

Getting Yourself Unstuck

Print to PDF

Contents

- 1.1. Introduction
- 1.2. Being a Data Scientist Means Knowing How to Get Yourself Unstuck
- 1.3. The Places to Go For Help, In Order
- 1.4. Method 1: Reading and Understanding Python Errors
- 1.5. Method 2: Using the Built-in Python Documentation
- 1.6. Method 3: Using Google and Other Search Engines
- 1.7. How to Avoid Toxicity in Online Communities
- 1.8. Method 4: Stack Overflow
- 1.9. Method 5: Using a Large-Language Model (a ChatBot) to Generate and Debug Code
- 1.10. Method 6: Interacting with the Global Community of Python Users

Table of Contents

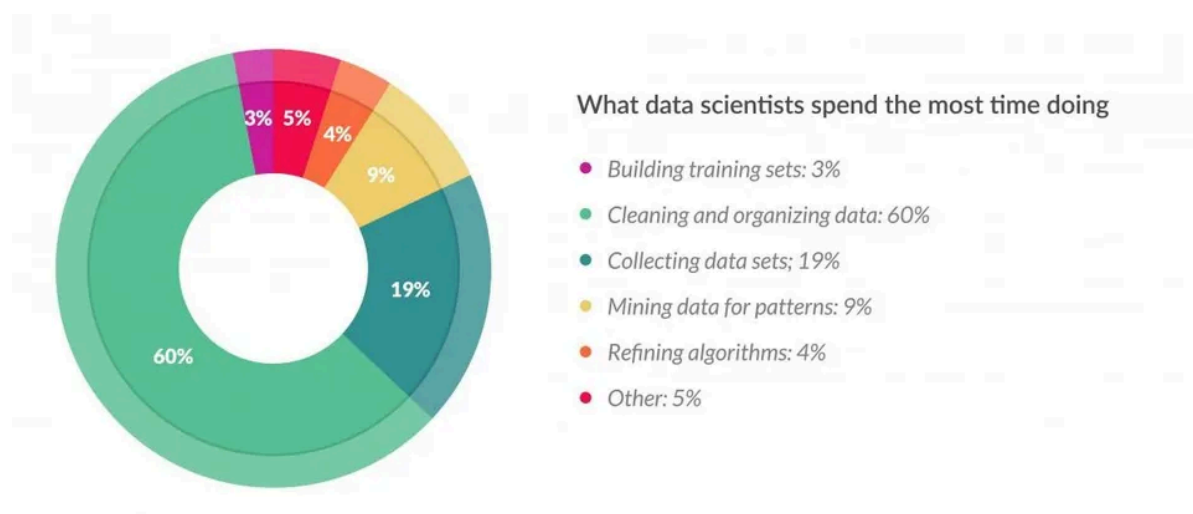
[Getting Yourself Unstuck](#)

- [Introduction](#)
- [Being a Data Scientist Means Knowing How to Get Yourself Unstuck](#)
- [The Places to Go For Help, In Order](#)
- [Method 1: Reading and Understanding Python Errors](#)
- [Method 2: Using the Built-in Python Documentation](#)
 - [Using a Console Window in JupyterLab](#)
 - [How to Call Up and Read a Docstring](#)
- [Method 3: Using Google and Other Search Engines](#)
- [How to Avoid Toxicity in Online Communities](#)
- [Method 4: Stack Overflow](#)
 - [How Stack Overflow Works](#)
 - [Asking a Question on Stack Overflow](#)
- [Method 5: Using a Large-Language Model \(a ChatBot\) to Generate and Debug Code](#)

- [Prompt Engineering for Generating New Code](#)
- [Example: Data from the World Bank](#)
- [Prompt Engineering for Debugging or Explaining Existing Code](#)
- [Method 6: Interacting with the Global Community of Python Users](#)
 - [Python Slack and Discord](#)
 - [Live Chats With Python Users on Freenode](#)
 - [Python Mailing Lists and Newsgroups](#)

1.1. Introduction

Modeling and analytics get the lion's share of attention in the classroom in data science programs, but in the real world, data is almost never ready to be analyzed without a great deal of work to prepare the data first. [This article in Forbes](#) describes a survey of data scientists in which the respondents claim to spend nearly 80% of their time collecting and cleaning data.



