

Tutorial overview

In this tutorial, we will learn to:

1. Download a file from the internet
2. Load data from a file on our computer into a spreadsheet
3. Explore columns in those data
4. Select one column to visualize
5. Make different sorts of text-based visualizations
6. Make a bar chart
7. Save our plots and other outputs

Estimated time to complete this tutorial: 30-60 minutes

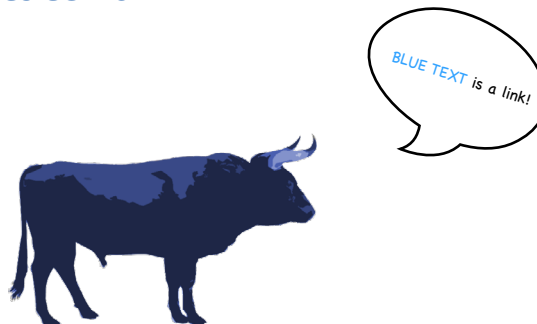
Key concepts used in this tutorial

Import from .csv format

Filter headings

Pivot tables

GUI-based figure design



Introduction

It's time to mooooooove along into this, our tutorial on how to analyze data using a spreadsheet program. This may seem herd at first, but if we calf some patience, it will work out udderly great!

Sorry, sorry. We'll stop now...moostly.

Luckily for us, [Open Context](https://opencontext.org/) (<https://opencontext.org/>) has a great [project](https://opencontext.org/projects/c89e6a9e-105a-4368-9e90-26940d7bf37a) (<https://opencontext.org/projects/c89e6a9e-105a-4368-9e90-26940d7bf37a>) on ancient cattle bones already online. So we can use this existing data set, and don't have to start from scratch.

However, before we can do anything we'll need to download the cattle bones data onto our computer. To do so [mosey on over to Open Context](https://opencontext.org/subjects-search/?proj=166-neolithic-and-bronze-age-cattle-data-from-switzerland#8/47.040/8.438/11/any/Google-Satellite) (<https://opencontext.org/subjects-search/?proj=166-neolithic-and-bronze-age-cattle-data-from-switzerland#8/47.040/8.438/11/any/Google-Satellite>) and save the file by clicking the big blue button that says 'Export or Map Records.' From there, click the smaller blue button that says 'Start Export.' It may take a minute for everything to download...

PART ONE: SPREADSHEETS

If we look at our download, we'll see that the file we just saved ends in ".csv". This stands for *Comma Separated Values*, and means that if we were to open it in a rich-text document reader, like LibreOffice Writer, Google Docs, Microsoft Word, or Wordpad, or a text editor like notepad, all of the values of the file would be separated by a series of commas (,) and new lines. While that pattern can be hard for humans to process, it's super easy for computers. They just search for that symbol and break the data up at those intervals. (We don't know how hard it is for cows. Yet.)

Thankfully, sorting this kind of data doesn't require a fancy algorithm, and most spreadsheet programs can do it automatically. Of course, we CAN tease it out using a fancy algorithm but maybe we'll explore that in the future...Like, maybe in the next tutorial? In general, a spreadsheet program can be a great way to explore our data. It allows us to sort and filter visually, seeing things be added or taken away. That interaction allows us to get a feel for our data.

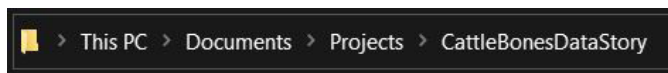
There are a lot of options out there for spreadsheet programs, so this tutorial will focus on common commands that should get us what we need regardless of the program we use. However, button placement and the names of particular tasks may vary, so don't worry if the output or directions in this don't exactly match what we're working with.

Specifically, this tutorial will be using screen captures (screen caps) and screen shots from [LibreOffice Calc](#). We're highlighting this program because it's free, offered open access, and the development team shares our interest in making computing simpler and more accessible for everyone.

Where are our data located on the computer?

The first thing we'll need to explore the cattle bones data in this environment is to locate the data on our computer. It probably didn't wander off, but we know how cows are. I mean data. We're talking about data, not cows.

From Open Context, we should have selected a location to download our table to. If it's something like a `Downloads` folder, consider moving it to `Documents` or another good folder so that any additional work we do with the data is saved in a place we'll remember. Maybe make a specific folder just for Open Context Data Stories and tutorials?



An example of a good naming structure that will associate all the parts of this project together.

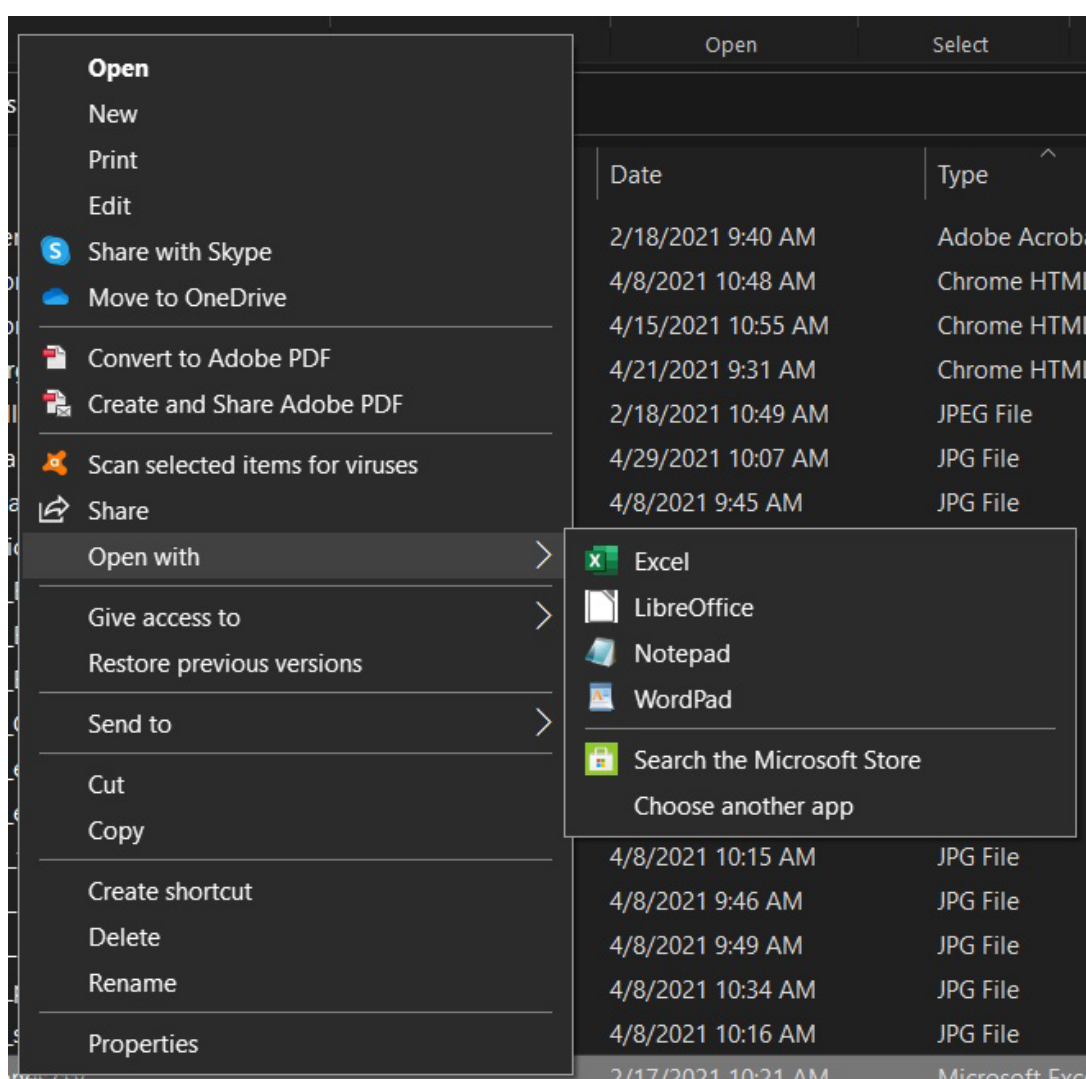
If we aren't in the habit of organizing things into folders, this tutorial is a good time to start. This way if we forget what something is called, or misspelled something, we can still find it based on knowing we're looking for the work we did for this tutorial.

PART ONE: SPREADSHEETS

So consider herding it into the same place that we've put this tutorial sheet. If we don't have this tutorial saved on our own computer, we might want to, just in case we lose internet access. It's also helpful to have all of the information for this tutorial and project close to each other, as that will cut down on organizational issues in the future.

Getting the data into our spreadsheet

Now that we've located where we put that file, we want to open it with our spreadsheet program. There are a couple of different ways to do this depending on the spreadsheet program that we choose to use for this exercise.



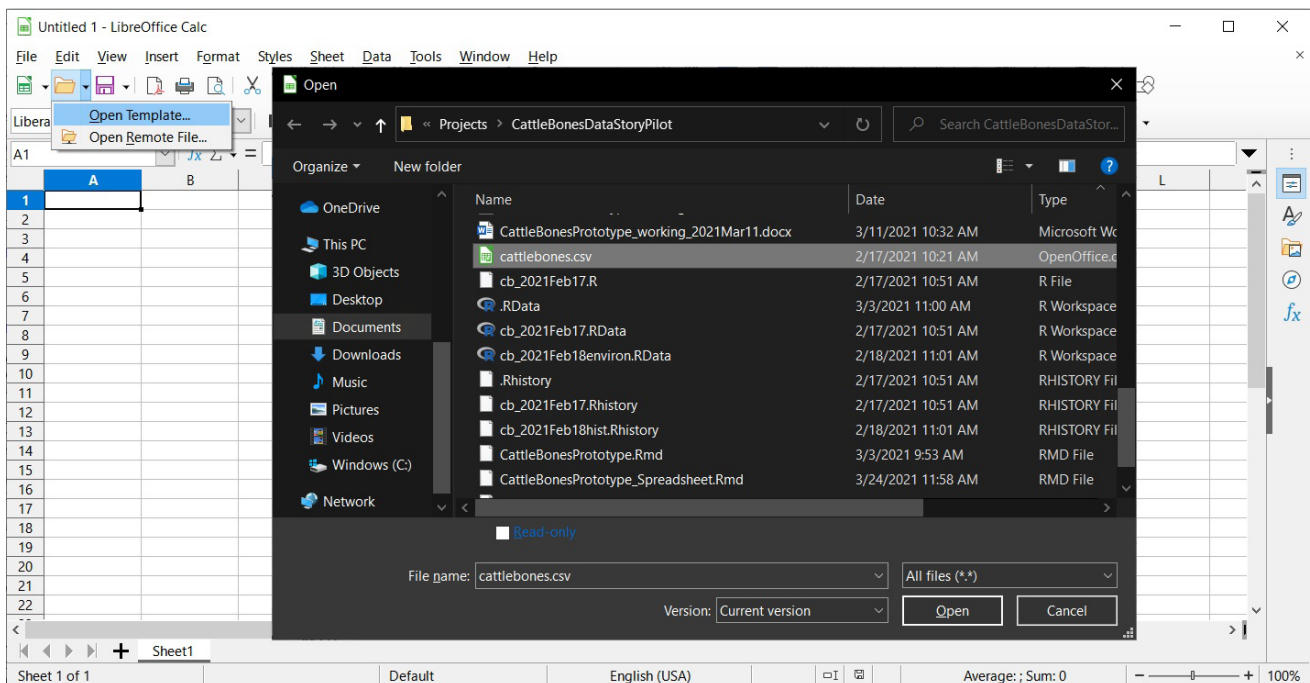
After opposite clicking, these options will show up within a Microsoft OS environment set to dark mode.

Opposite clicking (clicking like we would to copy and paste) and choosing *open with* will typically give us the option of selecting a spreadsheet program that we have on our computer. This is because .csv is a common document type that can be read by a variety of programs.

PART ONE: SPREADSHEETS

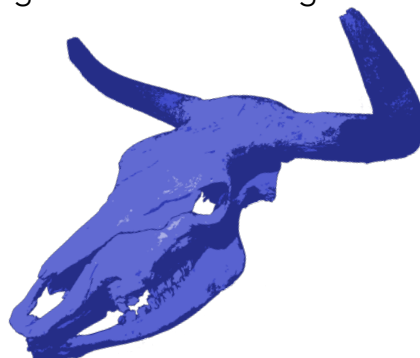
This will open the CSV directly into that program, detecting the separations between the values in each line and detecting that each new line of information should be in its own row. While we've probably seen tables before, usually we've got lines or boxes that separate out the information. In this case, the commas do the separating for us, and if we look at CSVs in a text editor (like notepad) we'll see that each line is reaaaaaally long. Also, were we to count the number of items between each comma we'd see that each row had the exact same number.

For the rest of this tutorial, the information in each unique line may be called an entry, a record or a row. The values separated by the commas may be referred to as columns, attributes, or fields, as they are the pieces that describe the specimen represented by the row. Everyone has a different familiarity with these concepts so we want to make sure that we are all in the same pasture.



In LibreOffice Calc, the open screen will pop up after we click Open Template. In this window, we'll navigate to our cattle bones file and click Open to start the process.

Otherwise, we'll need to navigate to that location within our spreadsheet program. Typically this will involve heading to a *data* or *import* tab, or an *open files* section within the program. We know the programs on our individual computers best so take a look around. We'll want to select the version that includes importing from .txt or .csv if given the option.



Text Import - [cattlebones.csv]

Import

Character set: Unicode (UTF-8)

Language: Default - English (USA)

From row: 1

Separator Options

☒ Fixed width ☐ Separated by

☐ Tab ☒ Comma ☐ Semicolon ☐ Space ☐ Other

☐ Merge delimiters ☐ Trim spaces String delimiter: "

Other Options

☐ Format quoted field as text ☐ Detect special numbers

Fields

Column type:

	0	10	20	30	40	50
Standard						
1	"URI"	"Label"	"Project"	"Project URI"	"Item Category"	
2	"http://opencontext.org/subjects/ba499699-c300-4b5e-96"					
3	"http://opencontext.org/subjects/9050955c-6878-49cb-a7"					
4	"http://opencontext.org/subjects/dc53038e-4fb3-4c0b-92"					
5	"http://opencontext.org/subjects/ac3e0090-c5d4-4b7a-b0"					
6	"http://opencontext.org/subjects/bfd2a79a-9f46-4223-8a"					
7	"http://opencontext.org/subjects/95dead7a-c376-4f2e-be"					

Help OK Cancel

Once we open the file, this window will pop up asking how the information in the file is organized. This is where we'll be asked if the information is at a fixed width or delimited/separated by a particular character.

Some programs will detect that the CSV has an established pattern, others will ask us to define how the data are organized. These are usually divided into categories like "fixed width" or "delimited". Sometimes, we may see them called "separated by". Fixed width means that the data are separated based on the number of characters within each column. For example, if there were no spaces within the file but we knew that every 3 characters and then every 5 characters started a new section we could define those typically by clicking on the scale at particular intervals.

A CSV, as the name implies, will use the delimited or separated type. 'Delimited' or 'Separated by' means that each column or attribute is separated by a specific character that should not appear elsewhere. In this case, it's a comma (,) but it could also be a forward slash (/) or a semicolon (;). As long as it's used consistently we can find that character until the cows come home.

Additionally, the program may ask us to define the *Character set*. Unicode (UTF-8) is a good choice because it can encode characters in most written languages, including Korean and Japanese. Once we've selected those options, the spreadsheet program will break the data into separate columns. The first row will describe the attributes accounted for in the following rows and our program may ask us if these are headings or headers. Say yes.

With that, we've successfully opened our CSV in a spreadsheet program and can start exploring the data!

What kinds of questions can we ask?

Moooooovelous, we've successfully added CSV data into a spreadsheet and can see the thousands of entries in this table. With these set, we can explore the attributes available for each row in the table. However, many of our questions depend on what we know about the data. Specifically, lots of the measurements are unique to a kind of bone and might require additional background information to utilize effectively and ask the right questions.

A question about this data set that does NOT require so much other information though is, "What periods are these bones from?". While an exploration of the Open Context website gives us a basic visualization of under the image labeled "[Chronological Distribution \(Years BCE/CE\)](https://opencontext.org/subjects-search/?proj=166-neolithic-and-bronze-age-cattle-data-from-switzerland&prop=oc-gen-cat-animal-bone#8/47.040/8.438/11/any/Google-Satellite)," (<https://opencontext.org/subjects-search/?proj=166-neolithic-and-bronze-age-cattle-data-from-switzerland&prop=oc-gen-cat-animal-bone#8/47.040/8.438/11/any/Google-Satellite>) we can look at this questions in other ways.

