DDAHS 2026 summer school Workshop 3 Exploring Hydrological Data-20260715_103311-Meeting Recording

15 July 2026, 09:33am

2h 2m 46s

◉ **Gianni Vesuviano** started transcription

Matt Dalle Piagge 0:03

First of all, we probably should introduce ourselves. Thanks, Gianni, for doing the recording. I'm Matt. Don't worry about my surname. Lots of people pronounce it differently, but fortunately, there's only one UKCEH Matt here today, so no need to worry about distinguishing between me and others because there are lots of Matt's at UKCEH, so that can be a problem. But today,
I can just be Matt, which is good. I'm going to be leading the session, talking you through different types of hydrological data, but we also have some facilitators to help out if needed for any technical issues or to answer any questions I don't know the answer to, which does happen. I believe you have already met
some of us.
Actually, I should give you some more information about what I do first, probably. As you might be able to see on my amazing t-shirt, it says that I am a research software engineer (just ignore the unicorn), which basically means that I'm doing a lot of the technical work that helps make the science that we do happen and happen well, to put it in a very simple, a simple space. All right, I'll let the other facilitators introduce themselves. I think we'll start with Kit, who you haven't seen before yet.

Kit Macleod 1:23

Cool. Thanks very much, Matt, and great to meet everybody online. So, I'm a colleague of sort of Matt and others at UKCEH. I sit within the sort of the hydrological forecasting and digital systems sort of group, and my role today is just to help out in the background.
I don't have as strong a technical background as sort of Matt and some others on the call, but I'm very interested in helping people understand and use our sort of data and digital solutions to improve water use and management. So thank you, Matt.

Matt Dalle Piagge 1:57
No worries, who wants to go next?

Amulya Chevuturi 2:03
I have met them because I've been in the background, but hi everyone, I'm Amulya Chevuturi. Nice to meet you all again. If some of you have not met me already, I'm a senior hydroclimate scientist at UKCEH. I work with all of these great people and I work on looking and understanding hydroclimate extremes.

Matt Dalle Piagge 2:07
Yeah.

Amulya Chevuturi 2:23
Thank you.

Matt Dalle Piagge 2:28
Go on then?

Tom Keel 2:28
Should I close?
Sure. So hi everyone. I met most of you yesterday. I'm Tom. I'm A hydrological data analyst. I've been running workshop 4 after lunch today. And I work on sort of UK rainfall and the data products associated with that.

Matt Dalle Piagge 2:52
And that's.

Gianni Vesuviano 2:52
And yeah, I guess finally I'm Gianni, you will have seen me in all the other sessions. I'm coordinating the summer school this year. Other than that, I do a lot of work with Flood Estimation Handbook, flood frequency analysis, rainfall frequency analysis and that kind of stuff.

Matt Dalle Piagge 2:54

All right.
That is everybody. And as Tom has mentioned in the chat, feel free to put your cameras on. We don't bite. It does make me feel slightly less like I'm talking to a screen. So if you feel comfortable doing that, much appreciated. But I understand if you don't, that's fine. It's nice to see so many of you here. And with that, let's get started.
Um...
a little bit of preamble before we launch into things. This is going to be, this workshop is roughly 2 hours long. It's going to be split into three parts. We're going to cover three different main types of hydrological data that you might come across in the future. That's textual data, vector data, and grid data.
There will be a break after an hour to give you, because there'll be a lot of information to give you a chance to breathe and step away from the screen for a bit.
If you have any technical issues, like getting the notebooks to work, then please type in the chat.
or raise your hand and me or one of our, one of my lovely facilitators will help you out and get you sorted. There will be time for questions after each part. So I will keep, I will probably allow five or 10 minutes for questions at the end of each of the three parts.
if needed. I think that is basically everything. Anyone, did I forget anything? I usually forget something.
Nope, silence. Excellent. Okay. In that case, I shall get started with sharing my screen and giving you a link in the chat to click on.

Amulya Chevuturi 4:58
All done.

Matt Dalle Piagge 5:00
All done already? Amazing. This is why I like you guys. All right, so there should be a, well, yeah, there are two links in the chat. The bottom one will take you directly to the first notebook.

Amulya Chevuturi 5:01
In the chats, yeah.

Matt Dalle Piagge 5:13

which I will also have on the screen, and we will go through it together. Let me work out how to share things. Every time I do this, I always seem to forget. Share window, screen tab.
Please share this one. Thank you very much.
All right, hopefully.
that is sharing.
Good, good, alright.
Okay, has everybody managed to load that page all right, is seeing what I'm seeing? I will generally assume silence is a good thing, but as always, if not, shout in whatever way is easiest for you and we'll get you started. So how this workshop is going to work,
is that we're going to go through these notebooks together. There's a lot of code that's already written that we'll just run, and I'll talk through that and explain what's going on. And then there'll also be a few bits where
I'll ask you to, or that will type in, type in some code together so that it's not just me and you clicking run endlessly on these bits of code, on these bits of code. But hopefully that will hopefully allow you to remember a few bits or pieces more easily as well.
So the first part is, as I mentioned, is textual data. Not the most exciting of the three data types, but you will see it around from time to time. It is essentially any data that you can open in the text editor, such as Microsoft Word or
Other open source equivalents, or whatever you want to use, or notebook note notebook plus plus is 1 I use, quite, no note plus plus is one I use quite a lot.
and actually see the data typed out. So you can actually see the numbers in a text editor there for you. That data is still used surprisingly commonly, even though it's not the most efficient these days. So it's worth knowing a little bit about the formats you're likely to encounter and how to work with them using bits and pieces.
introducing this, we should scroll down to the first cell, this one here, which has, I might actually just zoom in a bit so we can all see it. Yes.

Amulya Chevuturi 7:39

Matt, just, um... just copy to drive, if you could just introduce that.

Matt Dalle Piagge 7:46

Yeah, that's the one I forgot. I knew I'd forget something, which is why I have you all, I have my team, my wonderful team here, and there you go.
So as we're running this on Google Colabs, any changes you make to these notebooks won't be saved by default. If you'd like to make changes and refer back to those changes you've made, then there is this button at the top that says copy to drive. If you click that, it will save a copy in your Google Drive.
and any changes you make will be saved there. But the material itself will be available in the repository for you to view at any time. It's only if you want to make notes or changes yourself that you'd like to keep for yourself. That is a very good point. Thank you, Amulya.
Otherwise, I shall continue if everybody is okay.
Let's get going.

Matt Dalle Piagge 8:45

Let's run this cell. Oh, hello.

Amulya Chevuturi 8:48

Someone doesn't have copy to drive option. Even if you don't have the copy to drive option, as Matt said, all the material is available, so don't worry about it.

Matt Dalle Piagge 9:03

Is, yeah, okay, that's good.
All right.
Is everybody, the main question is so that we're just running this first cell here that just instals a couple of packages that we'll need to be using.
The question that I want to ask, I want to check is that is everybody able to run these cells all right? Because that's sort of the crucial point, I guess, of this workshop. If you can't run the cells, then it's going to be difficult to follow along. If there are problems, do shout.
But for now, I would assume that everybody's doing alright.
Let's run the next one. So you can run these cells using these little play buttons next to them, next to the bits of code, or a handy compute, a handy shortcut is also shift and enter on the keyboard. That will also, that will also run each cell for you.

Throughout the workshop, we're going to be using data that's stored remotely in a cloud storage system. I don't need to worry too much about that for now. I just wanted to include this next code cell here as an example of basically how to read in and show data that's stored remotely.
So running that will show you basically a list of data that is stored in the repository that we're using today and all the different data sets that we'll be working with. But you don't need to worry about them for now. That code is just there as an example. And that will be how a lot of this workshop works, is that there will be examples.
of how to do things.
how to do different bits of analysis, different bits of working with the data that you can refer back to if you need. So it's a resource you can come back to. You don't have to worry about remembering exactly all the different bits of code that we go through, because it will be a lot. It'll be there whenever you need it, if it's something that you need to do in the future. That's the idea of this workshop.
So let's scroll through that and actually get stuck in reading some data. The Python library that we're going to be using is called Pandas. I'm not entirely sure why it's called that, but it is at least a memorable name. It basically is a very powerful library that allows you to do all sorts of things
and types of analysis and processing and reading in lots and lots of different formats of text-based data. And it basically reads everything into a tabular format, which is easy to understand and read through.
So, we're gonna, we're gonna today, we're gonna read in some river flow data, as you as is typical for.
which is often in this sort of format, this sort of textual format. So we'll run these two code cells here to have a look at it.
Usually takes a couple of seconds as it's reading it in from the cloud somewhere. There we go, and the next one will show you what it is now.
This is, this data is actually under the hood is CSV data. The good thing about Pandas is it doesn't actually matter what the format of the underlying data is. You can see here that Pandas has a read CSV function with a lot of a lot of customizability, which we'll be looking at. But basically, whatever
textual data format you've got, Pandas can read it in and once it's read in it will look the same and you can work with it the same. That's the that's the one of the main advantages of.
of libraries like Pandas. See, I didn't actually say, but CSV in this case does stand for

