# Scaling Up: Thinking about Programming in the Social Sciences

I was pretty lucky as a child. I grew up in Gander, Newfoundland, Canada. It didn’t realise it at the time, but Gander was kind of an experiment in social planning. Located at the eastern tip of North America, high on a plateau away from the stormy coasts, Gander was selected as the site of an intercontinental airstrip for refueling flights between North America and Europe. During World War II, Gander was little more than the airport. In a few later decades when intercontinental travel became the norm, Gander was selected to be Canada’s first international airport servicing flights between New York, Washington, or Toronto, with places like London, Paris, and Berlin. As a child I remember the constant flight traffic and airplane themes on everything from the names of the streets to the town festival (the festival of flight).

One day I was given a map of the town and shown something that I find striking to this day. Three of the main streets encircling the town square, one of which I lived on, created the silhouette of the head of a goose (a male goose being called a ‘gander’). Here, inscribed literally in the fabric of the town was the name and the town’s symbol. I had played on these streets, walked, been driven, or cycled up and down Memorial Drive, Edinburgh Road, and Elizabeth Drive hundreds of times. Never would I have thought that these three streets together made the shape of a goose head. That this was intentional was a part of town lore, with maps highlighting the goose head for tourists. Yet, until I had seen a map or was old enough to read one, I did not realise it was the case – for I wasn’t looking at the right scale.

Social science is often about considering things at scales that are meaningful, like the goose head, but difficult to comprehend through everyday experience, like walking around the streets. As an example, think about the newsfeed on Facebook or Twitter. Consider viewing the feed one story at a time, scrolling endlessly until you switch to another task. It’s the level of human experience. Now, how was that feed determined? Which stories came first or second? Do you know why Twitter showed that politician followed by this specific friend followed by that advertisement? Do you know why Facebook decides which posts to place at the top or to bring back as memories many years later? Do you know how many advertisements users will tolerate in their feed? Has the number changed over time or does it change in different markets? The answers to these questions are partially knowable, but not at the level of simply reading a newsfeed like a normal user. Instead, a user would have to know how to make a claim with evidence…and also to know which evidence is the right kind of evidence. To do that sometimes we need to zoom out from everyday experience to see things at the right scale.

Data science is a relatively new field in terms of disciplines. Later on, I will give a little bit of an elaborate definition of data science and *social* data science. But for now, let’s just say that data science seems to have emerged from a new set of opportunities: much of our world is now being recorded or mediated. This means that there exists a profusion of what we call data. It is no longer difficult to zoom out and see things at a different scale and not difficult to find data that represents some large-scale phenomenon. This could be data from weather stations or tweets about YouTube celebrities. It could be data from shipping routes for trucks or data on the number of avocados grown in California last year. Regardless, it’s all about the notion that there is a profusion of data available and the growing need to manage this data to make interesting claims. In that sense, data science often overlaps with social science.

It is not too difficult to see a line of inquiry moving from social sciences towards more social computing and then directly into statistics and even physics. A key part of this movement between different disciplines is about how we handle things at different scales. An anthropologist might work for decades at a single village and feel they are unqualified to speak about the next village a few kilometres away, whereas a physicist might examine a network of Internet servers one day and a network of friendships the next day, and yet find similarities in both networks. This is not to slight the anthropologist. Their craft might emphasise a level of depth that demands they see things in very contingent and specific ways. The anthropologist might tell rich stories and emhphasise thick description at a very local scale. Nor is it a slight to the physicist, as despite all the interesting differences that get washed out when looking at huge scales, the fact that we still see similarities in networks at the most macro scale (like all web traffic on a given day versus the architecture of the brain) is extremely fascinating. Instead, it is merely important to highlight that different claims are made at different scales and require different approaches. The reason this is now important to the social scientist is because we have the ability to see and consider the social world at scales that were previously inaccessible. Some of this was inaccessible because of the cost involved in collecting the data. Sometimes these scales are part and parcel of the newer ways of doing things. Texting creates a record in the act of texting, speech does not. You can record and transcribe speech, but it is not quite the same.

With a little luck and some sound training, you will be able to operate at these scales as a part of your career. Some of my students have gone to work for many of the large tech companies. They make decisions about things like the Facebook newsfeed, Twitter’s search results, cryptocurrency marketing strategies, and YouTube’s comment algorithms. Others are working for academia or government asking questions about open data and more practical questions. Some examples include how to measure the prevalence of trans persons on Tumblr, how does a mental health diagnosis affect online conversations in a support forum, how does coming out affect the structure of online friendship networks, and how to organize unions on social media. The answers to these questions are not simply for our interest but often lead to decisions. The idea about seeing at scale is that it allows us to intervene at that scale. Hopefully, when we are through here, it will be clear that not only should we look at more macro scale work, but consider how our own biases intervene at these scales. If we are lucky our work can make a difference, no matter how small, to the lives and experiences of those around us. Often that means learning how aggregates become dumb mobs or wise crowds, how events seem to coalesce around focal ideas, and how the structure of our coded world makes a difference to how we engage in it.

