Python or *R*? Getting Started with Programming for Humanists

William Mattingly

Abstract This article describes how programming languages such as *Python* and *R* open up new research opportunities for the humanities by analysing large text corpora, visualising patterns in data and automating repetitive tasks. *Python* and *R* are probably the most prominent programming languages for engaging in the Digital Humanities. *Python* offers advantages for text analysis and machine learning due to its simple syntax and versatile libraries, while *R* scores with its statistical functions and visual representation options for data manipulation. The choice between *Python* and *R* therefore depends on the specific research requirements, although both languages are well suited to the humanities due to their strong communities and extensive resources. Learning strategies for getting started with programming and how to deal with potential pitfalls are also discussed.

Keywords Programming Languages, Python, R, Humanities, Digitization, Digital Education

1. Introduction

Programming is a fundamental skill in the hard sciences and maths. In these disciplines, programming languages like *Python* and *R* function as a tool to conduct research. They enable scientists to process large datasets, run complex simulations, and automate repetitive tasks. For instance, a biologist might use *Python* to analyze billions of genetic sequences from many organisms, or a statistician might use *R* to model and visualize patterns across data. In this chapter, we will explore how this concept translates to humanities data.

2. Data and the Humanities

Unlike the sciences, the humanities are traditionally associated with qualitative analysis, valuing a close reading of source material. For most of the twentieth century the majority of the humanist's data was available in analog forms, that is through the medium of print. The methods we developed for analyzing our data, therefore, were targeted at the practical limitations of our data and human anatomy. Even the most ardent researcher cannot possibly read the entire corpus of Latin literature in a single lifetime. Presuming one had the capacity to read such a quantity of material, one would have to physically obtain it. In manuscripts, these resources are scattered across different continents inside institutions that have limitations to access. Even presuming one could overcome these physical limitations, one would then have to synthesize it in some meaningful way. How can we retain all this information and translate that knowledge to an audience practically? We simply can't.

During the second half of the twentieth century, the medium for transmitting data changed. Text could be rendered as numerical data. Initially this was via punched cards, such as those used by *Index Thomisticus,* the first project that digitally rendered the collected works of Thomas Aquinas (approximately 10 million words, Busa 1980; cf. the chapter from J. Peters in this volume, p. 316). Because the digital data was rendered on physical punched cards, it still required physical access to the punched cards. These could, however, be transmitted differently than print and loaded on different machines across the globe.

As technology continued to develop, data could be stored not as physical punched cards, but as disks. As these disks became smaller and capable of holding large quantities of data, these disks could be transported more easily and reproduced more cheaply. By the 1990s, the *Index Thomisticus* evolved with this technology and made its data available via CDs (Economist 2020). This meant that scholars could purchase identical data and analyze it regardless of location. This data could be reproduced, distributed, and computationally accessed by a larger audience.

Over the last two decades, we have not only produced larger quantities of digital data, accessibility likewise increased thanks to the advent of the internet and the cloud. The cloud allowed for data to sit on a server in one place and be accessed by someone on the other side of the globe. We can, for example, study the entire *Patrologia Latina* from a beach in Florida via the open-access *Corpus Corporum* project at the University of Zurich.[1] So ubiquitous is this technology today, that it is sometimes difficult to appreciate how incredible this feat truly is.

Today, the limitations of quantity and access are gradually vanishing. As the scale of humanities research extends into the realm of *big data,* the ability to handle and interpret this information becomes crucial. What do we do with this data? How do we access it systematically? How do we use it to frame questions and translate that knowledge into something useful? As we will see throughout this chapter, programming affords us potential solutions to these questions. Programming is also an emerging skill that departments are including in their curricula, such as that of the recent Ph. D. in Digital History at Clemson University.

With the help of programming languages like *Python* and *R,* humanists can analyze large corpora of texts, visualize complex relationships in data, and uncover

1 See https://mlat.uzh.ch (Accessed: 22 June 2024).

patterns that would be impossible to find manually. Whether it's studying the entire collected works of Augustine (d. 430) or mapping the social network of Alcuin (d. 804), programming opens up a whole new world of possibilities for humanistic inquiry. For the humanities, much like the maths and sciences, programming languages are not necessarily used for creating software; scripting languages like *Python* and *R* function like tools by which we can analyze large data, interpret it, and visualize findings. Unlike software, scripting allows for researchers to leverage the quick utility of programming without long-term sustainability necessary to maintain software.