comma separated values. It's sort of quite an old data format, where in theory, numbers, all the numbers that are in the data set are separated by commas and that's how the format of the data works. Although you will find that nowadays CSV is used even when the data is not separated by commas, but actually other things as well.
But as I said, that doesn't really matter. The main thing is, if Pandas can read it, which it can, then your data will be able to reread it at a table like this. Okay.
Let's scroll down and continue.
There's a lot of text here that I won't be reading through directly, but that's for you to go through in your own time and refer back to if you'd like to, although I will sort of basically be covering it as I'm talking.
So what actually, what actually, I'm going to scroll back up again, sorry. So what actually is this data now that we've read in? I mentioned earlier it's river flow data. It's not, it doesn't seem to have a lot of metadata, which is not very helpful, but that is unfortunately true about a lot of data you would probably end up working with.
We can see along the top, we've got day, month and year. That's fairly self-explanatory. And we've got these five, 5 letter, no, 5 digit numbers along the top, which in this case refer to individual catchments. So this is a data set that is showing the river flow
a lot of different river catchments across the UK at different time points. So we've got time running down the vertical axis and different catchments along the top. Fairly simple, straightforward table.
Now, one of the great things about Pandas, it has many great things in my mind, but one of the great things is that you can, it knows how to deal with time.
And that's something that until this sort of library was developed, dealing with time in data was a pain and quite difficult. So Pandas has a standardised way of dealing with
dealing with the table, sorry, dealing with time. I just, I'm seeing comments flashing up in the chat. The one about why is data negative? I'll be getting to that. That's just, that's, that's jumping ahead and that's fine. But yeah, I will get to, I will answer that one in a minute.
Where was I? Oh yes, Pandas and time.
So.
In order to be able to get access to Panda's time functionality and do all the things that it enables you to do, like taking means over time or accessing maximum and

minimums or being able to extract out points or ranges in time without having to worry about
at the specific point in the data you're looking at or what the position of the row and column you want to extract is, we need to tell Pandas about how the time is formatted in this particular data set. It is quite clever, but it's not so clever that it can automatically determine what the time of a particular row is.
So that is one of the many options that Pandas' data reading in functions like read CSV have is to basically say.
what columns in this data set refer to time, how is it formatted? And then once you've done that, Pandas can take that in and will understand what is meant by the columns that you've told it our time and you can work with it there. So all that was me saying,
what this column is doing. You'll notice it's the same read CSV function as earlier. All we've done is added in a couple of options. In this case, what we're doing is saying we want to add a new column to our data set called times made-up of column zero, one and two. And that's
column one, two, and three. In Python world, everything starts at zero rather than one, just to be just to be confusing, but that's what that means. And we're saying that the ordering of the columns is that the day comes first before the year. And that's basically all we need to do to tell pandas what
how to work with the time data in this particular data set. So in other data sets, you'd need to adjust this to basically tell pandas which columns has the time data in it if it can't read it automatically. But that's something that, again, this is just an example. You and read CSV has a lot of different options for telling pandas how to deal with time.
And that's something you can go through later or in your own time. Once we've done that.
One thing we haven't done, we still need to do to unlock Pandas time, what I'm going to call Pandas time magic, is to tell Pandas that we want that to be the new index. And what I mean by that is that we want these, what I'm going to call row headers here, this stuff in bold, to be
the time information and not just numbers, as that would then allows us to extract out data based on different types, extract out data based on the time and much more easily, and allows us to unlock all the functionality that I keep mentioning. And so this is one of those parts where you stop looking at the screen and clicking, I keep

looking at the screen, but you stop just running code and actually have to type something in. So basically, this is where you can code along with me.
I'm going to type something in. In this case, it is to tell Pandas that we want this new times column to be the index. And that is where all the power is unlocked. So following along with me before we run this cell, I will show you how to set the index. And it's very simple.
Most of Pandas is quite intuitive. So the data set we're reading in, I've called it obsdata. I want to set the index.
to be, I can see the auto complete is being very helpful and basically filling this in for me. We want to set the index to be the times column.
And we want that to happen.
in place, which basically means it changes it without you having to assign it to a new variable. So if let's run that cell.
Wait for it to read in the data and do its processing.
And then, once that's done...
We can view it in the next cell.
There you go. And now we can see that instead of just numbers down this index column, it's now assigned the times that we saw in separate columns earlier into this index. Okay.
That's that. I think that's enough about me muttering about indexes and times. Now let's have a look at one of the one of the specific catchments in detail by way of learning how to select out different bits of data from this table. As you can see, it's There's a lot of data here going from 1961 to 2017 for a lot of different catchments. In fact, it's 737 catchments and 2000 and... around 20,000 time points. So let's just maybe focus on one particular catchment and a particular bit of time.
to make this easier to handle and to get our head around. And this cell here tells you basically how to extract out bits of the table that you might want to look at, bits of the data. And the good thing about pandas is that you can essentially
extract out based on the headers up here and the index down here. So instead of having to worry about the position of the data in the table, you can simply say, I want this column or this range of columns and this row or this range of rows.
using the values that are in bold rather than the values that are not going to worry about indexing and other faffy stuff. So we're going to have a look at one particular catchment and we are going to look at catchment 39001, although feel free to put another one in if you're feeling

adventurous and you happen to know the catchment ID of another catchment of interest.
There are, there is more information about.
all the different things you can see with catchments.
and all the different properties that come with catchments and which IDs correspond to which catchments on the NRFA website, which I'm not sure if you've been introduced to yet, but that's linked elsewhere in this notebook. And I think we will look at the winter of, let's say, 2013. So I'm going to fill in
some dates here. Let's go from 2013 and December the 1st to 2014.
at the end of March.
Temporarily forgot how many days March had, 31, I think.
And this little symbol here, this colon, that's basically saying a shorthand for in the whole of Python, actually, not just Pandas, but a shorthand for through to. So we want to select from 2013-09-01 through to 2014-03-31.
And this lock function that Pandas had is very powerful, is what allows you to extract out that sort of data, extract out data without using the headers and the indexes. So we run that very quick.
Now let's plot it. Let's have a look at it. Now that's enough talking about numbers and the rest of it. Actually looking at the data is probably what you want to be doing a lot of the time and visualising it. And the great thing about Pandas and many Python data handling libraries is that it's as simple as
calling the word dot plot, the function dot plot afterwards, and that will produce.
a very simple quick plot of the data you've just extracted.
In this case, this is data for Winter 2002 for the Thames catchment. That's what 39001 was, if you were wondering.
And you don't need to worry about doing any other fancy stuff to visualise the data.
The next bit of the code here is about...
Next bit of this, the next code cell, I should say, is if you want to make the plot a bit more fancy, which you might want to do, there are lots of ways of doing that. Not going to spend too much time on this because I think it's all fairly straightforward.
But again, this is just something you can refer back to.
if you want to make the plot look a bit nicer or customise some elements of the plot.
There are some simple commands you can use for doing so. And if you will spend just one or two minutes adding a few in here to jazz up the plot a bit, make it look a bit better.

Um...
So, for example...
What I think this one does. See, I've got to remember myself what I had in mind here, which doesn't always happen.
Ah, yes, yes. If you want to customise the plot, there is some, a little bit of Python magic we have to do first.
So in our plotting command, which if you remember is this, obsplot, in this case, data is stored in obsplot, we want to plot it.
We have to specify which axis we want to put, we want to make the customizations to. And that's what this line here is doing. It's basically saying whatever plot I'm doing, this is the plot I'm interested in and this is what I want to customize.
So this is create the plots.
Say this is the axis that we want to plot the data on. And now we can actually get around to actually customising it. Quite simple. If you want to, for example, change the y-axis limits, we can do set_ylim and the axis limits. So let's go from 0 to
Wow, there's a lot of, it was quite a lot of river flow in winter 2013, wasn't there? So let's go to 525, for example, setting the title.
Is as simple as...
plot dot title or plt.title. Let's just do river flow demo or whatever you like. Setting labels, again, pretty simple.
ax.set_ylabel.
And I'm going to skip over the legend one now, but you can also quite easily add legends as well. Let's run that. I have a feeling I have missed out a bit of code, and in which case we'll get an error and we'll see what that is.
Or we might not. No, it didn't. It was fine. I got ahead of myself. So now you can see we've basically just added labels and titles and other stuff like that. All simple stuff so far, but something you can refer back to if you forget how to do it in the future.
Now, if I'd selected the data out well, or created a better plot, we might have seen all those minus ones that someone highlighted earlier in our data. So if I scroll back up to the top, or back up to where we see the data, you see all these minus 1 values.
What that is,
as you may have guessed by now, is missing data, basically where they're saying that there are no observations here for this particular catchment in this particular time. So only a few catchments, it seems, have data in 1961, a lot of minus ones and only a couple of catchments I can see that have data.