This is a world built of code. And to understand it, we have to become familiar with some of that code ourselves. So in this book we are, of course, primarily going to be focusing on programming. But I want you to think that the reason we are doing this is so that you can build your own ‘socialscope’ so to speak. Some socialscopes are like long exposure photography, checking out the same context over long time scales. Some socialscopes are vast, like trying to take a photo of the whole earth. In either case, what we want is to be able to shape the data we can get into a form that allows us to observe things on scales that we couldn’t otherwise. Sometimes then we can test claims in ways that help us be confident in what we are seeing and thus what we are claiming. But first we simply need to be able to know what we are looking at.

# How to use this book

This book is meant for social scientists who want to scale up their skills and move from investigations amenable to everyday experience (discourse analysis, interviewing, focus groups) towards investigations that occur at larger scales. However, this is not a traditional book in quantitative methodology. Statistics are only going to play a minor part in the first half of the book (and only a supporting role in the rest). Surveys are going to be almost entirely absent. Instead, the book is going to focus most intently on programming skills and how to translate these to meaningful questions using the vast amount of data available, particularly on the web. That being said, this book will infuse this programming with a number of concepts from social sciences, particularly sociology, but also economics, communication, political science and even history. You do not need to be an expert in any (or even one) of these fields. You just need to be interested in the work of those who have already been thinking at scale.

This is very different from traditional quantitative textbooks. Those tend to start with a data set that has already been cleaned, outliers removed, labels nicely in place, and everything is nice and rectangular. Honestly, data sets like this tend to be kind of dull. For example, intro books for one non-python statistical package often use data on cars and their mileage. Personally, I don’t own a car, don’t like using fossil fuels if I can avoid them, and I would much rather look at some memes than automotive data. It is the 2020s; if we want to start adapting to this world, we have to start thinking in it. This means data can be fun, data is often messy, and data is definitely contentious.

This book is not a gentle introduction to the Python programming language, but you will not need much Python before starting here. There exist a ton of books for that purpose. What can be done for intro to python books has already been done very well by a host of others. Instead, this book begins with the DataFrame. This is Python’s version of the table. It has rows and columns. Both rows and columns can have labels and the entire DataFrame can have a name. We can filter, summarise, group, and merge these tables. In the end, much of what we will be doing is moving between data as it exists for collection and data that is processed in tables for analysis. Below is a brisk summary of each of the sections to give you a sense of where we are headed.

### Part 1. Understanding programming.

The ‘intermediates’ rather than the basics. We start here with a discussion of programming, what it means and how to think about it. We will look at the DataFrames, Python’s version of a table. We then look at a host of file formats. If you want to get data into a DataFrame, it is likely going to start in one of these formats or something similar. If you have a little experience with programming (perhaps having dabbled over a weekend or two, taken a short ‘intro to programming’ or ‘intro to python’ course) then it will be a much more rewarding section. If you aren’t sure what skills you need to get started here, check Appendix xx. There I list all of the concepts I expect you to know before we start. I also give a quick primer and then point to other places for more extensive learning.

### Part 2. Exploring and merging data

Data from the web was not made for our analysis. It was made for whatever is its core purpose. To convert this data into a form that is useful we often have to clean this data. The first chapter looks at cleaning up text files. The second chapter looks at distributions. Here we will be doing some rudimentary visualisations and statistics. This is practical applied statistics, not statistics theory. We will keep formulate to a minimum but point to places where the reader can learn more. Finally, we have a chapter on merging and grouping data. Often aggregating data or merging two different sources can be the site of some of the most interesting data analysis. We will primarily be using data from Wikipedia and the World Bank to be doing this.

### Part 3. Crafting data

People have asked, “why don’t you have the research questions at the beginning?” It’s because research questions are actually pretty difficult to get right. I struggle with them to this day. But by this point in the book we have some skills and the ability to merge in data from different sources. Now is the place where we can start to ask research questions that draw upon this data. We can use our simple stats to start thinking about “differences that make a difference” and start conceptualising what data sources we want as well as what data sources we can access. We will cover APIs. If data powers the web, APIs are like the electrical sockets that need to be the right shape and expect the right kind of data to transfer. We will also cover some issues with good data stewardship. Sometimes it is simply best not to collect some data or to consider alternative strategies in order to best respect our population of study.