Source: ['Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says' by Gil Press](#)

The [Anaconda State of Data Science](#) report in 2022 finds that data scientists “spend about 37.75% of their time on data preparation and cleansing. Beyond preparing and cleaning data, interpreting results remains critical. Data visualization (12.99%) and demonstrating data’s value through reporting and presentation (16.20%) are essential steps toward making data actionable and providing answers to critical questions. Working with models through selection, training, and deployment takes about 26.44% of respondents’ time.”

Whatever the exact numbers, the fact remains that as a data scientist you will be spending a great deal of time collecting and preparing data in advance of an analysis, and a lot of time interpreting and communicating findings. The modeling itself will be a comparatively small part of your work

And that's why traditional academic training for data science can lead students to be underprepared. In many classroom settings, a professor will give the students a cleaned dataset and will task the students with running a predictive model using the data. But working with data that is ready for analysis from the outset is rare; in practice, before this kind of modeling can be done, data scientists have to identify the data they need, get access to the data, and find a way for the data to be loaded into a Python environment. That might mean loading data from files or APIs, or extracting data from webpages, or accessing a database. Then data scientists have to manipulate the data so that it is exactly in the format an analytic model expects, and that often involves mastery of SQL and the `pandas` package to manipulate rows and columns of a data matrix, merge and pivot data tables, handle missing data, recode and combine categorical values, along with a myriad of other tasks. And after finally getting results, yet more work is needed to communicate the output in a clear way that allows that audience to draw fast and accurate conclusions using data visualizations and interactive dashboards.

The phrase **data pipeline** refers to all of the steps needed to go from raw, messy, original data to data that are ready to explore and analyze to the products we share to communicate results to an external audience.

The goal of this book is to make the steps in the data pipeline other than modeling - acquiring, wrangling, and communicating data - easier, faster, less frustrating, and more enjoyable for you. The techniques we will discuss are not the only ways to accomplish a task, but they represent fast and straightforward ways to do the work using Python.

This book is divided into three parts beyond this introductory chapter:

1. **How do we acquire data?**

- From external files with flat, tabular structure (Chapter 2)
- From JSONs, often from APIs (Chapters 3 and 4)
- From web-scraping using `beautifulsoup` (Chapter 5)
- From local or remote access to an SQL or NoSQL database (Chapters 6-7)

2. **How do we clean/wrangle/manipulate data to prepare the data to be analyzed?**

- With SQL queries (Chapter 7)
- With `pandas`, including merging and reshaping dataframes (Chapters 8-9)

3. **How do we explore data and communicate findings?**

- With summary and descriptive statistics tables (Chapter 10)
- With static visualizations using `matplotlib` and `seaborn` (Chapter 11)
- With interactive visualizations using `plotly` (Chapter 12)

However, prior to launching into the data pipeline, we have to talk about the single most important skill for a data scientist: **how to get yourself unstuck**.

1.2. Being a Data Scientist Means Knowing How to Get Yourself Unstuck

Most of you reading this book would describe yourselves as beginners with Python code and the practice of data science. Being a beginner is a wonderful thing. As a beginner, the rate at which you learn new skills is faster than it will be at any other point in your career. And the feeling of accomplishment you will feel by writing working code can and should be profound. But one drawback of being a beginner is the feeling of being judged or viewed as insufficient in comparison to more experienced coders who have much more of this programming language memorized. [Imposter syndrome](#), a persistent feeling of doubt in one's skills and the feeling of being a fraud, is especially common for students who are just starting out in a new field. And it's only natural to look at people who type and execute code at furious speeds and to think of yourself as an imposter, especially when you keep getting errors when you try to execute your code.

