, the way the chat will work is if you have a question, place it in the chat.

If you want to ask your question privately, you can use the private chat interface on Zoom to reach out to one of the helpers specifically. We also, I believe, are having our helpers. Their names should have helper in them.

If that is the case, that will help you out if you want to send a private message. Just feel free to put most of your questions in the chat publicly, and a helper will either answer you either privately or publicly. They'll answer publicly if they think that the answer will help other people who are following along.

I'd like to now really quickly share a link to the curriculum for this workshop. It is on a website called GitHub. My recommendation is if you're using NVDA, GitHub tends to add a bunch of buttons and other information before you actually get to the curriculum part.

I would navigate by heading to the heading one on the page. It may not be the first heading one. Navigate by heading or H1 to get to the beginning on each page of the actual curriculum.

We have our helpers. Let's go ahead and do a little introduction to the topic here. I'll just say who I am.

I'm background a little bit after the last one. I have retinitis pigmentosa, so it's a progressive eye condition. Over the last 15, 20 years, I've lost vision progressively, started outside it, and now I have about 2% vision remaining.

Some useful vision, I think, would be the word for that. I used to be very low vision, very focused on magnification, high contrast, and all of those assistive technologies. I still occasionally use those in some circumstances, but now I'm more of a daily screen reader user.

I also use Linux day to day. Sometimes, if you see me flailing around a little bit on Windows, then that gives some explanation. I come out of the digital humanities, which is a field in the humanities where we use computers to answer traditional humanities questions, so things like natural language processing, if you want to use a more programmer term, to analyze large data sets.

I also build websites and do other fun things like that. I started learning Python about 10 years ago, maybe a little more now, which makes me feel like time is really flying. I really felt when I learned Python that it really changed my perspective on using computers and empowering me to do things that I couldn't do before.

I really also found that I enjoyed teaching programming to other people because I like to share that feeling of empowerment, for lack of a better word, to give you new capabilities. I really enjoy that part of my job. I also want to thank NumFocus and the Pandas project.

We'll talk about Pandas in a minute, for funding these workshops. A special thanks to Patrick Hoffler, who is a Pandas core developer, who really stood up and allowed us to submit this grant. Thank you, Patrick.

(6:03 - 9:02)

All right. Let's just say really quickly, and we'll move on from this very briefly, because I don't think this is a very... In a way, it's not that useful a question, but we're moving now into the data science part of this workshop series. The question is, what is data science? What are we doing when we do data science? What are we talking about when we talk about data science? Somewhere, a little hazy, but somewhere in the 2000s, all the statisticians started calling themselves data scientists, and actually got probably a big pay bump in the process.

Basically, that comes around a time when specialized programming tools emerged for working with data. It could be large, could be small, but often large data, and applying statistical methods, programming, other analytical methods to data. Data science hasn't really been around that long.

You can say it's a discipline where people apply programming, statistics, math, and other techniques to gain insights from data, I guess, would be the fancier definition from it. I will add one thing to that, which is that I think that there's a really important part of data science that is sometimes overlooked, which is that of a data scientist, it really gets a lot out of the context and understanding where data comes from, what happened to it along the way, and also making connections between things you know about the world, things about history, about business, about how people interact, different kinds of domains, applying that to an understanding of data. I come from the humanities, so I do think that the data has a little bit of a humanistic element that sometimes we don't talk that much about, but it's actually one of the key elements of being a good data scientist is that qualitative element as well as a quantitative element.

Data scientists do very different things on a daily basis. Some data scientists are more like engineers, and they create what are called data pipelines, but basically flows from some kind of business process or other process that generates data into a format form and an application where you can analyze it. That's one thing you could be doing.

Some data scientists are really communicators, and others are very mathematical. You might just work with models, which is basically a fancy way of saying simplifications of the world that allow you to answer questions about data and transform data into a more

simplified form. There's lots of different types of jobs you can have as a data scientist and a lot of different ways, things you can call yourself within data science as well.

(9:04 - 11:09)

Now, what are we going to be doing today? Today, we're going to be continuing to use Python, which we were introduced to last week. We'll continue to learn IPython, which is a specific way of interacting with Python through the command line. It's a fancy Python interpreter, basically a program that allows you to have a conversation with Python.

It's a fancy version of that. We're doing all of that in our Anaconda distribution. Anaconda is the way that IPython comes to us by installing this Anaconda distribution, which is Python with a bunch of extra stuff included.

I know that's a lot of names that all have Python and stuff in them, but we're going to continue to use all that tools, but we're going to be adding something, and that is the Pandas library. We learned last week that a library is basically a big pile of code that you can pull into your code, and the Python-specific term, the technical term for a library is a module, and we will be using the Pandas library, Pandas module. People always ask what Pandas stands for.

It's more of a historical name. It's a cool name. It's a cute name.

We all like pandas. We all like bears. Maybe some of us don't, but I do.

But the name basically is short for panel data, or panel datas, I guess, and panel data is a type of data related to a series of data snapshots in time, so another kind of connected to something that we call time series data. For example, stock market data is often time series data, and that's what Pandas was originally designed for, was specifically for time series, so data that is mapped over time, but it's now used for a much wider variety of things, so that's why I say it's a bit of a historical name. It doesn't mean quite as much, and we work with a lot of things that aren't panel data in Pandas.

(11:11 - 12:35)

Cool. So, let's kind of just jump in, and the first thing we're going to do is, you know, the end of the last workshop, we learned about importing libraries. We imported the random library, and we got a little work done with the random library, which allowed us to, you can do things like generate random numbers, pull random items out of a list.

However, we now are going to import a new library, which is our Pandas library, so I'm going to start with sharing my screen, and it occurs to me, Sarah, if you wanted to, you know, back us up by recording locally, you know, if you want to ask for permission for that, that would be fine, too. We should be recording. Oh, yeah, so I'm trying to record through QuickTime player, because I'm worried that, yeah, that two Zoom recordings are

going to clash with each other in the middle, so hopefully... No, no, it's not.

If you're doing it one way, don't do it another way, then thank you. Yeah, yeah, yeah, so we'll hope for the best. Yeah, yeah, I think we're going to be good.

All right, so let's go ahead and share that screen. I'm going to make sure I share the sound, which is the most important thing. Share sound and share.

(12:37 - 13:16)

Screen. "You have started screen share." Okay, that's my notes, and we are going to start by opening... "Start window.

Participants can now"... The... We're going to start by opening the Anaconda prompt, so that is the command line application that comes with Anaconda, and it has a few little extra things added to it, such as IPython, so let's start typing "Anaconda. Anaconda prompt left"... It came up right away for me, but you may need to type a little more, so you want to type Anaconda space P, and then hit enter when that pops up. Remember, you don't want Anaconda Navigator.

(13:16 - 13:49)

Okay, you want Anaconda prompt. All right. Now, once you're in the Anaconda prompt, and I will expand it, and I will also get rid of... I think I found a way to get rid of that little thing, so you can just have the whole screen be our command line environment, and also, you can hear in the background, I already have NVDA running with a little bit of a nicer voice, just like I switched to last week, okay? So, let's do... Start by... We're typing IPython.

(13:49 - 15:04)

It's all lowercase to get into our IPython environment, just like we did last week. IPython, I, and then the Python, like the large snake. "Python 3.8.8 left..." Talking, but we hear the version is the first thing that we get when we start running this IPython process, and I am also... This is too much for me, so I'm going to first start by clearing the screen, which is to hold down the control and press L. Oh, and someone was asking that they don't hear anything when the screen clears, but I guess now that I'm thinking about it, I don't hear anything either.

What I do hear is the prompt. So, that is the in, and then a number. So, we talked about that last week, but basically, that tells you, hey, we're ready to type.

So, I probably won't do so much reviewing of that kind of work, but remember that the first workshop has a lot of detail on reviewing... We're typing stuff in, reviewing output, and parsing the data by listening. So, when we get output, then we want to make sure

that we're listening for the right things. So, let's first start by importing Pandas.

(15:05 - 16:35)

"I-M-P-O-R-T, space. P-A-N-D-A-S, space." And I don't seem to have word review on, so I'm going to try to turn that on.

Speak typed words on. There we go. "I-M-P-O-R-T, import."

P-A-N-D-A-S, Pandas." Okay, import Pandas. I did the space, just hear it. Import Pandas. So, now we have Pandas imported. If you type Pandas by itself, you will get back, it'll just say this is a module, a module object.

"P-A-N-D-A-S, out left bracket to right bracket, colon, less module Pandas from C colon backslash." Less module Pandas, that means the less than sign, and then it gives us some information on the object. So, when you get information on objects, they tend to be wrapped in a less than and a greater than sign.

So, now we're going to be using this Pandas module pretty extensively today. So, let's get started by talking first about thinking about one-dimensional data and two-dimensional data, and then I'll show you how to create one and two-dimensional data in Pandas, okay? So, what is one-dimensional, what are dimensions in data? What is one-dimensional data, two-dimensional data, what other kinds of dimensional data, and then also zero-dimensional data, okay? It sounds fancy, but it's not really as complicated as it sounds. So, imagine you have a list, and I want to use as my example here something that is consistent.

(16:36 - 17:54)

So, let me make sure that the list is the same as it is in our code. So, I think ... Yeah, let's use 10, 5, and 8 since that's what we use in the code, okay? So, imagine you have a list that has three items, 10, the number 10, 5, the number 5, and 8, the number 8, okay? And you can sort of, in your mind, imagine this, okay? And I'm not going to use the word visualize here, right, because it's not visual, it's spatial. So, all the data stuff we're going to be doing today is not visual, except incidentally, because we're using this program that was designed by, you know, people who do things visually.

It's spatial, okay? So, there's, you know, whether or not you're doing this visually, you have left, right, up, down, forward, and back, okay? Different dimensions, okay? So, in your mind, I'd like you to imagine that we have a line. So, we have a left item, a middle item, and a right item. And I don't know however it works in your mind.

My mind works different from your mind, but imagine that, okay? You can use your hands or whatever if you want to. You've got your hand, your head, and your other hand. And imagine, now I've already forgotten the numbers, but they are 10, 5, and 8, okay?

So, we have 10 on the left, 5 in the middle, and 8 on the right.

(17:54 - 19:37)

Now, that basically is, you could draw a line, okay? And place each of those items on the line. And that basically is one-dimensional data, okay? So, and the characteristic, the way you know it's one-dimensional data is that we can describe where something is in that line with a single number, okay? So, we could say, remember, we always start at counting and programming from zero. Zero is the item on the left.

One is the item in the middle, 10, 5, and 8. 10 is the item on the left. 5 is the item in the middle. 8 is the item on the right, okay? So, we think of that as like an address.

We're going to use a fancier word in a minute, but you can think about, it's kind of the location or the address of the object, okay? And what we use in programming is, in this Panda specifically, we'll use this word index to describe the location of an object in a data set, okay? But in a one-dimensional data set, you can describe where something is just with one number, okay? It can be a big number, small number, whatever, but it's just one number, okay? So, let's create a, well, let's, I'll hold, I'll quickly just describe what two-dimensional data is. And then we'll, we're going to go back and work with one-dimensional data. But two-dimensional data requires that you have two numbers describing where an item is in the data set.

So, if we had a list, so, imagine we have two lists, and I won't do, I won't create this in Python because we're going to be working with this in a minute anyway. We'll talk about it again. But imagine in your mind now, you have a list, okay? Or a line like we did before.

(19:37 - 23:48)

Now, imagine that instead of 10, 5, and 8, we now had each of those items be another list, each with three items, okay? So, we could, it doesn't matter what the items are, but imagine that it's three months. So, June, July, August is one of the items. Then we have numbers 10, 5, and 8. And then we have, say, another set of numbers, smaller numbers, 1, 2, and 1, okay? And we'll, we're going to create a data set like that in a minute.

But we're, let's not talk about what those represent in a minute. We're, it's going to be a simple budget, basically. But now, we have data existing in two dimensions, okay? So, we have, you can imagine a line going from left to right.

And now you can also imagine in that line, things are going up and down, okay? So, the first address tells you where it is in the left and the right. And that is, so, we could say zero, that's the first list. And then we give another number.

We could say, then we have that first list is June, July, August. That zero item list is June,

July, August. Then we could say one, that would be July, okay? So, we can now describe in this list of lists that we've created where, where items are.

But now we need two numbers, okay? And that's what we call two-dimensional data. If you can describe where an item is in a data set with two numbers, it's two-dimensional data. And then there's also zeroth-dimensional data.