### Part 4. Researching data

These are four discrete chapters demonstrating different aspects of data science. Each one is just enough to get you started on your way in that particular genre of research. There’s a chapter on natural language processing that explores some of the important tools and features of this field. This includes sentiment analysis and keyword extraction. The next chapter is on machine learning classifiers. This builds off the previous chapter to look at ways in which we can distinguish the text that we have cleaned. The next chapter examines social network analysis, a powerful approach that helps us understand the world like a map of relationships. These relationships might be based on who replies to who’s comment, who posts in the same message board or who tends to fight with each other online. Finally, we look at time series analysis. Python has lots of features for slicing up data by time. You might want to know if a person is more active in the winter, whether their comments happen mainly at night, or whether there are bursts of activity on the weekend.

### Part 5. Big Data and professional research

These final chapters are meant to get you on your way for the next chapter in your learning. By the end of this book you should be able to embark on more advanced texts in computational social science and statistical learning. You might want to go deeper into the techniques of data science or dive into NLP. I first consider some of the tools required to ensure your work is robust (for example by working on a server rather than a personal computer) and then look at some of the challenges of doing really big data research. I then bring it back to the practical side. I provide tips on where to publish and present such work as well as some of the ways in which theories of social science can really reinforce the interpretation and research designs of data science.

The rest of this chapter will be about the logic of coding within the social sciences. It’s full of helpful analogies and ideas about how to code that have come from over a decade of teaching it to people (who often were very hesitant at first but almost always get the hang of it and find their footing).

# Prerequisites for FSSDS

This probably should not be your first introduction to the Python programming language. But why not start right at the beginning? In reviewing this book, it became clear that for the basics of Python, there exist a huge number of books, YouTube videos, Reddit posts, blog posts, and MOOCs. Surely at least one of them is better than what I could do.

Despite the proliferation of these “Intro to Python” type courses, there seemed to be a gap just afterwards. Students might endlessly practice some simple skills or learn some clever library but not know how to put the pieces together. This makes the chasm between introductory texts and specialised texts a little too great. Take, for example, Cloffi-Revilla’s “Introduction to Computational Social Science”. This book is stats heavy and assumes you already know Python. Yet, what does it mean to say you know Python? In most cases, it’s more than just do you know what is a list or dictionary. Instead, it’s expected you’d know how to clean some data, make a few plots, maybe capture some data from the web, and definitely to put this data in a table for analysis.

So in your hands (or on your screen) is a bridge-building book. My goal is to help you move from the introductory courses on DataCamp, Coursera, and their ilk to the more complex work in simulations and models. This book might not get you all the way there, but I know for certain that the skills in this book can help you get as far as published research, even in computational social science. If you want to then embark on specialised topics, I think you will be much better prepared. And if you want to just hone these skills in descriptive and exploratory work, you might be surprised at how far you can get with what is included in here.

So what should you already know before beginning with this book? Below is a list of python programming concepts you should already be familiar with. In Appendix XX are pointers to further places where you can practice these skills.

I suspect that the introductory skills needed for this book could be picked up in a couple days, but it’s also hard for me to remember what it is like coming from no programming. Maybe 3-4 days of solid instruction is more realistic? Maybe a couple weeks with practice to get really confident. I would not expect it requiring much more. But here are the things worth knowing:

* What is a variable?
* What is a string? What is a character?
* What are escape characters (e.g., “”, “\”, “'”) and when to use them?
* How do you print text? How do you print a variable?
* What is a list?
* What is a for loop? How does it differ from a while loop?
* What is a dictionary?
* How is accessing data in a list different from accessing data in a dictionary?
* What is a function and how do you build one?
* How do you use if statements and boolean variables?
* How do you write to a file and read from a file?

In appendix Ixx, I provide small code snippets for these tasks if you want a refresher. However, if you are coming to Python with zero programming knowledge, check the Appendix for further resources. There are many, many free resources available for this. Then, when you feel like you can handle these concepts, come back and dive into DataFrames.

## What about statistics?

Data science involves statistics by nature. If we have many data points, we have a distribution of them. Some data points will be larger and others small. But how much larger or how much smaller? Statistics can help us answer that. Classic statistics can help us say whether something is ‘significantly’ smaller or larger. More recent statistics can help us say whether knowing something about a conversation can help us predict whether the next utterance is more likely to come from one speaker or another.

For many, this is immediately going to put them off programming. Many times I’ve heard students say they won’t like programming as they aren’t good at maths. And if they aren’t good at maths, then forget stats! Hopefully this book can help ease any anxieties about math and stats. That’s not because the book is math and stats free. This book definitely will have some statistics in here. But the stats are relatively basic and we will make only the barest assumptions about prior stats knowledge. We might think of the way we use statistics in this book is as a learning practicioner. Similarly, I try to keep the number of formulae in the book to a minimum. Instead, get ready for a lot of analogies, examples, figures, and practice data. We will use statistics as one tool among many, but what we are teaching is *interpretation of data*, not statistics. Have a flip through the book and see for yourself: lots of code, more than a few figures but not a lot of greek letters.