To do this, we'll need to look at the columns for headings that say something about time. Some keywords might be things like "period," "chronology," "temporal", "time", "date", "early", "late" and other similar words. We'll scroll to the right along the first row to identify those columns specifically.

To focus on the columns related to time, we can choose to "hide" all columns that don't include those words by opposite clicking on the column and choosing the "hide" option. Or, we can delete these columns by opposite clicking on the column and selecting delete.

Should we choose to go the delete route, make sure to locate our *undo* button within our spreadsheet program or the keyboard command for that. Ctrl+Z (or Command Z) will quickly become our best friend. Additionally, deleting columns will alter the data set, so we should probably save our file under a new name so that we don't need to re-download the original file. But if we need to re-download it, no problem, just go back to the head of the herd for the tutorial to follow those instructions.

PART ONE: SPREADSHEETS

While we scroll through the headings, how many attributes does each row in this table have? What's the easiest way to tell? Well, we can count how many columns there are, but most spreadsheet programs also label the columns, even if they don't have names. Unfortunately, these are often letters, to offset the numbers that identify the rows. This means after 26 we start getting into double letters like AA as names, so we'll need to do a little bit of addition anyways.

Knowing how many attributes each row has can help us understand how many different ways we can separate and explore our data. It can also give us an understanding of what attributes the creator of this data set thought were important to record, which defines what questions we ask. But why is that?

Well, we can't ask questions about information that we don't have data for. So we need to consider our questions within the scope of the data we're given. For example, we might have some questions about the order that these sites appeared in time. This is a really important question because it might relate to larger issues about cattle domestication or hunting practices! However, not all attributes about time allow us to answer that question in the same way or at the same level of detail. Hold that thought though, we'll come back to that later...

Now that we've explored how many columns there are and identified the ones that have to do with time, let's pick a specific column to explore in detail. We can select any we'd like and still follow this tutorial, but the displayed results may look slightly different. As a check, we're going to use the *Period* column to explore the data and answer our question.

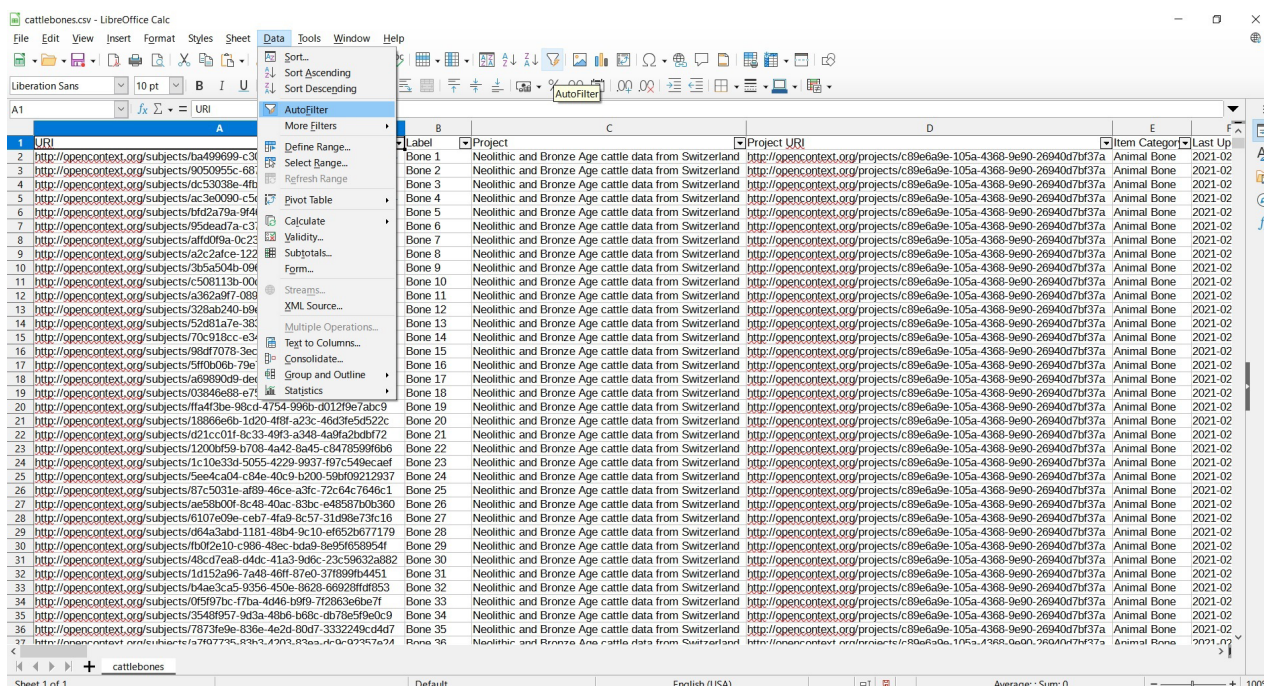
How do we visualize our question as a table?

Awesome, we looked through the various table headings and found the *Period* column, now we can figure out a few ways to explore what periods are included in this data set. If we're unfamiliar with archaeological terminology, 'period' generally refers to a particular chunk of time in the past. We may see the term "time period" but often that's flagged as redundant and just 'period' or 'time' will suffice. Periods are generally named chunks of time. However, around the world periods can be tricky, but we'll explore that in the supplementary material for this, and in the R based tutorial we'll be trying after this one.

Even focusing on this one column, the table is pretty intimidating. A list of over 10,000 entries is a lot, but we can use filters within a spreadsheet to summarize that data in a variety of ways. Filters allow us to explore a portion of the table at a time. In our case, only the rows that have a particular period associated with them.

To do this though, we'll first need to filter our table. Filtering will focus on the headings. Headings are typically the first row of a column and describe the kind of information stored in that particular column. When we filter, the program finds all the unique values underneath it and allows us to select a few of these to decide what rows we want to look at.

PART ONE: SPREADSHEETS



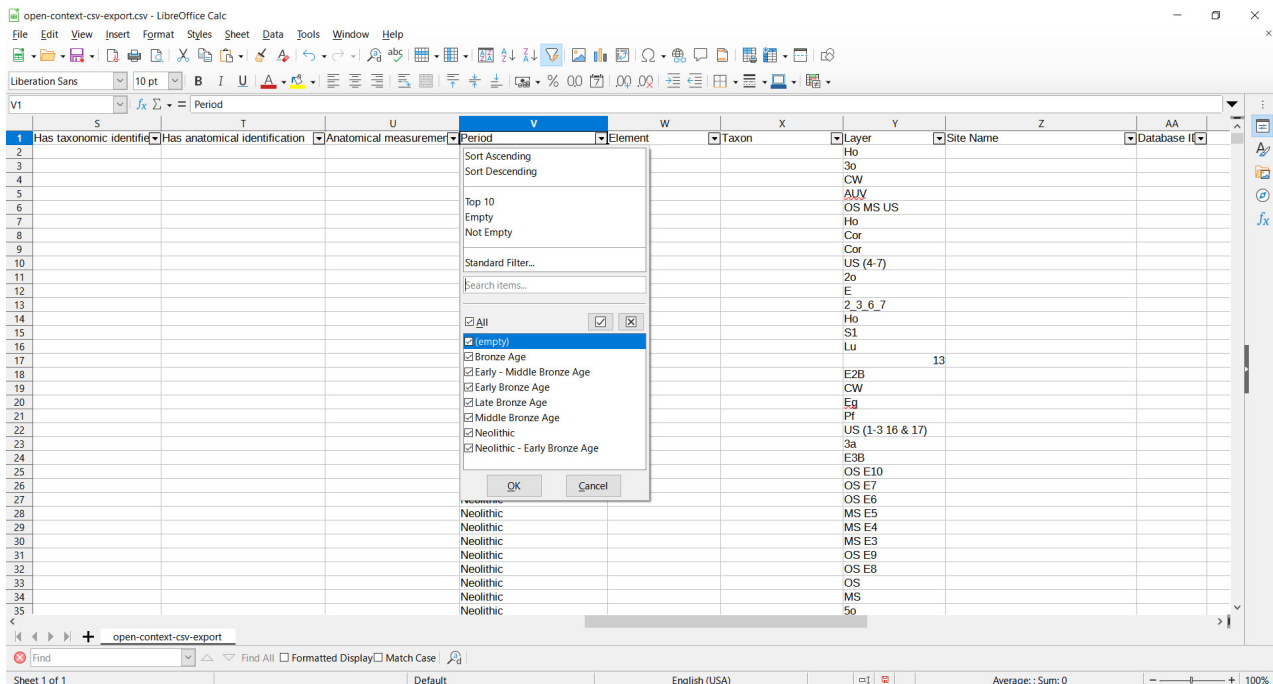
In LibreOffice Calc, we can autofilter by clicking on the “Data” tab and scrolling down to “AutoFilter” or we can click the autofilter icon in the main ribbon.

To filter our data, we’ll need to find that option. This can be a few places in a spreadsheet program, such as under the *Data* tab or in the main ribbon as an icon. However, if we have trouble finding it we can use our program’s help function or the internet to figure out where this feature might be. It may have a symbol like a funnel or an inverted triangle with horizontal lines on it. It may also be an automatic feature that adds a little caret or inverted triangle to the box where the attribute name box is without us doing anything.

Take a few minutes to figure out where this feature is before moving on, as switching between spreadsheet programs (something the author of this tutorial does do with some regularity) can be disorienting.

In the meantime, we’re doing great! Using spreadsheets effectively is a skill, and searching for tools like this helps us build these skills to answer questions in our own research in the future. We’ll be wrangling this data in no time!

PART ONE: SPREADSHEETS



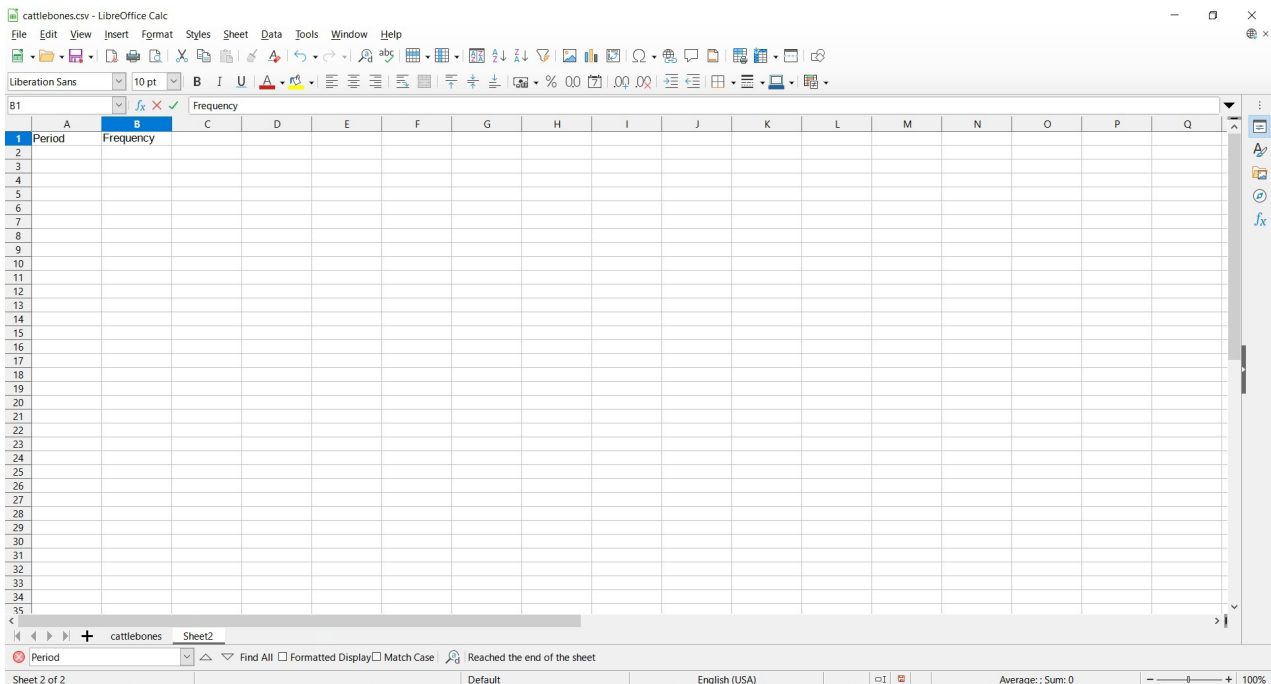
In the Period box, column V, a caret appears that can be clicked and will open this menu.

Once we've located the filter option, we can click on the caret it adds to the name box to see how many periods there are within that particular column. This helps us understand the diversity of what things can be called within this column.

Based on this, there are seven (7) periods in this particular table. Looking at them we might not be familiar with exactly what they mean, but they let us know that we can sort all, or at least many of the rows, into these particular categories.

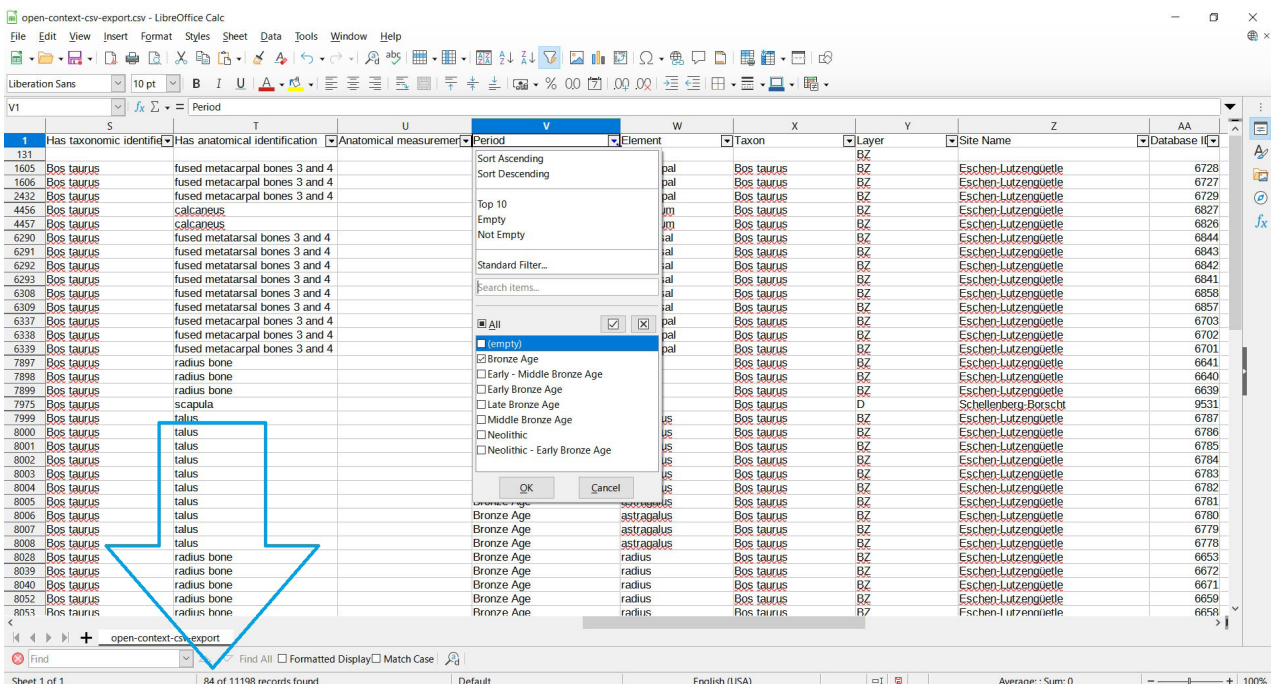
Because we decided to explore the periods that these specimens are from, we're probably interested in knowing how many specimens are from each period. There are a couple of ways that spreadsheets allow us to do this. The first is to do things by hand. However, we don't really want to count and tabulate each row, so what we can do is go through and filter by each period and then note the number of entries. Using these, we can create a new table, by adding a sheet to our spreadsheet, and call it Specimen by Period. Faaaancy!

PART ONE: SPREADSHEETS



We'll need to click the + button at the bottom of the screen and a new sheet will appear. This image already has new column names we are going to fill out.