It's not very useful to talk about. But any item that isn't a list or a sequence of items is one, is zero-dimensional data, okay? So, for example, if I create a string, like we did last week, quote, "H-E-L-L-O," hello. So, I created a string called hello, okay? Or with the text hello in it.

And that is zero-dimensional data, because there's no dimensions to it. It's just a point in space, okay? It doesn't even mean anything to describe its location. Or you could pick an arbitrary number to describe its location, because it's not, it doesn't exist in space, okay? It doesn't have a spatial dimension, okay? And then the other, there's other kinds of dimensions that are a little more abstract.

But basically, imagine, you know, we took our line and we added lists to it, right? Now, imagine if we took every item in those secondary lists, in the list of months and the two lists of numbers. So, imagine that we took those items, and then we replaced those with lists. And now we would have three-dimensional data, okay? And you could then, if you wanted to imagine it in your mind, you would maybe use forward and back.

That's another dimension. Because in our physical world, we have three dimensions. Then you could add more dimensions, because you could keep replacing items with lists.

So, you can actually have an arbitrary number of dimensions, but after a point, it's not easy to map it onto our physical world. So, it becomes harder to imagine it in that three-dimensional space. But it is possible to have those, and they're actually very useful.

And if you hear a lot about machine learning these days, and this is a gross oversimplification, but basically what all those models are, they're basically lists of lists of lists of lists. They're very complex. If you look at a neural network, it's basically these complicated lists of lists of lists of lists, okay? That's a gross oversimplification, but it's functionally what they are, okay? So, they actually are very useful.

We're not going to get into those n numbers. We're going to use one and two-dimensional data in this workshop series. So, let's get into it.

I'm going to show you how to work first with one-dimensional data and then we'll move on to two-dimensional data where things get really fancy. But you'll notice we switch. We're not going to do one-dimensional data and move on.

Once we learn about one-dimensional data, we're going to constantly be coming back to

this information because we're constantly going to be taking our two-dimensional data and making it one-dimensional, okay? And pulling out a one-dimensional element from the two-dimensional data set, okay? So, that's how we work in PEMD. So, let's go ahead and create this list, which is going to be a budget list. And I want to use the same numbers.

(23:48 - 24:27)

In left bracket, I search. Budget equals left bracket 10, 10, 5, 15, 15 right bracket. Budget equals left bracket 10, 10, 5, 15, 15 right bracket.

Okay. I just want to make sure things line up with the curriculum. Yeah.

I just remember that last time you made it a friendlier voice and slowed it down. Oh, I thought this was the friendlier voice. No, this is your normal voice.

"NVDA menu, preferences, tool submenu, code factory submenu, help submenu H. Configuration profiles dialog. Programming. Recording.

(24:28 - 27:31)

Nice line sounding left paren. Editing. Manual right paren." Let's try this and see if you hear a difference. I think this is the nice voice. I think this is your normal one, or your preferred one.

This is my preferred, so it's actually. Okay. I'll slow it down for people.

Yeah. That is slower than I do do it, but I guess I'm not. Okay.

I think it's audio, but I want to prefer a voice. Yeah. "Audio output device, colon, combo box, Microsoft sound mapper, colab, audio ducking mode, colon, combo, volume of NVDA, volume of NVDA, okay, button, audio, audio, audio, audio, three of speech, two of 15, speech property page, change, voice, colon, combo, rate, colon, slider 60 alt plus 59, 54, 52, 50."

How's 50, do you think? Did I get to make it even slower? Let's do 48. "48." That sounds pretty slow to me.

Is that good? Yeah. When you listen to robots all day, maybe your idea of what is a fast or slow voice gets a little scared. I do know people who listen faster than I do, but I wouldn't even say I'm that much of a speed demon.

It is easier to follow along when the voice is a little slower, so thank you for that. Okay. So let's go ahead and create our budget.

I think it was Emacs, budget equals left bracket 10, 10, 5, 15, 15 right bracket. 10, 10, 5,

15, and 15. Sorry, that voice is going to be a little faster because that's a different program, and that's really just my own notes.

So let's go ahead and create budget. "B-U-D-G-E-T, budget equals left bracket 10, 10, 5, 15, 15". I will review this.

So it's going to be budget equals space open square bracket, and then write the numbers 10, 10, 5, 15, and 15. Okay. So budget equals in left bracket 5 right bracket.

And you remember, you hear the in, and that basically means there's no output. It skipped the output. So when you assign a variable, you don't hear output.

(27:31 - 28:29)

And we can get our budget back by typing budget by itself. That's a technique we use all the time. "out left bracket 5 right bracket colon left bracket 10, 10, 5, 15, 15." 10, 10, 5, 15, and 15. Okay. So that's our budget.

Now, remember in regular Python, we can do things like use what's called slicing to pull out specific items. So basically, the syntax is this, and we're going to do similar things in this workshop. Type the budget variable. "B-U-D-G-E-T, budget." And we can type open square bracket and then give it a number.

"out left bracket 6 right bracket colon 10." 10. So 10 is our output.

We asked for the first item. It was budget, open square bracket, 0, close square bracket. So Python gives some functionality with lists.

(28:30 - 28:34)

And these are one-dimensional data types. Okay. Lists are one-dimensional data types.

(28:34 - 29:05)

But what we're going to do is we're going to create a new data type, which is a it's going to be called a series. Okay. So the series is basically you can think of it as a really fancy list.

Okay. That lets us do all sorts of things. So let's go ahead and load in our budget data type or our budget series.

Okay. Now, I am going to use a very short variable name here, and I'll give an explanation. So far, I've tried to use very descriptive variable names.

(29:05 - 30:53)

You don't see me doing X equals this, Y equals this, only very rarely. And I recommend

keeping to that. But I'm going to use a short variable name here, and then I'm going to kind of tell you why I chose that.

Okay. So I'm going to say S, like the letter S. S equals space. And then we want to use pandas.

Now, then do a dot. Got a reaction there. And there's our dot.

So now, so far, we have S space equals space pandas dot. And now, let's you type capital S series, and I'll explain that part too. And you heard that little blink, you know, that little beep.

That's how I have it set to get the capital letters. So when you hear that little beep, you're hearing capitals. Then open parenthesis, and then give our budget variable inside the parenthesis.

So B, U, D. And I'm going to let IPython complete it for us by pressing tab. Remember, that's something we learned in the first one. So I type B, U, D. Yes.

So it added B, U, D. It added G, E, T for me. And then I'm closing the parenthesis. So what I have here is S space equals space pandas, the library, dot, capital S, series, S, E, R, I, E, S. Okay.

Open parenthesis, and then our variable name, budget, close parenthesis. Okay. It takes a minute.

(30:54 - 31:03)

Computers. Okay. So it assigned the variable, so we don't get any output.

(31:04 - 36:08)

But we can get an idea of what the series representation is. Remember, the representation is a fancy word for what the output is like. Okay.

We call that a representation. So when we type a variable by itself, what we're getting back is the representation. So we're going to take a look at this representation of the series in a second.

First, let me just explain a few little things. So why did we use a short variable name? This is another unsatisfying data science-y answer. There is a convention, an informal convention in data science and Python that there's certain variable names that are used by conventions.

So if you only have one series, people often call it S. Okay. And there's another convention that's similar to that with data frames, which we'll talk about in a minute, but

we will cross that bridge when we come to it. There's also a convention to shorten many of the library names.

So often you'll see pandas called, if you follow a tutorial, you'll see pandas being called PD. Okay. So there's a way to shorten the library name, but I think it confuses the issue a little bit.

So we're just going to use the full pandas name. But if you see PD in a tutorial, they're talking about pandas. Okay.

These are data science conventions. I didn't make them up. I would also say anytime you have more than one series, go ahead and use a descriptive name, because once you start doing S1, S2, S3, then you're going to kind of get yourself confused.

Okay. So, but I want you to know about this convention, so I'm going to use it. All right.

So the other thing I want to explain is why is the series in this line up here capitalized? And the short version of that is that it is an object that we're using in Python, and the technical word would be it's a class that we're importing. We're using the series class, which we're not going to get into. But basically it's not a function that we're importing there.

It's another kind of object that we're importing from pandas library. You can basically just figure, hey, you know, it's pretty much the function for now. Later you might learn why the class is and what these kinds of objects and stuff are for now.

Just think of it as a way to make a series object. It's almost like a little factory. That's the short way of thinking about a class.

It's a prototype for making other objects with. Okay. So S equals pandas.series, and we give it our budget, pass the budget in, and now we have our series object.

So let's type S by itself. Press enter. S in left bracket, eight right "bracket, colon, S. Out left bracket, eight right bracket, colon.

Zero ten. One ten. Two five.

Three fifteen. Four fifteen. dtype colon.

In left bracket, nine right bracket." I kind of like that that was a bit slower, because it can be difficult to parse some of this output sometimes, and when you're hearing it for the first time, it's confusing. But what you heard was zero ten, then a pause for a new line, one ten, pause for a new line, two five, pause for a new line, three fifteen, pause for a new line, four fifteen, pause for a new line.

And then the last line said D type. It said it some weird way. It was like D type or

something.

But it said D type for data type, and then it said int64. So that's telling us the data that we have in the series is basically it's saying it's integers. Okay.

And this is the format of a series representation that you have two columns of numbers, one on the left spatially, one on the right spatially. The left column, let's talk about the right column first. The right column are the values, and that's the technical term.

They're the values that we had in the list. Okay. And value is a word for the actual data.

Okay. So when you're distinguishing the actual data from something else, like, for example, a label you're putting on the data or a variable name or something like that, then we say value to mean the actual data. And in this case, the actual data is the stuff that we put in budget.

Okay. So on the right column, so the second number in each line, when you're hearing it, you know, line by line, is the value. The left number is the index, and the index is basically it's a label for each row of the data.

Okay. By default, the label or the index, label is not the technical term, but it's an accurate way of describing it. The index on the left is a number that, an incrementing number from zero up to some other number.

Okay. So it's basically just a, it's the same as slicing. When we extract something from the list, it starts at zero, goes to one, two, three, four, and so on until the end of the data set.

(36:09 - 37:13)

Okay. That's the default index, but we're actually going to work with that in a minute. Okay.

Anything else to know about this representation? And, oh, the other thing I wanted to say is that what we call that in programming, when a number just increments like that, or we want to create something like that, we call that a range. Okay. So if I say, oh, it's a range, I just mean it's numbers going up.

Okay. And there are fancier ranges where numbers go up by two or something like that, but mostly they go up by one. Okay.

So, and we'll, we'll return to that later. Okay. So it's a range, the index is a range from zero to four, and the other, the right column is our actual values.

Okay. So now we actually get into some pretty cool stuff here. This is actually pretty neat stuff.

So I've kind of explained a lot of, you know, procedural stuff, but now let's do some cool stuff. So let's start by doing S, and we're going to use a method on the, this series object that we have. So S dot.

(37:15 - 38:45)

It takes a while for it to say dot. S dot. And let's use the mean method, M-E-A-N. "M-E-A-N." Open parenthesis, close parenthesis. Mean.

Right parenthesis. Skip saying the left parenthesis there, but it's S dot mean, left parenthesis, right parenthesis. We're not putting anything in there.

Usually when you don't pass anything into a method like that, it's going to operate on the, the object that the method is contained, contained inside. So if we say S dot mean and run that, then it's going to operate on S probably. Not all of us, but mostly, but usually.

Okay. So let's run that. "Out left bracket, nine right bracket, colon 11.0." So what was the mean? Maybe you remember from school, maybe you were mathematical, so this is obvious to you.

The mean is the average. Okay. The average is you take all the numbers, the sum of all the numbers, you sum them all up, you add all the numbers together, and then you divide by the number of numbers or the, you know, so in this case, you add up, you know, 10, 10, that's 20, 25, 15 and 15 is 30.

That's 55. And then we divide by five, we get 11. Okay.

That's the average. And the average is a very useful number. We'll be using this a lot.

Why isn't it S dot average? Well, it's just, that's just a decision they made when they created this library. Okay. Remember all of this stuff is created by people.

(38:45 - 42:42)

So they were made just different kinds of decisions. Sometimes they changed those decisions and then the language changed. So hopefully not too often, but so that's, so that is one method that we have inside our series object and it allows us to get the mean of all the items in there.