If you want to advance in data science, especially towards the hotly emerging fields of machine learning and artificial intelligence, you will inevitably encounter some pretty advanced maths (especially in linear algebra) and stats (particuarlly Bayesian statistics not often taught in social science). This book is not going to get you there, but it will hopefully get you started in your journey. In the later sections of the book, we will be doing some work on transforming variables, which will require a little maths, checking some exploratory stats for comparing distribitions, and even introducing some really simple classifiers. With these in hand, it is my hope you wil have a working literacy for these topics and a newfound appetite to dig deeper into them in other advanced books.

Data science is a set of practices that allow people to make sense of data, typically (but not necessarily) large volumes of data that make simple calculations difficult. Because of the scale of data produced, a lot of data science is rolled into a closely related field: machine learning. These are, however, two distinct fields. Machine learning refers to a set of practices that use patterns and inference to perform tasks on data. Machine learning algorithms are often very hungry for data. Training neural networks to come up with speech patterns, or average faces might require huge sets of data (as in millions or billions words or photos). Thus, getting that data and making sure its the right kind of data are very much tasks suited for data science as a precursor to machine learning.

Yet, not all data science leads to machine learning. Much of it can lead to visualisations, more traditional statistics, network analysis and just descriptive tables with clear summaries. A lot of books tend to rush to show off the cool classification algorithms or show some complex patterns in the data but we are going to resist this here. Instead, we will be focusing on how to be thoughtful and critical of data. As I’ll lay out in chapter one, our goal is insight and wisdom from data, not just a data analysis for its own sake.

# Why Python (and not R, Stata, Java, C, etc…)

In the winter of 2002 I took an undergraduate computer science course called ‘Vocational Languages’. This course was meant to teach undergraduate computer science students (and those like me who were taking a ‘minor’ in computer science) some languages they would likely encounter in the field. This year the professor came into class stating that this year we would not be learning Perl, the previous reigning champ of text processing and very dense code. We would be learning this new language, Python. It seemed fun, easy to use and bound to catch on. For my final project I built a video game playable online using a Python web server. It was a small miracle I got it working but I was really pleased with the results. A year passes with me trying to do my own projects in Java (my programming ‘mother tongue’) and realising that I just liked Python’s syntax better. I got increasingly proficient at Python and used it in virtually all my papers that had some coding involved. I used Java less and less. Meanwhile the world around me was also discovering Python. For example, Eytan Adar had written this clever network visualisation package called GUESS. It was basically a window to see a network and a command line to use Python-like commands to change the look of the network. It was used in one of the most famous network diagrams of all time, Adamic and Glance’s picture of blog traffic between democrats and republications (xx). As an image of political polarisation it was a harbinger of the extreme political environment to come. Meanwhile, Guido Van Rossum, the creator of Python began working at Google, giving the language a boost in legitimacy. Students started picking up this language, often as a second language alongside Java (for computer scientists), R (for statisticians), Stata (for econometricians) and so forth. Python started feeling like the glue that kept us all from drifting into our own silos.

Today, Python is one of the most sought-after languages in data science. New algorithms emerge in this language incredibly quickly and libraries such as sklearn allow people to use Python to do some of the most cutting-edge machine learning. It turns out Guido’s intuition was right - many of the bugs we face happen because the language is too hard to read. Unlike many of its predecessors such as Java and C, Python is a little looser as a language. This means you can say less and let the computer make more assumptions. For example, you can just say x = 5 without first telling the computer that x is going to be a number (whereas in Java you would have to cast the variable by saying, for example, int x = 5. The fact that Python is ‘weakly cast’ means you run the risk of some input / output errors. What if you send "email.admin@example.com", which is a string representing an email address rather than 42 which is a number? There are times in Python where this might throw an error. But errors are not actually very common. So, we take such risks in order to use a language that is organised visually, pretty succinct, and very well supported.

Data scientists want a language that is approachable, with a smooth learning curve, easily rendered across platforms, and extensible with libraries for different purposes. Python now fits that bill. You want to access data on Reddit, there’s PRAW (the Python Reddit Wrapper). Making code that acts like a user on a browser? There’s now Selenium and mechanicalsoup. Neural networks? There’s tensorflow. Natural language processing? There’s NLTK. Python is widely supported, popular and has, to the extent any computer language has, a friendly and helpful community. Indeed, as of 2019, Python is now the most popular language on StackOverflow, suggesting a vibrant community of learners and experts alike (xx). Further, Python is not a language of computer science machismo. It’s not designed to win speed tests or compile into the smallest file size. It’s meant to be clear and useful. And good news for those on a budget, every library and package we use in this book will be completely free to download and use.