3. Why Learn to Code?

One of the most important questions we can ask is *why learn to code?* Behind this question lie two other questions: *What is the utility of coding and how does it benefit me directly?* To answer these questions, let us consider a problem. Imagine we needed to identify all named persons in the writings of Augustine. We could, of course, spend months or years going through and manually tagging each person. This would be time consuming and repetitive. If we knew programming, however, this problem becomes far simpler to solve and the solution can be developed in hours (or days depending on its complexity). We could either construct a set of rules for all the people we expect to find in the letters or (if we did not know this), we could develop a machine learning solution that would learn the features of the words that correspond to people and classify them for us automatically. This task is known as *named entity recognition* (cf. the chapter from E. Gius in this volume).

This process by which we create a solution that can be implemented repetitively over similar data, is known as automation. Automation is one of the key reasons to learn to program. As humans, we are very bad at doing the same task repetitively and consistently. Computers, on the other hand, are perfect at both of these tasks. Learning to program is, in part, learning to automate tasks and is one of the key benefits. It allows the researcher to spend more time researching and less time performing repetitive tasks.

In addition to automation, programming also affords us the ability to develop solutions that work at scale; this means being able to perform practically the same solution across millions of data. If we can identify all persons in a text of 10,000 words in five seconds (a reasonable time), then we could easily repeat this process across millions of texts in just a few hours, depending on computational resources.

Through programming, we can also develop and apply machine learning solutions with a few lines of code. Continuing with the example above, imagine we needed to solve the same problem with the works of Jerome. Here, we could take the machine learning solution we developed for the works of Augustine and apply it to the works of Jerome and have comparable results.

Programming allows humanists to achieve far more than the creation of novel solutions. It fundamentally changes the way we frame problems. A knowledge of programming brings with it knowledge of solutions or potential solutions to unknown problems (or unasked questions). Those who have a knowledge of programming and know what is possible to do with it know that they can ask new questions and pursue novel research directions. It makes it feasible to conduct studies at a scale and depth previously unimaginable. With programming, researchers can identify patterns and trends across large datasets, leading to new insights and understandings.

Selecting the right programming language largely depends on the problem at hand. If the researcher needs to build a website with custom functionality, then learning HTML, *JavaScript,* and *React* (a way of building JavaScript components easily) may make the most sense. If the researcher needs to work with and manipulate data in any way, there are two dominant languages that should be considered: *Python* and *R.*

4. The Benefits of *Python* and *R*

Python and *R* have emerged as the primary programming languages for many fields, namely data science, machine learning, natural language processing, statistics, the sciences, the maths, and the humanities. *Python* was created by Guido van Rossum in the 1980s with the idea that code should be easy-to-read and easy-to-right with a syntax, or style of writing, that was concise and simple (Van Rossum & Drake 1995). Against *Python,* sits *R* which was created by Ross Ihaka and Robert Gentleman in the 1990s (R Core Team 2021). Unlike *Python, R* was designed purely for statistical analysis. Its syntax is a bit unconventional, but it affords users the ability to apply statistical methods to quantitative data easily and visualize the results.

Both of these languages have increased in popularity largely due to their excellent and active communities. These communities write packages, or libraries, for each programming language. A package is a collection of classes and functions (think of these as large blocks of code) that can be leveraged by others in the community. This means that new users to the programming language can do complex tasks with very little code, making them ideal programming languages for beginners. For example, students new to *Python* can load a machine learning model and perform *named entity recognition* on the entire corpus of Thomas Moore (for NER cf. the chapter from E. Gius in this volume). This is thanks to the natural language processing library *spaCy*[2] and the contributions to that project by Patrick J. Burns who created *LatinCy* (Honnibal & Montani 2017; Burns 2023). If one were interested in doing *transformer-based topic modeling,* which is the newer approach to this decades-old methodology, a researcher can do so with just 2 lines of code using *BERTopic* (cf. the

2 See https://spacy.io (Accessed: 22 June 2024).

chapter from M. Althage in this volume, esp. p. 259, n. 19). While further methodological steps are necessary to refine a researcher's approach, such as topic identification, validation, adjusting hyperparameters, the code required to do so remains minimal. This is because *BERTopic* employs many advanced methods in sequence for the user. It even allows for the quick visualization of a topic model. Contributions like these make *Python* and *R* appealing to those with limited and advanced coding experience alike. This appeal, in turn, fosters a healthy community which continues to grow. As the community grows, more members contribute their own packages. As time progresses, the cycle continues to repeat.