And so let's do a couple of others that are along these same lines. So let's do S dot median. "S S M E D I A N" dot median, open parenthesis, close parenthesis. I'm going to type a little faster here. Just S dot median, open parenthesis, close parenthesis. 10.0. Okay.

So the median is the middlemost number. So if you sort all the items from least to

greatest, and then you pick the middlest item, it would be 10. Okay.

If there's an even number of items, then you'll get the average of the two middlemost items. Okay. So if we had six items, maybe we would have got the average of the two items in the middle.

Okay. Now this is not that intuitive, right? Because our list is 10, 10, 5, 15, 15. You're like, oh, it's in the middle of number five.

No, it sorts first. And in fact, there's also a very useful method to sort our values. So let's do S dot S S dot, it takes a while to do that, sort underscore values.

And I'm going to try to fill it in with tab. So it's S dot sort underscore values. Okay.

And I do encourage you to use that tab. And it does take a while for it to finish it, filling it in. But what it does, it gives you some peace of mind that you're not going to do a typo.

You type a little in, and if it fills it in for you, then probably you're on the right track. Okay. So I do recommend using that tab a lot.

So it's S dot sort underscore values, open parenthesis, closed parenthesis. And sometimes it gets that. I'm going to stop it and explain what we're hearing.

So we're getting our series, but now the items have been sorted into a new order. And it kept the old index, which is the range from zero to four. And now, you know, the first item used to was the third item, it's two, is five.

So five was the lowest, it gets moved to be to the beginning item. Okay. So the index tends to stay the same when we run things like this.

Okay. So we kind of get the original position of these items rather than the new position. Okay.

So the index doesn't automatically update. That's actually usually good. And then now we have a series, it goes from five, 10, 10, 15, 15.

Okay. So it's sorted from least to greatest. That's what it does by default, least to greatest.

Now, if you kind of don't like how Pandas is printing so much out to us, we can also do S dot values and only get the values. We don't hear the index. Now, the index is often useful to hear, but sometimes it's not.

So we can just do S dot values. S dot values, no parentheses. "Out left bracket, 12 right bracket, colon, array left, left bracket, 10, 10, 5, 15, 15."

So we just got the values from the original list, not the sorted. You'll notice that that

didn't update our variable when we sorted the variable. We would have to overwrite the variable if we wanted to update the variable.

Okay. That's It's usually good. But so the values is pretty useful.

(42:42 - 42:52)

And what did it return back to us? It said it returned an array. Basically, a lot of pandas will return things that look like lists. They look like lists.

(42:52 - 42:57)

They quack like lists. They walk like lists. They smell like lists.

(42:58 - 43:06)

They sound like lists. But they are not lists, OK? And in fact, there's a word for that in Python. It's called duck typing, because it quacks like a duck and walks like a duck.

(43:06 - 43:21)

It is a duck. So instead of saying list, I will say a list-like object. And that means it's an object that basically, for all intents and purposes, lets you do the same things with it that you could do with a list, OK? In most cases, that's true.

(43:21 - 43:29)

In some edge cases, it might not be true. In most things, you'll be able to do with it, OK? So I'll say a list-like object. And then there's a bunch of them that we'll work with.

(43:30 - 43:39)

But basically, you can think of them for now as lists. But remember, they're not technically lists. I just don't want to tell you anything that's not true, OK? So in this case, it's an array, which is a list-like object.

(43:40 - 43:49)

And it gives us our values that we have. So it skips the index, OK? So that's pretty useful if we want to just know what the values are. Let's run through a couple of extra cool ones.

(43:49 - 44:03)

I'm not going to dwell on these. s.min. s.min, open parenthesis, close parenthesis. "Out left bracket, 13 right bracket, colon, 5." 5, that gave the minimum value.

(44:03 - 44:25)

s.max. "Right, out left bracket, 14 right bracket, colon, 15." 15 gave the max value. What else is useful? s.std. Is that what you're thinking? "Right, in left, out left bracket, 15 right bracket, colon, 4.1833." The standard deviation, that's std.

(44:25 - 44:30)

s.std, open parenthesis, close parenthesis. This is maybe one you haven't heard of. Maybe you have.

(44:31 - 44:47)

It's a measure of the variability of the data. So it tells you basically how spread out or distributed the data is. If the data is clustered around the mean, then it will have a low standard deviation.

(44:47 - 45:03)

If there's a lot of outliers or a lot of the data is away from the mean, then you're going to get a high standard deviation. It's actually a pretty useful number once you learn to interpret it a little bit. And then there's others.

(45:03 - 45:23)

There's s.count. "Out left bracket, 16 right bracket, colon, 5." 5. But we might throw a few more at you over the course of this. But basically, if there's anything you would kind of conventionally do with a list of numbers, you can sum things. s.sum is provided to you.

(45:23 - 45:34)

So there's a lot of methods here. s.sum. "Right out left bracket, 17 right bracket, colon, 55." 55 is the sum of all the items in the list.

(45:34 - 45:42)

So there's a lot here. But we're going to move on. But remember that there's just a lot provided to you as methods in here.

(45:42 - 45:48)

And remember, we had s.count here. But we can also use len. Remember, we had len also works on our series.

(45:49 - 46:07)

L-E-N, open parenthesis, close parenthesis. Or sorry, L-E-N, open parenthesis, s, close parenthesis. L-E-N, len, s, "out left bracket, 18 right bracket, colon, 5." And that will also

work on those list-like objects that might not necessarily have a count method inside them.

(46:07 - 46:19)

Len will work on those as well. So like our s.values, it will also work on those. So what I want to do now is we're going to keep our budget variable around.

(46:19 - 46:34)

And what we're going to do is we're going to think a little bit about two-dimensional objects. Before we move on, I just want to double check that it's not anything I wanted to show you that we're moving on. So let's go to other.

(46:35 - 46:48)

... ... We did that. ... ... Describe.

(46:48 - 46:54)

... ... Describe is one I almost skipped. Because I don't personally use it. But I want to use it to illustrate a point.

(46:55 - 47:14)

So let's learn one more, and then we'll move on. So it's s.describe, "d-e-s-c-r-i-b-e," open parenthesis, close parenthesis. And now it's going to give us a lot of output.

(47:14 - 48:04)

Let's listen to some of it. "count 5.0000 mean 11.0000 std 4.1833 min 5.0000 25% 10.0000 50% 10.0000 75% 15.0000 max 15.0000 type colon float 64 dtype float 64 in left bracket 20 right dtype cortana window" dtype float 64. So that means these are floating point numbers in this.

(48:04 - 48:28)

What is all this? Now I kind of let that all run. So you'd hear all of it to kind of make a point, which is that. So a lot of times the functions like this or methods rather like this that print out a whole lot of information are pretty widely used by sighted data scientists, especially when they're starting out, they're doing what's called exploratory data analysis.

(48:28 - 48:38)

And you don't even know the questions that you're asking. You might just say, hey, print out all the normal statistical things. So describe prints out the standard deviation, the

mean, the median, the count.

(48:39 - 48:55)

And then it also the 25th and 75th percentiles. That's kind of a it's basically it means if you divided it, the data set in half, the first half, the median of the first half would be the 25th percentile. The median of the second half would be the 75th percentile.

(48:56 - 49:23)

And another way of saying that is it would be the item closest to the quarter and the 75th percent mark in the data set. OK, so these kinds of functions are commonly used by sighted data scientists, and we can use them to especially if we're preparing something for sighted people to use. But in my experience, and I also my recommendation is that you learn the specific ones and you ask for more specific information.

(49:24 - 49:52)

Why is that? Because the answer is, do you really want to wait for all of all of this stuff to print out when you just maybe want one or two of these things? It's actually faster for you to print out each one individually, probably, and to listen to that, because a bunch of these, you're almost certainly not going to want all of them every time. When I teach sighted students, they ask me, why don't we always use describe? Because it prints out so much information. And my answer is describe is not that useful.

(49:52 - 50:07)

If you say we're going to do something with the mean, you want to do something with the median, which you often do want to do. Also, it's just a lot of information, you know. If you're preparing something for someone else, you may just want to be like, hey, what's the mean? And your intent is more obvious.

(50:07 - 50:25)

That's what I say to my sighted students. And for you guys, I say, you might say, well, why would we use describe? And my answer is, well, sometimes you're going to be working with sighted people, and sometimes you're going to be preparing something like an exploratory data analysis that you know other people are going to read, so you'd want to use it. But I would say generally for your own use, we're not going to want to use stuff like this.

(50:26 - 50:40)

And we're also not going, we'll talk more about this, but we're also not really going to use those big, long representations that much. You need to know what they're like, but we're not going to use them as much. Once you're a little more experienced, once I

teach you a few more things, you're not going to use that long stuff as much.

(50:40 - 51:13)

We're going to get small amounts of information back, very focused, which is good for screen reader users who don't want to hear so much stuff, okay? Okay, so that was the point I wanted to make with describe. Okay, moving on now, we're going to work on making our two-dimensional data, okay? One thing we touched on here, and please, if there are questions that are coming up a lot in the chat, maybe one of the helpers could kind of get on and kind of filter it up to me. Anything like that, this is probably a good point for that.

(51:14 - 51:36)

I think it seems like people are following along pretty well, though again, a reminder to everyone that if you do have questions, don't hesitate to pop them in the chat. One thing we did discuss briefly is that these, for instance, this s.sortValues won't actually change the order of your series. So don't worry, you're not messing around with your series.

(51:37 - 52:00)

You can save a new variable as s.sortValues. So for instance, s.sorted equals s.sortValues, and that'll save it saved. I think there's also a way to put in place equals true, and that will overwrite your original variable for the series. But all that, that's a little bit of a tangent to say, not to worry, you're not changing anything here just by printing these things out.

(52:04 - 52:17)

Thank you so much, Sarah. Okay, and if people have other questions, drop them in the chat, and if people are having the same question, I can also answer. Or if it's an involved answer, just let me know, helpers, and I'll explain on the mic.

(52:18 - 52:29)

So let's talk about, let's work toward creating our two-dimensional dataset. First, we're going to spend just a little time on this idea of labeling data. That's not a technical term, it's just a descriptive term.

(52:29 - 52:50)

Basically, giving names to things is a very common thing in programming, okay? And the index is one way we did that. So now we're going to have to learn one other way to do that, and that is, it's what's called a key, okay? And a key is a way you can look up some data. So since we're moving on to a new section, I'm going to clear the screen, Control-L.

(52:52 - 53:33)

We already have 19 inputs, so that's pretty cool. And then let's do, I'm going to explain a new datatype to you, but this datatype is kind of just a stepping stone, okay? And this datatype is called a dictionary, okay? And basically, I'm going to show you the dictionary, but for the interest of time, if you think you can go ahead and put it in, put it in, but we're going to do another one right after. So you may just want to sit back and actively listen, and then you'll have the practice to make the dictionary in the next section, okay? Just for the interest of time.

(53:36 - 54:48)

So what we're going to do is we're going to create a little phone book, okay? And if you remember, this makes me feel very old, but many, many years ago when I was a There was such a thing as a phone book, and it was a big, thick book that you could use to put down a whole lot of them to stand and reach a high shelf or something, which actually probably made it the most useful, but you could also use it to look up people's phone numbers. And it would basically be their name, and then you would have, it would be their name on one column, and then on the other side, it would be their phone number, okay? And you could use their name to look up their phone number, because the names were all in alphabetical order, okay? So what we're going to do is we're going to create a phone book, because that's essentially what this thing I'm going to show you is, this dictionary. It's a way to put, it's a way to contain data so it can be easily looked up, okay? So you have, and what it consists of is, it's basically like, imagine it's kind of like a list, so it's one-dimensional data, but each item has its own name, okay? And that name is called a key, and we can use the key to look up what the value, which is the data.

(54:48 - 55:16)

So the key is the name, the value is the data, and we call those key-value pairs, okay? So we're going to, I'm going to create a phone book, and then I'm going to look something up in it, but you can decide if you want to follow along, go for it. I'm going to do it kind of quickly to explain, and then we're actually going to create our own dictionary in the next section, and you'll need to create that dictionary, so up to you if you want to create the phone book, okay? I'm just pressing delete a bunch of times. So I'm going to type phone underscore book.

(55:17 - 56:00)

"P-H-O-N-E," phone, that's our variable name, space, book, equals, space, equals, and now we're going to use a curly brace, which is a, if you're using, we're doing, using square brackets a lot. If you hold, if on an English keyboard, if you hold down shift and you press the left square bracket, it'll create a curly brace, and it says left brace, so it's a new kind of syntactic structure, okay? Now, you can, once you start a brace, you're

allowed to create your own new lines, okay? And sometimes that can help you keep track of things. So I'm going to hit enter, but you don't have to do this.