# Working with Anaconda and Jupyter

One of the best things about Python is that it now has an entire ecosystem for scientific computing. In this book we will be using the incredible Anaconda package for Python. In this package is not only the Python language and a bunch of useful libraries like many of the ones mentioned above. There are also dedicated software programs that help you develop code. Two in particular stand out: *Spyder* and *Jupyter*. For those who have used Stata or RStudio, Spyder will be familiar. It’s a program for developing Python scripts that are run on data. It has a window for typing commands, viewing output, looking at what data and variables have been assigned, and viewing source code to be run all at once. While I have made use of Spyder for tasks in the past, what really excites me about Python these days is Jupyter. It feels like exactly what I’ve always wanted; It’s like a long Microsoft Word document where the paragraphs can be code that you run right in the document.

Jupyter is a browser-based tool for viewing Python code alongside text, figures and results. It’s like a *Microsoft Word* document where paragraphs can be run and results can be inserted directly into the document. There are a number of flavours of Jupyter, but I prefer “Jupyter Lab” bundled in the *Anaconda* python package (xx).

As you have probably noticed, programmers tend to enjoy puns and in-jokes. The language Python apparently came from the creator’s love of the show ‘Monty Python’. Now, despite it being named after Monty Python’s Flying Circus, people still like using snake related puns. Anaconda is one of those. And it’s also an excellent package of Python libraries and apps. By downloading and running Anaconda (which is several hundred megabytes) you get access to all manner of scientific Python routines, particularly Pandas, which we will use extensively, as well as beautifulsoup, networkx, nltk, and many more featured in this book.

You can install all these libraries yourself by hand, but you will have to look elsewhere for instructions. That being said, if you want to see how I recommend installing packages in Jupyter, check out how I install our first external library on page xx. There is a small box with a code snippet I recommend using for importing routines in Jupyter should you wish to import your own.

When you download and install Anaconda, you will get a program called Anaconda Navigator. This is an application hub that allows you to open and run a variety of applications for scientific computing. The first application in the upper left corner is probably Jupyter Lab and the second is Jupyter. How does Jupyter Lab differ from Jupyter? The original Jupyter can present one notebook at a time in a single column. Jupyter Lab, on the other hand, has a panel on the left hand side for navigation and multiple notebook tabs open at once. Through Jupyter Lab you can open and navigate files and work on multiple Jupyter notebooks at the same time.

## Getting started with Jupyter Lab

When you click on the Jupyter Lab icon in the Anaconda Navigator it should open up your default web browser and navigate to a page with the following URL: http://localhost:8888/lab . This URL is a little different from conventional URLs. Instead of a domain name like www.eff.org, it is just the word localhost. This is your machine. It turns out Jupyter lab is actually a small server running on your computer that is serving you the application through a browser. The “8888” is a port number. A server communicates using different ports, often for different services. Standard unencrypted web traffic runs through port 80, while email often runs through ports 25 and 587. From now on, when we say open Jupyter Lab, we mean navigate to that particular browser tab that is running Jupyter Lab. Remember, however, if Jupyter is not running in the background then localhost:8888 will just display a blank page. You must launch the server first and then you can use it. In addition to doing this by navigating to the Anaconda Navigator application, you can also do this in the terminal on MacOS / Linux and using the Anaconda Prompt app on the xx.symbol(windows) menu on Windows.

When you open Jupyter Lab for the first time, you’ll be greeted with a navigation pane on the left and side and a single tab open labelled ‘Launcher’. You can use the Launcher to create a new Jupyter notebook. Create one using the first button (labelled “Python 3”). This is the default Python Jupyter notebook. There are a few ther types of notebooks that you can create at this point. Later (see Pg. xx) we will show you how to create a notebook that runs R code instead of Python.

<fig 1. Jupyter blank screen - xx.REVIEWERNOTE(Waiting until near book completion to get shot to have most up to date, but it’s basically going to be a flat screenshot in the browser with other details cropped) >

Now that you have a new Jupyter notebook, have a look at the numbered diagram in Figure 2. Here I point out the various controls on the screen.