5. Comparing *Python* and *R* for Humanities Research

The relevance of *Python* and *R* to humanities research is multifold. *Python*'s simplicity and readability make it a great starting point for humanists new to programming. Its extensive libraries such as *Pandas*[3] for data manipulation, *spaCy* and the *Natural Language Toolkit* for natural language processing, and *Matplotlib*[4] or *Seaborn*[5] for visualization, provide valuable tools for various research tasks in the humanities. On the other hand, *R*'s strong data handling and built-in statistical capabilities, as well as its powerful visualization libraries, make it particularly suited for humanities researchers who work extensively with statistical data or need to create complex visualizations. When it comes to choosing between *Python* and *R* for humanities research, the decision often boils down to personal preference, specific project requirements, and the kind of data you'll be working with. Both *Python* and *R* have their strengths and are capable tools for handling humanities data.

Python has a few key advantages. First, its syntax corresponds to other programming languages and is generally easier for those new to programming. Second, *Python* allows users to quickly build websites via libraries like *Django*[6] and *Flask* (Grinberg 2018). Through *Streamlit*[7], a novice Python programmer can build a custom data-based application and put it in the cloud with only a few lines of code. Third, Python is often the first choice within the machine learning community. This means that most recent machine learning developments are available first in *Python.* Fourth, most natural language processing advances are often developed in *Python,* making it ideal for tasks like text classification, *topic modeling,* and *named entity recognition.*

3 See https://pandas.pydata.org (Accessed: 22 June 2024).
4 See https://matplotlib.org (Accessed: 22 June 2024).
5 See https://seaborn.pydata.org (Accessed: 22 June 2024).
6 See https://djangoproject.com (Accessed: 22 June 2024).
7 See https://streamlit.io (Accessed: 22 June 2024).

R, on the other hand, was built for statistics. It has several advantages over *Python.* First, *R*'s syntax and functionality are tailored for statistical modeling, allowing for complex analyses with concise code. Second, *R* provides an extensive collection of packages like *ggplot2* (Wickham 2016) and *Shiny*[8] that enable high-quality data visualization and interactive web applications. While *Python* boasts of good visualization libraries like *Plotly*[9] and *Seaborn* (Waskom et al. 2017), the visualizations in *R* are easier to produce, tend to look nicer, and are easier to customize. This means that users can not only analyze data but also create visually appealing representations with ease. Third, *R*'s integration with various data sources and its data manipulation capabilities through packages like *dplyr* make it a powerful tool for data wrangling. Fourth, although *R*'s machine learning capabilities may not be as extensive as *Python*'s, packages like *caret*[10] and *randomForest*[11] still provide robust tools for machine learning.

Both languages have active and supportive communities, so you'll find plenty of resources and help for both. Choosing between *Python* and *R* will depend on your specific needs and goals in humanities research. There are three main questions to consider. First, consider your own research needs. If your research involves a lot of text analysis or natural language processing, *Python* might be the better choice due to libraries like *spaCy.* If your work involves heavy statistical analysis or you need to create detailed visualizations, *R* might serve you better. Second, consider your own learning style and your experience with programming. If you're new to programming, *Python* may be easier to learn. It will be important to look at a couple snippets of code in *Python* and *R* to get a sense of how different these two languages are. Third, consider the communities behind both programming languages. Those communities will be there to help you when you run into challenges. Both *Python* and *R* have strong communities, but depending on your area of study, one might have more relevant resources and discussion boards than the other.

In the end, there's no definitive right or wrong choice between *Python* and *R* for humanities research. It's about choosing the tool that best suits your needs and complements your research. A humanist interested in using programming in their research will likely learn to write in both languages but typically prefer one over the other. This is because most things done in one of these languages can be done in the other, even though it may not be as easy. If you have a project entirely written in *Python,* for example, but need to produce a nice visualization, it may not make sense to introduce *R* into the workflow for one step. Instead, you will write more code and use the Python library *Seaborn.* Likewise, if you are presenting your findings on text

8 See https://shiny.posit.co (Accessed: 22 June 2024).
9 See https://plotly.com/python (Accessed: 22 June 2024).
10 See https://cran.r-project.org/web/packages/caret/vignettes/caret.html (Accessed: 22 June 2024).
11 See https://cran.r-project.org/web/packages/randomForest/randomForest.pdf (Accessed: 22 June 2024).

analysis and statistics, it may not make sense to use *Python* and *spaCy* for lemmatization, or the reduction of all words to their root form.