(56:00 - 56:48)

I'm going to hit enter, and then I'm going to add my first key-value pair. Okay, I hit enter just for my own sanity. And then I'm going to do quote, I'm going to say my name, Patrick, and this is going to be my phone number, okay? "Quote," so that's quote, Patrick, quote, and then that's the key. So that's what I'm going to use to look up this data. Then I do a colon, and the colon is, it's kind of in the middle of the middle row on the keyboard. If you go all the way to the right on the letters in the middle on the English keyboard, it's the first special character to the right of the middle row that has ASDF on it.

(56:49 - 56:55)

And it's, you hold down shift to access it. That's colon. And then I'm going to do another double quote.

(56:56 - 57:19)

Quote. And then I'm, oh, actually, you know what? I'll make these integers. So I'm just going to say my phone number, which is, of course, and you guys all know my phone number is 999-999-9999. Now, you guys can call me anytime. Patrick and my phone number is 999-999-9999. Colon.

(57:20 - 57:31)

And then I did a comma and I hit enter again. So that, so the phone book is, it's phonebook equals Curly Brace. And then I made a new line just for my own sanity.

(57:31 - 57:40)

You don't have to do that. Then I have a string, Patrick, colon, and then an integer, which is our value. That's the actual data we're storing.

(57:42 - 57:47)

999-999-9999. And then I do a comma. And now we're doing another key value pair.

(57:47 - 57:58)

So the pairs are separated by commas. And within the pair, you have a colon to separate the key and the value. Now we're going to do Sarah, my co-instructor.

(57:58 - 58:27)

Quote, "S-A-R-A-H," Sarah, colon, space. So Sarah, as a string, quote, Sarah, quote, colon. And of course, we all know Sarah's phone number, which is 111-111-1111 Of course, you have to remember the plus 44 for the UK. Yeah, otherwise you're not going to get through to her. OK.

(58:29 - 58:36)

Now, I'm not sure, Sarah, make sure you call in the right time frame. So it's five hours forward there, too, as well. So press Enter.

(58:37 - 58:47)

And then go through the right brace. Remember, the brace is above the square bracket. You hold down Shift, right brace.

(58:47 - 58:55)

And I know this is probably the most complicated thing we've put in so far. So no shame if you get it wrong. I get these wrong all the time, because I kind of start rushing.

(58:55 - 59:01)

And I'll leave off a quote. I'll leave off a colon. It's very easy to mess these up.

(59:02 - 59:24)

But I didn't do it correctly. So phone book equals left brace, new line, the string Patrick, colon, the integer, not very long integer with a lot of nines, comma, the string Sarah, colon, very long integer with a lot of ones. I put another comma there.

(59:24 - 59:37)

It's not necessary to put a comma at the last item. But I recommend it, because you'll not mess yourself up as much. And then I did a right brace, OK? That's our dictionary.

(59:37 - 59:45)

And now that we have the dictionary defined, we can type phone book. Phone book. I just said book.

(59:45 - 59:56)

It's phone underscore book. "Out left bracket, 21 right bracket, colon, left brace, Patrick, colon, 9999999999." This is the representation of the object.

(59:56 - 1:00:10)

Now, to pull out my phone number, now, there's only two numbers in here, right? But

imagine there were thousands of numbers, OK? And this would be more useful. But let's do phone underscore book. And I'm not going to go slow.

(1:00:10 - 1:00:26)

But phone underscore book, left square bracket, book. And then I type in quotes Patrick, OK? So quote, "quote, Patrick, quote, right bracket." And then I end with a bracket.

(1:00:26 - 1:00:37)

So it's just like our slicing syntax. But instead of giving a number, we give it a string that is our key, OK? And you can use other things for keys. You can have keys that are integers.

(1:00:37 - 1:01:20)

You can have keys that are actually more complicated things that I actually don't recommend choosing. But the most common ones would be integers, floats, or strings, OK, to pick as keys, all right? So in this case, it's a string, which is Patrick. "Out left bracket, 22 right bracket, colon, 999, 999, 9999." OK, just in case you've forgotten my number, there it is, phone underscore book, open square bracket, quote, Patrick, my name, which is the key, close the quote, close the square bracket. And I hit Enter. And then it printed out the data, which is in that key value pair.

(1:01:20 - 1:01:48)

So remember, the Patrick was kind of a label on the data, which is the phone number, OK? And this is very common. A lot of things in programming are really just about giving things names, variables, indexes, and now key values, OK, key value pairs, all right? So now I've kind of shown you how to do that. Now let's create together a new dictionary, OK? And this is going to be our two-dimensional data set, all right? I just want to do a time check.

(1:01:52 - 1:02:02)

"2 colon 6 PM. 2 0 6, right? 2 colon 7 PM." 2 7 PM, that's, OK, it didn't say 0 7, but that's fine.

(1:02:03 - 1:02:31)

I'm assuming it's 2 0 7. I think we should be OK for time. OK, so now let's create our dictionary that we're going to use for our two-dimensional data set, OK? Let's create three variables with each a list of five items. And then we're going to use those to create the dictionary, because I think that's easier than creating it all in one go, OK? So we already have our budget.

(1:02:32 - 1:02:45)

But if you haven't created the budget, it was budget equals square bracket 10 10 5 15 15, OK? That's our budget. But if you already have the budget assigned, you don't have to do it again. And now let's get our other two.

(1:02:45 - 1:03:01)

And I want to do it the same with the tutorials. I'm going to pull it out of my notes. ... ... "June, July, September, October." So we have June, July, September, October, November. So it's going to be a list with five items.

(1:03:01 - 1:03:09)

Each is a string. June, July, September, October, November, OK? So let's create that one. And we call that month, I think.

(1:03:09 - 1:03:12)

Month equals left bracket. June, July, September, October, November, right bracket. IPython.

(1:03:12 - 1:03:20)

So let's do month equals. "M-O-N-T-H month equals space." Open square bracket.

(1:03:20 - 1:03:29)

Left bracket. Quote. And then I'm going to say, it started in June, right? Month equals left bracket.

(1:03:29 - 1:03:53)

June, July, September, October, November, right bracket. That's a lot of text. "June, comma, space," July, "July." Remember, they have to be surrounded by quotes, and you separate them with commas. Quote, space. June, July, September, October, November.

(1:03:54 - 1:04:08)

I'm not going to type all that in one character. "Quote, right bracket." But it's month equals, open square bracket, June, comma, July, comma, September, comma, October, comma, November, comma.

(1:04:09 - 1:04:12)

Actually, don't put the comma at the end. But you can if you want. I won't mess it up.

(1:04:13 - 1:04:24)

And then close the square bracket. I press Enter. "In left bracket, 24." OK, now I'm safe in my month variable, OK? So now we should have a budget. The budget should be five items long. The month should be five items long.

(1:04:24 - 1:04:36)

We need to create one more, OK? And it's going to be cookie budget, OK? Because I like cookies, OK? Maybe you like cookies too. I don't know if you do. But hopefully, you do.

(1:04:38 - 1:05:04)

And our little data set here is going to represent the budget. We have the total budget for each month and the budget we have for cookies each month. And you can kind of imagine this as like, imagine you're like a child in like 1964, OK? And your budget is $7 a month or something like that, OK? Just we're keeping things really simple, OK? So our cookie budget is going to be, and I'm going to use the same value.

(1:05:04 - 1:05:44)

"Month equals left bracket, June, July, September, October. Month equals left bracket. Month equals cookie underscore budget equals left bracket, 3, 2, 0, 4, 5, right bracket." 3, 2, 0, 4, and 5. So let's do cookie underscore budget. C, budget, space, left bracket, space, space, 5. I think it was 3, 2, 0, 4, 5. I went down fast there because I knew I wasn't going to remember the numbers if I didn't. But it's cookie underscore budget equals open square bracket, 3, 2, 0, 4, 5, OK? And if it's a little different for the curriculum, it's not the end of the book.

(1:05:45 - 1:06:00)

And then we want to close the square bracket. "Right bracket." Should I close it? "Right bracket." Yeah, do it. 5. OK, so now we should have three lists. In left bracket, 25, right bracket, colon.

(1:06:01 - 1:06:18)

OK, but no output because I saved a variable, OK? And now one thing I do want you to check, I know we're all frantically inputting lists right now and not really listening to me. But what I'm going to do is I'm going to use my len function to check all three lists. If any of them don't have five items, we're going to have a problem, OK? We're going to have a problem.

(1:06:19 - 1:06:32)

So let's do len budget. Len. "Out left bracket, 25, right bracket, colon." 5. Len. Cookie budget. Underscore budget.

(1:06:32 - 1:06:35)

"Cookie. Budget. 37.

(1:06:36 - 1:06:42)

In left. Cookie. Len." I got an error there, but I. "Cookie. Right. Name error.

(1:06:42 - 1:06:48)

Trace back." Did I call it something else? "In 37. In left.

(1:06:49 - 1:06:56)

Cookie slash. In left bracket, 29, right bracket, colon. Cookie slash.

(1:06:57 - 1:07:01)

In left. In left bracket, 29." Let's just see what I did wrong there.

(1:07:03 - 1:07:07)

Yeah. In left bracket. In left bracket, colon.

(1:07:07 - 1:07:12)

5, right bracket. Cookie. In left bracket, 29, right bracket, colon.

(1:07:13 - 1:07:24)

Cookie line budget equals left bracket, 3. "In left bracket, 37 dash." Am I doing caps or something? "Name error. Trace back.

(1:07:25 - 1:07:30)

In caps lock on. Caps lock off. Cookie slash.

(1:07:33 - 1:07:37)

T. 37." Yeah, I don't know what I did wrong there. I'm going to save the cookie.

(1:07:38 - 1:07:47)

Yeah, you tell me what I did wrong. I was going to say that I don't see caps lock on, but yeah. Yeah, odd, right? Yeah, the chat is saying you missed an O in cookie.

(1:07:48 - 1:08:07)

"Space." There's only one O in cookie? IE line budget. Yeah, it was cookie with one O. So

this really is like a small child in 1964, OK? But let's do, you can imagine all the R's being backward and stuff like that.

(1:08:07 - 1:08:20)

Let's do, I'll just do this. I'm going to do cookie budget spelled correctly. And there's something about programming that makes my spelling ability massively decrease.

(1:08:20 - 1:08:37)

My already poor spelling ability just decreases precipitously when I'm doing programming. It must be like a left brain, right brain thing. "Equals cookie." ... I just fixed it, basically. If you got it right, I just overwrote the variable with a new variable, OK? Or I reassigned it.

(1:08:38 - 1:09:09)

All right. So len cookie budget. "Cookie line budget. Out left bracket 33, right bracket colon." And then let's do len cookie budget. And then we're going to do len month, OK? "Out left bracket 34, right bracket colon." So now let's create our data frame. And then I'm going to explain a few things about how we work with two dimensional data. So it's a two step process.

(1:09:09 - 1:09:19)

So we first create a dictionary. And then we create the data frame from the dictionary, OK? So we have our three variables. And now let's use them to create the dictionary.

(1:09:19 - 1:09:37)

So we're going to call it monthly underscore budget. "M-O-N-T-H-L-Y. Monthly budget." Monthly budget. Monthly underscore budget equals. And then let's do an open brace.

(1:09:38 - 1:09:54)

And I'm going to press Enter just for my sanity. And then what we want to do is, in quotes, I'm going to put budget. And these are the labels for what are going to be columns in a minute.

(1:09:54 - 1:09:59)

But we'll talk about that in a minute. But basically, remember, the key is the label. Budget.

(1:09:59 - 1:10:08)

And I did lowercase. I keep everything lowercase, OK? Don't confuse yourself. And then I

did a colon and then the variable cookie budget.

(1:10:09 - 1:10:15)

And then we'll go over this again. "Cookie line budget. Cookie.

(1:10:17 - 1:10:20)

Cookie line budget." I'm too lazy. OK.

(1:10:21 - 1:10:27)

And then let's do quote. Oh, no. That's completely incorrect.

(1:10:27 - 1:10:37)

That should be budget. ... Budget, in quotes, colon, budget, the variable, comma.

(1:10:37 - 1:10:44)

And let's do, in quotes, cookie underscore budget. ... Colon.