<Fig 2. Jupyter notebook screen xx.REVIEWERNOTE(Ibid.)>

1. *The browser address bar.* This is where you would type a URL. At the moment, it probably shows localhost:8888/lab. This means that you are looking at a webpage that is run from your local computer. ‘localhost’ is a shorthand for the Internet Protocol [IP] address for one’s own computer. The ‘8888’ means that we have asked for data on Port 8888. You can also see Jupyter notebooks via the web, for example, Google have a service at https://colab.research.google.com that allow you to run Jupyter notebooks in the cloud.
2. *The file menu.* This is where you can click on commands for Jupyter such as **File**→**New Launcher** so you can create a new notebook. Notice one of the file menu items is called ‘Kernel’. This is the term for the instance of Python that runs the code, stores data, and returns a result. Each lab notebook has its own kernel. Sometimes we will need to restart the kernel, for example if we accidentally run a command that has no end, such as “count every number”. The other important thing to note in kernel is that we can clear output. Sometimes, we will want to start our Jupyter notebooks fresh. Clicking **kernel**→**Restart Kernel and Clear All Outputs…** will make sure that all the cells are treated as if they have never ran. It’s good to do this and re-run all your code from start to finish before sharing with other people.
3. *The navigation sidebar.* On the left-hand side is where you can select a file or check to see which files are currently running. The top icon is the file icon, it points to a file browser. The navigation is pretty similar to what you would get with a file browser on your computer such as ‘Finder’ on Mac or ‘Explorer’ on Windows.
4. *The tabs panel.* These tabs work like browser tabs. You can click on one to start working on that tab, drag the panels to change their order and check whether they have been recently saved by seeing whether there is a circle in the tab name on the right-hand side. You’ll notice that there’s a new file you just created and a second tab called ‘launcher’.
5. *The actions panel*. This small panel has some important and common actions like ‘save’, ‘run’ and ‘stop’. I tend to use keyboard shortcuts for these actions. You will definitely want to notice on this panel where it says the word ‘code’. That’s where we assign the ‘type’ of a cell. You can change the type of a cell by selecting a different type from the drop-down menu that appears when you click where it says ‘Code’. On the right hand side it says ‘Python 3’ which means that this particular notebook interprets code as Python 3 code. The right most dot is a status meter. When the computer is busy running code the circle is filled in and looks like a spot. When the computer is idle the circle is empty and looks like a ring.
6. *The main panel*. This is where the work gets done. In this panel you’ll see that content is organised in cells. Each cell can be either ‘code’, ‘Markdown’ or ‘raw’. **Raw** text is not highlighted and the computer just ignores special characters and code. **Code** means that the contents of a cell are treated as Python code. **Markdown** is text that has extra characters to denote formatting. For example, Markdown uses two asterisks on either side of a string to indicate it should be bold: When I type \*\*this\*\* into a Markdown panel the text is rendered like **this**. Markdown is discussed more just below.
7. *A Python cell*. You can tell this is a Python cell on your computer because it has *syntax highlighting* that indicates Python-oriented words and variables. For example, the word ‘print’ will show up in green and comments will show up in blue. It also will say ‘code’ in the cell type in the main panel. To run the cell you can do any of the following:

* The file menu. Click “Run”→“Run Selected Cells”.
* The triangle in the ‘actions’ panel.
* (My favourite) Shift-enter on the keyboard.

1. *Code numbers*. When you run a cell, it will report a number off to the left of the cell. That number represents the order in which cells were run. If there is no number that means the cell has not run yet. If you run a cell a second time, it will increment the number, so the number could actually go much higher than the number of cells in the notebook. You can also see a blue bar to the left of the number. Click that bar and it will collapse the output. This is handy if you have just printed out a lot of output but you want to hide it while you work on the code underneath.

## How to add text to a Markdown cell

A markdown cell is one that has text in it. Markdown is a simple way to add features to text, like *italics*, headers, ~~strikethrough~~, and **bold**. In a Jupyter notebook that is rendered, you can click on a cell to see the Markdown that produced the text. Some of the more common things you will see in Markdown:

1. Use two tildes (the ~ character) for ~~strikethrough~~
2. Use two asterisks (the \* character) for **bold**
3. Use underscores (the \_ character) for *italics*.
4. Lists are auto generated by having several lines, where each starts with an asterisk and a space.

* Here is a list item
* A second item
* These are indented because we had a space before the asterisk

1. You can also embed code in a Markdown cell. It won’t run but it will have syntax highlighting and a monospace font. This is using three tildes and then the name of the language like so:

* ~~~ python
* print(“Hello World”)
* ~~~

and it will be formatted on the screen like:

print("Hello World")

1. Use hash (the # symbol) at the beginning of a line to make it a heading. You can use two hashes to make it a subheading (or three for subsubheading, etc…).
2. Use the dollar symbol on either side of a formula to use math notation like (and I use it for ).
3. Three or more dashes at the beginning of a line create a straight line across the page

## How to create a new cell / navigate with the keyboard

At any given time only one cell might be in focus. To say ‘in focus’ means that the cell is editable. It also means keyboard shortcuts and the computer’s list of undo actions refer to the text in that specific cell. When a cell is out of focus, it is still indicated with a blue strip on the left-hand side, but keyboard shortcuts and the undo actions refer to the Jupyter file and cells rather than their contents. It is important to be able to navigate into and out of focus for a given cell with the keyboard if you want to be a fluent user of Jupyter. Being able to have cells of different types organized in your notebooks is where Jupyter shines. For example, you can have one cell of code, then graphical output, a well-formatted table of numbers, and your notes just below. Moving around these cells can help you navigate not just the code, but the overall analysis. It also helps you to think about chunking your code and organising your questions.