I want to again, one of the good things about Pandas is that it can handle this missing data quite easily.
And that is basically handled again by adding a function to the reading in of the data, which I will scroll back down to underneath the plot we just made.
into the function that we use to read in the data, we can say that a certain set of values are missing and that we want pandas to ignore that and to do all the calculations and all the analysis and plotting that follows, ignoring those values.
So, to do that, we type in this function called na_values, and what that means is...
Please set these values to NA, which in this case means not a number or not data.
And -1 is the those values that we want to do. So now if we run, oops, gosh.
Apologies, I forgot to put a comma in.
That is a classic error that you'll probably make all the time. I still do, years after I started learning this stuff.
So, if you put...
Add that line into our region function, you should now find.
that it has converted all the minus ones to NANs.
And if I had selected out a better region of data, I would be able to prove this to you a little bit more easily, in that you would no longer see any, the minus ones being plotted on the data. So you wouldn't see suddenly, say, if there was missing data in December, you wouldn't suddenly see
the data dropping down to minus 1 and coming back up again. It would just be a blank section of plot because it knows to ignore those values.
Okay.
So that is what I would have meant to be showing in these two plots. We were still running, we'll still still run them.
and see what they look like.
That is a lot of data.
But the main point I wanted to pull out is that we are not getting any minus 1 values in the plot from all the way from 2000 to 2015. So it automatically just ignores them.
And if I select out a smaller bit of data.
you can see what Pandas is doing, where it is basically a small little gap around April, where my cursor is, which I hope you can see. But if you see basically between April and May, there's a small little gap and that's what Pandas is doing. It's just missing out the data. And when you do any further calculations with the data,
That's basically what Panda tells. It will ignore those values for you. You don't have to

worry about them.
So that is a very quick introduction to reading in some text data with Pandas and what it does. There is plenty more material in this notebook that you are welcome to go through in your own time. I will just introduce a couple of little bits of it here so that it is something you can look at in
if you want to, and so you know what's there. Some of it is discussing, if you scroll down in a notebook, you'll see it. And some of it is discussing and showing you how to work with a different type of textual data, which actually doesn't work so well with pandas and that you might see come up.
in your work in hydrology or whatever from time to time. And that is something called ASCII grid files, or they usually have the extension dot GRD instead of dot CSV or something else.
There's a lot of information here about how you would read in that those data files like that and work with them. And basically you would read them into a using a different library, using a library called a Python library called Xarray, which we will get to work with in a little bit. So teaser for that.
But if you ever see gridded data, it will look at data with a dot GRD suffix. It will look a bit like this. It has an opening in the text editor, I mean, and it will look a bit like this. It will have some five or six lines of headers, which basically describe the domain. the grid, where it is in space, and how far apart the grid points are. And then the data itself is sort of stored in a tab separated format underneath that. I'll stop there on that one. There's plenty more information in the notebook if you want to go through it in your own time later. As I said, there's a lot of resources that are also linked from these workshops and these notebooks for you to use if you need to or just are interested. And one other thing, if I scroll down past this section to the section that starts with supplementary material is that I alluded to but didn't really get into heavily some of the stuff that pandas can do with
time handling.
And this and supplementary material in the notebook will show you and talk you through some of the other stuff you can do, such as calculating means and maxes, minimums, sorting data, doing a whole lot of other stuff with it.
That's there if you need it and if you want to.
look at it later and in your own time. It's a resource that you can use. But I will take a breath, pause for a little bit before we jump into the next section of this notebook, of this workshop, and I will actively say and ask if we have any questions or if anyone

had any issues that they want to do.
talk about. Feel free to speak up, put your hand up or put anything in the chat.
Otherwise I will just keep talking and I don't know about you, but I do get sick of my own voice after a while.
My lovely co-facilitators, is everything alright?

Amulya Chevuturi 34:09
Yes, everything is going great.

Matt Dalle Piagge 34:09
Awesome.

Amulya Chevuturi 34:14
People who, some were having issues with the Google account, I hope that is all resolved, but feel free to kind of unmute and let us know. We can troubleshoot any issues before we jump into the part 2.

Matt Dalle Piagge 34:35
I did not know that's where the name Pandas came from. Thanks, Sebastian. There you go. Been doing this job for however many years, many years. I did not know that. Very cool.
All right, I'm going to guess that everybody is okay.
In which case?
It's time for the second part.
So, once again, there will be another link in the chat for you to go to, and it will be a second notebook.
Amulya, on the case, as always, amazing.
It will be look very similar to the first one, but it will be slightly different.
And I will open it up myself.
and it should have, oops, that wasn't meant to happen. Part 2, vector data at the top to make sure you've got the right one open.
I'm just gonna have a drink of water.

YS

Yelesen, Sebastian 36:04
I didn't know about the Polars Library,

which you mentioned at the bottom. That's new to me. Thanks for putting that in there.

Matt Dalle Piagge 36:07

Yes.

You're welcome, yeah. I don't know, Tom, do you talk about Polars in your workshop? I can't remember.

Tom Keel 36:18

I do, yeah. We use Polars and stuff, so we'll be very quickly ramping up the difficulty, I think.

Matt Dalle Piagge 36:19

Yeah.

Yeah.

YS

Yelesen, Sebastian 36:21

Yeah.

Matt Dalle Piagge 36:24

So you get a chance to look at it, yeah.

YS

Yelesen, Sebastian 36:24

Because, like...

Tom Keel 36:27

Yeah.

YS

Yelesen, Sebastian 36:27

Yeah, because I've been trying to look at how to, when you start to work with things on larger and larger scale.

I was like, oh, this is taking too long. And then I need to kind of learn how to use, like, like, I've been trying to learn how to use Zarr and, like, Dask and things like that for like dealing with, like, like dealing with things on a larger scale.

Matt Dalle Piagge 36:48
Yep.
Oh, fantastic, yeah.
Yeah.

Yelesen, Sebastian 36:55
Just got too impatient for doing things the old way.

Tom Keel 36:58
Thank you.

Matt Dalle Piagge 37:00
Yeah, yeah, yeah. And that's one of the reasons those libraries got developed right, is that data is getting bigger and we need to find out new ways of working with it.
Yeah, and yeah, part three of this workshop will mention Xarray, Dask, Zarr, those file formats a little bit more and point you to some more resources.
for those as well. I hope that's helpful.
Yeah.

Yelesen, Sebastian 37:23
Thank you.

Matt Dalle Piagge 37:24
You're welcome.
All right, in that case, let's crack on with part 2, which is talking about vector data. So data that you can't just open a text editor and look at, but data that basically describes essentially different shapes and where they are, to put it very bluntly, very simply.
That's what I mean by vector data. The classic...
The classic vector data, why did I keep doing that? The classic vector data is a shape file, which again, basically contains a set of shapes in a file, as the name might suggest, and a little bit of information about, potentially anyway, a little bit of information about each of those shapes, such as where they are on the planet and what they actually signify.

So in hydrological data, what you'll find is that that, in the world of hydrological data, what you'll find is that we like shapefiles a lot because we often use them for
Uhh...
Stuff like rivers, where all the rivers in the world, river catchments, where are they in the world? And in fact, the river catchments is what we'll be looking at in this part of the workshop today. Country boundaries even, and all the other sort of administrative areas you might find or might need to work to know about.
when you're dealing with your hydrological data. So all sorts of data can end up in shapefiles and hydrological catchments and river catchments is only one of them, but it is likely you are likely to come across it. So let's, as before, let's run these first two, first two cells together.
run anyway. This basically just instals a few packages that we will need for dealing with shapefiles.
I just remove that and hide some of things on my screen. Might take a few seconds to install as we're installing a few libraries this time. And as well as talking through the different types of data.
A couple of different.
a couple of different data formats. I'll also, again, show you how to visualise them and how to plot them. Yeah, and as Tom says, if you wish to save your changes, the copy to drive option is there for you if you need to. Okay, that only took a few seconds to install. That's good. Let's import these libraries so that we're ready to work with them.
And let's go. Right, so to work with shapefiles, very handily at some point in the not too recent past, someone wrote an extension to the Pandas library called GeoPandas to make working with shapefiles a lot easier. So what you'll find is that
If you, once you've got the hang of working in stuff with pandas, you'll find the working with shapefiles is just as straightforward because basically all the commands are the same. It's all very similar. It has the same interface. The only difference being, as you'll see, if we run these two
These few cells.
Drum roll, please. But it looks exactly the same, but it looks very similar. It's another table. Pandas loves tables. The only difference being you've got this geometry common geometry column that sits in the table, which basically contains the spatial information. So that's containing basically the
the vector that describes your shape and each row is a list of shapes and each

column is information, each other column is information about that shape that is stored in the shape file.
So hopefully that's straightforward and makes sense. The first thing we're going to do is just
Pick out one of those rows and see what it looks like.
And for that, it works basically in the same way as collecting out data in Pandas does. We can make use of Pandas ilock or lock functions, which are its spatial indexing and subsetting functions. So we have called the shapefile very helpfully.
shapefile. So if you type along with me, we can extract out a bit of the data. Let's say, let's do ilock. Now ilock differs from lock by basically
selecting out data based on where the index or where the position of the row or column that you want to extract out. So rather than using the names of the columns and the names of the rows, which are in fact just numbered anyway, in this case, we're going to use their position.
So, let's just select out the first row.
which because we're using in Python, that's zero, the 0th row. And let's use this colon shorthand for saying let's select out everything in that first row.
So if I run that, you'll find we've just pulled out this one row of our data set because we've pulled it up as the first row and every column. To see what it looks like, we can once again just add basically add dot plot to it.
and run that. And we'll see that's what our shape looks like, which is at the moment not very descriptive or helpful. I happen to know this is a particular river catchment somewhere in the world, or somewhere in the UK, but on its own, not very helpful.
So if I run the next one below, just to highlight this row again, if this was a decent shapefile, you would see some more information, more helpful information, shall I say, in these columns along with it that would describe what the shapefile is or some information about it.
I sort of, I picked this file because even though it doesn't have that helpful information in it, this is unfortunately some of the data that you will be working with probably at some point. It might not have all the information you want or need. Because hopefully that is getting better, but there's a lot of legacy data sets out there that don't contain much metadata, which is
can be difficult to work with. But you recognise this ID, these ID columns as hopefully being the same as or similar to what we saw in the last part, as basically that is the catchment ID of each shape, which in this case is a river catchment.