In that new sheet, we can create two columns. The first will be *Period* and the second will be *Frequency*. This means that for each period, that is how many specimens are assigned that period. We'll now go back to our original table and start noting down information. We'll want to click the filter caret in the period column and then deselect all of the periods we're not interested in by clicking the check boxes. Un-selecting them all would return us no entries.



In this, we limited the number of rows to those that had Bronze Age as their Period. At the bottom, we found the number of rows that were labeled that way.

PART ONE: SPREADSHEETS

Period	Frequency
Bronze Age	81
Early - Middle Bronze Age	248

Once filtered, I added the Bronze Age as a row under period and then the number of rows associated with it to the table in the new sheet

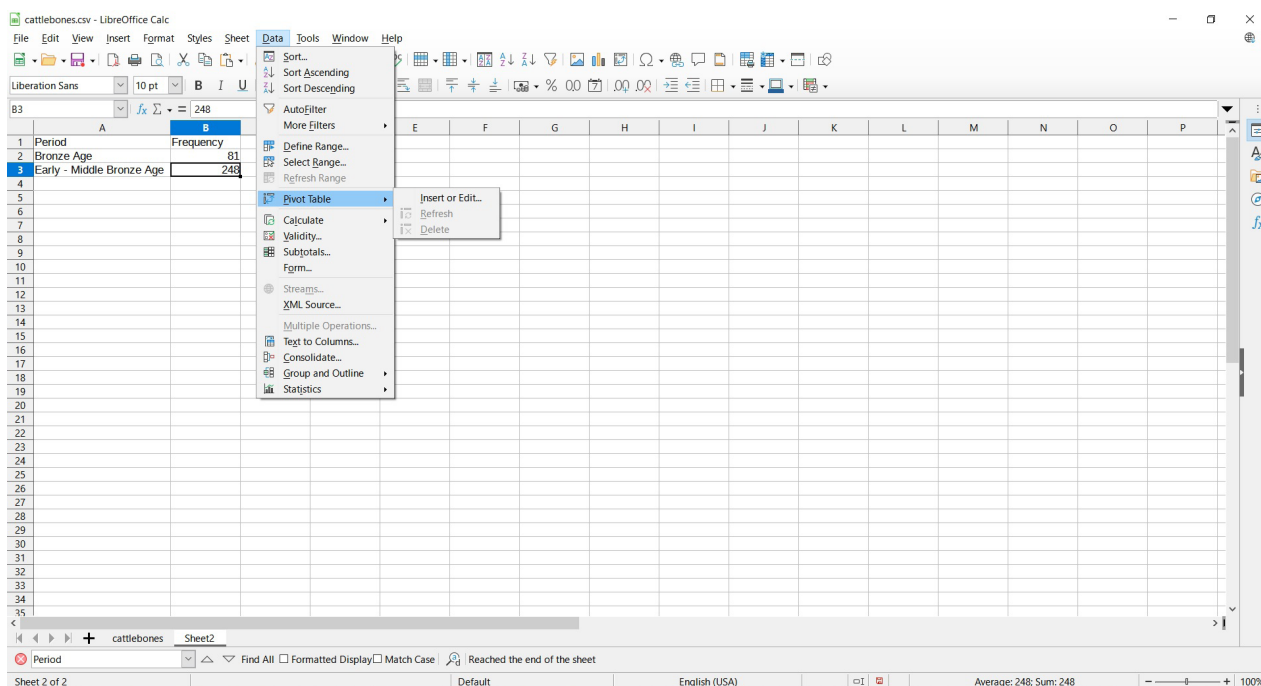
Then, we'll want to select one period at a time, clicking the check box, and counting the number of rows. (If we're lucky, our spreadsheet programs will state at the bottom of the window how many records were returned.) We'll then put the number for each period into the table under frequency. Once we've finished this, we can check that we counted or recorded the correct numbers by adding all the numbers under frequency together. We should get the same number as the amount of records in the original table.

Great! We've made one version of this table. Or we skipped this, which is totally fine, because it's tedious and is prone to errors. To skip the potential for errors we can create this table in a different fashion.

To do this, we'll explore another tool within spreadsheets after we turn off any previous filters. Specifically, we'll create what is called a 'Pivot Table'. Pivot tables allow us to summarize information from a more complicated table (like the original cattle bones one) based on parameters we establish. In this case, we want to know how many specimens are from each period. We'll create a pivot table that counts all of those for us, and lets us know how many specimens are from each period.

To do this, we'll need to highlight the column with the period information, either by clicking the heading above it or manually highlighting all the values in that column. Then we'll want to look for the pivot table option in our spreadsheet program. This might be under data, insert, or elsewhere depending on our program. Take a few moments to explore this before moving forward.

PART ONE: SPREADSHEETS



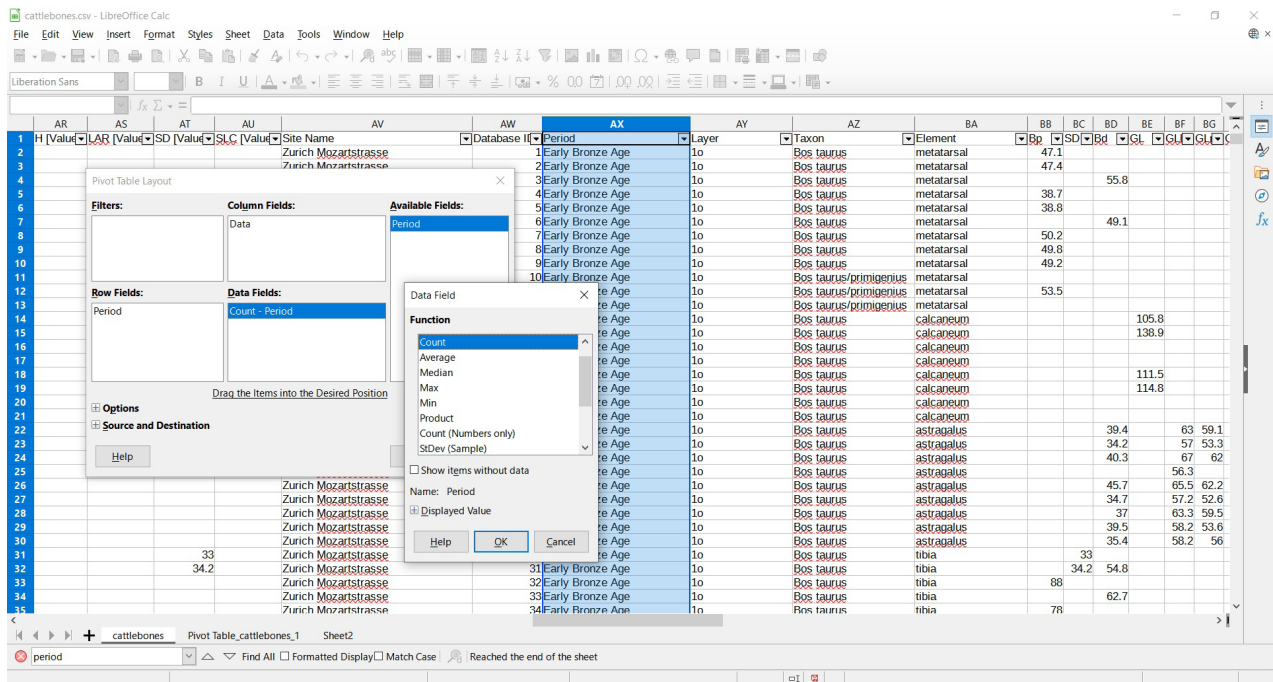
To add a pivot table, we'll once again navigate to the "Data" tab and select pivot table or click the icon in the ribbon.

Once we've found where to make a pivot table, go ahead and click *Create new* or whatever variant options we have and make sure to save it to a new worksheet, sheet, or similar so that we don't replace anything we've already made. This will create a blank pivot table space that should give us options of what data we're using, possibly represented as the columns and rows by an alphanumeric designator within a range such as (V2:V11199), or just by the name of the column. There should be options for filters, columns, rows, and values.

We will want our rows to be the period column. We'll either drag that selection to that area or check mark it. That should then generate a column with just the period labels we identified previously. We'll then put that information in values, by either dragging or selecting.

Once we do this, the space might automatically populate with a particular function. This function will be a variation of **COUNTA** or **COUNTUNIQUE**. What this does is it adds all the rows together that have that label. Since this column only has alphanumeric values most other functions won't work here. However, doing this will put those frequency values in a column by their appropriate label.

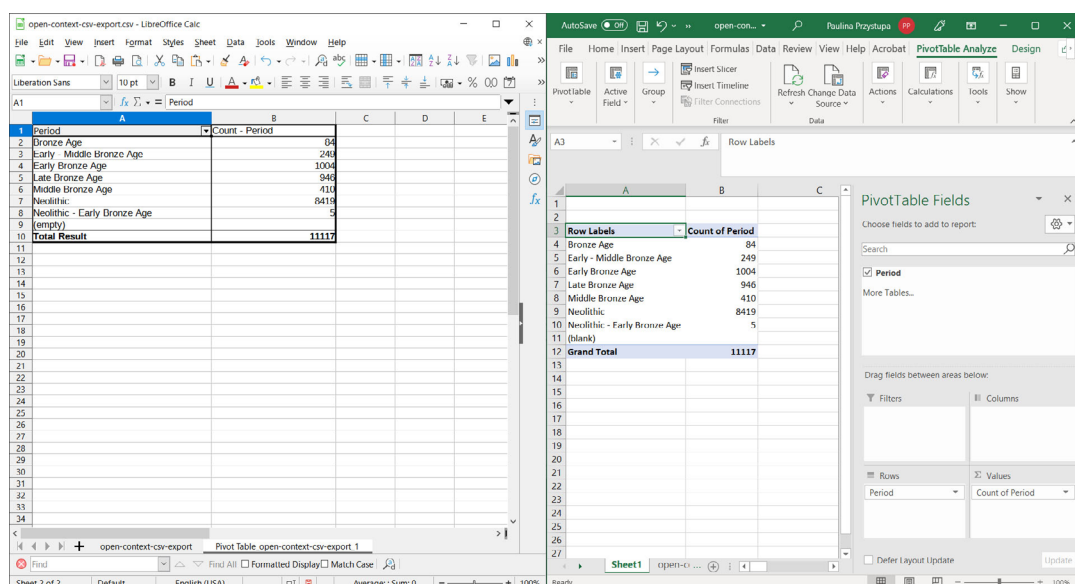
PART ONE: SPREADSHEETS



Once we've selected add or edit pivot table we'll need to specify some information in the design window.

If we did these by hand, how do these values compare? Did we count them correctly? Hoof we made a mistake? Are they the same?

They should be because the only difference is that we (humans) counted the values versus the computer doing it for us. Hopefully, they look pretty similar. If they don't, take a second to retrace our steps and see what might have happened.



Here's what a completed pivot table should look like in LibreOffice Calc on the left and Excel on the right.

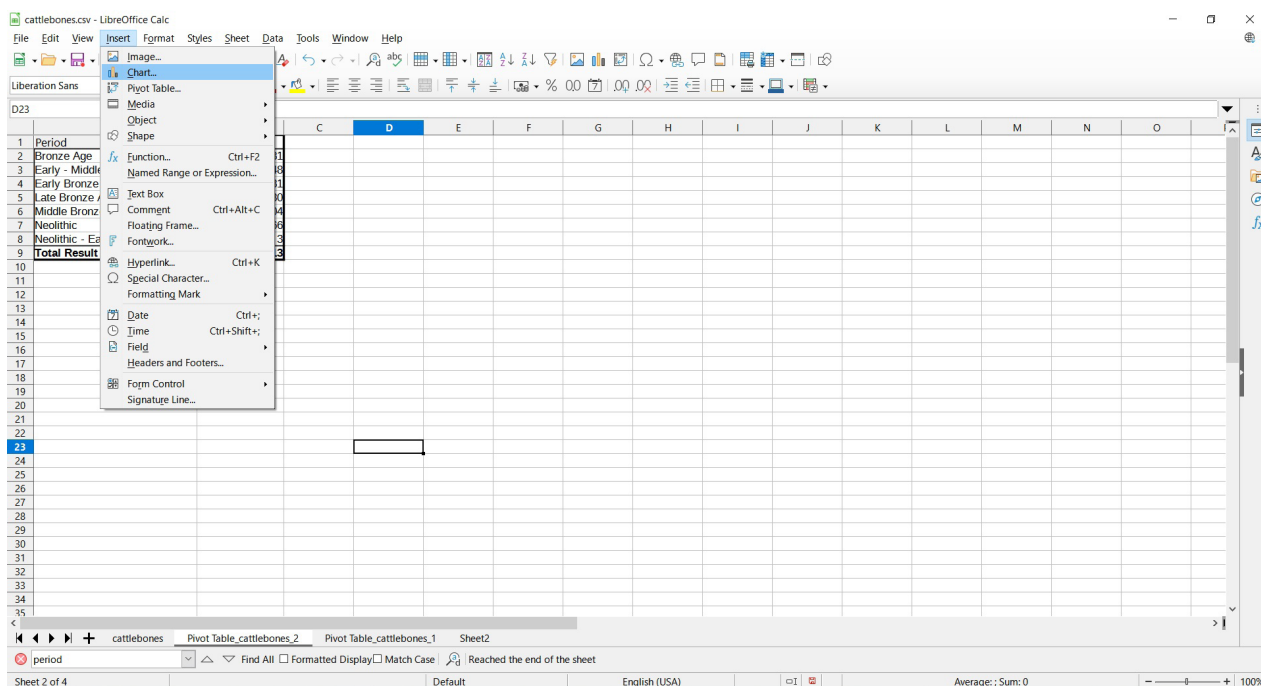
How do we visualize these specimens as a chart?

However, if they do look similar, congratulations! We made a pivot table by hand and used the automatic functions in our spreadsheet program as well. Using this table, we can look at the distribution of specimens by the period. Specifically, it looks like there are many bones from what's called the *Neolithic*.

While this table is nice, it doesn't really convey how different these values are. The Neolithic clearly has a lot more values than the other periods, but looking at the numbers doesn't display that magnitude of difference as well as a chart might.

So let's make one.

The most basic way to symbolize the magnitude of difference visually without a table is to create a bar plot, sometimes called a bar chart. Bar plots use the frequency of a particular value to stack up and compare these values visually. Bar charts are a good visualization in this case because we can use the frequencies of the data directly rather than summarizing them using percentages. Additionally, humans are really good at comparing things across straight lines so it plays to our interpretive strengths. To do this we'll utilize the pivot table version of the data to create a bar plot. First, we need to create a chart within our spreadsheet program.

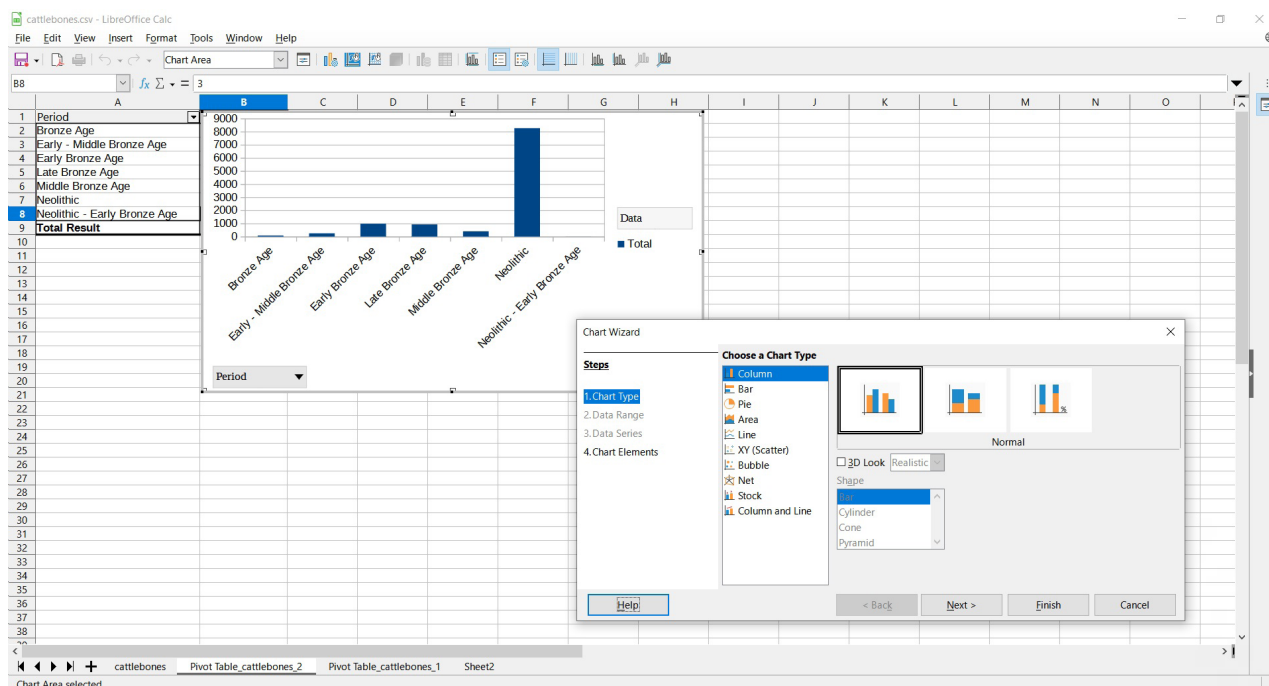


We'll go to insert, or click the create chart button in the ribbon, to start working with the bar chart.