(1:10:44 - 1:10:59)

And then let's use our cookie budget variable. ... ... And then let's do month, in quotes. "Quote." I'll go over this again.

(1:10:59 - 1:11:05)

I know this is confusing. "M-O-N-T-H, month, colon." And a new line.

(1:11:05 - 1:11:12)

And now I'm going to do the right brace. Right brace. So what do we have here? We have monthly underscore budget.

(1:11:13 - 1:11:21)

That's our variable name. Equals left brace. I did a new line, but you don't have to.

(1:11:23 - 1:11:32)

Quote, budget, end quote, colon. Then our budget variable. Budget with no quote, OK? Comma.

(1:11:33 - 1:11:42)

Next line is, quote, cookie underscore budget. Quote, colon. And then we have our cookie underscore budget variable.

(1:11:43 - 1:11:47)

Comma. Then we have a new line. And then we have month, in quotes.

(1:11:48 - 1:12:00)

So quote, month, quote, colon. And then we have our month variable, M-O-N-T-H. And then we, I did put a comma, but you don't need to, on the last one.

(1:12:00 - 1:12:16)

And then you can do a right brace, OK? I'm going to hit Enter. Hopefully it'll work for me. Now, so what we have is a variable, monthly budget, with three key value pairs.

(1:12:17 - 1:12:51)

And the key is each describing it. But we basically just kept it the same as the variable names, OK? So it's budget, in quotes, budget, the variable, cookie underscore budget, in quotes, cookie underscore budget, the variable, and then monthly, or month, in quotes, month, the variable, OK? If people in the chat want to kind of paste this in so people can copy it in case they're having issues or something, then go ahead and paste it in for people. So they have something to copy if they're having trouble.

(1:12:52 - 1:13:05)

But it is good practice. Yeah, put it in the chat. And then it is good practice to type this all in and just take your time, OK? I know it can be very easy to get syntax errors in something like this.

(1:13:05 - 1:13:13)

It's totally normal when you start learning programming to get constant syntax errors. And you know what? You'll be missing a quote. There won't be a colon.

(1:13:13 - 1:13:38)

It'll be something annoying. And the things to try are you can start again or go character by character, OK? And using the review, OK? And in fact, I think in some ways, I think it makes us more patient with stuff like that. I feel like a lot of times I've noticed when I teach sighted people, it's like they're like, they don't really think.

(1:13:38 - 1:13:45)

They're seeing it in their brain so much that they're not seeing what's on the screen. So they'll kind of be like, oh, no, but I did everything right. But it's really hard to miss.

(1:13:46 - 1:13:52)

It's really easy to miss something visually. I'm not saying it's not useful to see things visually too. But it's really easy to miss something visually.

(1:13:52 - 1:14:06)

But often, if you review character by character using NVDA, you're actually fairly liable to catch things because you're like, oh, wait, oh, whoa. You're not going to have, there's not really as many mirages that can win. OK, so we have our monthly budget.

(1:14:06 - 1:14:17)

I was just letting, giving you some time to catch up there. So we have our dictionary now. And remember our type function, our old faithful type function? We haven't used it in a while.

(1:14:17 - 1:14:41)

Let's just run it on monthly budget. P. So T-Y-P-E, open parenthesis, monthly. ... "Right paren, out left bracket 36, right bracket colon, dict." It pronounced it kind of funny, but dict. It's short for dictionary.

(1:14:41 - 1:14:55)

That's our data type that we're using here. So remember, dictionary, it's really just labeled data. And so what we did here was that we had our three lists, each with five items, saved as variables, and then we gave them names in the dictionary, keys, as keys.

(1:14:56 - 1:15:04)

Now we're going to use this monthly budget. We could pull out data from this, but we have no reason to. So what we're going to do is create a data frame in Pandas.

(1:15:04 - 1:15:10)

So we're going to do Pandas. "P-A." OK, so sorry, let's save it to a variable.

(1:15:10 - 1:15:31)

So D-F, and I'll explain. This is another conventional variable, short variable. So it's D-F equals, and now do Pandas dot data frame.

(1:15:32 - 1:15:48)

And it's capital D-A-T-A, capital F frame. "F-R-A-M-E," and we call that camel case or word case is what people call it sometimes. The camel case is a little more evocative.

(1:15:48 - 1:15:56)

Then open parenthesis. "Data frame." And then we're going to pass it our monthly budget variable.

(1:15:57 - 1:16:17)

... So it's D-F equals Pandas dot capital D data, capital F frame, open parenthesis. And then we pass in our monthly underscore budget.

(1:16:17 - 1:16:26)

Then we close the parenthesis. And that should create a Pandas data frame from our dictionary and assign it to D-F, the variable D-F. It's a very short variable.

(1:16:28 - 1:16:36)

"In left bracket, 38, right bracket." Because we assigned a variable, we don't get any output. But now we have a data frame variable.

(1:16:37 - 1:16:49)

And just like with S before, D-F is a conventional variable name for a data frame. So S was a conventional name for series. D-F is a conventional one for data frame.

(1:16:49 - 1:17:09)

If you only have one data frame, then people will conventionally use the variable name D-F for the data frame. If you have more than one data frame and you're doing other things like that, you should probably give them other names, more descriptive names. I figure people are kind of taking a while to catch up.

(1:17:10 - 1:17:21)

But I'm going to start explaining a little bit about the data frame here, OK? I'm going to just do a little time check, whatever you like. 2, or "2 colon 22 PM." 2.22, OK.

(1:17:22 - 1:17:46)

So yeah, so now I'm going to show you a couple of the data frame fundamentals, OK? So the basic things you can do with the data frame. So the first thing to know about a data frame is, well, we'll look at the representation in a minute. But before we look at the representation, I want to, which is only marginally useful for screen reader users, I want to talk about the format.

(1:17:47 - 1:18:09)

So the data frame, it's two-dimensional data, OK? So you can kind of think about it as if you imagine a line going from left to right in your mind. And then for each item on that line, there's also another list going from top to bottom, OK? And if you've ever used Excel, it's the same as a spreadsheet. So you have from left to right, you have columns.

(1:18:10 - 1:18:31)

And from top to bottom, we have rows, OK? It's a table, OK? So we have an x-dimension and a y-dimension. And so our x-dimension has columns, which are going to be things like our monthly budget, our cookie budget, our budget, and our month. Those are the items from left to right.

(1:18:31 - 1:18:55)

We have three. And then from up to down, the rows are going to be our five items that are going to be data for each of those, and each month's different kinds of budget, OK? So let's try out a few different things, OK? I want to kind of get to some stuff at the end that's pretty cool. So I'm going to try to pick up the pace like 20% here, OK? Let's still try to explain things fully.

(1:18:56 - 1:19:01)

So let's do clear. I'm going to clear things because we're going to experiment with our data frame now. So Control-L.

(1:19:02 - 1:19:14)

"In left bracket, 38, right bracket, 12." We now have our data frame variable that we're going to work with. So let's first do just listen a little to what the format of the data frame is, the representation.

(1:19:15 - 1:19:36)

D-F. "D-F. Out left bracket, 38, right bracket, colon. Budget cookie line, budget month." That was the column names, OK? Now let's hear the actual table stuff. "0, 10, June 3." 0, 10, June. And then it said 3. But it said June 3rd. It's trying to predict things in a not useful way.

(1:19:36 - 1:19:50)

So 0, 10, June 3rd, OK? So the first item is the index. Remember, it's the range from 0 to 4. So that first item is not going to really mean anything. It just means it's the first item.

(1:19:50 - 1:19:56)

That's what the 0 is. Then 10 was our budget. The month was June.

(1:19:56 - 1:20:03)

And then the cookie budget was 3. So let's hear another row. "1, 10, July 2." 1, 10, July 2nd.

(1:20:04 - 1:20:09)

So 1 is the index. 10 is the budget. July is the month.

(1:20:09 - 1:20:14)

And 2 is the cookie budget. So we're going to be spending $2 that month on cookies. We'll just do one more.

(1:20:17 - 1:20:23)

"2, 5, 0, September." And then so it's so on. It'll do that for each line, OK? I won't review the whole thing.

(1:20:23 - 1:20:30)

And at the end, there's no D type or anything like that. That is because there's different data in each column. So a D type doesn't mean anything.

(1:20:30 - 1:20:45)

There's all sorts of D types in here. We have integers. And we also have strings, OK? So now, I will say what we're going to learn right now is how to learn a whole bunch of stuff about the data frame without using the string representation.

(1:20:48 - 1:21:04)

And I'll talk more about why the string representation, why you're actually not missing out on that much by not reviewing the string representation. It actually, you're not missing out on very much compared to a sighted person. But I'll kind of return to that topic in a minute, OK? When we have a longer data set where it's more obvious.

(1:21:05 - 1:21:30)

All right. So let's do a couple of basic functions with the data frame. The first one is, and this is what I always do within your data set, is how big is the data set? So let's do df.shape. df.shape, S-H-A-P-E, no open and closed parentheses.

(1:21:30 - 1:21:36)

So it's an attribute, not a method. Some of these are attributes, not methods. So they're kind of like little variables inside the data frame.

(1:21:36 - 1:21:52)

They're not functions inside the data frame. "Out left bracket 39 right bracket colon left paren 5 3 right paren." So it said left paren 5 comma 3 right paren.

(1:21:52 - 1:22:05)

So it gives us two numbers in parentheses. And the first number is 5. And you can maybe guess because you know how many, you know the dimension, you know what we put in. So the first number is how many rows.

(1:22:05 - 1:22:16)

And the second number is how many columns. So it's 5 for the number of rows, 3 for the number of columns. That's the shape of the data.

(1:22:17 - 1:22:30)

And you can kind of think that makes sense. You know, like if you imagine it like, oh, if we had a lot of columns and only a few rows, then the data is really long. If we have a lot of rows and only a few columns, the data is long in a top to bottom dimension.

(1:22:30 - 1:22:41)

So it does kind of change the shape. Or the data could be perfectly square if we have five columns, five rows, et cetera, et cetera. So that shape tells us we have five rows and three columns.

(1:22:41 - 1:22:58)

Now you're like, well, Patrick, I already knew that because we created the data ourselves. But don't worry, we're going to be in a minute using a real life data set that will pull in much faster than we even created this toy data. And that'll be much more useful then.

(1:22:59 - 1:23:08)

So df.shape, that tells us the number of rows and columns. Let's now do this. And this is honestly, it's the thing you will do the most often in Pandas.

(1:23:08 - 1:23:20)

So pay attention to this. The thing you will do the most often in Pandas is pull out a column from a data frame. This is the most commonly performed operation in Pandas, in my opinion.

(1:23:21 - 1:23:55)

So df. and then give any of the names. Let's do our month, df.month. "D-F, dot M-O-N-T-H, out left bracket 40, right bracket colon, zero June, July 1st, September 2nd." Reading it in a weird way because it tries, you know how screen readers are. They try to be smart, but sometimes they're not always smart. So it's like September 2nd, because it's a two, then a big space, and then September.

(1:23:55 - 1:24:22)

So it's like September 2nd, but it's kind of guessing wrong, but that's fine. So we have other ways of doing things with this, but the, so basically what we did was we pulled out, we did df.month. And what that gives us back is that column as a series. And this is why I say, we're not gonna neglect our series stuff because now we can start combining things together.

(1:24:22 - 1:24:34)

So what if we do df, what if we want to know the average cookie budget for each month? Okay, now we're gonna combine a couple of things together. Okay, and this is called chaining. So let's do df.

(1:24:37 - 1:24:56)

... Cookie underscore budget. "C-O-O-K-I-E, cookie, B-U-D-G-E-T." And another dot. Budget, dot. So it's df.cookiebudget, cookie underscore budget, dot mean for the average.

(1:24:57 - 1:25:06)

M-E-A-N mean, right parenthesis. Open the parenthesis and then we'll close the parenthesis. Do we have a question or anything there? I heard someone on the mic.

(1:25:06 - 1:25:27)

I think maybe someone accidentally unmuted themselves and we might've been hearing a screen reader. So maybe proceed with caution. Out left bracket 41, right bracket colon 2.8. So what we did there, and now we're kind of getting, things are getting a little interesting here, right? So we're combining some stuff together.

(1:25:27 - 1:25:53)

We're doing df.cookie underscore budget, dot mean. So in one action, we pulled out the column and got the mean of the data in that column, okay? When you start combining the methods one into the other, we have a word for that and it's called chaining, okay? And Pandas has a lot of functionality that allows us to facilitate chaining. That's a little bit more for intermediate users, but it's very cool stuff.