To.
I just noticed that re-show the code that generated the shape. Certainly can. It's up here. I'll pause there for a moment.
Oh, thanks, Tom.
So that catchment is apparently the River Forth in Gargun… oh God.
Now I have to pronounce it. Gargunnock in Scotland. Who knows if I pronounced that correctly?
And again, if you wanted to find out more information about the catchments and what those IDs correspond to, then the NRFA website is your friend. And that is linked to somewhere in this notebook, I think.
Okay, thanks, Sam. Okay, let's continue. But do shout again if I'm going too fast or you need me to pause or something like that.
So, we've pulled out one row of data. I've included this line of code here below it, as sometimes shapefiles might have extra global attributes that, and that states the metadata associated with the file as a whole, not the individual shapes.
As I have alluded to, this one doesn't, but just so you have that for reference in future, you might find there is some more information in the attributes which you can access in this way.
Right.
That's what one particular catchment looks like. Now let's get to exploring what the rest of the data looks like in the shapefile. We can, if we wanted to, just very naively plot the entire thing and see what happens. I quite like doing that sometimes as a starting point. To do that, it's, yeah, very simple shapefile, which is what we've called. Our data set.
Dot plot, as always.
press shift and enter or the play button to run that code. And you should see something like this. And that is something that looks a bit like the UK to my knowledge, a sort of jaggedy bit of the UK with bits chopped out of it, cut out of it. Um…
But okay, that tells us something about where the data is, but it doesn't tell us much about the individual catchments. We can't really see them. We can't really see what the individual shapes are because there's so many of them and they're so close together. So I wanted to show you a
thing you can, customization, you can add to the plotting commands to make it look nicer and make it easier to understand what's going on. And that is in the next cell.

We can add in, basically we can add in something that says, let's remove the fill colour of these shapes and make them transparent.
so that we can just see their borders. And you can do that with this.
face colour argument and we will set it to none.
in quotes and a capital N.
And that should.
Old press.
produce a plot where all the shapes are transparent and the colour has been removed. Arguably, there's so many shapes that it's still not very helpful. You can't actually see what's going on very well. So we're going to make some further customizations and focus on one catchment in particular.
to demonstrate that. So continuing on.
We're going to look at the River Thames, which has catchment ID 39001. Oops, I keep doing that, don't I? Catchment ID 39001.
which is basically the majority of the River Thames catchment flowing through London and a lot of the southern, southeast part of England. So the first things we want to do is to select that out. We might remember how to do this from the previous workshop.
Top workshop, the previous book, previous notebook, the previous part of this workshop, using...
Pandas lock function, except I'm going to show you a new thing here. Instead of doing shapefile.lock, we can also select out using Python Python using Pandas' dot where function. Another way is another way of pulling out bits of the data.
that I wanted to show you.
So, we...
Google, that's not very helpful. It's just disconnected my runtime.
Why did you do that? Interrupting me mid flow. So I will continue typing in this code box and then I may have to pause for a moment to restart and rerun some of the code. But I will just finish where I'm at with this particular cell. So instead of using lock or ilock,
to pull out data. We can also use where to basically select out rows and columns based on a particular feature or a particular bit of information. So we can say we want to select out where
The row.
Sorry, where the yes.

Where the ID column?
which is what I just typed in, where the ID column.
Is equal to...
39001.
And I'm going to call that Thames.
Now, I will...
Pause for a moment where, because I will have to rerun some other code cells. So you can stay where you are. I'm just going to scroll up to the top and rerun some things because for some reason my...
Runtime disconnected, very helpfully.
OK
Bear with me a moment.
Shouldn't need to run anything else.
Right, okay, back with you. All right, so we're in this cell here, which we've just typed into. I've just typed this in. And what this is doing is showing you another way of selecting out data from using Pandas, where basically, instead of using the row headers and the row and the
Sorry, the column headers and the row headers to select out data. We're saying we want to select out any row where the ID column is equal to 39001. So if everything works well, that should run fine. It did.
And if we actually look at that data, that has not worked correctly. Now, what did I do wrong?
It has selected out a certain number of rows.
I am confused. See, this happens in, this happens to, this happens to me all the time. I just hope it didn't have happened and I was on camera in front of you all. But I need to remember why that doesn't work, why that hasn't worked properly.
check file.
OK

Yelesen, Sebastian 51:58
Don't worry about it.

Matt Dalle Piagge 52:01
Happens to us all, right? Yeah, yeah.

Yelesen, Sebastian 52:03
I've wasted many afternoons on the...

Tom Keel 52:10
Is it because you need to drop it because the where is the boolean?

Matt Dalle Piagge 52:13
Yes, yes, that is it. Thank you, Tom. Amazing.

Tom Keel 52:19
I'll see first.

Matt Dalle Piagge 52:23
I think that's this drop equals true. Is that drop equals true? Let's try that. No, it's not that. So, or dot drop.

Tom Keel 52:28
No, that doesn't work.
What was wrong with lock?

Matt Dalle Piagge 52:33
I wanted to show a new a different type of doing things, but now I regret that.

Tom Keel 52:39
Okay.

Matt Dalle Piagge 52:40
dropna, and then is that it?

Tom Keel 52:44
Ohh, is it in place?

Matt Dalle Piagge 52:45
No.

Matt Dalle Piagge 52:49
dropnan
I have a, I'm going to, again, I'm going to cheat because I wrote this, I wrote down the answer somewhere else. So everybody pause whilst your amazing host goes to look up what he forgot.

Amulya Chevuturi 53:04
But as Matt said, this is part of research, right? Like we make mistakes all the time and it's perfectly fine. We learn from our mistakes.

Matt Dalle Piagge 53:08
Let's see, let's you just...
This happens all the time.

Amulya Chevuturi 53:24
Oh, do you have to put like inverted commas?
You have to like put 39001 in like...
Commas, Matt.

Matt Dalle Piagge 53:34
Shouldn't have to.

Amulya Chevuturi 53:36
Okay, sorry. I tried.

Matt Dalle Piagge 53:37
But thank you. And it turns out that this is all my own doing, because in my own example, I've not bothered to use where. I've just used lock like Tom suggested.

Amulya Chevuturi 53:46
And also you used the inverted commas. I'm saying you should try it. Oh yeah, you used ID instead of ID string. Fair enough.

Matt Dalle Piagge 53:52

I know, I know, but I used a different... Oops.
I used ID string rather than ID.


Amulya Chevuturi 53:59
Yeah.


Matt Dalle Piagge 54:00
And also, just to add to the hilarity, my lovely stand-up desk has just collapsed on me, so I'm going to sit down and attempt to continue as if nothing happened. Sorry, everybody.


Amulya Chevuturi 54:06
Oh.


Amulya Chevuturi 54:11
Well, if workshops, if like the things don't collapse on us, is it an actual workshop that we have delivered?


Matt Dalle Piagge 54:16
Ha ha ha!
Indeed, indeed. Oh, gosh.


Tom Keel 54:22
Yeah, Matt, you can just do dot drop now at the end if you want to keep where.


Matt Dalle Piagge 54:23
Welcome back.
Oh, that's what it was. It was dot drop. Thanks, Tom. Thanks, Tom. Sorry, everybody. Bear with me a moment. I seem to have managed to lose the mouse underneath the desk. Oh, wow.


Tom Keel 54:28
Yeah.


Matt Dalle Piagge 54:38

Amazing.
Right, there we go, that will do.
Right, hopefully we are back in action. I will switch back to the workshop that we were looking at.
and continue as if nothing happens, which is why I'm very glad I have a background on today. So you didn't see me suddenly drop about two feet. Right.
As Tom said.
Now that we've if everyone's type managed to type in this code, which I'll highlight again so you can see it.
We can get rid of everything else by using .drop, and then if we run that.
Oh, shush. Tom, what did you do?

Tom Keel 55:24
dot, dropna

Matt Dalle Piagge 55:26
Drop that.

Amulya Chevuturi 55:27
Drop.

Tom Keel 55:30
Mm.