We'll want to look for the area in our spreadsheet program where we can create a chart. This will typically be under *Insert* and will have different options. We'll want to select the bar chart option when we are on the sheet for the pivot table with the label frequencies.

PART ONE: SPREADSHEETS

If we're already in there, the automatically generated table will probably pop up populated with the values from the pivot table, named in the appropriate fashion. It might include a "grand total" column but we can ignore that, or we can remove it by changing the data range that the chart includes. However, if it has frequencies charted and labeled in a way that makes logical sense in relation to the numbers in the original table, congrats and we can move to the next section. (We really resisted making a cow pun here, but failed.)



Once we've clicked the button with the correct information highlighted, we'll need to confirm the automatically generated styles. We need to confirm these before the chart is created, so click next when this or a similar window appears.

If it doesn't have frequencies charted and labeled, we'll need to select the values we want to include. To do this, we'll want to select the column with the *Period* labels as the x-axis (or horizontal categories) then the column with the frequencies (or counts or **COUNTA** or whatever our program called it) as the "series," "y-axis", or whatever is opposite the x-axis. This will then auto fill our table.

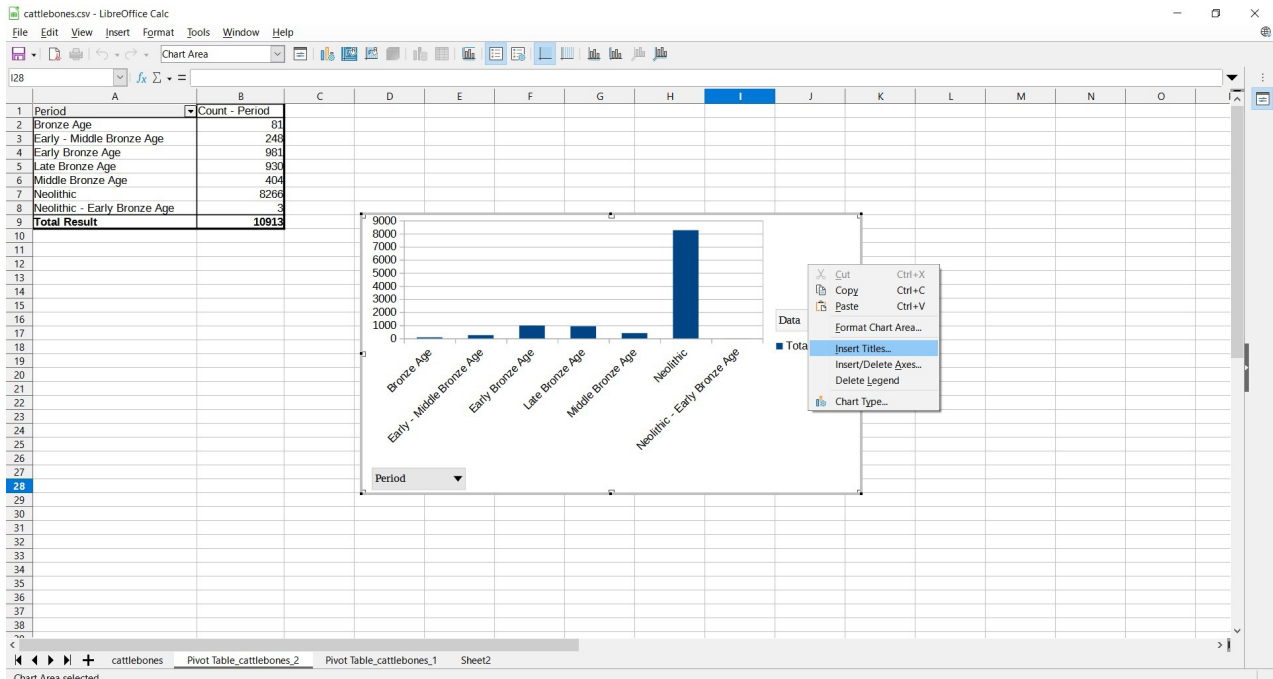
And there we have it, a basic chart that visualizes the huge numerical differences between the values in this particular data set when we look at periods.

How do we make our visualization nicer to look at?

Yay! We've got a chart, and while it does show us what we wanted, there are a few things that aren't quite right. It uses complex jargon that we don't need like possibly referring to the frequencies as "**COUNTA OF**" rather than just frequency or "number of" which make a lot more logical sense.

PART ONE: SPREADSHEETS

Additionally, this extends into the y-axis as well. If we wanted to show this chart off, we'll probably want to fix some of these issues. Most spreadsheet programs will now just allow us to click on the area to edit it directly, so why don't we do that.



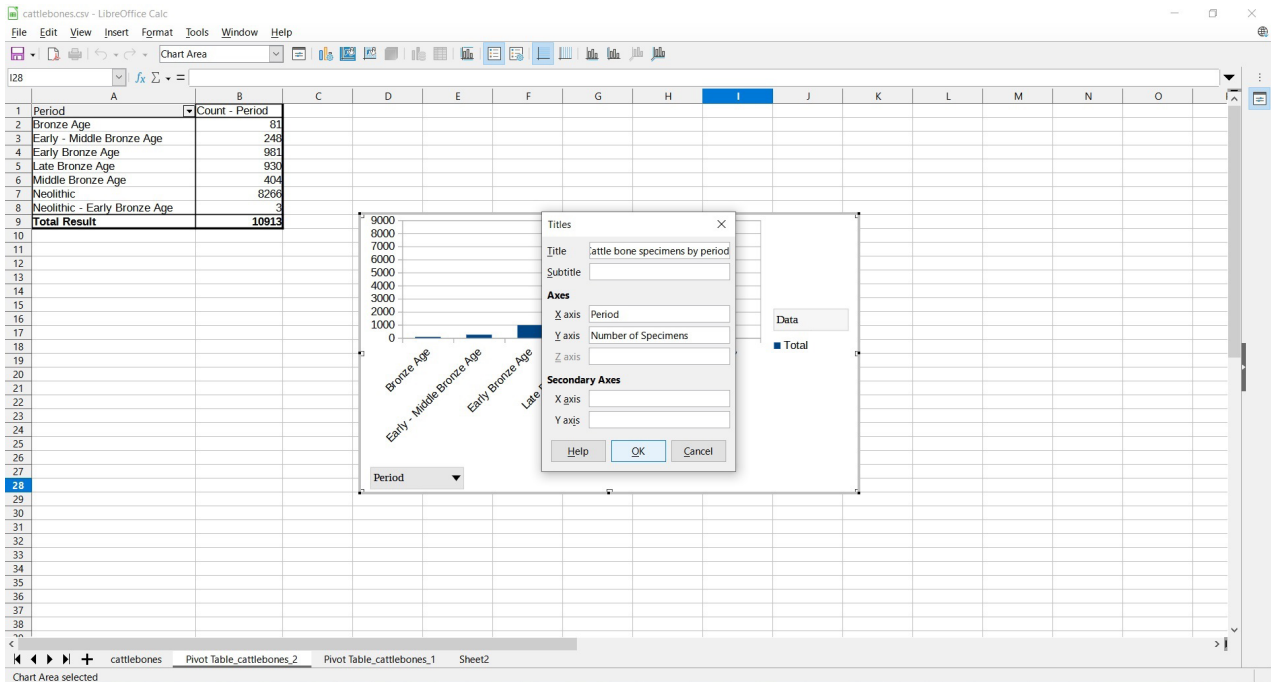
Use our opposite click to open up this window and get the option to edit the titles in the chart.

Specifically, let's tackle that title for the chart first and replace whatever is there with something like "Number of specimens per period". We may also have noticed that the x- and y- axes could be labeled in a clearer manner. If we have labels there already, just click on the y-axis label and replace it with "Number of specimens".

If we don't have labels for either axis we'll need to look for a section under design or a similar concept that allows us to add a chart element. Then we'll find a section for axis titles that will add these to the chart. For the horizontal or x-axis, we'll want to put "Period" and for the vertical or y-axis, we'll want to put "Number of specimens".

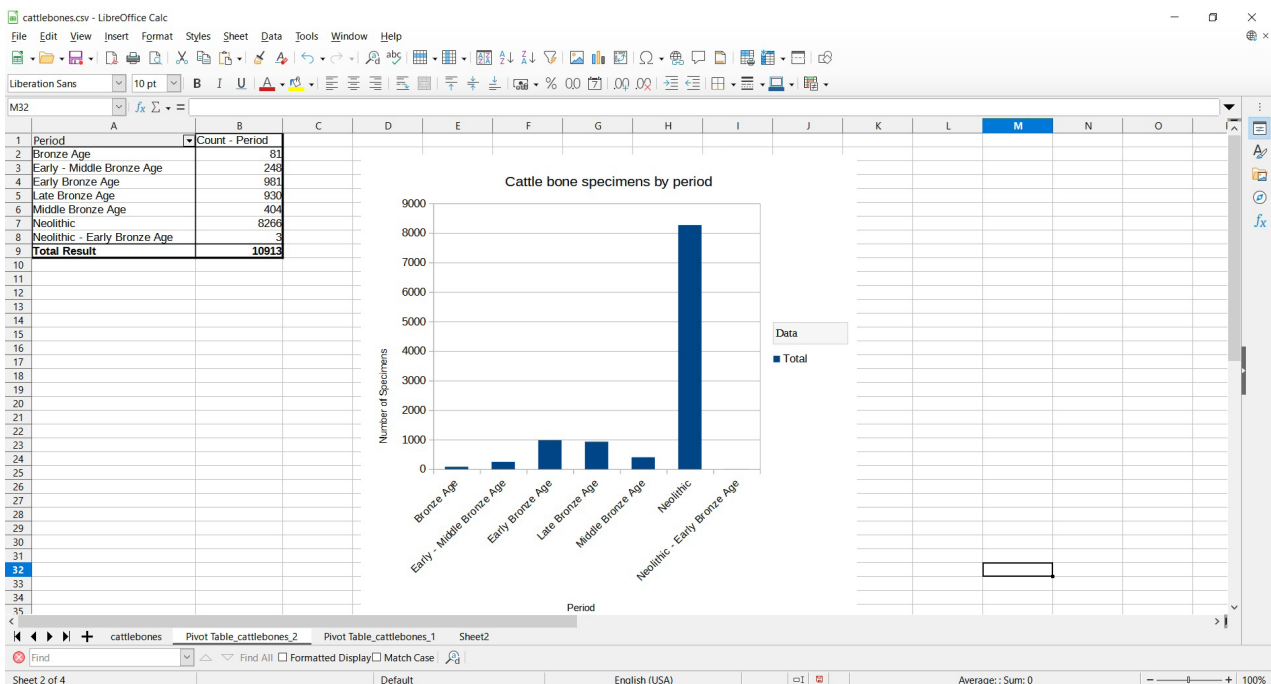


PART ONE: SPREADSHEETS



We can click elsewhere on the chart to add the titles by typing them into a box like this.

We can also play around with these features to change the colors of the bars should we desire. (And let's be honest, playing with colors on a chart is sometimes the most fun part of the process.) Additionally, we can use the graphic user interface (GUI or gooey) to stretch, pull, and rearrange or center elements. This allows us to resize the chart or chart elements to the specifications that we want. Play around with these features until we're happy with the appearance of the plot.



Here's an example of a chart with the correct labels.

PART ONE: SPREADSHEETS

When we're done, hooray cowpoke, we used our spreadsheet program to make a cool looking bar chart from over 10,000 rows of cow bones! Now that it looks nice, based on these charts and the numbers in the tables, what can we say about when cattle bones appear in this data set? What other questions can we ask? What conclusions might we come to?

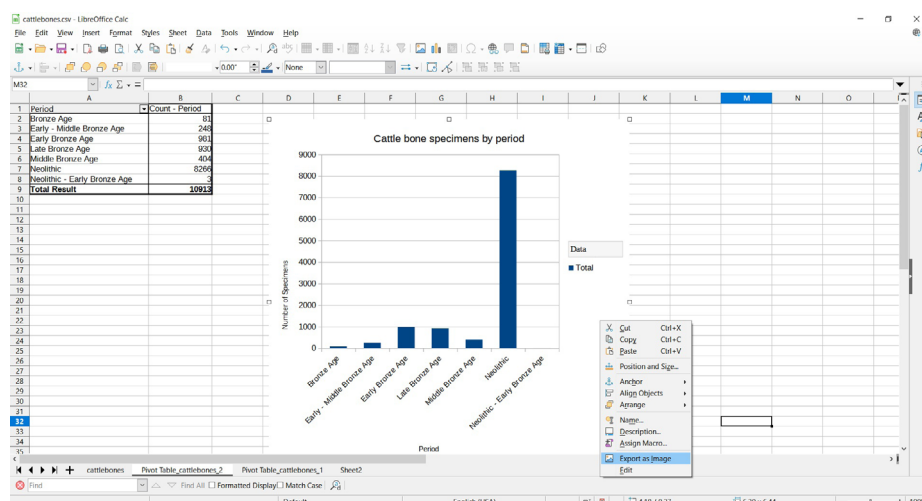
Once we've considered these, consider another column that looks at time, such as "Temporal Coverage.Label." to see if we get similar results. This column is another way that the specimens were assigned time. Do the patterns look the same? Do the names in that column sound familiar?

These should have similar patterns and the names should be familiar because the categories assign the specimens into similar time chunks. However, this column allows for specimens that could not be assigned a specific period between Neolithic and Bronze Age to be assigned generally to the period that includes Neolithic through the Bronze age. These previously would have had "empty" assignments in the period column.

What to do now that we've finished

Congratulations! We've successfully used a spreadsheet program to find a file on our computer, loaded that file into a spreadsheet, opened it as a table, explored what columns were in the data, picked one column to visualize, looked at that table in a couple of formats, and played with creating a bar chart based on that data! We did a lot, y'all.

We may have even tried this tutorial again with a different column heading. Now, we might want to save some of our progress. The charts will of course save when we save our spreadsheet, but if we'd like to save them separately we can either copy and paste them into a new document or utilize options that pop up when we opposite click to export the chart. We may need to click on a set of vertical dots (a vertical version of an ellipsis ...) and have the options pop up there.



We can opposite click on another part of the chart to export or save it as its own image.

Pick whatever format works best for us and make sure to save it with a good name so we remember what this beautiful chart was for. Or, just save our spreadsheet and come back to it later for comparison.

Do it again?

Now that we've gone through this process, once, or possibly twice if we explored a different column, could we easily do these steps again? If the data changed, would it be easy to retrace our steps? Or if we wanted someone else to do it could we guarantee that they'd get the same results?

If we answered no to any of these questions it may be worthwhile to explore these data in a different statistical context. Specifically, utilizing a statistical package or coding language to quickly adapt or edit the process that we took.

If we're not too cow-herdly, we can take the next step with these data and try our R-based tutorial. Or skip ahead to that if we know spreadsheets just aren't for us.

Key concepts used in this tutorial

Import from .csv

Learned how to import a different file type into a spreadsheet program to read the table more easily.

Filter headings

Used the first row/record/entry of information to filter according to other values found in that column/attribute/field.