6. Getting Started with Programming

One of the most challenging aspects of learning to program is installing the programming language on your computer. Each operating system, such as Mac, Windows, or a Linux distribution (like Ubuntu), requires you to install the language differently. Each operating system also has unique steps. On Windows, for example, you need to make sure that *Python* sits in your system's PATH (often manually). Each operating system also introduces small, but critical differences. On some Macs, for example, you will have *Python* 2 pre-installed on your system. This means that when you install the recent version of *Python* (*Python* 3.12 as of writing this), you will have two versions of the same programming language on your computer. This means that you need to use the command "python3" in the command line to execute a *Python* file on some Macs. On Windows and Linux, however, "python" will be the command you use. Many of these issues are negated, however, if one uses virtual environments or *Conda*[12].

Installing the programming language is only one hurdle. Users usually want to also install a custom way to interact with the programming language. For *Python*, this typically means installing an integrated development environment (IDE) like *JupyterLab* or *VS Code.* For *R,* this means installing *R Studio.* These are tools that allow you to write and execute code in a single space. They allow you to learn more easily and also it will be the way in which you typically engage with a programming language. Installing and setting up an IDE will vary depending on your system requirements. Again, this is a step that can lead to confusion and issues as well.

In my experience, the frustrations that new students encounter during this process can dissuade them from wanting to learn to program. These are some of the most important moments in a student's career. It is when they are curious about programming and eager to learn. These frustrations can quickly dim the light of curiosity. To avoid this, I recommend that all new students skip the installation of the programming language and the IDE entirely. Instead, there are numerous companies that offer cloud-based solutions to these issues. They allow you to access a server remotely and run *Python* from your browser.

Constellate from ITHAKA[13] is one such solution, but it requires institutional access. It provides students with a virtual environment with enough resources to even do machine learning should they wish. It comes pre-installed with libraries commonly needed on humanities projects. Each user instance also comes pre-installed with

12 See https://docs.conda.io/en/latest (Accessed: 22 June 2024).

13 See https://constellate.org (Accessed: 22 June 2024).

JupyterLab, an IDE that facilitates data management and learning (via *Jupyter notebooks*). This makes it ideal for classroom settings. I have taught for three years with *Constellate* and highly recommend it.

Not all students will have institutional access to *Constellate,* however. If this is your case, there are comparable services available. The most popular is *Google Colab*[14] which has several tiers, including a free version. It can link to your *Google Drive.* This means you can load data onto your *Google Drive,* interact with it, manipulate it, and save it. The free tier tends to drop occasionally, but works well for getting started nonetheless.

7. Resources for Learning

Learning to program as a humanist is often an individual endeavor. Even if you have a formal education in *Python* in a college classroom setting, you will have to rely on your ability to self-teach for the remainder of your programming career. This is because real-world data and problems are messy. Clear and easy solutions are rarely evident. You must be prepared to learn new aspects of a programming language to solve novel problems as they emerge. Imagine you have learned to program in *Python* to do *named entity recognition.* Now, you need a way to visualize those named entities in a network graph. How do you do that? In this case, it would be time to learn *NetworkX*[15] to collect the data and either *Matplotlib* or *PyVis*[16] to visualize it.

Because *R* and *Python* have large communities and a lot of libraries available for solving common problems, there are numerous resources available for furthering your education. One great resource is Walsh (2021), an open-source textbook titled *Introduction to Cultural Analytics and Python.* This textbook not only teaches the basics of *Python,* it also provides an introduction to a few key methodologies, such as text analysis and network analysis. The standard textbook for Humanities data and *R* remains *Humanities Data in R* by Arnold & Tilton (2015). It has open-source online resources, but these are meant to be used alongside the purchased textbook. Another *R*-specific resource is the open-source textbook *Computational Historical Thinking. With Applications in R* by Mullen (2018). Unlike recent open-source textbooks (such as Walsh's), this is not designed with *JupyterBook.* Nevertheless, it still has useful supplementary material, such as worksheets.

After you have acquired a basic understanding of either *Python* or *R,* you will find yourself frequently needing to learn a specific library. Often, the documentation for these libraries is sparse. It is usually written by experts for more advanced users.

14 See https://colab.google (Accessed: 22 June 2024).
15 See https://networkx.org (Accessed: 22 June 2024).
16 See https://pyvis.readthedocs.io/en/latest (Accessed: 22 June 2024).

At these times, it can be necessary to have a relatable tutorial. If you want to stick with academic tutorials, the *Programming Historian*[17] is probably the best resource available. As of now, there are 101 lessons for both *Python* and *R.* These often center around a specific problem or methodology. The lessons published here are both open-source and peer reviewed via *GitHub.* They tend to be more targeted at specific problems or specific libraries.

Some of the best resources available, however, are written by non-academics. These resources are published on *YouTube* and *Medium.* The easiest way to find resources for your specific problem is to search on these platforms for something in which you are interested.