Matt Dalle Piagge 55:33
OK, that seems to have worked without an error. There you go. All right, back in business. Let's go. There you go. Now you can see we've just got one catchment, which is what I was aiming for. Why do I do this to myself? I don't know. Because I had a simple answer set out for this and then decided to invent things on the fly anyway.
All part of the fun, everybody. I hope you enjoy the entertainment. In the next line, we will plot it to see what it looks like.
Again, very simple, just dot plot will produce a very simple plot for you, showing you what the Thames catchment looks like. I'm very familiar with this plot by now, I've seen it many times, so I can almost feel like I can almost identify it by site, but that's

what the Thames catchment does look like.
if of interest. Now, as always,
What I'm going to do is overcomplicate things and tell you some even tell you even more indexing tricks. This one I've typed in beforehand, so I don't have to do the embarrassment of live coding and forgetting what I'm supposed to be coding. So if you scroll down to this particular cell that has
quite a lot of code in it and these weird lovely lambda rows and run that it is basically showing you how you can use
Use the lock function again to select out ranges of data, or...
Yeah, let's stick with that for now. Ranges of data. I won't go through it in detail for the sake of for the sake of time.
Because we're 11:30 already, halfway through everybody.
But it is there for you to look through in the future and what I'm and adapt to suit your own work if you need to do something like this in the future. But what I am actually doing is selecting out a range of IDs rather than one in particular. In this case, I'm selecting out a range from 39,000
to 40,000. And why would I do that? And that is because all the Thames catchments, because there are many of them within the big river basin that is the Thames, all start with the number 39. So everything that's between 39 and 40,000.
will be a Thames catchment. So if you wanted to look at, for example, all the catchments within the Thames.
This is a way we can do it. But we're just going to run that code. I'm not going to make myself or you type it in. If we just run that.
Check that it looks sensible. It does. Let's see all the IDs start with 3900. Excellent. And then plot it.
And that plot looks very familiar. It looks the same. That for once is not a typo, an error. It was meant to look the same. And that's because...
Why is that because? No, that's because.
What we've done is essentially plotted all the Thames catchments. They're all sub-catchments of the biggest Thames catchment, which we plotted earlier. And because they're all opaque shapes, they've plotted on top of each other and you can't see them. So once again, just as a reminder, we're going to do this face colour none.
trick to see all the different catchments.
which is maybe a bit more helpful, maybe not. So you basically, basically showing you all the different sub-catchments like and tributaries of the Thames that flow into

the Thames and make it up and make up the overall river basin that is the Thames.
One more thing I had to show you is some of the geographic and vector handling capabilities of GeoPandas.
So if you take another look at this table, you'll see that the data does have the shape area and shape length columns, but very hopefully someone has decided to blank them out or set them all to zeros. I don't actually know why that is. This is just the data I pulled out.
But fortunately, one of the things GeoPandas can do, and it can do many things, is calculate the area of each shape, or even the length of the border of each shape, or many other things as well, like also a centroid or the centre of each shape, and other things like that as well. So I'm just going to show you very quickly how to do that. If we scroll back down,
To this squiggle of a plot.
We can run.
We can use very simply, again, not too complicated, this shape dot area function.
And if we run that, we get given back a column that basically is all the different, the areas of all the different shapes in the column in metres squared, in the data set in metres squared, I think. Now on its own, obviously, that's not much helpful. If that is detached from all the rest of your data, you've just got a column of numbers.
So one other thing we would need to do is to basically create a new column in our data set.
and add this column to it, which is what we're doing here. So we're going to create a new column, which is done using this syntax here. And we're going to call it catchment areas and assign this area calculation to it.
And if we now look at the shapefile by running the next cell, we can see we've added a column one here, which has all the catchment areas to it. And then you could use that for help in your analysis.
if you wanted to. And as I said, I recommend checking the Pandas documentation for all the other things it can, GeoPandas, sorry, documentation for all the other things it can do regarding the shapes. So as I said, the centre, the circumference, all the stuff like that, which I don't have time to go through now.
But I thought I would just demonstrate that it can do calculation like that easily for you without having to worry about more complicated GIS based approaches, for example.
For the sake of time, I'm afraid I am going to stop there as a...

as my desk decided to interrupt me at a fun moment. I will just give a quick teaser of other things I've included in this notebook that you can go through in your own time or use as a resource when you need them.
I'm showing how to sort data by columns, as you might be used to doing in Excel. So for example, how to sort by the smallest and largest catchment. That's what I probably would have gone through if I had a bit more time.
And then there's a little bit more information on ways of making the plots look nicer, but I'm going to skip over that because in the next part of this workshop, after the break, I will also be doing that and showing you how to plot things nicely.
But one thing I think I can't miss off is this.
Very cool explore function that Pandas has.
which again I recommend you check out to do in more detail. I thought I was going to skip it, but I've decided I'm not going to because it's too cool.
Basically, the idea is if you want to get a really a shareable...
Plot of your data.
Sorry.
If you want to get a shareable plot of your data on an interactive zoomable map like Google Maps looks like, for example, then you can do that essentially in one line.
And that's if you scroll down to the cell that looks like this, that is what that is doing.
And I'm going to run that for you now.
I think I need to do...
I might need to do something first, or maybe it will just work.
It just works. There we go.
So, with one line, you can create a...
an interactive zooming in and out plot, which is having some interesting API issues right now, but we'll not worry about that. And I've just plotted the Thames catchment on a, I think it's open street map, judging by these error messages that are coming up, that you can zoom in and out.
of click on for more information and plot multiple different shapefiles on. And that's all done using this explore function, which I recommend you check out. And it has a huge array of customizability and things you can do with it.
But I will stop there, 'cause I think...
We'll probably need a break, me included. Although I will take, you know, I think we've got, I could do 5 minutes of questions beforehand if people would want them.

And that is enough of me talking for a little bit.
Guys, anything, anything in the chat to pick up on?

Amulya Chevuturi 1:05:12
I think we have answered that question. With a little too much information, I feel.

Matt Dalle Piagge 1:05:15
You've answered it amazing, handled it with aplomb.
Oh, really? Oh, yeah, I can see now. I'm reading through it. Yes. All the all the all the dtypes. Yeah, yeah, yeah, that happens. I.

Tom Keel 1:05:25
I think we were all typing at the same time.

Gianni Vesuviano 1:05:25
Yeah.

Tom Keel 1:05:30
You have got a question, you got a raised hand.

Gianni Vesuviano 1:05:31
Yes.

Matt Dalle Piagge 1:05:33
I can see I have a hand up. Go for it.

Yelesen, Sebastian 1:05:34
Yeah, hiya.
I guess I know you can always display the contents of a data frame, you know, like a table, but one of the things I found a little bit frustrating about, you know, like, for example, like grib or, you know, other file types you open with NetCDF is once you get into like,
quite large data sets where you've got, you know, for a given, you know, geographical area and multiple measurements and multiple measurement time series and all that stuff. Is there any guidance out there for like how you can do quality

control? Like if you've got like a source to like and you want to inspect it to see if, you know, there's any...
problems to do it like how do you do that scale?
you know, for with these, what do you call geographic data sets, or is it really just sit down and check things manually?

Matt Dalle Piagge 1:06:35
And...
I don't know, I think the others might help, might know more than me. I think there are some quality control libraries that exist out there that do some basic kind of checking on data sets, including spatial data sets, I think. I don't know what they are off the top of my head.

YS

Yelesen, Sebastian 1:06:52
Yeah.
Yeah.

Matt Dalle Piagge 1:06:56
Trying to think, but the way, because yeah, I know others in UKCEH are involved in quality control and have been doing that sort of thing. Just trying to think, nothing comes to mind.

YS

Yelesen, Sebastian 1:07:03
Yeah.

Matt Dalle Piagge 1:07:09
My approach, if I couldn't find a library, would be essentially to write a script that does that processing and checks for things automatically using some of the stuff we've gone through. Yeah.

YS

Yelesen, Sebastian 1:07:17
Yeah, I suppose so.
Yeah, it's like one of the challenges.

Tom Keel 1:07:23
Depends on the data, as well.

Matt Dalle Piagge 1:07:25
Gonna say, yeah, Tom might, you might have done some stuff like this, right?

Tom Keel 1:07:31
Oh yeah, well I was gonna say it completely depends on the data.

Matt Dalle Piagge 1:07:34
Yeah.

Yelesen, Sebastian 1:07:35
Yeah.

Tom Keel 1:07:36
Flow data, QCing has automated and manual steps, a lot more manual steps, say, but then rainfall data has lots of automated steps, you're just checking for spikes and things. You might summarise spatial information so you could view it as a time series over a catchment or something like that.

Yelesen, Sebastian 1:07:58
Do you in the UKCEH, do you guys have a software tool you use to just quickly open and inspect different spatial, like I use, what's it, Panoply, the one that NASA made, but I know there's more out there just because

Yelesen, Sebastian 1:08:16
It's free to use, and you can quickly open, you know, files like grib to see what's actually inside, but yeah.

Matt Dalle Piagge 1:08:27
Amulya, did you want to jump in?

Amulya Chevuturi 1:08:28

So yeah, so a lot of people, so there is a slight differentiation here between the NetCDF and the shape files or the spatial data files. Tom, sorry, Matt will be talking about and NetCDF files in the next one. Shape files, a lot of people do end up using QGIS and stuff.
Panoply, a lot of people do use for NetCDF. In the olden days, there used to be other software, but a lot of people use ncview, ncdump, those kind of, but those are command line driven.

Amulya Chevuturi 1:09:09
code. So I don't know any other ones where you can inspect. I just open up it in Python, I Python, interact Python and quickly check personally speaking. Yeah.