But here's the secret: problems, errors, and anomalies in your code are inevitable. And that's true whether you are a beginner or a programmer with decades of experience. It would be accurate to describe a workday of a data scientist as moving from one error to the next to the next. It is simply the nature of using a big programming language like Python that includes so many user-written and open source extensions. So you will feel at times like all you do is generate errors with your code. This feeling is natural, and it is a universal feeling that all data scientists share regardless of their experience level. What sets experienced data scientists apart is their skill in using help resources fix bugs and to get to the right answer, **as quickly as possible**.

As a student, you often have access to an instructor who can answer questions and help you work out the bugs in your code. But as a practitioner you might not have access to someone with the experience and time to help you with your code in a one-on-one way. Some proprietary software sells access to helplines and consulting, but open source projects like Python depend on the community of Python users to provide this support. The more popular an open source software package the bigger the community and the better the resources to help programmers solve problems. And it just so happens that [Python is the most popular programming language in the world](#), and there are some incredible resources available to anyone working with Python. So the most important skill to master to become an advanced data programmer is participating in this community to quickly find answers to the problems that arise as you code.

In this chapter, we will discuss the various methods and resources at your disposal for finding help. Some of the methods provide quick assistance but focus on smaller issues, some methods are slower

general, it is best practice to start with quicker methods and to only move to slower methods if you can't find a solution with a quicker one; so there is an *order* in which you should employ each resource. If you practice using these help methods and memorize the order in which to try using each one, you won't have any trouble squashing all the bugs in your code.



Source: [Ph.D. Comics, "Debugging", by Jorge Cham](#)

1.3. The Places to Go For Help, In Order

There are many resources available to you. I suggest using the following resources in a particular order:

1. Reading and understanding Python errors
2. Python documentation
3. Google
4. Stack Overflow
5. Large language models such as [ChatGPT](#)
6. Places to connect with the larger worldwide community of Python programmers, such as PySlackers and Python Discord, internet relay chat (IRC) rooms, and various Python mailing lists

Several of these resources are ones which are officially recognized and recommended by the Python Software Foundation, the nonprofit organization that maintains the official Python distribution. See <https://www.python.org/community/> for more information about these resources.

Many new programmers use Google and Stack Overflow as their first options for getting help, and recently many people feel the pull to consult a ChatBot right away. I strongly suggest that you do **not** look to Google, Stack Overflow, or a ChatBot before trying to use the official Python built-in documentation or try to understand an error message you receive. The documentation and error messages aren't as intuitive and easy to access as Google or ChatGPT, but the answers that the official documentation will provide are guaranteed to be correct and specific to the functions you are trying to

[Skip to main content](#)

Overflow, and ChatBots, they can be wrong or misleading because there is a lot of information out there that isn't especially helpful. Defaulting to these resources as a first recourse will slow you down a great deal.

1.4. Method 1: Reading and Understanding Python Errors

Usually, when you are stuck, it's because you ran some code that resulted in an error. Reading and *understanding* the error is the single most important and useful way to get yourself unstuck. These errors are not aesthetically pleasing, and they are written in technical language, but the intention of these error messages is to tell you exactly what went wrong. Many new Python users skip reading the errors entirely, which is a shame because the error might indicate exactly the fastest way to solve the problem.

There are two reasons why people skip over reading the errors:

1. The error appears in a big pink box with pea green and cyan text inside of it. It's pretty ugly.
2. The first several lines of code are reserved for a function's **traceback**. The traceback is an attempt to isolate the particular line of code within a larger function that causes the error. However, the traceback is often very technical and not especially useful. After reading the first few lines of the traceback, many people give up on the entire error message.

But the useful part of the error message occurs **at the bottom of the message**. The traceback is useful when developing new functions and debugging original software. But the vast majority of the time, whenever you are using pre-programmed methods and functions, only the bottom of the error message matters.