Pivot tables

Took the information in one column and turned it into a pivot table that summarizes data from a more expansive table.

GUI-based figure design

Created a variety of charts to visualize the data from the pivot table in a different fashion.

Tutorial overview

In this tutorial, we will learn to:

1. Find a file using RStudio on our computer
2. Load data from a file on our computer into R
3. Save those data as a variable
4. Explore columns in those data
5. Select one column to visualize
6. Make different sorts of text based visualization
7. Make a bar chart
8. Save our plots and other outputs

Estimated time to complete this tutorial: 30-60 minutes

Key commands used in this tutorial

```
setwd()
```

```
read.delim()
```

```
colnames()
```

```
levels()
```

```
factor()
```

```
table()
```

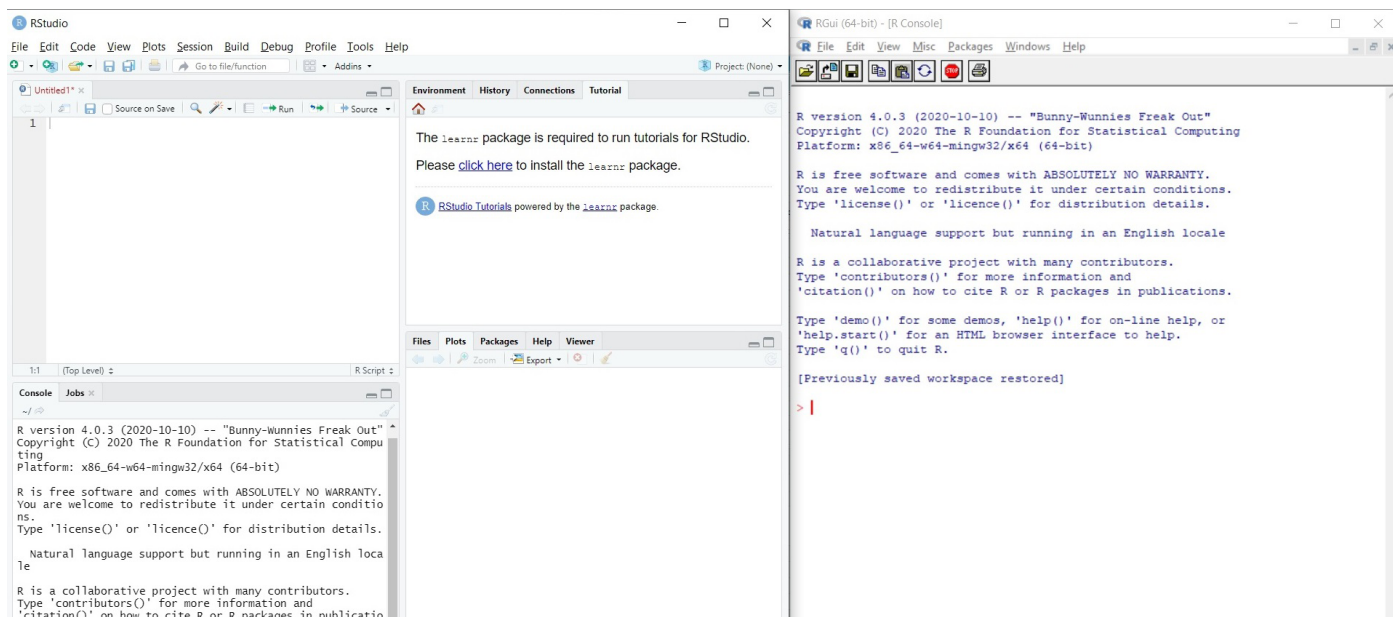
```
as.data.frame()
```

```
barplot()
```



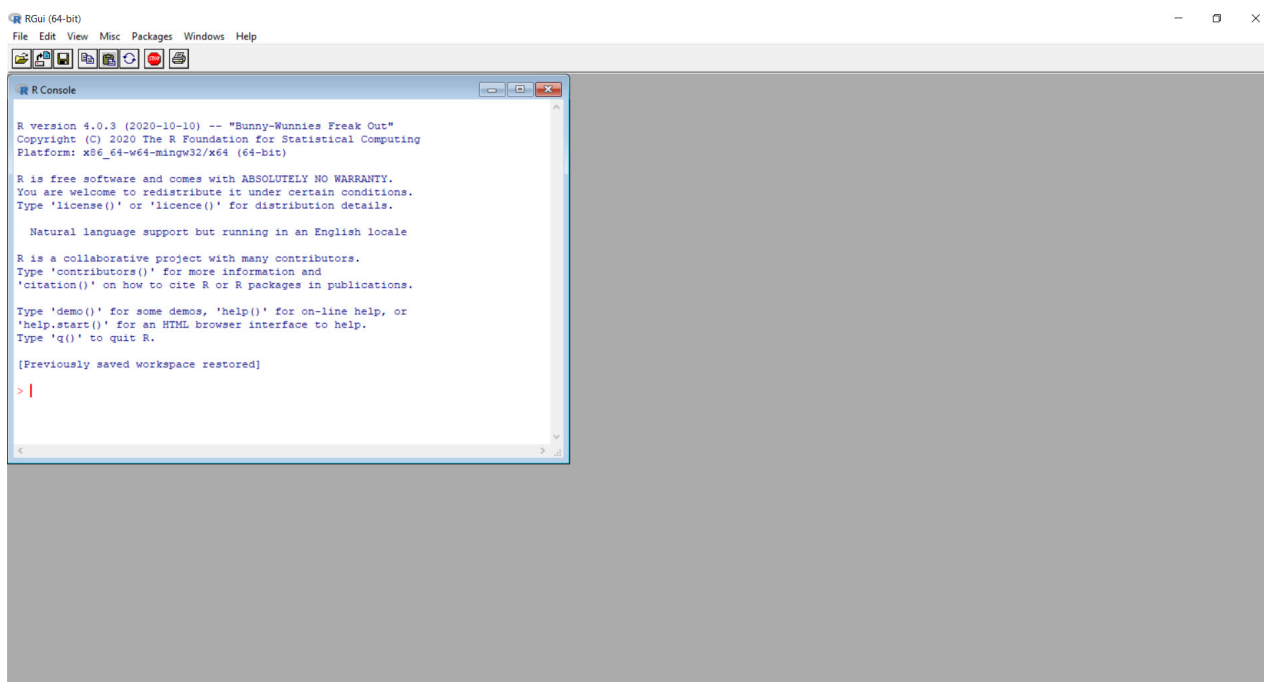
Introduction

Now that we've played around with the cattle bones data from Switzerland in a spreadsheet let us see what we can do in [R](https://www.r-project.org/) (<https://www.r-project.org/>). Using a spreadsheet program is a good way to explore data but isn't always easy to replicate. In contrast, doing the same sorts of processes in a statistical program, like R, can seem intimidating but once we've done it, scripting can be an easy way to retrace our steps.



A side by side comparison of R and RStudio demonstrates the difference between a friendly graphic user interface and using the command line.

R is a programming language and allows for a limited graphic user interface (GUI) to do our coding. While many people use this version of R, using a command line system when we haven't done any coding before can be really intimidating.



This is what the R command line program looks like in a Windows environment. If this is what we opened, close it out and look for RStudio instead.

However, R can be combined with [RStudio](https://www.rstudio.com) (<https://www.rstudio.com>) to create an approachable GUI experience while still utilizing the statistical power of R. This tutorial will run out of RStudio so please make sure to download both R and RStudio before starting this tutorial.

Additionally, RStudio is a great scaffolding tool to get us familiar with command-line style codes that can lead to scripting more powerful tools. It also has a lot of other functions but for now we'll stick with a highly interactive and iterative process for exploring these data. However, before we can do anything with R we'll need to download [R](https://www.r-project.org/) (<https://www.r-project.org/>) and [RStudio](https://www.rstudio.com) (<https://www.rstudio.com>).

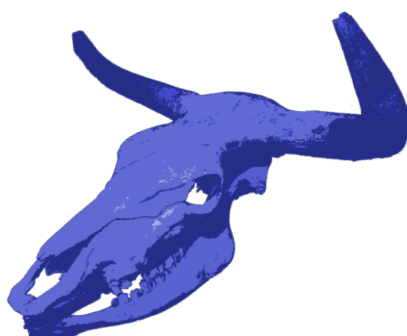
This document, the one we are reading, was generated by R utilizing a package called [R Markdown](https://rmarkdown.rstudio.com/) (<https://rmarkdown.rstudio.com/>). This file contains both text (like what we are reading), lines of code (which we will see below), some inline commands that will appear like `this`, and the products of certain commands. It's a really great way to keep track of what we are doing and run and rerun statistics as we explore them.

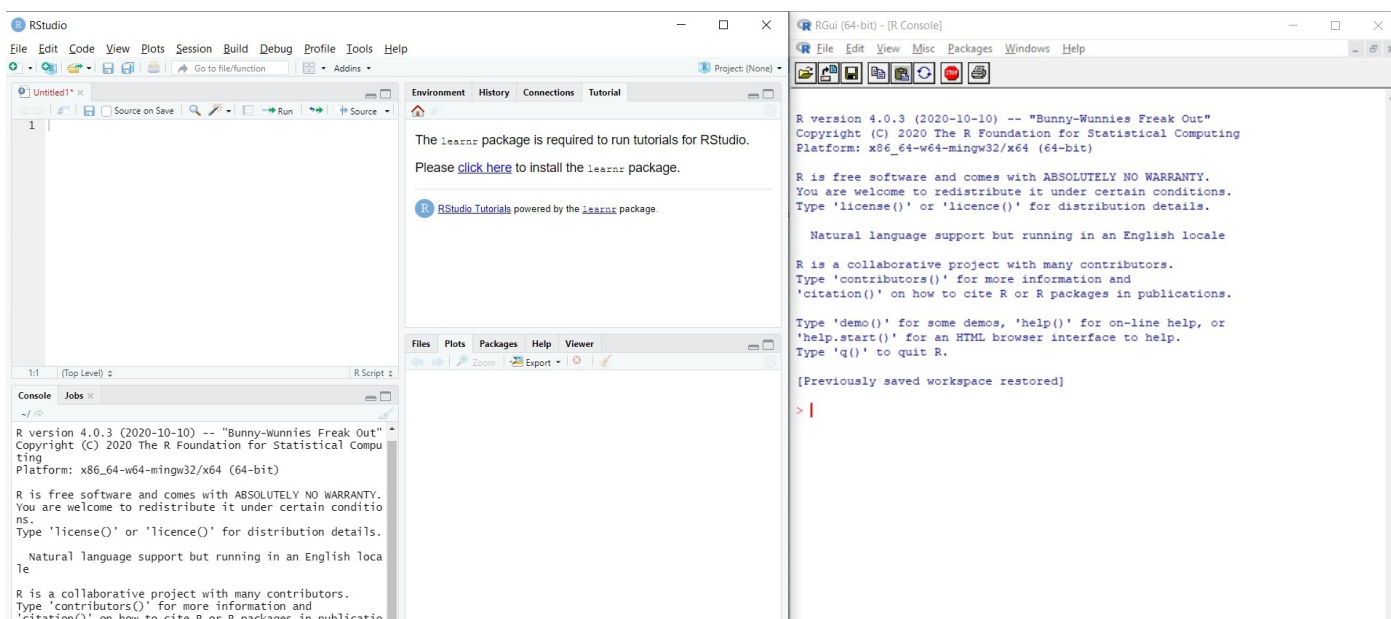
The R Markdown file that generated this text is available at the Open Context codeberg.org repository for this [data story prototype](https://codeberg.org/OpenContext/DataStoryPilot/src/branch/main/cattleBones) (<https://codeberg.org/OpenContext/DataStoryPilot/src/branch/main/cattleBones>). (Currently, we'll get a 404 error if we try to navigate there as the permissions are set to private for now.)

If you didn't try the Spreadsheet tutorial, the data file you'll need is available [here at Open Context](#). You can save the file by clicking the big blue button that says 'Export or Map Records.' From there, click the smaller blue button that says 'Start Export.' It may take a minute for everything to download...

A note on using R

Now that we've downloaded the necessary data and programs take a second open RStudio. Does it look like the image on the left or the image on the right? If it looks like the image to the right, we're working in the wrong space. That's the command-line R environment. Instead, search for RStudio and open that program up.





A side by side comparison of R and RStudio to demonstrate the difference between a friendly graphic user interface and using the command line.

Throughout this tutorial, we will run a variety of commands within the RStudio console environment. Once we run these commands we will get a variety of printouts or other outputs. The exact look of these may look be different than how they appear within this tutorial sheet. Certain commands may printout information below our command that will print horizontally then add new rows wrapping to the width of our specific console.

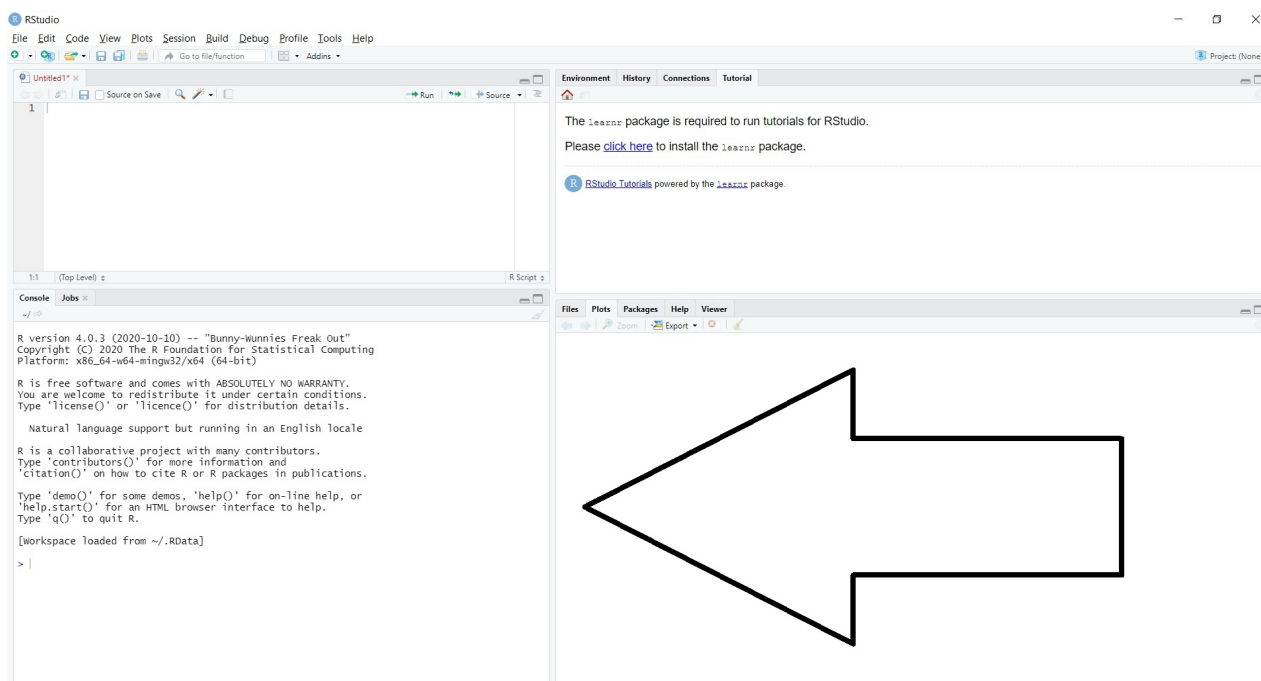
Additionally, as we go through this tutorial we may find ourselves mistyping various commands. If that happens we can use the up arrow to have the console repeat the last command. We can then use the side arrows to enter back into that line and delete or add letters to fix it.

Where is our data located on the computer?

The first thing we'll need to explore the cattle bones data in this environment is to set our working directory. Our working directory is the location on our computer where we will be looking for files and will be where things will save to when we choose to export files or save the history or environment of the session of R.

We can set our working directory in RStudio by going to **Session > Set Working Directory > Choose Directory** and visually navigating to that location. Or we can set a working directory by using this typed command within R or the RStudio Console section:

```
setwd("~/Projects/CattleBonesDataStoryPilot")
```



The arrow points to the console area, where we'll want to type our commands in relation to the RStudio GUI.