8. Common Pitfalls with Programming

As you progress in your programming career, you will encounter many pitfalls. One of the most common is a bug in your code. Most programming languages will give you an error message to indicate why a piece of your code failed and it will often point you to the specific line. As you work with external libraries and write more complex code, however, debugging can be trickier. Fortunately, there is (or was) a healthy community at *Stack Overflow* which allows users to post bugs and ask for help. Usually someone in the community will respond within a few hours. Since the advent of *ChatGPT,* however, traffic on *Stack Overflow* has decreased. The implication is that more users are asking questions via *ChatGPT* or some other similar service. For basic coding issues, *ChatGPT* will provide fairly good and specific advice for fixing a bug. There is a good reason for this; it was trained on a lot of *Stack Overflow* data. When using *ChatGPT* or *Stack Overflow,* it is always important to not simply copy-and-paste the solution, but to understand *why* the bug surfaced in the first place so that you can correct the issue *and* learn from your mistakes.

Another common pitfall is the use of algorithms that are not fully understood by the programmer. While a programmer does not need to know how neural networks work to leverage them and generate useful output, it would be unwise to rest an argument on the statistical output of a model whose algorithm the researcher does not understand. When you do not yet understand the methods fully, therefore, programming should be used as a tool to assist in research, never as a tool to validate arguments.

17 See https://programminghistorian.org (Accessed: 22 June 2024).

9. Conclusion

Programming is not meant to replace traditional humanistic inquiry; instead, it provides humanists with new avenues to conduct research. It allows us to frame questions that we could not otherwise answer. It allows us to automate tasks in hours that could otherwise take years. And it allows us to glean new insights from large quantities of data. As the humanities evolves and technology becomes ubiquitous, tomorrow's humanists will likely acquire more technical skills, just as they did with the advent of the Word processor. Tomorrow's Word process is programming.

References

Arnold, T., & Tilton, L. (2015). *Humanities Data in R. Exploring Networks, Geospatial Data, Images, and Text.* Cham: Springer. DOI: https://doi.org/10.1007/978-3-319-20702-5 (Accessed: 22 June 2024).

Burns, P.J. (2023). LatinCy. Synthetic Trained Pipelines for Latin NLP. *arXiv.* DOI: https://doi.org/10.48550/arXiv.2305.04365 (Accessed: 22 June 2024).

Busa, R. (1980). The Annals of Humanities Computing. The Index Thomisticus, *Computers and the Humanities,* 14(2), 83–90. URL: https://www.jstor.org/stable/30207304 (Accessed: 22 June 2024).

The Economist (2020). How data analysis can enrich the liberal arts. URL: https://www.economist.com/christmas-specials/2020/12/19/how-data-analysis-can-enrich-the-liberal-arts (Accessed: 22 June 2024).

Grinberg, M. (2018). *Flask web development. Developing web applications with python.* Sebastopol: O'Reilly Media.

Mullen, L.A. (2018–). Computational Historical Thinking. With Applications in R. In *Computational Historical Thinking [Blog/Preprint].* URL: https://dh-r.lincolnmullen.com (Accessed: 22 June 2024).

R Core Team (2021). R. A language and environment for statistical computing [Software]. Wien: R Foundation for Statistical Computing. URL: https://www.R-project.org (Accessed: 22 June 2024).

Van Rossum, G., & Drake Jr, F.L. (1995). Python reference manual. Version 1.2. Amsterdam: Centrum voor Wiskunde en Informatica. URL: https://ir.cwi.nl/pub/5008/05008D.pdf (Accessed: 22 June 2024).

Walsh, M. (2021). Introduction to Cultural Analytics & Python. Version 1. Zenodo. DOI: https://doi.org/10.5281/zenodo.4411250 (Accessed: 22 June 2024).

Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D.C., Augspurger, T., Halchenko, Y., Cole, J.B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., Meyer, K., Miles, M., Ram, Y., Yarkoni, T., Williams, M.L., Evans, C., Fitzgerald, C., Fonnesbeck, B. Ch., Lee, A., & Qalieh, A. (2017). mwaskom/seaborn. Version 0.8.1 [Python-Package]. Zenodo. DOI: https://doi.org/10.5281/zenodo.883859 (Accessed: 22 June 2024).

Wickham, H. (2016). ggplot2. Elegant Graphics for Data Analysis. Cham: Springer [= *Use R!*]. DOI: https://doi.org/10.1007/978-3-319-24277-4 (Accessed: 22 June 2024).