For example, in chapter 2 we will discuss loading electronic data files. Many things can go wrong. One error we will contend with comes from the following code:

```
from pandas import read_csv
url = "https://raw.githubusercontent.com/jkropko/DS-6001/master/localdata/anes_example_topli
anes = read_csv(url)
```

What this code is supposed to do is not important at the moment. It produces the following error output:


```

ParserError                                Traceback (most recent call last)
/var/folders/8k/ps7r1m2x3_zcd4_dznr8ns1m0000gp/T/ipykernel_11927/2849456955.py in <module>
      1 from pandas import read_csv
      2 url = "https://raw.githubusercontent.com/jkropko/DS-6001/master/localdata/anes_example_toplines.csv"
----> 3 anes = read_csv(url)

/opt/anaconda3/lib/python3.7/site-packages/pandas/util/_decorators.py in wrapper(*args, **kwargs)
    309         stacklevel=stacklevel,
    310     )
--> 311     return func(*args, **kwargs)
    312
    313     return wrapper

/opt/anaconda3/lib/python3.7/site-packages/pandas/io/parsers/readers.py in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    584     kwds.update(kwds_defaults)
    585
--> 586     return _read(filepath_or_buffer, kwds)
    587
    588

/opt/anaconda3/lib/python3.7/site-packages/pandas/io/parsers/readers.py in _read(filepath_or_buffer, kwds)
    486
    487     with parser:
--> 488         return parser.read(nrows)
    489
    490

/opt/anaconda3/lib/python3.7/site-packages/pandas/io/parsers/readers.py in read(self, nrows)
   1045     def read(self, nrows=None):
   1046         nrows = validate_integer("nrows", nrows)
-> 1047         index, columns, col_dict = self._engine.read(nrows)
   1048
   1049         if index is None:

/opt/anaconda3/lib/python3.7/site-packages/pandas/io/parsers/c_parser_wrapper.py in read(self, nrows)
    221     try:
    222         if self.low_memory:
--> 223             chunks = self._reader.read_low_memory(nrows)
    224             # destructive to chunks
    225             data = _concatenate_chunks(chunks)

/opt/anaconda3/lib/python3.7/site-packages/pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read_low_memory()

/opt/anaconda3/lib/python3.7/site-packages/pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()

/opt/anaconda3/lib/python3.7/site-packages/pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._tokenize_rows()

/opt/anaconda3/lib/python3.7/site-packages/pandas/_libs/parsers.pyx in pandas._libs.parsers.raise_parser_error()

ParserError: Error tokenizing data. C error: Expected 1 fields in line 5, saw 168

```

None of this output helps me understand my mistake until I reach the final line that begins

`ParserError`. With a little bit of experience, I can understand this error: it says that it expected the fifth row of the data file ("line 5") to have one column ("Expected 1 field"), but instead it found 168 columns ("saw 168"). That's useful - it tells me that the first four rows of the data file do not contain the data I need, probably because the data authors put text or citation information at the top of the file - but I had to wade through dozens of lines of technicality to arrive at the useful error message.

One way to make reading error messages easier is to turn off the traceback by loading the `sys` library

```
import sys
sys.tracebacklimit = 0
```

Now, when I use the same code, I get a different error output:

```
ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.
```

```
ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.
```

```
ERROR:root:Internal Python error in the inspect module.
Below is the traceback from this internal error.
```

```
Traceback (most recent call last):
pandas.errors.ParserError: Error tokenizing data. C error: Expected 1 fields in line 5,
saw 168
```

I can't say that this error output is pretty or intuitive in and of itself, but relative to the previous error message this output has some nice properties. First, it is shorter (and there's less pink). Second, and most importantly, the useful part of the error message appears much closer to the top of the output, so there's less scrolling involved and the error takes up less space.