To change the focus from one cell to another, you can click on a new cell or ‘run’ the current cell. To run the current cell, press .

To get out of focus you can either: \* Run the current cell (), \* Escape the cell (). \* outside of the cell.

To get in focus you can either: \* Press , \* with the mouse.

If the cell is not in focus you can tell because there is no cursor and pressing up or down will move the blue bar on the left hand side up and down. If it is in focus you can tell because pressing up and down will move the cursor within the cell.

To create a new cell, you have to be out of focus. You can do this by pressing for above and for below the current highlighted cell.

To delete a cell, you press , that’s twice. Remember that this only happens when you are not in focus, otherwise you would just be typing the letters dd into the cell.

To change a cell from ‘code’ to ‘Markdown’ you can either: - Use the menu at the top, - When the cell is not in focus you can press for code and for Markdown.

# How to write formulae in a cell

Throughout this book, you will see a few formulae. For example, here is a formula for getting the average (or specifically the ‘’arithmetic mean’):

This formula was not written with Markdown but with a special typesetting language called $\LaTeX$. Technical papers are often drafted in $\LaTeX$ as are many books in STEM fields. It is less common in social sciences, but it is really handy. I wrote my dissertation (like this book) in a combination of $\LaTeX$ and Markdown.

The formula was given its own line because it was enclosed with $$ characters. Here is what the code looks like: \bar{x} = \frac{1}{n} \sum^{n}\_{i=1}x\_{i}. If we enclose it with single $ it will be a formula inline, like so: $ {x} = ^{n}*{i=1}x*{i} $. I use inline formulae for most numbers in this book as well as commands.

If you click on this cell, you can see the formatting underneath. Here we are just using MathJax, which is a subset of LaTeX used for formulae. StackExchange have a nice brisk tutorial of the syntax of MathJax [here](https://math.meta.stackexchange.com/questions/5020/mathjax-basic-tutorial-and-quick-reference). Basically, there is syntax for: \* Superscripts , x^i , subscripts, x\_i , \* Fractions, with \frac{NUMERATOR}{DENOMINATOR}, as in \* Parentheses (using \left( and \right) to scale properly) as in \* Summation, product, and related symbols. \sum for , and \prod for . \* Greek symbols. Use their name for the symbol such as \alpha for or \omega for . \* A host of diacritics, math symbols, and fonts. Check the tutorial above for clear examples.

## The big Jupyter Gotcha

There are many advantages to running code in Jupyter but there are a few caveats. One in particular is really important to discuss right up front: You can run cells in any order even if you do not mean to. From this you can run into some pretty common issues.

1. Running code out of sequence. Imagine that in cell one I clean up some text (for example, I remove all the periods and commas). Then in cell two I run some code on that text (for example, make it ALL CAPS). Now imagine I then go back and do something else in cell one, such as change my code to remove apostrophes as well. So I run cell one again, but skip the second cell. Now my data is not in ALL CAPS and subsequent cells will not get the data they expected.
2. The “I’ve changed my variables to make them read better” issue. This one is my number one gotcha. As an example, I might change a variable name once I get my code working but want to make it more readable. But because Jupyter does not have a great ‘find and replace’ system, I might forget to change *all* of the instances of a variable. So if the variable was called tl but I want it to be tweet\_list, then I replace the variable name. But what if I do not change it *everywhere*? There might still be a tl left in the code somewhere. The program sees this and keeps running (since tl was already created). But the next time I restart the program, that tl will not be created, tweet\_list will. So the program will throw an error that any remaining tl is an unrecognised variable.

To practice navigating in Jupyter as well as see some of the issues that might happen if you’re not careful, you can download and run <xx.exercise\_ch00\_s1.ipynb>.

In general, - Run your Jupyter cells in order, unless absolutely necessary. - If you change a variable, be sure to change it **everywhere**. - If you change a cell further up in your code, run every line afterwards. - If you send notebooks to other people, then from the menu: “Kernel”→“Restart Kernel and Run All Cells…”. If you get an error then debug it before you send the code to people.

# Github: A home for collaboration

Ever wonder how computer programmers keep their code up-too-date? They use a version control system. Such systems allow you to download code, modify it, check in those modifications and remember different versions of the code so you can roll back any changes. There are numerous version control systems out there. The classic one was simply called cvs or “Concurrent Versions System”. Despite its many advantages it was eventually superseded for most programmers by a system called “Git”.

Git is a system for managing files. You can run a git server on your own computer. It is doable but not especially common. Instead, people tend to use a git platform run by someone else. The dominant one is GitHub. Another would be BitBucket. If you find code on Github you can download it directly to your computer. However, you can go further than that and ‘fork’ the program. This means you have now created your own variant of the existing software. If you make changes to a fork of a software program those changes only show up in your forked version. If you want those changes to appear in the original that you forked, you can create a ‘pull request’, which asks the maintainer of the original program to integrate (or pull in) your code.