Tom Keel 1:09:14
Yeah.

Matt Dalle Piagge 1:09:22
Yeah.

Tom Keel 1:09:23
Yeah, I mean, I mean, if...

Matt Dalle Piagge 1:09:23
Yeah, me too.

Tom Keel 1:09:25
Like ncdump, ncview, and all the NC tools are very much what Panoply is written in, I believe. I think it's just a front end for those tools, as with a lot of these things, like with viewing spatial data that's built on a command line tool called GDAL.

Matt Dalle Piagge 1:09:34
Mhm.

Yelesen, Sebastian 1:09:37
Yeah.
Yeah.

Tom Keel 1:09:45

And GeoPandas just sort of interprets that and turns that into a data frame, basically, that looks like Pandas. But yeah, the very bare bones command line tools that they have for viewing this sort of data haven't really changed. They're still just like essentially bash progress that are very efficient.

But yeah, everything that does exist to view this sort of thing would probably be built upon ncview. I think, yeah, Amulya just shared a link to that. So like learning how to use that very well would be good, yeah.

Yelesen, Sebastian 1:10:15

Thank you.

Yeah, I've just seen the EarthKit ECMWF just released EarthKit recently, which is nice. I haven't actually experimented with it yet.

Amulya Chevuturi 1:10:35

Yeah, and...

Matt Dalle Piagge 1:10:35

Yeah, me neither. I'm checking that one out at some point.

Amulya Chevuturi 1:10:38

Yeah, Kit has posted the EarthKit link, but the great part about Earth link is EarthKit is that they have put in very many different, like there's one hydrology EarthKit, there is like other, so there's a lot to explore. So I would recommend having a look at it.

Actually, I myself haven't looked at it in detail. I've just gone, skimmed it.

Amulya Chevuturi 1:11:01

But it looks very good.

Yelesen, Sebastian 1:11:03

Yeah.

Yeah, thank you.

Matt Dalle Piagge 1:11:07

You're welcome. All right, we'll hold it there, I think, given in the interest of time. I'll let everyone have a breather and a break.
I say be back in 5 minutes, 6 minutes. At 10, we'll start again at 10 to 12.
See you soon.

All right, I think we're probably ready to get going again.
This is my favourite part of this workshop. This is where we get to talk about large gridded data sets and introducing Xarray and Dask for working with those two really powerful tools. So,
As before, we'll follow the same idea. I will share a link.
to the notebook that we'll be using in the chat.
I know that works, okay.
Yeah, good.
I will share my screen so we're all on the same page.
Alright, I'll give everyone a moment to load up.
Load up the workshop, the notebook for this final part and copy it to their Google Drive if they want to say, if you want to save it.
And yeah, then we'll follow along. We'll continue in the vein that we started as. We'll go through the notebook together. I'll talk through what some of the code is doing. And there'll be a couple of places where we will type the code in together, just to make it a little bit more interesting than me.
me talking at you for the next half an hour and you just clicking, clicking, running, clicking and running code. But also, there's not too much of that so that it can be a resource that you can come back to. I did forget to mention previously that if
you get stuck or I'm going too fast or you've missed a bit of where I'm typing in the code and you need to see it again, do drop a message in the chat. Do feel just to quickly say that. We can always go back to it. That's not a problem.
All right, without further ado, let's get going.
As always, the first thing to do is to run these first two cells, which install the packages that we're going to be using. Xarray is core amongst them, but actually that comes with everything else, so we don't need to install it. But there are some other stuff that we need to run.
This will probably take a few more seconds than the previous one, as there are a few more packages to install. So whilst whilst those packages are installing,
I'll talk through what we're going to do.

Oh, sorry. So we are going to read in an example of a large gridded rainfall data set, which in this case will be the rainfall data set that CEH have produced called GEAR. It is an acronym that stands for something, and of course I can't remember.
but it is a grid data set derived from observations that is on an hourly time step that spans many years and is a key data set that can that is used in all sorts of in all sorts of places and is a great resource. It is a very large data set. In a few minutes, we'll see just how big it is.
And so some of the things we'll go through here will be how to cope with that and how to work with such large data sets.
Looks like the code installation and the package import has run already. Again, didn't take too long, but sometimes, because this is a free compute resource, Google Colabs can take a bit longer on other machines or for different people.
So as before, oh, thanks, Gianni. As before, if you're getting behind, do shout and we can I can stop and we can we can pause for a moment. But assuming everybody is OK, I will scroll down a bit and we will continue.
with the next cell, which is showing you how to read in these large data sets when they are too large, essentially, to store on your local machine. So...
The gear data set, I think, itself is several terabytes in size. It would probably not be the wisest choice to try and download the entire thing so that you could just access bits of it, or even if you wanted to access the whole thing for whatever you needed to do. That is several terabytes of disc space you need to somehow find.
So what I'm going to show what this cell does and what this workshop is based on is how to access that data without having to download it to your computer first. This data is stored on object storage or basically in the cloud and we've made it available for
for people to access that way. There are a few tweaks you might need to and gotchas to be aware of perhaps that come with working data that's stored remotely and not on your computer locally. But that is the point part of the point in this workshop and we'll go through that together.
This first code block basically reads in the data set from the remote storage solution. I will explain why there seem to be two blocks of code in a moment. But first of all, just wanted to explain what some of what this code is doing. This is based, this line here is basically the path to where the data is. And this
line here, this endpoint URL is basically the website on which this data is stored and the path to it is local to that particular repository and that particular website. The

reason I've got two
two sets of reading in functions here, this top one versus this bottom one, is that one of the key things about working with data that is stored in this way, remotely, is that for the...
reading of that data to be efficient and for your analysis to be quick, given it is essentially you instead of accessing data on disk, it is accessing data that is stored on the internet so that it has a longer, much longer latency and lag times. You need to be very,
No, you don't need to, sorry, that was the wrong phrasing. The way the data is stored, specifically the way it is cut up into lots of different pieces, which is how it works and how this particular format works and how it makes and how it is easier to access and work with the data, how you do that,
is key to how performant your analysis will be.
I'll show you a diagram of that in a minute to make that a bit clearer. But for now, we're just going to run this code cell and read in two identical versions of the data set that are essentially what we like to call chunks differently. So they are basically this, the terabytes of data are split up into different.
into different blocks and different chunks in a different way. And which one you use for different analyses has a big impact on how performant neural analysis is when working with data that are stored remotely. So once we've run that, we can now have a look at the data, or at least have a look at the metadata, because no data has actually been read in yet.
We can do that by running the next two code cells.
And let's look at the first one to begin with. This is what Xarray displays data as. So anything that Xarray can read in, which in this case is a large group of data set, but there are many different formats and data that Xarray can read in. They will all look like this.
NetCDF is the classic data format. We are working here with Zarr, which is very similar to NetCDF, but the format is designed specifically for accessing, being accessed remotely with these, this chunking thing that I keep talking about. So
Having a quick look at this, the metadata before we move on, the top line is showing you the dimensions that the data is on and how big they are. So we can see that we've got a lot of time points to about 200,000 to be precise. And we also have our Y&X dimensions as well.
More information on the of the coordinates, including these time, Y and X dimension

is in the first bit of the display under the coordinates heading, which we can expand and contract, so it would seem. The main coordinates are highlighted in bold, but also the ancillary coordinates, which are basically derived from
the main base coordinates that the data is on are also displayed and are also available for use. We won't get into that here, but there is more information in some of the links that are scattered throughout the workshop and at the end. And then the data itself, the data variables themselves is in this bottom section.
In data variables.
Um...
In this case, we have free data variables, and we will be looking at the one called rainfall amount.
as that is hopefully named and is about the rainfall. And if you wanted to find different, find out what the other ones were about, a little bit more information is available by clicking on the page icon next to there.
next to the variable of interest. What I'm particularly interested in here is the other icon that you can click on, which is this, I think, representing a disc platter, and I would like a hard disc drive platter, but anyway, icon next to the one I just clicked, which shows you
information about this chunking thing I kept talking about. If we expand that, click on that button for both data sets, for both the rainfall amount variables in each data set, we can see a diagram that represents how these data sets are split up.
how they are chunked differently. So in the top one, we can see that it has been chunked in what I like to call a spaghetti chunking fashion. I think that I can't claim the, I can't claim, I can't copyright that one. I think someone else came up with it before me, but it is memorable where we have these thin strands
of data where it has been chunked, i.e. split up into long, thin strands over the spatial dimension, and those chunks run along the time dimension. So you have small bits, small chunks that cover a small bit of the spatial domain, but the entirety of the time domain.
Whereas if we look at the other one, which I have hinted to with the naming of the variables being calling it time chunk, is chunked differently and continuing with the Italian pasta based metaphor, I'm going to call lasagna chunking as it's chunked in big fat layers.
Something like that anyway. So instead of being chunked over space, it is chunked along the time dimension, this long one, going back and looks a bit like a lasagna.