To turn the traceback back on, simply type:

```
sys.tracebacklimit = None
```

1.5. Method 2: Using the Built-in Python Documentation

Packages, modules, classes and functions in Python have built-in documentation that you can display directly in the console or in the output of a notebook. These built-in documentations are called **docstrings**. The docstring, and not Google, should be the first place to look for help with specific, pre-built Python code as it conveys guidance directly from the code's authors. In contrast, going to Google first often turns up blogs and posts in which another secondary user describes the code. ChatBots like ChatGPT are built in large part from scraping these secondary uses from various webpages. Either way, these approaches are more prone to misinterpretations and mistakes.

1.5.1. Using a Console Window in JupyterLab

In my experience working with new Python programmers, beginners are very reluctant to use the

[Skip to main content](#)

existing code.

One reason for this reluctance is that calling up a docstring is annoying in a Jupyter notebook. Ideally, you should be working with two windows side by side: one for the notebook or script that contains your code, and one for viewing docstrings. Another reason new users don't use docstrings is that the help documentation isn't accessible until the packages that contain the relevant functions are loaded into the workspace.

The following is a procedure you can use to make viewing docstrings easier with Jupyter notebooks. With a little practice it can become a good habit that saves you a great deal of time in the long-run:

1. Use JupyterLab and open the notebook you are working on.
2. Run the entire notebook (or at least everything up to an error you are trying to fix).
3. Right click somewhere in the notebook and select "New Console for Notebook".
4. Click on the tab for the new console window and drag it to the right so that it occupies the left-side of the screen.

I strongly suggest using JupyterLab (not just Jupyter Notebook), which is available as part of [Anaconda Navigator](#) or as a stand-alone package that can be installed with `pip install jupyterlab`. JupyterLab has a number of features to make it easier to use relative to the regular Jupyter Notebook interface. One of those features is the ability to open consoles with the same kernel as an open notebook, and another feature is the ability to move tabs so that they occupy a portion of the screen to the left or right, on the top or bottom.

The procedure listed above gives you a visible and separate place to view docstrings. It also loads the same kernel as the notebook you are working on, so that if you've imported `pandas`, for example, `pandas` is also loaded into the console so that you can call the docstring for any `pandas` module or function. Keeping this console open will help you more easily integrate calling and reading docstrings into your workflow.

1.5.2. How to Call Up and Read a Docstring

There are many different kinds of code objects in Python that have attached docstrings:

- A package itself, such as `pandas` or `matplotlib`,
- A module: a named subset of the code within a package that focuses on a specific topic. One example is the `matplotlib.pyplot` module which handles all of the functions to display the graphics constructed with other functions within the `matplotlib` package for data visualization,
- A function, either within the base Python code such as `print()`, or within a package or module

- A Python variable that you, the user, has created. Some types of variables have associated **attributes** and **methods**. An attribute is another Python variable that can be extracted from the one you created. For example, if you create a data frame with the `pandas` package and name it `mydataframe`, it has an attribute `mydataframe.columns` that is a list of the names of the columns of your data frame. A method is a function that operates on this data frame. For example, typing `mydataframe.corr()` calculates the correlation between all pairs of columns in the data. (One easy way to tell the difference between attributes and methods is that attributes do not use parentheses, and methods must always use parentheses.)

To display the docstring, replace the word “helpme” with the name of the relevant package, module, function, or variable in one of the following:

- `help(helpme)` displays the entire docstring for the package, module, function, or variable in question. If `helpme` is a user-defined variable, this method displays the attributes and methods available for the variable, if any.
- `helpme?` or `?helpme` displays an abbreviated docstring, as well as the signature (the complete function syntax, including all arguments and their default values). However, if `helpme` is a user-defined variable, this method does not display the attributes and methods available for the variable.
- `helpme??` or `??helpme` is the same as `help(helpme)` but shows the internal code for the function and methods if applicable and possible.