(1:25:53 - 1:26:08)

And sometimes you'll just be like, df.this, dot this, dot this, dot this. And you combine a whole bunch of things together and in one line, you get kind of almost like a whole little data analysis. So, and we'll be getting by the end, maybe the last thing we do in this workshop will be kind of a little bit of a longer chain like that.

(1:26:09 - 1:26:18)

So, okay. So that pulls out the column. Okay, so we've pulled out the column and then you can also in the same action, do things to it.

(1:26:18 - 1:26:42)

So like use our methods, okay? So df.cookie budget, which is a series. And then we can use a method that we normally use on a series, the mean method in the same action, okay? And then we get, it smoothly get the mean of that column, which is very nice, honestly. Like, and how much typing around would that take you to do in Excel? You'd have to like set up a whole thing, you know, and type into it.

(1:26:42 - 1:27:10)

It's just, you know, it's very quick. When you start getting fast with this, you start getting data really fast and you're kind of at your fingertips, okay? So what else do I need to show you here for data frames? We pulled out the column. I'm gonna show you really quick how to pull out a row, okay? And then we're going to kind of move on to the next thing.

(1:27:10 - 1:27:19)

Okay, so let's do df. There's two ways to pull out a row. I think we're getting a little tight.

(1:27:19 - 1:27:32)

So I'm gonna only show you one of them, but I'll kind of point to the other one, okay? Two colon 32 PM. We're not doing too bad. We're actually, so, okay, I'll show you the full version here.

(1:27:32 - 1:28:23)

I'm sorry, I just want to make sure that we get to everything. So what I want to show you is, right now we have it on this data frame, we have an index, but it's kind of useless. It just counts zero, one, two, three, four, you know? So, and we can actually pull out the index and listen to what the index is by itself by doing df.index. df.index, no parenthesis, "out left bracket 42 right bracket colon, range index left paren, start equals zero, stop equals five, step equals one right." So basically this says, it's a way of representing

counting up. It says range index, start at zero, stop at four, count up by one. Okay, it's a function that allows us to create, and we can actually run this function.

(1:28:23 - 1:28:43)

It exists in pandas, range index, which allows us to create like an index, which counts up for us, which is actually pretty useful. We're going to use this in one of Sarah's workshops to create some data that we'll then use for soundification. But basically it means, okay, right now the index counts up from zero up to four.

(1:28:43 - 1:28:58)

Okay? That's not that useful. What you want with an index is something that is, it's ideally it's unique, okay? And descriptive. And this is unique, you know, if each row has its own number, but it's not descriptive.

(1:28:58 - 1:29:16)

And when you look for descriptiveness, what you want to think about, like what is this row? So remember columns, they represent attributes of our data. So different facets of the data. So for example, the month, the cookie budget, the budget, those are facets of the data.

(1:29:17 - 1:29:35)

The rows, they represent the, an entity. So a specific item or a thing or object or concept or something, okay? An entity. And in our case, you know, each, so each the columns they're actually, they're facets and they're facets of a month.

(1:29:36 - 1:29:47)

Each row represents a different month, okay? In our budget. And a month of budget. And so what would actually be most descriptive here would be the month.

(1:29:47 - 1:29:54)

Because, you know, we'll always want to know, okay, well, yeah, that's June's cookie budget. That's June's regular budget. So let's do DF.

(1:29:56 - 1:30:19)

Let's, what we want to do is overwrite the index with the, a more useful series, okay? And so we can actually do that by picking out a column and over, using it to overwrite it. So this is pretty cool. DF.index. "DF, dot, I-N-D-E-X.

(1:30:19 - 1:30:21)

Index. Space. Equals.

(1:30:21 - 1:30:23)

Equals. Space." Space.

(1:30:23 - 1:30:44)

And now let's do DF.month. "DF, M-O-N-T-H." So it's DF.index equals DF.month. And what that does is it's just like variable assignment. What you're doing is you're overwriting the index with another column, okay? And that column that has more descriptive data.

(1:30:44 - 1:30:51)

It will actually keep the month column. There's no reason to get rid of the month column. But from now on, the index will be replaced with the month.

(1:30:52 - 1:31:05)

If your dataset had like, you know, if your dataset was longer than a year, you would need to have the year and the month to be the unique index. But that would still be a great index. The year and the month would be a very common index for like a dataset that involved time.

(1:31:05 - 1:31:15)

"In left bracket 44, right bracket colon." And remember, we're assigning something so we don't actually get any output. Remember when we assign variables or we overwrite stuff within data, this is the first time we've done that.

(1:31:15 - 1:31:34)

We don't get an output. So let's do, now if we do DF.index, you should get the month. "D-F dot I-N-D-E-X. Out left bracket 44, right bracket colon. Index left paren, left bracket June, July, Sep." The month is overwrote it correctly.

(1:31:34 - 1:31:50)

So, okay. And now if we look at the data frame, DF, the representation. "Out left bracket 45, right bracket colon. Budget, month. Those are the columns. Month, June 10th.

(1:31:51 - 1:32:02)

June, July 10th." So, okay. It's a little difficult to parse that, but what it's saying is that the column on the left, which is the index and the representation has changed.

(1:32:02 - 1:32:36)

Okay. Now this is actually even more clear. So now if you want to, say we pulled out the cookie budget, DF.cookiebudget. ... So DF.cookiebudget. And this is where it'll strike you as a little more useful maybe. "Out left bracket 46, right bracket colon. Month, June 3rd, July 2nd." So it's saying the June 3rd is pretty annoying. June 3, that's our cookie budget for June. July 2nd too, that's our cookie budget for July.

(1:32:38 - 1:32:55)

"September zero." It said September zero. "October 4th, November 5th." Okay. So it's, you know, September zero, October four and November five. Okay.

(1:32:56 - 1:33:05)

So now we have our data. When we pull out a column, it continues to be labeled, which is actually really useful if you're like reading this data or something like that. You want it to continue to be labeled.

(1:33:06 - 1:33:21)

So having a useful index like that is very good. Okay. Now, the last thing I'd like to do here is pull out the, well, one more thing I want to show you, and then we're going to pull out a row very quickly using this index.

(1:33:21 - 1:33:28)

And then I do want to move on to our big data set because that's cool stuff. But we're going to be working with our big data set next week too. All right.

(1:33:28 - 1:33:40)

So what if we want to calculate, or we want to get an idea of how much we're going to spend on cookies each month. Okay. And this is very cool.

(1:33:40 - 1:33:52)

Pandas makes this very straightforward and in a very cool way. So we have two columns. We have our budget column and we have our cookie budget column.

(1:33:52 - 1:34:08)

And what we're going to do now is a small thing and it will tell us the percentage of each month that is taken up by cookies in that month. Okay. So we're going to do df.cookie underscore budget.

(1:34:12 - 1:34:51)

So df.cookie underscore budget space divided by, so I'll use slash for divided by

df.budget. out left bracket 47 right bracket colon month, June 0.300. 0.30. So that's 30%. Okay. So it's a floating point number that tells us out of one, if it says 0.3, that means it's out of one.

(1:34:51 - 1:34:59)

So 30% of our budget is taken up by cookies in June. Let's try it. Let's listen to July.

(1:35:00 - 1:35:20)

July, 0.20 So 0.2, remember that's the same. If you convert that to a percentage, that's 20% of our budget is going to be taken up by cookies in July.

(1:35:20 - 1:35:31)

Let's do September. September, 0 so we didn't spend any money on cookies in September. That's zero spent that month on cookies.

(1:35:31 - 1:35:40)

Okay. So that's pretty cool. And that, so, and it's becomes, it was made more useful by the fact that we labeled, you know, we changed the index to be the month.

(1:35:41 - 1:35:47)

So now we can kind of keep track of the months even as we do these operations. Okay. And we've answered that question for ourselves.

(1:35:47 - 1:36:14)

We were like, yes, what, what, you know, month is, like what month is a, you know, September? How much did we spend on cookies or what percentage of our budget did we spend on cookies in July? So we answered that question for ourselves by doing this. And what we did was, you know, we divided one column by another. And so what it does is it goes through and it goes row by row and it divides each by each.

(1:36:14 - 1:36:29)

So it says cookie budget divided by budget, cookie budget divided by budget, cookie budget divided by budget. And then it gives us a new series that is actually the, that is the division of the two columns. Okay.

(1:36:29 - 1:36:54)

And imagine that would also take a ton of work to do in Excel, right? I mean, if you were going to make a new column and also very error prone, I've never done something like that without creating a ton of errors and stuff like that. So, and we're not going to do it in

the interest of time, but you can, now I could take this series and add it back into our data frame as a new column, which would be like percentage, we could call it budget percentage on cookies or something. And we can add it back in.

(1:36:54 - 1:37:04)

I won't do it now in the interest of time. Okay. And one last thing I promised I'd show you is what if we want to plot only the data for September, for example.

(1:37:05 - 1:37:10)

So we could do df. ... Dot.

(1:37:10 - 1:37:19)

... L-O-C. "Dot. L-O-C. For location. And then open square bracket.

(1:37:20 - 1:37:26)

L-O-C." Quote September. "Quote. ... Quote. September.

(1:37:28 - 1:37:30)

"Quote. Right bracket." Close square bracket.

(1:37:30 - 1:37:33)

So it's df. L-O-C. Open square bracket.

(1:37:33 - 1:37:34)

Quote. September. Quote.

(1:37:34 - 1:37:38)

Close square bracket. "Out left bracket. 48.

(1:37:39 - 1:37:42)

Right bracket. Colon. Budget five.

(1:37:43 - 1:37:47)

Cookie line budget zero. Month September." So the month is September.

(1:37:48 - 1:37:54)

Remember our cookie budget for September is zero. It's a very sad month. And then the budget is five.

(1:37:56 - 1:38:14)

Very, pretty cool, right? So the, and that is another series. And when you do, when you pull out a row, it gives you a series where the index is a, is the column name and the value is the actual value of that data. Okay.

(1:38:14 - 1:38:23)

So if you imagine in your mind, in your mind, just lifting out the row. Okay. We have our row.

(1:38:23 - 1:38:38)

Imagine it kind of flying out of the data frame. And then we only have that data now. So it then takes it and it says, okay, what were the column names? It turns it around and says, what were the column names? Okay.

(1:38:39 - 1:38:49)

And now the column names become the index. Okay. So, and so they tell you what that, you know, it gives you a nice little label for that particular item of data.

(1:38:49 - 1:38:56)

So in this case, it was, our budget was $5. We didn't have that much money in September. So we weren't able to spend any money on cookies.

(1:38:56 - 1:39:12)

Okay. Now we really will move on. There's also a df.iloc, which we won't try out, but that will tell you if you know the number of, you know, where in the data set it is, regardless of what the index is.

(1:39:12 - 1:39:22)

So you can pull out the first item by doing df.iloc zero, and that would pull out the first item, no matter what the index is. So that's df.iloc for index location. That's what they named it.

(1:39:22 - 1:39:27)

Okay. So that's another useful one that we won't do. Cause I do want to pull in this data set really quick.

(1:39:30 - 1:39:33)

... I think we're actually going to be good. Okay.

(1:39:33 - 1:39:48)

So now we're going to do another cool thing. I hope that this, I feel like this little bit should be a little bit exciting for you guys. So what we're going to do is, we're going to, we're done with this data frame, but we're going to produce a new data frame with, it's a real data set.

(1:39:48 - 1:40:02)

So what's that we've done so far is toy data, right? It's data we made up. Now we're going to work with data that actually reflects something in the real world. So what I want you to do is type d, I'm going to clear the screen because we're starting something new, and we're going to overwrite our df variables.

(1:40:03 - 1:40:40)

So you can kind of say goodbye to the df variable that we made that you worked so hard on. So I'm sorry for that. We'll do df equals, "df equals space" equals space, pandas, "p-a-n-d-a-s, pandas," read underscore csv, "r-e-a-d, read, c-s-v," and then open, open parenthesis, "csv," okay, "left paren" quote, and then we do, I'm going to give you a URL.