And it's a weird analogy, but I'm running with it. I like it.
And yeah, that is important to get our heads around in some sense or to remember when we're working with this remotely stored data. It's not usually something you have to worry about if you're working with data stored on disk, but if you're working with data that's stored remotely, it becomes very important to how fast your data access and analysis is.
is.
I think that is probably enough about looking through that. And let's get to some plotting and...
accessing this data. So scrolling further down.
I am going to skip over this cell here, so no need to run that one.
these little cells here.
Again, I'm just showing you that information I looked at earlier. It's another way of accessing that. You can pull it out directly with the dot data command. And one way of subsetting the variable of interest you want to look at is kind of like it was with pandas, is using the square brackets and notation.
where you type in the name of the variable of interest.
All right, now there is a there is a wall of text next, so be prepared for that. I will I will talk through it.
but I will talk through it using examples. So let's scroll through this computations and visualisations set with our Xarray and Dask text and get to the code cells beneath it.
Because I will explain a lot of that stuff with the examples that come next. So we are now on the
Insert code here.
So, let's see if I can remember what it was I wanted to show here. Now, fortunately, I have like given myself instructions in the previous cell. So, what we're going to do is just do a simple extraction of some data like we would with pandas. So, we will use...
Why has it done that? Sorry, it just jumped to the top. I'm going to...
Re-go back to this cell, so...
Everyone assumed that didn't happen.
Um...
everyone and type along with me if you would like.
We are going to select out a time point.
from this time chunk version of the data set, and I will explain why in a moment. We want to look at the rainfall amount variable, as that's the one of interest. And we can,

without getting too fancy, do some very simple extraction of
Data, so we're going to just pull out randomly the 37th or 36th time point.
And these colons again mean that we want all of the other dimensions, which are the X&Y spatial data dimensions. And we want to look at what that is.
So that's where the dot block comes in. So far, I hope fairly straightforward.
We run that, it takes a little bit of time, but not too long, and produces a very quick plot of the data, which you haven't, as a reminder, isn't stored on your machine, but is stored up in the cloud.
Now, the reason I used the time chunk version of the data set here is that we're pulling out one time point. And if we were to use the space chunks version of the data, if I scroll back up and show you what that looks like,
Here, if I used this chunk of the data, I would have to pull in the, because of the way the storage system works, it can only read in units of these chunks. It would have to read in the entire time series of one of these spatial chunks. So let's say these spatial chunks are 10 kilometres by 10 kilometres.
it would have to read in for each spatial bit of the data set, for each, you know, 10 kilometre by 10 kilometre bit of the data set, it would have to read in the entire chunk, the entire time series as well, which would mean you would essentially end up trying to extract out the entire 1.5 terabyte data set.
just a few one time series at a single point. Whereas, if we use this other chunking, the time chunking method, or...
to reuse my lovely pasta-based analogy method, the lasagna chunking method.
we only need to extract out one chunk and we only pull out, we still use slightly more, pull out slightly more data than we want to because the, you know, the time chunks do span a particular bit of time, not just one time point, but we are still pulling out only one chunk versus
the entirety of the data set. I hope that made sense. I will happily explain more about this at the end when we have questions, because I appreciate it's quite a lot of new information.
But that is the one key part of working with large group of data sets when they're stored remotely.
So let's now go back to that plotting bit of code underneath this lovely wall of text.
The.
And now...
scroll to the one underneath where I introduced one further bit of new information.

which I hope isn't too much. And again, there's plenty of more. If this is not making too much sense, I'm happy to take questions on it. And there is more information scattered in the links in that text above and further and further down at the bottom of the notebook as well.
Because...
We, when we extract out data from the cloud, we need to do it in parallel because that is how it becomes performant. Xarray library that we are using to read in the data uses a parallelization library that is called Dask. Basically, this speeds up.
the access to the code. It's not always needed. It's there in the background as it helps you and it automatically speeds up data analysis you're trying to do by automatically running different bits of your analysis in parallel as it sees fit. It doesn't always work at speeding up.
But for simple tasks, it is now the de facto standard for reading in data in this way and runs automatically in the background and you don't have to think about it. The reason I'm bringing it up, and I explain this more in the text that's written above, if you feel the need to read through that later,
Is.
It does not necessarily, it doesn't work quite as you would always expect it to do. And what I mean by that is...
When you do a calculation with data in this way.
you will find that the result is not always stored in memory as you would expect it to be. So in this case, it doesn't matter too much because I am just extracting out one particular plot, sorry, one particular time point, which is not much data. But if I was working with a larger subset of the data, and I then needed to re-access that data, or do something, do another step with the data I've extracted, when I ran the next step in my analysis, that data would be extracted from the start and reprocessed from the start, rather than the intermediate step being saved in memory.
To get around this and to help it behave a bit more intuitively, Dask has this dot compute function. What that does is that it does the calculation and then saves the output.
to memory, into the memory, so that it's there for you to work with later on, rather than you running an analysis, perhaps then plotting it to check what it looks like, and maybe the analysis took a few minutes, and you think, okay, I plotted it, the data's there, ready for me to run and ready for me to see what it looks like.
Sorry, and I'm ready for me to continue my analysis with. If you then did the next

step of your analysis, even if you've saved the variable to a, you know, saved data into a variable.
that the analysis will be run right from the start again. And it would extract out everything and read out, read in the data from the start and redo your entire analysis, including now the 1st and the second step, rather than just doing the second step. That is genuinely a useful behaviour in some situations.
where the intermediate result is too large to be stored in memory.
The great thing about the Xarray and dash combination is that it does that, handles that automatically for you. If you are reading in a huge amount of data and need to analyse lots of different bits of this 1.5 terabyte data set, it can, it knows how to handle that.
and can process all of that data, even though it is larger than would fit into the memory of your machine. And that is a feature of it that I really, although it's a bit complicated to get your head around, I really wanted to highlight before moving on.
Right, that is enough of me talking. I think let's actually run some code again now.
This dot compute function, as I mentioned briefly, just to reiterate, saves the intermediate, saves an intermediate output in memory so that you don't have to then recompute it later. So
We can now run this code. And as you can see, this analysis doesn't take, this extraction only takes a few seconds. And so there's no need to what we wouldn't need to really worry about using this dot compute function here. I'm only using it to show you what it does.
And we don't have time to do longer analysis in this workshop. But it is definitely something that if you're working with large, very large grid like this one, you will find yourself needing to use. Now, I realise that it's a lot of information. So again, just a reminder,
But that is, I tried to explain that well in the text above this particular point in the workshop, in the notebook, and there are plenty of resources, one of which Kit helpfully put in the chat for you to look through and are linked to from elsewhere in the notebook. So getting your head around this functionality.
which you wouldn't otherwise have to worry about if you were working with data that's stored locally on disk.
Okay.
Let's carry on. I think we will skip over the next code cell in the interest of time. And we will talk a little bit about.

How to make, uh, we will, yeah, we will.
Stop talking about the all this lovely parallelization and large data stuff, and we will do a little bit of...
I spent a little bit of time showing you how to plot it nicely, rather than just the default plots that Xarray can produce of these large data sets. And then I'll show you An analysis, one example analysis that I particularly like, which is how to extract out bits of this data set using the shapefiles that we saw earlier in the workshop in the previous part.
Right, let's talk about some plotting.
If you're following along with me, we're on the code cell that mentions and imports a package called cartopy. That is essentially the, I would say, almost a default plotting package for spatial data in Python. It is a way of producing nicer plots than Xarray or pandas can produce on their own.
So let's import that code.
And let's...
type some, let's actually type some code. And the reason I slowed down there was because I have to remember, of course, off top of my head what we should be doing here. We're just looking at the top line of the next cell. The other lines I will very briefly talk through, but they are essentially there to
show you examples of how you can customise plots.
and add stuff like coastlines and make it look nice. But in order to do that, the first thing we have to do is tell the plotting library that we want to produce a map plot using cartopy. And how we do that,
is we take the data we've extracted, which I think I called the plot point.
Let me just have a quick look.
As I extracted it out here.
We are going to do doc plot as we always do.
But this time we're going to add a few bits in as well. We're going to say we want to produce.
This, I think, not very helpfully named plot type called pcolormesh. It's essentially producing a, you know, a colour plot of the data. Nothing to it than that. It's just something that you don't have to remember, but that you can copy down in future. But it is the easiest plot for producing map plots.
And then we add a few, a few, sorry, a few arguments to the plotting command. The most important one being this.