The exercises for this book can be found on Sage’s GitHub repository. We will not cover most of the details of Git here, but it is worth noting that GitHub now renders notebooks on the site. This way you can go to an archive with a Jupyter notebook and see it on the site rendered as if you were running Jupyter. You cannot run the cells however. For that you would have to run the notebook elsewhere such as http://colab.research.google.com.

If you want to download the notebooks, GitHub allows you to downlod the file directly, or to download it into an archive on your computer that you can sync with GitHub. While many people use the terminal for this, GitHub also has applications for Windows and Mac that make it very easy to manage files as they are edited by you and collaborators and synced through GitHib.

I would actually recommend that you create your own clone of the files for this book and work off of that clone. To note, simply because you “clone into desktop” this project does not mean you can push changes to the original repository. For that, you would need to be a project member and not just a reader of this book. If however, you wanted to alert a repository maintainer of a typo or otherwise want to contribute, you can always create a pull request.

# Jupyter Extras: Table of Contents and GitHub integration

There are two extras that do not come with the Anaconda version of Python but should hopefully work with Jupyter Lab given your specific version (which will be more recent than the one I am using here, by nature). Both are extra tabs in the side bar that help you manage your files: “Table of Contents” and “GitHub integration”. These might be a bit tedious to install, but I have found that they both work pretty much as expected given the instructions. Also, both come from the Jupyter Lab team, so they are not especially experimental or fragile, just optional.

Before you install them, however, you might need to install some of the ‘dependencies’. A dependency is a package or program required for a program to run. The most likely package you will need is the NodeJS package. NodeJS is an internet server architecture. It runs code in JavaScript on a server and renders it for a client, like a web browser. To install NodeJS so you can install the following packages, open Anaconda Prompt (on Windows) or a terminal window (MacOS / Linux) and type:

conda install -c conda-forge nodejs

## JupyterLab Table of Contents.

This is a very handy tab for the sidebar. It uses the header depth of markdown to create automatically numbered sections. If you notice in Jupyter that when you click on a header it is like regular text except it will have a hashtag in front of it. One hashtag means header 1. Two mean header 2. Three mean header 3, etc…

Therefore, if your text looks like this:

# Big Headline is Big  
## But that doesn't always mean it's better  
### Sometimes good things come in small headers.

then on the sidebar you will see:

1. Big Headline is Big   
1.1 But that doesn't always mean it's better  
1.1.1 Sometimes good things come in small headers.

Note that it only works with headers at the beginning of cells. So if you place a subheading in the same cell, Jupyter Table of Contents will not find it.

To install the latest version of Jupyter-toc run the following code in console:

jupyter labextension install @jupyterlab/toc

Now how do you find a console? On windows type WINDOWS-KEY to bring up the search, then type “anaconda prompt”. There might be a couple prompts on your computer, but this one will know where to find python and install the extension in the correct place. On MacOS, search for ‘Terminal’ and then run the above command in the terminal.

Please note, this code does not always work as it is sometimes out of sync with the version of Jupyterlab in Anaconda. But it should work and is very handy. Consult the ‘updates’ section of the GitHub page if this doesn’t work. There I will provide updated solutions for installation if possible.

Once you run it make sure that you do not get any errors. Note that sometimes you might get a *warning*; these are ok. They often advise caution but do not mean the program failed to install. *Errors*, on the other hand, will stop the program from installing. If you do not get any errors, refresh your JupyterLab window. You should see a new tab on the left-hand side with three lines and three dots. That’s the table of contents tab. Now open a notebook file with Markdown headers (or create one) and see for yourself how it works.

## GitHub integration

There are many exercises and sheets that go along with this course. You can download them from github.com. But wouldn’t it be nice to have the most updated version ready immediately in JupyterLab? The Github extension allows you to type in a repository and view the files right from JupyterLab. Thus, by installing this extension you’re getting much closer to viewing the exercises right when you need them. As a bonus, this book is not the only one that does this. On Github you can see many repositories of python notebooks. Some of my favorite include Van Der Plas’s Data Science in Python (which is like this book but with less social science and many more techniques) and the Bokeh library which shows how to create interactive visualisations to reveal data. We will be loading some Bokeh code later in the book.

To get started, first run the following code, again from the prompt/terminal like with Table of contents above:

jupyter labextension install @jupyterlab/github

Then refresh your browser. Now you can type in the name of a Github user and see the code in each of their projects. Unfortuantely, you cannot save your code as it would overwrite what is on the server. If you want to save your progress I recommend right-clicking the file name, select “Copy” and then navigate to a home folder and paste. Alternately, just download the entire repository from the GitHub webpage or the GitHub app and use that directory.