(1:40:41 - 1:42:15)

So it's going to be http, ... colon, "h-t-t-p," slash, slash, bit.ly, for a short, this is a short, a URL shortening service, ... "bit, dot, l-y," forward slash, ... n-y-c-b-n-b, for n-y-c-b-n-b, "n-y-c-b-n-b," and I'll go over this again, n-y-c-b-n-b, and then close the parenthesis, "right paren," I'm going to make sure it worked for me, and then I'm going to explain it again. ... ... pandas, dot read, ... that was the longest three seconds of my life. Okay, so, what did we do there? So we do, we typed df, for our new data frame variable, equals, pandas, dot read, underscore, csv, r-e-a-d, underscore, csv, open, parenthesis, quote, and then we have a URL.

(1:42:15 - 1:42:55)

So it's going to be, h-t-t-p, colon, slash, slash, bit, dot l-y, b-i-t, dot l-y, forward slash, n-y-c, New York City, b-n-b. So it's n-y-c-b-n-b, okay? And helpers can, one of the helpers copy the line and paste it into the chat, and no shame in copying this one, because it's the URL, okay? I tried to make it simple so people could type it if they didn't want to copy. I know copying can sometimes be clunky, but we do have it right there in the, hopefully someone will share it in the chat, okay? So df equals, pandas dot read, csv, and then a URL.

(1:42:55 - 1:43:12)

What is this doing? I've prepared a dataset at that URL, which is a commas, it's basically

a spreadsheet. It's commas separated values data. So it's a spreadsheet that I uploaded to that URL, and pandas very neatly reads it in and imports it all into our DataFrame variable, which is pretty cool.

(1:43:13 - 1:43:26)

So in that little line, we created a new DataFrame, and we're going to learn a little bit about it. So let's run through really quick, because we're just reviewing. So I'm going to run through really quick some of the stuff you would do when you get your hands on a new DataFrame.

(1:43:27 - 1:43:51)

Hopefully you guys are being able to import the DataFrame, okay? Remember it's bit.ly forward slash NYCBNB, okay? B as in Bravo, N as in November, or B as in Bravo, okay? Let's do df. So I'm going to, now I'll do the representation, but it's not going to be that helpful. Remember our representation is kind of overwhelming.

(1:43:51 - 1:44:03)

So let's just do df. df, "out left bracket 51, right bracket colon, 48,891." Did you hear that? 48,891.

(1:44:03 - 1:44:34)

So that's, we're going to talk about this in a minute, but we don't know what that is yet, actually, but we're going to confirm in a minute. Something's there. "36485057, 36, 48,892, 36485431." My question, you're probably like, what the hell? It's just basically, it's just printing out random stuff. That's because what we have now is a big DataFrame, and it has, well, I'm going to tell you now how big it is. So it's going to be, how do we tell how big it is? We don't just look at the representation because that's too much information.

(1:44:34 - 1:45:21)

It's not useful. Let's do df.shape. "df.shape, S-H-A-P-E," no parenthesis. "Out left bracket 52, right bracket colon, left paren, 48,895." So this has 48,895 rows, each representing an Airbnb listing, and it has 16 columns, okay? So this is a big, it's not big data because it still runs in our computer or whatever, but it's a much bigger data set than we've used so far. And it's real data, okay? This is all Airbnb data from New York City in 2019. So pre-pandemic, pre-recent reform of Airbnbs in New York City.

(1:45:22 - 1:45:37)

And so it represents actual data from that year that we can look at. What's the first thing we're going to want to do after we find out the size of it? And it is useful to know the

size. We have 16 columns to work with and we have almost 50,000 rows, okay? Let me just double check the time.

(1:45:38 - 1:45:54)

"2 colon 50 PM." We're going to do some cool stuff. Okay, so what we want to do is we want to know what the columns are, okay? Because then the columns will tell us how much, what kind of data we have to work with, okay? So we want the column names.

(1:45:54 - 1:46:26)

And pretty much if you're working with a new data set, you're going to do this in more or less this order. You're going to want to know how big is the data set in terms of, you know, X and Y, rows and columns, how many, and also the second thing you're going to want to know is what is the data we have, which the column names will tell you. So let's do df.columns, okay? df.columns. df.columns. And there are a lot of these, so we'll listen to some of them.

(1:46:26 - 1:47:12)

"Neighborhood, latitude, longitude, room line type, price, minimum line nights, number line of line R, reviews, last line review, reviews line per line month, calculate," and of course the, it's New York City, there has to be a car alarm going off outside, but you're getting a little local flavor. But okay, so we have, in our, so now we have our, that is very annoying. Okay, so we pulled out our column names and I would say there's a couple that stood out to me, okay? That's df.

(1:47:15 - 1:47:25)

Oh my God. df.theprice, that stood out to me as being really interesting. I'm always attracted to stuff like price, okay? So that's some interesting numerical data.

(1:47:26 - 1:47:49)

We have df.neighborhood, that probably tells us the different neighborhoods in New York City, which is pretty cool. And then there's a couple of other ones that stood out, maybe minimum nights. But the other one that I found really interesting was df.name, okay? And so each of these represents a different kind of data and df.price is numeric data, it's an integer.

(1:47:50 - 1:48:24)

And then the df.name, we can check. Let's check these really quick. So df.price. "48,890.70." ... So I heard the word, I heard 70 there and 48,890, that's the index. And then 70 is the price. So it's telling us integers, so it's numeric data of some kind, okay? And we could do that with name and in the interest of time, I'll tell you, name is the

name of the listing as it appears in Airbnb, which is really interesting, textual data.

(1:48:24 - 1:48:59)

And then the neighborhood, it's kind of like grouped data, which we'll get into. It's a special kind of data called categorical data that we'll talk about next week, okay? But for now, what I'd like to be interested in is let's figure out a little bit about the price, okay? So what, first of all, I'd like to know what is the average price of an Airbnb in New York City? So it would be df, and we kind of did a little bit of this before, df.price.mean. ... Remember, this is a real data set and we're running this over almost 50,000 rows.

(1:48:59 - 1:49:08)

We're getting the mean of almost 50,000 rows. "Out left bracket 55, right bracket colon 152 points." So, okay, 152.

(1:49:08 - 1:49:23)

So the average price of an Airbnb per night in New York City in 2019 is a little over $150. And that sounds really high. So maybe you're like, oh, wow, like New York is expensive as I have heard.

(1:49:24 - 1:49:58)

But before we make any judgments, let's also try the median. So let's do df.mean, df.price.mean. "df, dot price, dot ... right paren. So df.price.median. Out left bracket 56, right bracket colon 106.0." 106.0. So the average was almost a little over 150 and the median was only 106.

(1:49:58 - 1:50:11)

So this is interesting, right? And maybe you're like, oh, that's interesting. That sounds a little more reasonable. I mean, maybe it still sounds expensive to you, but you're like, oh, that's actually less than I expected from the average.

(1:50:12 - 1:50:27)

So this actually does tell you something. And this is the kind of thing you learn as you do more with data science. When you look at the mean and the median and the median is significantly lower than the mean, it tells you something about the shape of the data.

(1:50:28 - 1:50:38)

And they call this right skewed data. This specific situation is called right skewed data. And what that means is there's a couple of, there's some items in the data set that are bringing up the average.

(1:50:39 - 1:50:51)

So some, in this case, very expensive items. And I'll draw you an analogy. Like imagine all of us together, we imagine each of our incomes, right? So imagine, and we probably all have fairly normal incomes.

(1:50:51 - 1:50:58)

Maybe some of us are whatever, you know, I don't know. Maybe there's some millionaires here. We say we all have pretty normal incomes, right? All of us who are in this workshop.

(1:50:59 - 1:51:13)

And we collect us all together in a data set. So our data set is all of our, or let's say our wealth, okay? Okay, so we have each of our individual net worth or whatever. And then we take someone like Jeff Bezos or something like that.

(1:51:13 - 1:51:29)

Someone who has a gazillion, trillion, billion dollars, okay? And we add Jeff to the data set, Jeff Bezos, Mr. Bezos. Suddenly the average in that data set is gonna shoot up by like probably more than a billion. I think there's something like 50 people in this room.

(1:51:30 - 1:51:41)

Jeff Bezos is definitely worth more than $50 billion. So the average is gonna go up by more, at least a billion dollars. However, the median will only go up very slightly.

(1:51:42 - 1:51:56)

And that, why is that? That is because the mean, the average is very sensitive to high numbers that kind of like outliers that throw it off. Whereas the median is the middle most number. So the middle most number in the Jeff Bezos example didn't change too much.

(1:51:56 - 1:52:29)

So the median often tells you more about the most representative data in certain data sets. Whereas the average is kind of like tells you it's a little more thrown off by big numbers that are included or big outliers, okay? In cooperation with each other, when we run on both of them, we kind of configure and some people, a sighted person might try to learn this kind of information by creating a bar graph. And then the sighted person would see visually that there's a whole bunch of big values at the end of the data sets.

(1:52:29 - 1:52:43)

But we can also make that intuition based on the mean and the median, which is these kind of statistical numbers that we can use. And now let's kind of, this will be kind of the last thing we do in the workshop. I kind of want to tell you what we're doing next time.

(1:52:43 - 1:52:52)

And I'll give you a little kind of philosophy of the non-visual stuff before we go. But this is the last Python we'll do. Let's confirm our hypothesis.

(1:52:52 - 1:53:26)

So our hypothesis is that there's a few very large values in this data set that are throwing off our analysis, but that are not kind of skewing the data over toward the expensive side. So let's do df.price.sort underscore values. So that sorts the price from least to greatest.

(1:53:26 - 1:54:05)

Now I'm going to run this, and then I'm going to tell you to add something to the end of it, but I'm going to run it really quick. "40,433, 9,999. 12,342, 9,999." So that tells us right there. So it's actually, it's telling us from highest to lowest. So I heard there, oh, you know what it is? What's confusing here is that what Pandas usually outputs is it's from the least to the greatest, and it prints out the first five of the least, then it skips all the rest of the data, then it prints the last five.

(1:54:05 - 1:54:31)

But because I made the text so big in this, it's only printing out the last five, even though it's supposed to print out all of it, okay? So it's being a little deceptive here, and it's only because of me magnifying this command line environment so much, okay? It's printing out the last five, but really what it tries to do is print out the first five and the last five. So what I want you to do is, so it'll work on your computer because you may not have magnified to the same extent. Press up.

(1:54:32 - 1:54:55)

"In left bracket 58, right bracket colon, df.price.sortline values left paren, right paren." Add to the end of that. "In left bracket." Add to the end of that dot, dot tail, T-A-I-L, open parenthesis, close parenthesis. "T-A-I-L, left right paren." So now we're chaining three dots together.

(1:54:57 - 1:55:09)

"Out left bracket 58, right bracket colon, colon, left paren, right paren. Out left bracket 58, right bracket colon." So this is just a bunch of preamble.

(1:55:09 - 1:55:43)

We haven't gotten to the data yet. "40,433, 9999. Name colon, price, type colon." So it printed out the last five and basically it says there are a bunch of them with, we can review to see, but there are a bunch that are 9,999 and a couple that are 10,000. "Blank. Name colon, 29,238, 10,000.

(1:55:46 - 1:55:54)

9,151, 10,000. 17,692, 10,000." So there's three at least that are 10,000.

(1:55:54 - 1:56:04)

And then I know from experience. 12,342, 9,999. Okay, so we have a whole bunch of really, and probably there's some weird reason those are in the dataset.

(1:56:04 - 1:56:16)

Probably people aren't really paying 10,000 for their rooms. They're probably doing something like jacking up the price temporarily or something like that so that people don't rent the room in some day that they want, or something like that. Something weird is going on there.

(1:56:16 - 1:56:30)

And that's why I say the logic and the context sensitiveness comes in when you start looking at that real life data like this, which we'll do more next time. But now, and that last one's kind of cool. So what we did was we did df or data frame.

(1:56:31 - 1:57:07)

We pulled out the column, df.price. Then we sorted the values in the column.sortValues. Then we pulled out the last five so we could get an idea of what the last five most expensive items are. Okay, so, and this is kind of, as we go in pandas, you'll see we'll sort of chaining and chaining and chaining and combining together more and more items like this. Okay, so I kind of just want to say that just for the little bit of the non-visual philosophy before we get into the next workshop.

(1:57:07 - 1:57:25)

And we're basically done with the Python here. So, but it's really important when you're a screen reader user to control how much information is coming to you and exactly what information. And luckily this environment that we're in and pandas really does give you full control over that.

(1:57:26 - 1:57:49)

Okay, and in fact, I would say we as non-visual people, we're not really at that as much of a disadvantage as you would think in terms of doing things totally non-visually. And that is because, for example, when a sighted person prints out that data frame, the data is too big to look at visually anyway. There's 16 rows, there's 50,000, oh, sorry, 50,000 rows, there's 16 columns.