Both of these are valid ways to navigate to our data and more than how to do it, remembering to set the working directory is an important habit to build. Each word after a "/" (forward slash) is the name of a folder that encapsulates the thing that follows after it. In our case, the folder the tutorial author set our working directory to, "CattleBonesDataStoryPilot," is within a "Projects" folder.

The above command has two important parts we'd like to explain. The first is the command `setwd()`, which is the name of the thing we are doing. And that thing is setting (set) the working (w) directory (d).

Within the parentheses () for this, and other commands, will be the thing, or things, that we want the command to work on or set. In this case, it's the pathway where we'd like to save files. We'll notice that the pathway is in quotation marks "" indicating that the thing is a string data type, best understood as collection of alphanumeric symbols.

It also starts with a tilde "~." This indicates that there are some parts of the path are omitted. In this case, it's the beginning of the path, which will be specific to each user. For example, if we navigate to that location by going through our files and copying the path from the navigation bar in file explorer with a windows environment we'll notice that there are backslashes "\" in that path. These have a unique function in R, and other programming languages, so they need to be swapped with a forward slash "/" if we want to use it to find and save files.

If we forget that, we'll probably get an error that looks like this:

```
Error: '\U' used without hex digits in character string starting "C:\U".
```

And appear under our console area. This lets us know we need to change something. We may also notice that there is a '>' symbol at the start of each line in the console. This indicates the line where we are typing our commands. If we hit enter after typing a command and a '+' pops up, this is probably because we forgot to close out the command with a parentheses ')'. Once we type that and hit enter, our command should run smoothly.

Our pathway will be unique so don't worry if it looks different than what we've put in this tutorial and congratulations! We've gotten started with R.

Getting the data into R

Once we set our working directory, we won't have to type the beginning of the path should want to call or load files from that particular folder. We still can, in case files are saved in different places, but it's not necessary.

The next thing we'd like to do is to add the data from the table for cattle bones that we've downloaded. R is able to read a variety of file types and there are multiple ways to load data into the statistical environment. If we remember from looking at the spreadsheet, the data on cattle bones was saved as a .csv, which stands for comma separated file. If we haven't heard of these before, go back to the spreadsheets tutorial for a refresher. If we have, the way we're going to add these data into RStudio is to use the command `read.delim()`.

`Read.delim()`, which we can also find more information about by clicking on the help tab in RStudio and searching about, reads the data into the R environment for manipulation. Below we have the line of code we'll want to use to get the data. We'll want to copy everything from `read.delim` through the `)`, pasting this into the console line after the `>`, then hitting enter and having R read the code. Any portion of the code with a `\{ }` or `#` or a `>` at the beginning will not be things we'll need to type or copy.

While it might seem hardly worth noting for some folks, adding in a `>` before our code was one of those things the writer of this tutorial didn't understand when they first started coding!

```
read.delim("cattlebones.csv", header = TRUE, sep = ',')
```

If we haven't renamed our downloaded csv file, the automatic name will be "open-context-csv-export.csv" so we can replace "cattlebones.csv" with this if that's the case. Also if we have any typos, we can use the up arrow to go back to our last command. Then we can use the left arrow to enter that code and change the typos.

We'll notice that `read.delim()` as a command has multiple inputs separated by commas `,`. In this case, the comma without apostrophes is used to separate different inputs into the same function. This function takes a variable number of inputs but the three we added were the file name, in quotations; a section stating that the file had header = TRUE, meaning it had column names that are distinct from the data we want; and indicates that the pieces of the table were separated by commas using `sep = ','`.

What's cool about `read.delim` is that we could have entered a different data type, for example a text file `".txt"` and indicated a different kind of separator for our data, such as a space, period, tab, or any other character; and it would still be able to read it within RStudio.

Once we type that in and hit enter a whole herd of text should appear underneath. SURPRISE!

In this tutorial sheet, we just printed the code and had the code run in the background so this worksheet didn't get filled with thousands of cattle bone entries! It would have been a stampede! At the end we may see something like:

```
[ reached 'max' / getOption("max.print") -- omitted 11161 rows ]
```

It may have slightly different numbers in the last section but it's just letting us know that there are data that are part of what was read that were not printed. We might get fewer rows because of display settings or maybe we added an extra column when we were downloading our data that added some more rows to the original data set.

Regardless, getting different numbers at any point during these tutorials is ok. It's not right or wrong as long as we're working through the problem of learning to play with this data. Additionally, we can take a break to search for more information or just get our eyes away from the screen because sometimes we just need some space and time to process what we just learned. And that's definitely worth doing!

Anyways, these thousands of entries appear because `read.delim()` just reads through all the data and prints it. For a data set with fewer entries this may be a good way to look at the data. However, with a data set of this size it is not helpful. Also we did a lot of that exploration using our spreadsheet earlier. There we could scroll through or over the records or rows and columns, also known as attributes or fields, to take a look at the observations.

Instead, it will be better for us to have these data saved so we can use other commands to call parts of the table to explore particular things. To do this we'll want to save this table as a variable.

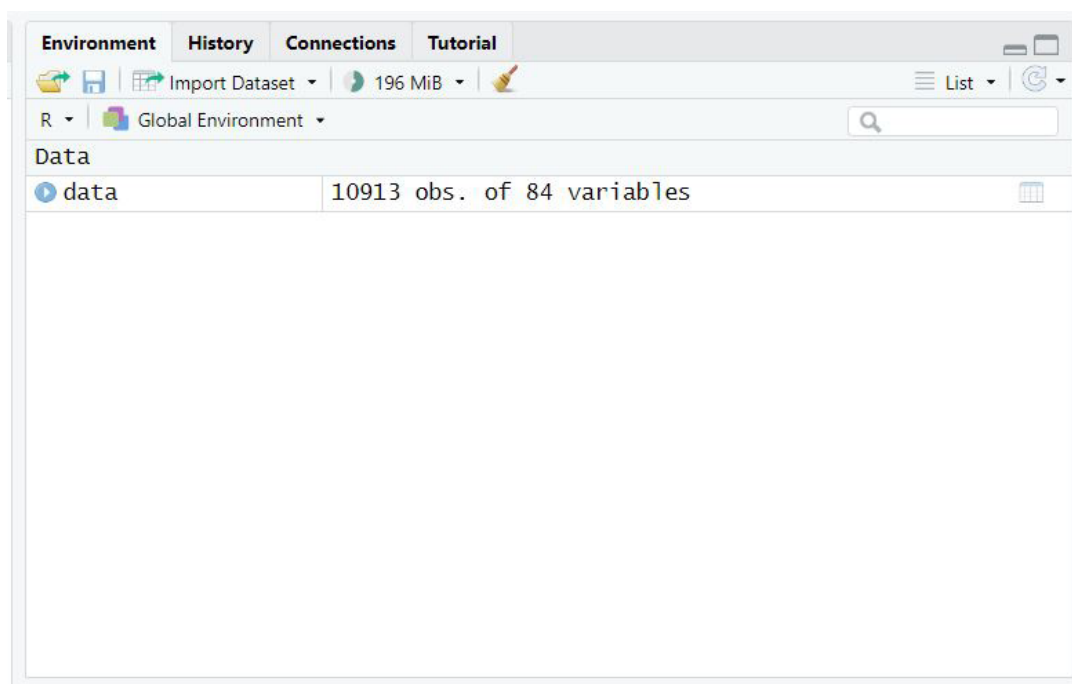
A variable is a nickname for a particular set of data that we can use instead of typing out the whole name for that particular set of data. Essentially, it's like in an algebra class where a letter could stand in for a number. But in this case, that letter can stand for another letter, a word, a number, a set of numbers, or a variety of other items as well.

We'll do this by giving the data set a name that we can reference. This can be any name we'd like to use and can be done by setting the name equal to "=" the data table or by pointing the data table at a name "<=". Both of these commands are valid and R programmers tend to use "<=".

Although this is the standard practice for R, it is not a standard for many other programming languages. For example, if we were to venture into python or java assigning variables, another way to say we "named" a data set or number, is only done using "=". There are a few other idiosyncracies like this in R but just know that both ways are valid.

```
data = read.delim("cattlebones.csv", header = TRUE, sep = ',')  
data <- read.delim("cattlebones.csv", header = TRUE, sep = ',')
```

When we copy and paste either of the above commands into the console and hit enter, we may notice that nothing printed out...



Typically in the upper right corner we have the Environments tab. Here is where we can check if our data saved as that variable.

Which is exactly what we wanted. Rather printing all the parts of that table, we instead get that table saved as something we can reference in the future. If we're nervous about whether we did the right thing though, we can check the environments area of RStudio located, usually, in the upper right hand section of the screen. There under Data should be a new variable called data that has appeared.

What kinds of questions can we ask?

Moooooovelous, we've successfully created our own variable and now won't have thousands of data entries pop up unexpectedly. With these set, we can explore the attributes available for each row in that table. However, a lot of those questions are dependent on what we know about the data. Specifically, many of the measurements are specific to the kind of bone and might require additional background knowledge to utilize effectively and ask the right questions.

A question about this data set that does NOT require so much other information though is, "What periods are these bones from?". While an exploration of the Open Context website gives us [a basic visualization of that](https://opencontext.org/subjects-search/?proj=166-neolithic-and-bronze-age-cattle-data-from-switzerland&prop=oc-gen-cat-animal-bone#8/47.040/8.438/11/any/Google-Satellite) (https://opencontext.org/subjects-search/?proj=166-neolithic-and-bronze-age-cattle-data-from-switzerland&prop=oc-gen-cat-animal-bone#8/47.040/8.438/11/any/Google-Satellite) under the image labeled "Chronological Distribution (Years BCE/CE)," we can look at this questions in other ways. We may also already know the answer if we did the spreadsheets tutorial.

When we originally read the table, not all the rows were visible and the formatting made it difficult to see the names of the columns. If we just want to look at the columns we can use a different command to just see those.

```
colnames(data)
```

```
## [1] "URI" "Citation.URI"
## [3] "Item.Label" "Project.Label"
## [5] "Project.URI" "Context..1."
## [7] "Context..2." "Context..3."
## [9] "Context..4." "Context.URI"
## [11] "Latitude..WGS.84." "Longitude..WGS.84."
## [13] "Early.BCE.CE" "Late.BCE.CE"
## [15] "Item.Category" "Published.Date"
## [17] "Updated.Date" "Temporal.Coverage"
## [19] "Has.taxonomic.identifer" "Has.anatomical.identification"
## [21] "Anatomical.measurement" "Period"
## [23] "Element" "Taxon"
## [25] "Layer" "Site.Name"
## [27] "Database.ID"
```


This prints a list of the column names in the order that they appear in the table. The numbers on the side identify the place in the list for the first column in that row. Can we identify which of these might help us explore when these cattle bone specimens come from?

Looking at that it looks like "Temporal.Coverage..URI," "Period," "Early.Date..BCE.CE.", and "Late.Date..BCE.CE." are some that have something to do with time. To look at all the entries from that column, we'll use a command that looks like `data$COLUMNNAME`.

The first part is the variable we're calling from and the `$` allows us to specify the column name. So we need to put a valid column name from that list after it. When using such a command, we do not need to include the quotation marks `"` around the column name. We may also notice that column names that had included spaces or other special characters like `"(`, like "Early BCE/CE", look slightly different (aka "Early.Date..BCE.CE."). This is because R doesn't like special characters in column names, so it replaced all of those with periods. So they are the same names, just with some characters replaced.

```
data$Period
```

For cheese of use, we picked the same column to explore in this tutorial and it once again generates a whole herd of data as that command calls all the cows, we mean entries, in the column. SURPRISE! And once again, while in a data set with fewer entries this might be useful, looking at the list might not really help us get anything meaningful out of it because it's just too many entries.

How do we visualize our question as a table?

Rather than be trampled by a stampede of over 10,000 entries, we can use R to summarize that data in a variety of ways. We explored this by hand using our spreadsheet and picking different filter options, so we'll do a similar thing here by looking at how many periods there are. To get the diversity of what entries can be called we're going to combine two commands together.

```
levels(factor(data$Period))
```

```
## [1] ""                "Bronze Age"
## [3] "Early - Middle Bronze Age"  "Early Bronze Age"
## [5] "Late Bronze Age"           "Middle Bronze Age"
## [7] "Neolithic"                "Neolithic - Early Bronze Age"
```

We'll explain what this code does by reading the commands from the inside out in the above code. The innermost piece is `factor(data$Period)`. If we had just typed that in we would have generated another long list, as it factors every entry in the original `data$Period` call. Factoring will go through the data and identify each observation as within the same possible category, identifying similar values and associating them in a set of discreet data. This will give each entry

an integer value.

The next level is the `levels()` command that goes around the `factor()` command. By putting ``levels()`` around `factor()`, we tell R that we don't need to see that factor list and instead just want the list of the unique names within the data. Those are the exact observations we want to look at. Seeing that there are only seven (7) levels for period means that this might be a good variable to use to explore time in this data set.

However, like with the spreadsheet, we may want to know how many of each kind there are. One way we can visualize that is by taking that column and turning it into its own table.

What this does is allow R to summarize the information within that column and turn it into a frequency table. What this means is that it will turn those levels into rows and the only column will be how many rows from data fall into/utilize that particular description. The basic way we can do this is by calling the `table` function

```
table(data$Period)
```

```
##
##                                     Bronze Age
##                                     81          84
##      Early - Middle Bronze Age      Early Bronze Age
##                                     249          1004
##               Late Bronze Age      Middle Bronze Age
##                                     946          410
##               Neolithic Neolithic - Early Bronze Age
##                                     8419          5
```

This tabulates how frequently each name is used, like counting how many cows are named Daisy, but the formatting is a little odd. For a more visually appealing version of this, we can turn the table into a data frame, which is a different way of storing the data from the table, similar to what we did when we imported the whole herd of data.

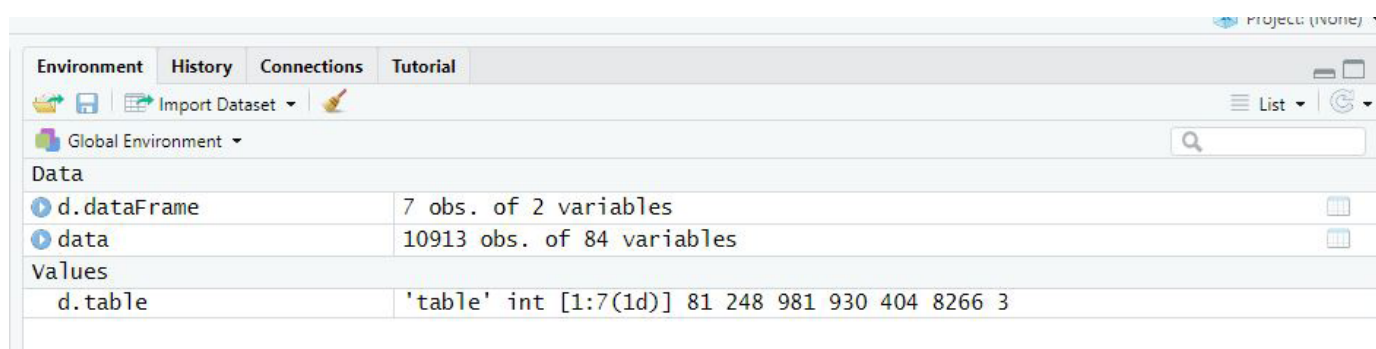
```
as.data.frame(table(data$Period))
```

```
##               Var1 Freq
## 1                                     81
## 2      Bronze Age    84
## 3      Early - Middle Bronze Age 249
## 4      Early Bronze Age 1004
## 5      Late Bronze Age  946
## 6      Middle Bronze Age 410
## 7      Neolithic    8419
## 8 Neolithic - Early Bronze Age    5
```

This is much easier to read. And from this we can see that there are a few periods that aren't very common, some that are sort of common, and then one period that steers the data these cattle bones come from.

Because we'll want to do more with this in the future, we'll save both the table and the data frame as variables so that we don't have to do more typing in the future. Tables and data frames display similarly for us in R but are different ways of saving and accessing the data. This means that different commands work on one and not the other. There is more to it than that but for now just remember that they are different.

```
d.table = table(data$Period)
d.dataFrame = as.data.frame(table(data$Period))
```



This is what the environment pane will look like once we've saved our variables there.