slightly unhelpfully named subplot keyword arguments or keyword arguments. function. Don't worry, I know it's a bit complicated. Don't need to worry about remembering it. It's just there. It will be there for future reference. But this is an important line of code to remember how to get access to cartopy's plotting functionality.
And inside that argument, we will write projection to say we want to plot a map projection equals CCRS, which we imported as part of the cartopy package earlier. Dot.
I think we just do, I think it's just OSGB.
This is where, again, I can't remember what exactly I meant to put here. And the reason I put OSGB is that CCRS package contains the list of all the possible projections you can produce. And I recommend exploring that yourself. If you're doing
plotting stuff across the world and in different maps and different projections. But for now, we're just going to use OSGB, which is essentially a particular map projection designed for the UK.
Um...
I will scroll up, but leave that on the screen because I reckon I realise that that's quite a lot of typing and quite a long line to run to type. But when I run it, we will find that Matt has remembered to do not remembered to do a particular thing. And that is to add to this top line.
to give the plot a name, assign it to a variable so that we can then customise it and add coastlines to it later.
Wait a few seconds.
And there we go. Not an awful lot happening that we can see in this particular plot. So we can customise it further.
By adding arguments to this plotting command, I'm going to make the colour map a little bit easier to to to read essentially, and colour maps can be changed by adding. this argument.
To the command, in this case, cmap stands for colour map, and we're gonna make it. blue, essentially, which I think is appropriate for rainfall.
Wait a few seconds.
Still quite hard to see what's going on here. It looks like we have a colour bar that is going out quite a long way, but not very much data on the map is actually at that level. So a lot of the other data seems to be faded and misted out. To get around

that so we can more easily see the lower values of data,
There's one more function I wanted to show you, one more thing I wanted to add, and that is that Xarray has a...
function called robust or an argument called robust that we can add to these, to your plots, which make sure the colour bar does not use the highest and lowest values, but uses like the 2nd and 98th percentile so that it isn't, so that the
the highest values in your data set don't dominate everything else. I mean, you can't see the underlying variability. You'll see what I mean when I run this and plot it.
So for example, so now we can see a lot more of the variability of the data by cutting off that very, very top bit of the data of the scale. We can see that actually there was a fair amount of rain at this particular time step in Scotland and in the northern England.
It was just being, you couldn't see it because this, or because one of the data points I think down in the southwest had particularly has very high rainfall and was crowding out everything else. And
The rest of the lines in this code block are other things you can do to make the plot look nice. So I've showed you how to add grid lines, how to add the coastlines, and how to customise it and add a title, and all the other customization that I've showed you, like, you know, labels and all the rest of it is very similar. So I won't go through that again.
And again, this code is here for your reference to refer back to when you need to. if you need it.
I think that's probably enough about plotting. Let's do something a bit more interesting now. My favourite bit of the entire workshop, which is showing you how to extract
is doing, is doing more, is more subsetting. We've done quite a lot of subsetting in this workshop. We show, you know, showed you how to extract out using the indexes, which by the way, works in your Xarray. I haven't shown you because it works quite similar to pandas, but in Xarray you can also extract out data based on its coordinates or its indexes if you're using pandas.
So you can select out based on the time, based on this X dimension, Y dimension, without having to know where in a data set specifically it is. But I thought for this final bit of the workshop, we'd do something a little bit more exciting.
and that is showing you how to extract out shapes. So if you have a shape file that, for example, is a river catchment, and you want to just look at the river catchment,

extract out the river catchment from a larger data set that perhaps covers the entirety of the UK, which we'll be doing here,
I wanted to show you how that is possible, how you can extract out using a given shape file. It doesn't have to be a catchment, it could be anything else that you might have to hand and show you how you can do that by providing you with a function and the code with which to do that.
Again, I will go through it very quickly. This is something that you can go through more in your own time. That code is there for you to use in the future if you need it. I noticed I'm proud of my past self. It is quite well documented as it's a function that I wrote a few years ago. But yeah, it will be available there for you to use at any time.
So let's download that code with these next two cells. Download and import it.
The next cell is just reading in a shapefile. And as a bonus point, it's actually, no, sorry, I got that wrong. It's not reading in a shapefile. It's reading in a NetCDF file now, not a Zarr file, a NetCDF file. It's a single file, which you can also do from reading in from the cloud.
And this cell was showing you how to do that if that's something you might need to do in the future. But I won't spend time on it now as it's not the interesting point. So that first
cell reads in the NetCDF data that we're going to be extracting from. We also need to read in a shapefile and that's what the next bit of code is doing.
I'm going to show you quickly by typing it in. Again, you will please do type with me to help you remember some of this stuff.
how to read in the shapefile that's stored remotely in the cloud.
Uh, and that is...
using GeoPandas like last time and its ability to read in files. So that's GPD.readfile.
We are reading it in from a remote storage, so we need to provide those storage parameters and credentials. And we do that by providing it with...
those in the form of the variable that I set up in the previous cell.
and then open.
the path, then include the path in the form of sfname, include the path to the variable, which in this case is a shapefile, is the name of the shapefile. Again, I won't spend too much time on that. That code will be there if you need it. Most shapefiles are small enough that you'll probably never get it.
read it in from object storage or from the cloud, but if you need to, I want to include that code as well.

So let's have a quick look to check that's written OK.
It looks familiar. It looks like a GeoPandas table. That's all good. And it is that same file that we read in earlier, which is the catchments.
And now we get to use this catchment subset shapefile function, which is the function that I wrote, which does some of the fiddly stuff around extracting out a catchment's worth of data from larger net CDF or any other gridded data set.
I also wanted to highlight this functionality of any function in Python, is that if you are not sure what it does, you can put a question mark next to it.
and run the cell or run that code and it will give you lots of documentation on how to use that function.

Matt Dalle Piagge 1:55:11
Again, I won't go through that all now, but because there's a lot of information there. But that is how you could find out a bit more information about this function or any other function. And I'm going to very quickly

Matt Dalle Piagge 1:55:26
because I'm aware time is crawling on.

Matt Dalle Piagge 1:55:33
Type in.
show you how to use this function to do something like extracting a catchment.
So I will quickly highlight from the documentation.
These are the things, the variables we can add to the arguments, we can add to the function.
So those are the things we are going to be adding to the catchment subset shapefile function.
We need to provide it with the data that we want to subset from.
That, I've forgotten what I called it.
I have called very helfully climo. I think that's because it's a climatology. So that's the NetCDF data that we're subtracting, extracting from. We need to.
Tell it where the shape file lives.
I have already defined that above in the sfname variable, if you remember, that's here.
I also need to provide this endpoint variable, which is, again, telling it where the

remote shapefile in this case is stored.
I should do that on the new line.
Not helpful, Google.
And I'm just going to copy that from above for the sake of time.
Hopefully, it is giving me...
Several.
quotation points. One other thing we need to put in is the catchment that we want to extract out. And this is done by using those, looking at the shapefile that we've read in using GeoPandas.
and telling the function what column we want to use to identify the shape or the catchment we want to extract out. In this case, we'll use
We will use.
The ID string column.
and we will extract out because it's familiar to us now.
The Thames catchment called 39001.
And if I've managed to type that in correctly.
I also forgot to add in one thing.
Okay.
Just drop thing is another functionality of this particular function.
And plot that.
We should find.
Using these next cells.
That we have managed to.
somewhat not helpfully. Extract out the Thames catchment. You might just recognise that, but it looks a bit weird. I think that is because I forgot to run this drop function, add in this drop bit, which removes all the blank space around the catchment we've extracted. So I will rerun that.
It is still doing it. Oh, well, there you go. So, that's what happens in these in these workshops sometimes. What am I doing wrong there?


Tom Keel 1:59:16
Is it drop equals 1 to the tree?


Matt Dalle Piagge 1:59:18
Let's drop equals 1 equals true. There you go, Tom. You were actually reading the

documentation. Well done. I wasn't.
Okay.
And now it should be a better plot that shows we've just managed to extract out the Thames catchment from a NetCDF data set. And I won't again run through.
how to plague the plot look nicely, because I think we've done a bit of that, but there is some code below that show, again, would plot this on a map like the one we saw earlier with coastlines and nice other stuff.
Um...
That is where I think I'm going to have to leave it. I'm aware it's half past time run away from me, as it always does, you know. But again, I will spend one minute highlighting the supplementary material, which is at the bottom of the workshop of the notebook, showing you how to do quite a few more things with these large grid data sets and Xarray and
or the rest of it. There's some code that shows you how to do animations, so how to animate a plot to move forward and back in time or in space, even in space. Quite a lot of code, but I talk you through it if that's something of interest, show you how to produce a climatology using Xarray's
extraction and time handling and capabilities, which is a bit like very similar to pandas.
And then there are a huge number of further resources at the bottom for reading more about all these things that I've talked about in sometimes great detail. And for you to look through should you wish to and want to in future. I'm aware I haven't left an awful lot of time for questions, but it is important that we both have a good lunch break or a good break away from the computer, given we've done a lot of work just now.
and that you also have time for questions. So if...
The way, if people would like to post any questions in the chat before disappearing for lunch until the next time, that is fine. And then we can answer them at the start of the next workshop, if that would be okay, Tom. If you've got enough time, amazing. Either or we can answer them in the chat offline.
Outside of that workshop, but I think in the name of...
making sure of setting good practise and making sure we don't sit at our computers and screens all day. I think I will stop there. Hope some of that was useful. You'll be using some of it in the workshop to come and learning even more stuff about all these different data types and libraries we use to work with hydrological data.

And just a final reminder, this, I know I talked through a lot of stuff today. Don't expect you all to remember it, especially not the code, but these resources will be there for you on GitHub. We will link to them and send you that link to share you with that, share that link with you later for you to go back to and read through. Any at any point, as well as all the other resources that are linked through. Alright, have a good lunch break, everybody. I hope that was useful, and I will see you in the next workshop.

YS **Yelesen, Sebastian** 2:02:39
Thanks for your time. Thank you.

Matt Dalle Piagge 2:02:41
You're welcome. Thank you very much.

MM **Martha McKee** 2:02:44
Yep, thank you.

◉ **Gianni Vesuviano** stopped transcription