(1:57:49 - 1:57:59)

It's just too much to look at as a practical matter. And in fact, pandas knows this. And when a sighted person prints out a data frame that's long like this, it just shows the first five rows.

(1:57:59 - 1:58:17)

I dot, dot, dot, and the last five rows because it knows it's not useful. And my sighted students often ask me, hey, how do I change it to see all the rows? Because they just wanna go looking through the data. And I say, it's actually, no, do not do that because you too will become overwhelmed.

(1:58:18 - 1:58:55)

They need to learn the same thing that I'm showing you, which is to pull out the most specific information that they can and to be efficient that way. I would also say that the other thing, and we're gonna get more into this next time. And we only kind of scratching the surface of it here by learning some of the fundamentals, but that what I'm gonna try to show you is that there are ways of working with the data that will give you the same information or it's very similar information that a, say a pie chart, bar chart, line chart would give you.

(1:58:55 - 1:59:24)

But instead of using a visual app, a visual approach, we use, we try to explore the data by having a conversation with the data, by being like, oh, the median is this. Well, that makes me curious about the mode. What's the most common value in there? And then you build up a mental model of the dataset in your mind, but without like that kind of visualization approach.

(1:59:25 - 1:59:39)

And it's very, it's honestly, it's very practical. It really can build up a mental model of your data through this conversational method. I'm not gonna say it's gonna be exactly as good all the time, but then we also have sonification, which we'll do in the last two workshops.

(1:59:40 - 1:59:52)

So we can access some of that condensed information that some people get in a chart. Okay, so that's my pep talk for next time. We're gonna learn some, we're gonna lean into this conversational style with the data.

(1:59:53 - 2:00:26)

We're gonna do this exploratory data analysis to get a feel for this dataset and to answer some actually quite specific questions. Like for example, what's the neighborhood with the most expensive apartments and so on, okay? We're gonna all do that next time. For people who are feeling adventurous this week, between now and Tuesday, I have in the curriculum, in addition, in the curriculum, I have added a challenges section to the end.

(2:00:27 - 2:00:55)

So it has three little challenges. They only use, or they can be completed with items that we learned in this tutorial. There may be easier ways to do it to solve those challenges and pandas, but you can definitely solve them with the methods, attributes, techniques, and everything we've learned in this tutorial, okay? And they're like real questions about the data, this dataset that you can try to figure out if you want to challenge yourself between now and Tuesday.

(2:00:55 - 2:01:13)

Okay, and I'll also say maybe just cause you're not, maybe it's not something you're used to, the curriculum that's been created for the first and second workshops, they've really been designed for independent work. So if you haven't looked at them, they're very descriptive. I'd say they actually have more information in them than I can get across to you in the workshops.

(2:01:14 - 2:01:28)

And they're very talky, just like I'm talky, and conversational. So they're not just like a dump of the information in this workshop. They're actually like designed so people can follow along independently and without even looking at these workshops.

(2:01:28 - 2:01:40)

So if you haven't looked at those as a resource, they're there for you. If you're feeling a little overwhelmed, you wanna review. And then finally, if you wanna kind of, you know, drop off the meeting or whatever, I'm just gonna talk about procedural stuff now.

(2:01:40 - 2:01:52)

On Thursday, we will be, have another office hours. And every week after the Tuesday workshop, we'll have a Thursday office hours. It's at the same time during the day.

(2:01:52 - 2:02:15)

So it's 1 p.m. Eastern time, 6 p.m. Grand Meridian time, or GMT, okay? And, and thank you. I'll stop the recording there. And I'm gonna stick around for questions, okay? Actually, if people have questions about the data science portion, I'll leave the recording running so people can benefit from that.

(2:02:16 - 2:02:41)

So thank you all and have a great day. Okay, so helpers, were there questions in the chat that came up that would be good to address? Or does someone wanna get on the mic and ask a question? Let's see. I think, I think just if anyone wants to hop on the mic.

(2:02:42 - 2:02:53)

Just to be clear, I wanted to make sure, people have permission to leave now. If you're worried about social permission to leave, you have permission to leave now. This is, this is, you know, even more optional than the workshops.

(2:02:53 - 2:03:02)

Just wanna make sure that's clear. Sorry, we have a question from Liam. Can we go through saving this as a file like last time? So I believe the iPython session.

(2:03:08 - 2:03:11)

There were my headphones fell out there and I didn't hear you. I didn't hear it. So I wonder if you could.

(2:03:12 - 2:03:15)

Sorry about that. We have a question from Liam now. I'm waving my hands around.

(2:03:17 - 2:03:23)

Oh, can you still not? Oh, are your headphones still not on? I can hear you now. Oh, okay. Whoops, okay, whoops.

(2:03:23 - 2:03:30)

Okay. Liam asks, can we go through saving this as a file like last time? So I believe saving the iPython session. Yes, okay.

(2:03:30 - 2:03:39)

That's great to review and thank you, Liam. Okay. So to save, we use what's called a iPython magic command.

(2:03:39 - 2:03:44)

Okay. Those all start with a percentage sign. Okay.

(2:03:44 - 2:04:05)

So the first thing you have to do is you have to know what the last line, the number of the last line you input was, because we'll need that for this command. So I'm just going to press enter so I can hear my input. "In left bracket 59, right bracket." So I heard 59. So that means we've entered 58 lines of code. So we want to save from one to 58.

(2:04:06 - 2:04:21)

It's kind of annoying that it makes us put that in, but it's just how it's designed. I want to do a, maybe I'll do a pull request and be like, it's supposed to actually save everything by default, but there's a bug where it doesn't. All right, so let's do %save.

(2:04:23 - 2:04:37)

"%save." Space. "Save." And then. Space. And then now we want to put in a file name and it will, if you don't put a .py at the end, it will add the .py for you.

(2:04:37 - 2:04:56)

So what will be created is going to be a Python text file, a .py file, which is a text file with Python code in it. But I'm going to call this pandas. "P-A-N-D-A-S." Pandas. "F-U-N-D-A-M-E-N-T-A-L-S." I just called it pandas underscore fundamentals.

(2:04:57 - 2:05:03)

And I didn't put the .py because it will add that for us. "Fundamentals." So it's %save.

(2:05:03 - 2:05:05)

Sorry. %save. Space.

(2:05:05 - 2:05:11)

Pandas underscore fundamentals. And then we have another space. Then one.

(2:05:11 - 2:05:13)

Number one. One. Hyphen.

(2:05:13 - 2:05:17)

One. Two. 58, I think was the number.

(2:05:17 - 2:05:22)

Five. Eight. Let me just, I think that hyphen.

(2:05:23 - 2:05:30)

"158 is neither a string nor a macro." Yeah, I didn't. "58. Five." I didn't properly put the hyphen in. Dash.

(2:05:30 - 2:05:32)

"Five. Eight. D-F.

(2:05:33 - 2:05:53)

D-F dot shape." And what gets printed out is all the, so what gets printed out to the screen here, and what we're hearing, is all of the stuff that we have done in the workshop. But it actually, it should have put it in the file, okay? And we can actually check that using another magic command.

(2:05:53 - 2:06:02)

So I'll double check it. "Percent. ... L-S." ... ... And I call that pandas underscore fundamentals.

(2:06:03 - 2:06:23)

... ... "Volume in drive C has no label. Volume serial number is 36F6-B." Okay, printing out a lot of stuff from the review. "Volume in, in left, volume, blank, directory of C, blank, file not found." Oh, cause it made it a dot P-Y file. Well, whatever, don't worry. It did save it anyway.

(2:06:23 - 2:06:34)

Let's just not get too deep into that. But there's ways to check if the file is actually created. But you can go, by default, it will save it in your users folder and your name.

(2:06:35 - 2:07:01)

So if your name is Liam, it would be users forward slash or backslash Liam. Then you have to look in that folder in whatever the Windows Navigator, whatever they call the program where you look at the files and folders, and you should see a file in there, whatever.py, whatever you gave it. Sorry.

Thanks for asking that because it's useful to review and people like to save their sessions. Now, I can share this with people if they ask for it, so that's also good. I might have forgotten.

(2:07:03 - 2:07:23)

Anyone else have a question? Yeah. Hi. I have two questions.

This is Juan. Hi, Juan. Question one is a silly question, but I noticed when you're using your IPython, when you start entering multi-line input, you hear a colon, but in mine, it uses a dot, dot, dot ellipses.

(2:07:23 - 2:07:33)

Is there a reason that mine is different than yours? There's a good question. Let me just see what I get just to confirm. So if I make a dictionary.

(2:07:35 - 2:07:38)

Left braids. It's really odd. I think visually.

(2:07:38 - 2:07:52)

Because I like the colon. It's less verbose. You like the colon.

Well, the thing is, yours is accurate and mine is inaccurate because it's mine. That's what it's actually putting on the screen is an ellipsis. I can see with my little vision here.

(2:07:53 - 2:08:08)

Oh, so it's maybe NVDA reading ellipses as a colon. I don't know how NVDA would read it. I mean, NVDA does mess up output, but I don't see it doing a reading as a colon.

(2:08:08 - 2:08:28)

I would say the possibilities, it's order of likelihood are one, that I'm using an older version of IPython, which I know I am because I was too lazy to reinstall it, but there was really no reason to do it. And they fixed this, but this was a bug and they fixed it. So that's a fairly likely possibility.

(2:08:29 - 2:08:55)

Two, so if I installed a new, I updated my version here, which honestly I've had on this for years, so it's probably kind of out of date. Can I get version information about that big version? I notice you're running Python 3.8 and I'm running 3.12. You're probably on 3.12. Yeah, so that's a lot of versions out of date. So, you know, so I would say it might've, there might've, they might've fixed, that might be a bug that they fixed.

(2:08:56 - 2:09:16)

That seems fairly likely given how out of date mine is. I would also say another possibility

is I have magnified or like I've made the font like 70 something, which is kind of all source of a lot of bugs with NVDA right now because, you know, it's cutting things off, it's doing weird things like that. So it could be a source of something like that.

(2:09:17 - 2:09:36)

And yeah, I could have a different NVDA setting, but I think that's almost a less likely possibility because it's so weird. Okay, it's probably because, yeah, your version is way older than mine. Okay, my second question is something you kind of very touched very lightly on, like a single sentence last time, but I was just curious.

(2:09:38 - 2:09:53)

So the magic command per edit, so it opens Notepad. Is there, do you know how, do you know how to change that? Because it's opening Notepad. It's kind of a little more like an office hours thing because it's a multi-step process.

(2:09:54 - 2:10:19)

Yeah, but basically what we're gonna do is change a setting. There's a functionality here, a magic command to change settings and we will use it to change the setting, which will be the command line command for the node. Probably you want, do you want VS code? You probably want VS code, right? Yeah, well, it's okay.

(2:10:19 - 2:10:35)

I don't need it exactly. I just want to know, I just want to make sure it's not like something I change in Windows. If it's something I change in IPython, or is it something that changes? Yeah, you can change it in, yeah, I don't think it, it may open the default text editor for your operating system.

(2:10:35 - 2:10:53)

So if there's a way to change it in Windows, that's worth trying. But I do think it does require, I suspect it requires a change of setting in, or in either the command line or IPython. But I will- I can Google that, I can Google that.

(2:10:53 - 2:11:03)

Yeah, yeah, exactly. But you know what? It's something that might be useful for people. So I might look it up and include it in the resource that I'm creating here.

(2:11:03 - 2:11:13)

So people know to change it. And it's useful, like if VS code opens and then you can edit stuff and use your full screen reader and everything, save it, and then it runs it in there.

That's a really, kind of a nice workflow.

(2:11:16 - 2:11:33)

Yeah. I mean, it was okay to use Notepad, but- And Notepad stinks, you know, it's like- I mean, for simple, like setting up structures, like simple dictionaries, isn't a big deal, but the auto-completion in VS code is way better. And so it just makes coding a lot easier.

(2:11:34 - 2:11:39)

Notepad++ is also pretty good. It is accessible. So, cool.

(2:11:39 - 2:11:43)

Awesome. That's Juan, right? Thank you, Juan. Yes.

(2:11:44 - 2:12:08)

Anyone else have a question, want to get on the mic? Well, if that's all the mic-friendly questions, feel free to jump in before I end here. I guess we'll end the recording and then I'll stick around for a little bit if people want.