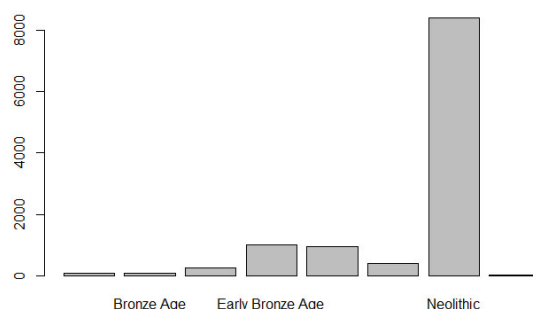
Make sure when we save data as variables that if we are going to use a lot of them we utilize a standardized naming process for ourselves. Try to keep variables short so that we don't have to type as much but also so that we don't confuse ourselves with what we've saved so far. If we are moving through this tutorial within RStudio the Environment tab will have a list of the variables that are in play during the current session.

Cow tips aside, the difference between these two formats for the data aren't huge. So, another way we can visualize these differences in time is by turning that table into a bar chart. A more moooving way to visualized these differences.

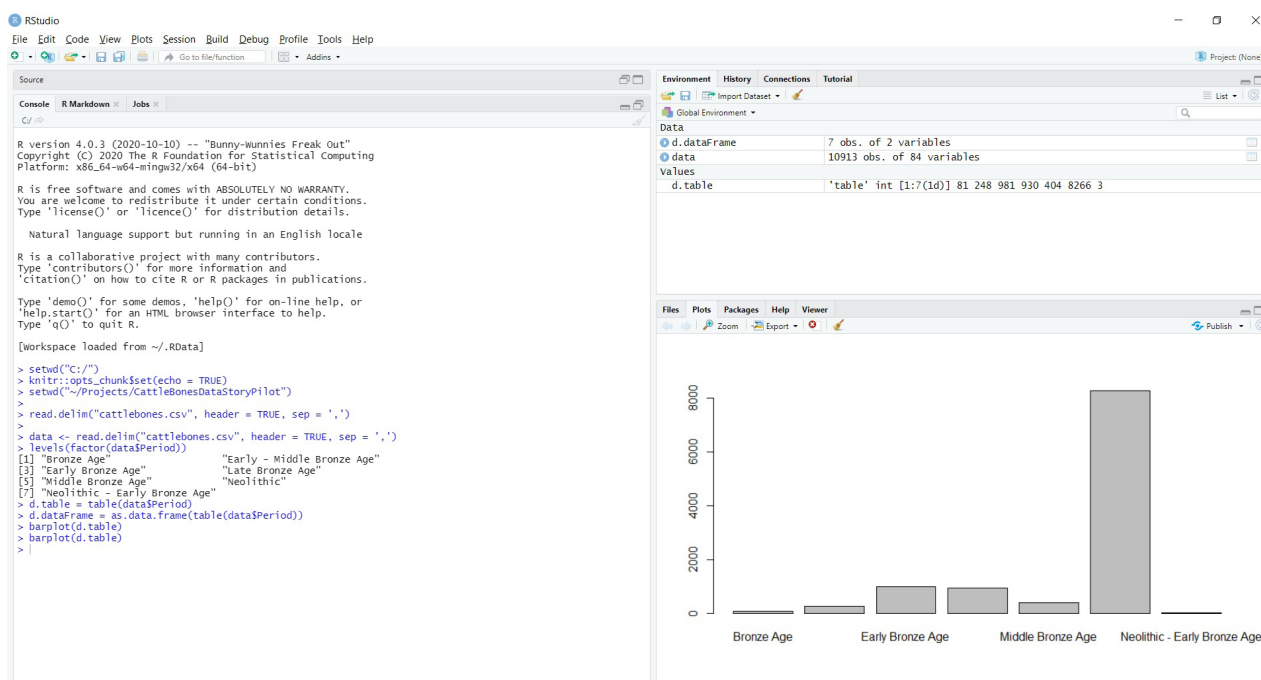
How do we visualize our question as a chart?

The best way to do this is to use a bar plot. Bar plots use the frequency of a particular value to basically stack up and compare these values visually rather than numerically. To do this we'll utilize that table version of the periods to create a bar plot, utilizing the `barplot` function.

`barplot(d.table)`



Our images might look different in this tutorial than what pops up in RStudio and that is ok! We can click zoom on the plot or resize the pane to see more or less of that particular plot. Regardless, the plot really accents how one period has more representation than the others in this particular data set.



Within the tutorial and in the GUI the plot looks different, depending on the width of the panes we have.

If we typed the `barplot` function into our own RStudio environment, getting more of the labels to show is a matter of making the plots pane area larger by hovering over the gutter between the console and the plots windows until it has the arrow cursor and dragging it over until it is larger.

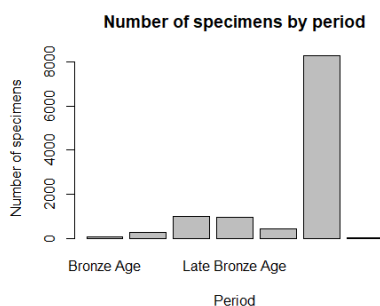
How do we make our visualization a little nicer to look at?

Have patience though as we may notice that there are a few things missing from this plot. For example, the X and Y axes are not labeled and we don't have a title. Also, some of the bottom column names are missing. In this section we'll play around with these features, rustling up some better visualizations.

For most plot based functions in R, adding labels to the x and y axes involve using the `xlab` or `ylab` function. These will pass a data type called a string, which are essentially alphanumeric symbols that do not have an inherent value. For example, the data type ``integer`` is a symbol with a specific value so that we're able to do something like $2-1 = 1$. However the string values 2 and 1, typically depicted as "2" and "1", cannot be manipulated in the same way. It is similar to trying to subtract the letter a from b.

For `xlab` and `ylab`, we'll need to make sure we put all our values in with quotation marks (") around them so that the function understands that it wants to print those values. We'll also add a title, referred to typically as `main`.

```
barplot(d.table, main = "Number of specimens by period", xlab = "Period", ylab = "Number of specimens")
```

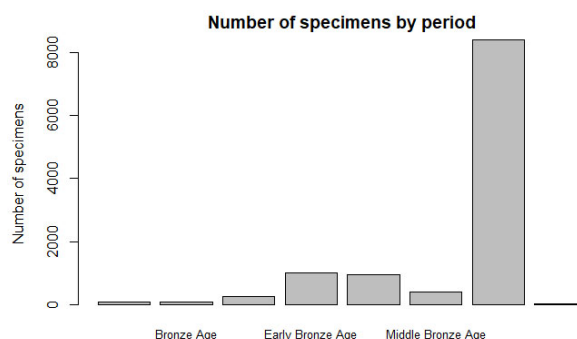


This function added the labels to the x and y axes and a title. However, it looks like some of our labels are still being missing, probably because of their size. We can look up all the details for what can be edited in the command for the function by using `?barplot`. This will search for `barplot` and then RStudio will automatically open the help tab with the entry for that particular function.

Specifically, we'll be playing with the values for the `cex.names` portion to get all of the labels to show up correctly. First we'll try setting `cex.names` to `.75`. The number, `.75` in this case, is a proportion of magnification for the particular label so we could also make them larger or smaller proportionally by changing this value.

PART TWO: R

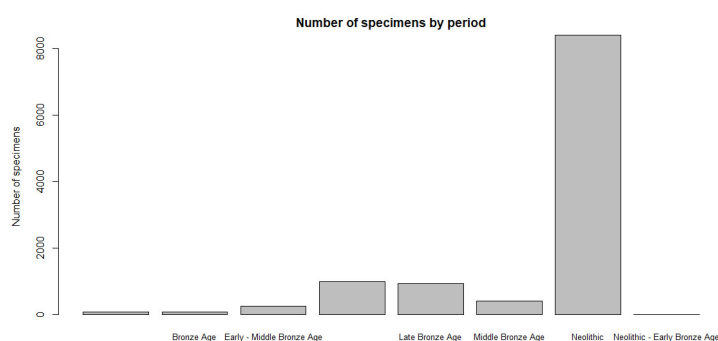
```
barplot(d.table, main = "Number of specimens by period", xlab = "Period", ylab =
"Number of specimens", cex.names = .8)
```



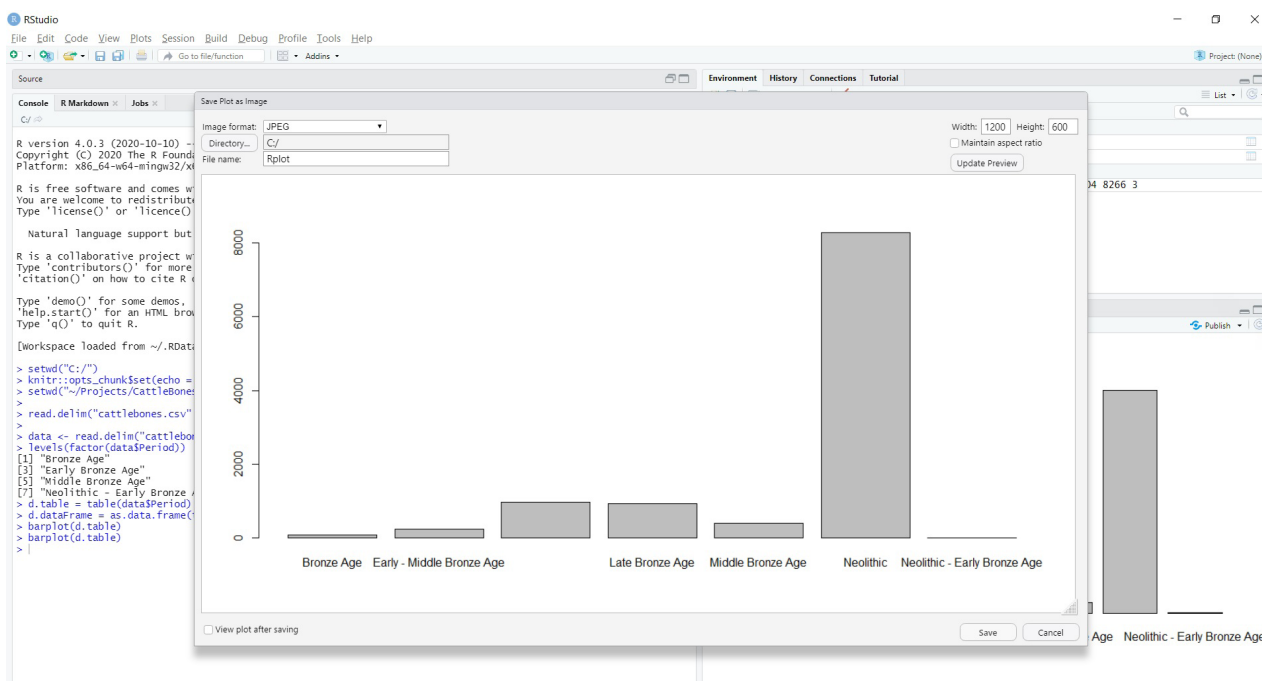
This has made the labels a little smaller and more of them show up in the plot but it looks like we could also play with the size of the plot to allow for more space for the labels to show up.

Within this tutorial sheet, we did this by changing the figure width and height within the program that connected the code and outputs together. So before our code we have a section that looks like this `{r barplot4, fig.width=12, fig.height=6}`. That is why although the code below is the same, the image looks different than the one above.

```
barplot(d.table, main = "Number of specimens by period", xlab = "Period", ylab =
"Number of specimens", cex.names = .8)
```



That's a little more behind the scenes information than we probably need right now. But as we improve our skills with R knowing these differences exist will help us make the best plots we can for how we're exporting our visualizations.



We can hit the export button and have this window come up. Then we'll alter the dimensions by pixel and update the preview to see if it's at the proportion we want.

However, as someone coding through this within R or an RStudio environment, these proportions can be changed by altering the width and height of our GUI plots area. Or when we export the file changing the dimensions in the pop up window. Otherwise, we need to reset the default plotting sizes on our computer environment but that can be a little tricky so we'll graze over that for now.

Based on these charts and the numbers in the tables, what can we say about when cattle bones appear in this data set? What other questions can we ask? What conclusions might we come to?

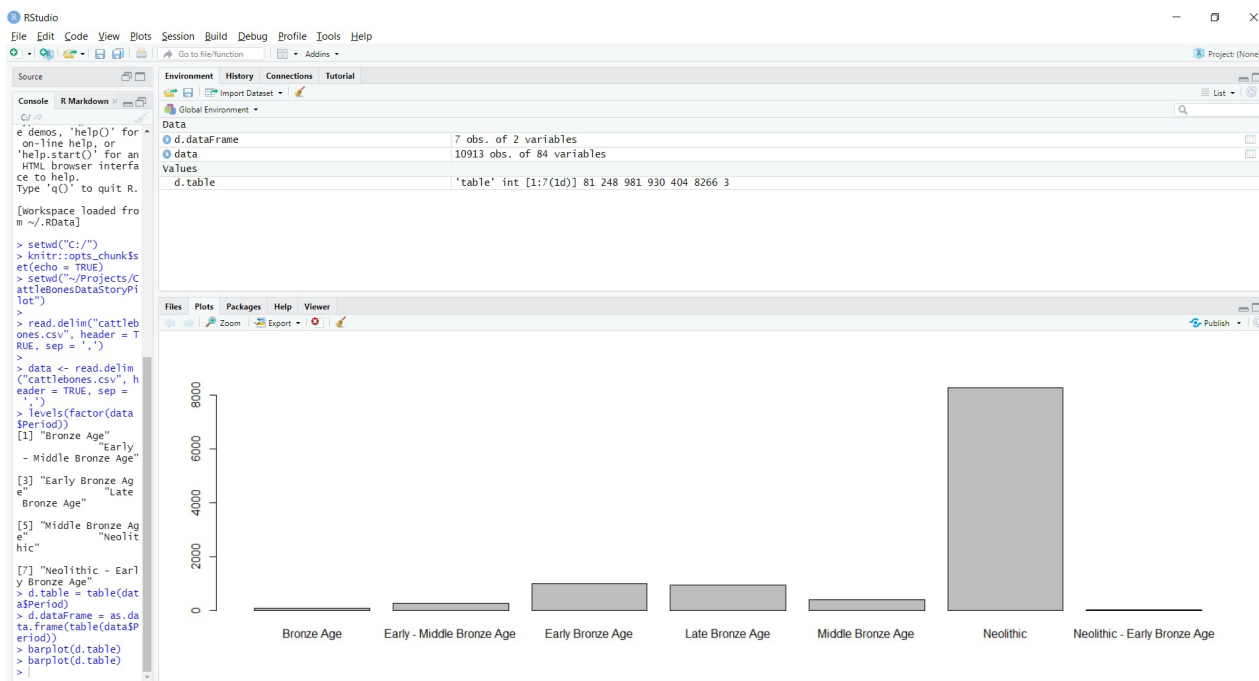
Once we've considered these and that we've been introduced to these visualizations in R, consider another column that look at time, such as "Temporal.Coverage..Label." to see if we get similar results. Do the patterns look the same? Are things named the same thing? Did R count them correctly? Hoof we made a mistake?

What to do now that you've finished?

Congratulations! We've successfully used R to find a file on our computer, loaded that data into R, explored what columns were in those data, picked one column to visualize, looked at that table in a couple of formats, and played with creating a bar chart based on that data! Hoof, hoof, mooray! We may have even tried this tutorial again with a different column heading. Now we might want to save some of our progress.

PART TWO: R

If we'd like to save the plots we generated we can do this in the plots tab of RStudio and hit the export button. Save to image will give us a variety of file types that we can choose from. Here is where we can manipulate the aspect ratio to make sure all of our labels show up correctly. This can be done by adding in different values in the width and height boxes and updating the preview.



We can drag the sides of each pane left and right and up and down before we save our plot.

If we did this manually by dragging the borders of the plot screen to our desired size we can just opposite click with our mouse and copy the image to paste it into a different location. That takes care of saving our finished plots, now what about our coding and the data we loaded?

If we exited out of R or RStudio at any point, in frustration, accidentally, or for some closure; we may have been asked if we wanted to save our workspace or something along those lines. This allows us to save all the data we've loaded, variables we've created, and a history of all the commands we called within this particular session of R. This can be really handy if we're poking around on a single data set over multiple sessions. However, it can also be annoying if we're switching between data sets and using up all the good variable names!

Instead, we may want to save our environments and history uniquely for the project we are working on and load them when we start a new session of RStudio. To save our environment, we'll click the Environment tab and hit the save button. This should open up a new window within our working directory where we can save the file. Pick a good name for it, maybe not Daisy, and then save. Next we'll want to click on the history tab and do the same thing.

Once these are saved we can call the cows home in future sessions by clicking the folder with the arrow pointing to the right. This means that we won't need to reload the data or variables from this session. This can be very helpful in case we forgot to save some figures, so we can find the exact commands we used to create it, or if we want to rerun our analyses with different data.

Congratulations folks! We've wrangled R for our cattle bones questions. Now though, what do they mean? To find out check out the narrative tutorial written to complement these numbers.

Review of commands used in this tutorial

`setwd()` This sets the working directory where your files will be saved to.

`read.delim()` This reads data to put it into the R environment.

`colnames()` This creates a list of the column names within a table.

`levels()` This gets the levels of a factored piece of data by only picking up the unique expressions within attribute variability.

`factor()` This factors the values within a column of data.

`table()` This creates a table that counts the frequency of a particular.

`as.data.frame()` This turns whatever is p.

`barplot()` This creates a barplot of information.

Howdy and welcome back!

In this third part of the Digital Data Story, we're going to change focus a bit and look at what we do with data after we've completed an analysis. Together, we've used a [spreadsheet](#), we've used [R](#), and now, we're going to use the data to make a compelling case for the results of our research. This is called creating a 'data narrative'.

Learning to create a 'data narrative' is important because at some point we'll want to share our results in a way that the public can understand. Sometimes the public might be a supervisor, other times it might be a funding agency. Often the public is a publication, or a presentation to stakeholders. No matter who the specific public is, it's crucial that we provide our results in a way that is clear, appropriate to the audience, and engaging.

But hold up there, cowpoke. How is a data narrative different from data, and how is it different from a regular narrative? To answer the first question, a data narrative differs from data because it puts some of the data together to tell a particular story that answers a particular question. And the second question? A data narrative is different from a narrative because where a narrative is the storified version of events, a data narrative is the storified version of data.

It's important that at this stage, we think critically about the data we've analyzed, and about our analysis. We know that we did our spreadsheet and R exercises 'right', or we skipped here directly for a good tail, but it's still possible to end up with an inaccurate conclusion. An inaccurate conclusion will end up being a 'data narrative' that shares those inaccuracies, and the public won't be served. As researchers and data analysts, our work should always in some way benefit the public and giving them inaccurate information isn't helping them! So let's talk about the things that can go 'wrong' when we've done everything 'right'.

Interpreting data considerations (bias)

First up, we need to think about bias. Often, the word bias has a pretty negative connotation. In our case, when we're talking about bias, we're not necessarily talking about something bad, like racism, or sexism. For our purposes, bias is anything in our personal experience of the world that influences our results in a way that changes those results without the data to back them up.

So let's look at the potential kinds of bias we might encounter with our cattle bones data. We're going to talk briefly about sampling and selection bias, confirmation bias, and outliers.



Sampling and selection bias

Sampling and selection bias has to do with the kind of data that are, and are not, included in the data set we're working with. These kinds of biases can be due to several different things; the kinds of sources that were selected may cause bias, the screen size in the field that specimens were sifted through may cause bias, how we filter our data in a table or spreadsheet may cause bias. Even the questions that drove the data collection team to do the project in the first place may cause bias.

Often, these are the kinds of decisions that happen even before we open our data set. They're events out of our control. However, because this is so out of our control, it can be hard to sort out what all of the potential bias factors are, especially if we haven't heard of these sorts of osteological (bone) remains before.

A great way to try to approach the issue of selection or sampling bias is by looking at the data... about the data set. This 'data about data' is sometimes called metadata. To do this we need to go back to the original source where we got the cattle bones data. Looking there, we'll find information about the creation of the data set that can help us understand what choices were made when it first was created. So go ahead and open a browser window [here](#) to the data set on Open Context.

The starting clue to the sort of limitations that might be in this data set comes from its name. While we've been just calling this data set 'cattle bones,' the actual project was called '[Neolithic and Bronze Age Cattle Data from Switzerland](#).' This tells us a couple of important things.

First, all the bones are from present-day Switzerland, so it means that we're not dealing with cattle data from all over the world, or even all over Europe.

Second, it calls this 'cattle data,' not cattle bones. This hints that there might be some other kinds of data within the data set.

Third, it describes the data as 'Neolithic and Bronze Age'. Now, those terms came up when we were making graph labels related to time, but what do they actually mean? We're trying to work with data here, but in a pinch we can do a search for it online. (If we're interested in using a more data-centric source, we can mosey on over to a site like [PeriodO](https://periodo.do/en/) (<https://periodo.do/en/>) for a quick detour and we'll wait until the cows come home to continue.)

Because the terms 'Neolithic' and 'Bronze Age' refer to time, this means that we only have cattle data from Switzerland that come from those two periods of time.

Specifically, if we look at the subheading of the project it has particular dates: c4500 - 800 cal BC. The dates refer to a particular method for dating the periods in question, and we'll explore that in a future tutorial. For now, we just need a general idea of when those periods were.

PART THREE: NARRATIVES

The project description also specifies that these are cattle 'biometrical data'. A quick online search for '[biometrical](https://www.thefreedictionary.com/biometrical)' (<https://www.thefreedictionary.com/biometrical>) should ensure that we understand what sort of data were collected. The project description also mentions something called [NISP](https://en.wikipedia.org/wiki/Number_of_Identified_Specimens) (https://en.wikipedia.org/wiki/Number_of_Identified_Specimens), though? That's something we can set aside for now...but it may be important in the future.

Another part of sampling and selection bias can come from the kinds of data we choose to use to answer our particular questions from a particular data set. For example, if we purposefully, or accidentally, only filtered for a particular kind of bone or for only one period, and then tried to answer the question, 'What period are these bones from' we would only be able to answer that in relation to the sample of the data set we filtered for. We wouldn't be able to answer the question for the whole herd of data.

So, how can we deal with this kind of bias?

The easiest thing to do is to make our question smaller by adding specific information about the data set we're looking at. This allows us to be explicit about how applicable our conclusions are. It also allows us to be explicit about what populations our narrative and answer applies to. We don't necessarily want to create long questions that could confuse our reader, so we can separate out these biases in the narrative and state them explicitly.

Confirmation bias

The next kind of bias we need to consider is called 'confirmation bias'. This is the tendency to give more weight to evidence that aligns with our previously existing ideas and thoughts on a subject. This can result in listening to evidence that supports our hypothesis and ignoring evidence that refutes our hypothesis.

As an example, if we have a hypothesis that Neolithic people consumed more protein derived from cattle sources than modern day people, on average, and we think that Neolithic people didn't live in big groups, we might choose to compare the potential amount of beef consumed by Neolithic people against the amount of beef consumed by modern people in rural areas, where the population density is low. But that would ignore the infrastructure required for the amount of cattle found in our data set, and infrastructure indicates a potentially high population density.

So, how can we deal with this kind of bias?

Confirmation bias is tricky, and we have to think about where our personal beliefs may be influencing how we analyze our data. There are ways to avoid falling into confirmation bias though. We can use multiple methods and forms of analysis to test our hypotheses. We can make our research reproducible, so that someone else can run the same analysis we did to see if they come to the same conclusions. (Remember, working in R is good for this!) We can avoid ambiguity in our research by making our terms and choices clear. Finally, we can be okay with our results not being what we wanted!

Outliers

The final kind of bias we're going to discuss that can cause our narratives to move away from accuracy are what are referred to as outliers. These have a particular definition in statistics. For now we'll operate under the definition that outliers are a kind of 'extra special' data.

It's important to highlight that 'extra special' does not necessarily mean 'more important'. In fact, the exact opposite is more likely! Sometimes, a particular specimen is so one-of-a-kind, so individual, so unique that it disrupts the story being told by the other specimens within a particular data set. An outlier may be the single human tooth in a data set full of bear femurs. It may be the one artifact that dates to the medieval in a data set of artifacts that all date to the Neolithic.

So, how can we deal with this kind of bias?

While we don't want to ignore the existence of outliers, it is important to understand that they don't describe or accurately portray the majority of cases. Which, in archaeology, is typically what we're going for. So resist the urge to throw out the discussion of all of the data that makes sense together in favor of focusing on the one piece that doesn't. Exploring the outliers can be a future task, and a chance to ask different questions once our main project is complete!

Interpreting data considerations (creating a narrative)

Whew. There's clearly a lot to consider after we've done our analysis before we're ready to share our work with the public. But having considered all of that...how DO we share our work with the public? Once we're at this point, it's time to start creating our data narrative.

Remember our driving question for this work was 'What periods are these bones from?'. So we want to create a narrative based on our data that answers that particular question. Some of the ways we could display that narrative include:

- a list
- a table
- a chart
- a picture
- a presentation
- a podcast
- a twitter thread
- a cow-drama
- an interpretive dance
- a table-top role-playing game

PART THREE: NARRATIVES

While some of our data narratives are ‘traditionally’ academic, such as visualizations like lists, tables, and charts, those traditional methods of sharing results aren’t the only ways to display a data narrative. One big factor that would dictate choosing to make say, an interpretive dance over a podcast, or over a scatter plot, would be the audience. Who are we trying to reach? Why are we trying to reach them? What would be the method of sharing data that would answer both ‘who’ and ‘why’?

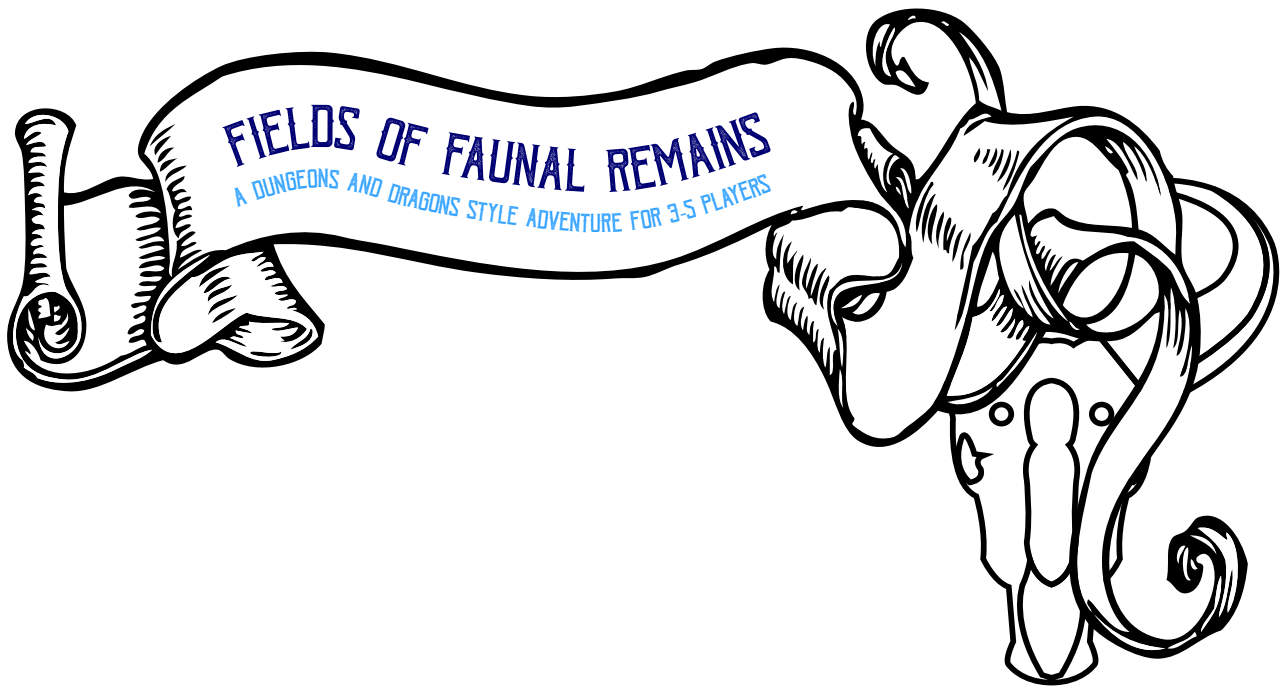
After the ‘who’ and the ‘why’, it’s time to consider the tone we set for our narrative. While we are doing data-driven science, that doesn’t mean that our narrative should be unapproachable. Our intention is to communicate our results and based on our list, there’s lots of ways to do that.

While we tend to think of scientific writing as objective, if we recall the sections on bias we just considered...that’s not quite right. We can try to be objective, and we can take steps to minimize influences that bias us, but there are always a few things out of our control that make it hard or impossible to be 100% objective. Also, we can try to be honest and open about them by disclosing potential sources of bias. That may help inform others and, over the long term, maybe people can find approaches that reduce the impact of those identified biases. This is the ‘interpretation’ part of archaeology, and it’s critical!

Additionally, attempts to write in an objective way can also reveal particular biases and do have an amount of subjectivity to them as they imply a kind of authority and create a certain kind of narrative edge to them. Archaeologists refer to these as theoretical perspectives. These drive the kinds of frameworks we use to interpret our data. They also help us give our data meaning (or to try not to give our data meaning) depending on what our goals are. A few of these narrative or theory-driven types include:

- objective/scientific
- descriptive
- subjective
- perspective (could you share your data narrative from the perspective of a Neolithic cow?)
- question driven

Let’s look at one quick example of a data narrative told via a role-playing game adventure, with an interactive, descriptive format.



Your party is stopped in front of a heavy wood and metal gate. It's locked. Coils of barbed wire fence stretch off in both directions. Beyond the gate, and through its slats, you see a sweeping green field.

What does your group do? Do you try to unlock the gate? Do you try to lift the barbed wire and slither through? Do you attempt to find the landowner?

Once you make it through the gate, you step into the grass. It's knee-high in places, forcing you to step carefully. As you pick your way through the field, you notice piles of white...something. Bones? Smaller pieces of bone are spread out throughout the grass, making white drifts in the vegetation.

Are the bones human? Roll your dice and make an investigation or medicine check.

If you roll up to a 9, failure. Oh no. You're unsure if these are human bones. They probably are? Maybe. Or not. You don't know.

If you roll over a 10, success! You discover that the bones are distinctly non human. They're much heavier, and appear bleached in the sun. These are definitely cattle bones.

If you roll a 20, well done! A study of the bones shows that not only are these not human bones, they're larger than any modern cattle bones you've ever seen. These are aurochs bones!

Once you've considered what kinds of bones you're looking at, the party may wonder...why are there so many bones?! They can spend some time investigating and counting the bones, looking for particular features and sorting them by type.

Roll your dice and make an investigation check.

If you roll up to a 9, failure. You discover a well-preserved piece of paper, but it's unintelligible. You should really do a tutorial on data literacy like that wizard suggested. I heard there are some good ones offered by the Alexandria Archive...

If you roll over a 10, success! You discover a well-preserved piece of paper, folded under a rock. It's some sort of inventory of all of the bones in this field. It will take more research to fully understand it though. Lucky for you, you've done some work with spreadsheets and with R, so you know how to get started.

If you roll a 20, well done! A study of the paper reveals it to be a curated data set of cattle bones, based on a project called, 'Neolithic and Bronze Age Cattle Data from Switzerland.' The project appears to have been carefully recorded and curated by Elizabeth Wright, Margherita Schäfer, Barbara Stopp, Elisabeth Marti-Grädel, Francesca Ginella, Manar Kerdy, Miki Bopp-Ito, Sabine Deschler-Erb, and Jörg Schibler. Because you've worked in spreadsheets and R, you quickly get started querying your data and preparing visualizations.

Next steps

It's time to bring the cows into the barn, as we've reached the end of this Digital Data Story. We hope that over the course of the last three pieces of this tutorial we've learned a bit about improving our data literacy, and have gotten some practice in with cattle bones data.

Remember, the methods we discussed in this tutorial aren't limited to this data set. What we've learned can be adapted and used on our own project data, or on any open data that we have access to. We'd love to see how we use these methods and how we've created our own data narratives. Please get in touch and share with us!

– The Data Literacy Program



NATIONAL
ENDOWMENT
FOR THE
HUMANITIES



This work has been made possible in part by the National Endowment for the Humanities and The Mellon Foundation. Any views, findings, conclusions, or recommendations expressed in this work do not necessarily represent those of the National Endowment for the Humanities or The Mellon Foundation.