For example, consider the docstring for the `print()` function, which displays results to the notebook or console output. We can call:

```
help(print)
```

Help on built-in function print in module builtins:

```
print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
        string inserted between values, default a space.
    end
        string appended after the last value, default a newline.
    file
        a file-like object (stream); defaults to the current sys.stdout.
    flush
        whether to forcibly flush the stream.
```

Or alternatively:

```
?print
```

And finally:

```
??print
```

In this case, `?print` and `??print` yield the same output because the internal code of `print` is C code, which Python calls but does not display.

Docstrings often have **sections** that convey particular information.

1. The header (only appears with `help(print)`): tells us that the `print` function exists in the `builtins` module (the module that loads automatically when we launch Python)
2. The signature: lists all of the parameters of a function. Each parameter in the signature is set equal to its default value. If the user doesn't specify the parameter in the function call, it's set to the default.
3. The short description: A one-or-two sentence summary of what the function does. For the `print` function, this summary is "Prints the values to a stream, or to sys.stdout by default" which is technical-speak for prints to whatever the output medium happens to be. In a Jupyter notebook that means printing just below a cell of executable code.
4. The parameters: **this section is the most useful for learning how to use a function.** The parameters section lists the parameters, in the order in which they appear in the signature of the function, along with information about each parameter. Each parameter is described in a sentence or two to explain what the parameter does. Sometimes the parameters may be noted as either required or optional in a call to the function, and the docstring might also list the type of each parameter: in this case, the `sep` and `end` parameters are denoted as strings.

Other docstrings might include sections that describe:

1. Returns: describes what the output should look like and contain
2. Attributes and methods for a user-defined Python variable
3. See also: a list of related functions
4. Examples: Examples are meant to be run, not just looked at. Copy-and-paste the examples into your notebook or script, run the code. Then see if you can do more things with the given objects than the examples do.

1.6. Method 3: Using Google and Other Search

Engines

Many of us have a habit of going to Google first whenever we have a coding problem. Please make a concerted effort starting right now to break this habit. If you know the functions you need more information about, using the built-in docstrings first the best habit. The docstrings are what the code's authors provide to guide users. If you know where to look within a docstring, you can find the correct answers to your questions quickly.

Google is a comparatively slow method for getting help. With open-source environments such as Python, there are many ways to do the same thing, and there are way too many presentations on the internet of any one topic to sift through efficiently. Many of the resources you find this way come from a field or a point of view that differs from your own, leading you to have to work hard to translate the language that explains the method. And some of the information is out of date or simply wrong.

Use Google or another search engine if

- you don't know the functions you need to do a task,
- you don't have a textbook or set of notes that you trust for guidance,
- or if the built-in docstring doesn't give you the information you need.

The format for Google search terms that usually works best for me is

```
python (a specific package if you know which one to use) (the function/method you want help
```

Starting the search with "python" usually narrows the search to Python-specific documentation, message-boards, and blogs. Specifying the package is important when multiple packages can do some version of the same task: for example, if you are trying to use `plotly` to create a bar plot, you will have to sift through results that use alternative packages `matplotlib` and `seaborn` unless you state `plotly` directly. Typing the relevant function or task third narrows those results to more relevant posts. For example, suppose I try to solve the issue that led to the error above when using `pd.read_csv()`. I determined that the problem occurred because the first four rows of the data file are taken up by an unnecessary header placed there by the data authors. I would like to know how to skip these lines. If I turn to the docstring first by typing `?pd.read_csv`, I see that there is a parameter `skiprows` with this description:

```
skiprows : list-like, int or callable, optional
    Line numbers to skip (0-indexed) or number of lines to skip (int)
    at the start of the file.
```

So I can just include `skiprows=4` as a parameter within `pd.read_csv()`. But if I turn to Google, I can issue the following search:

```
python pd.read_csv how to skip rows at the top
```

Google will often take you to Stack Overflow (<https://stackoverflow.com/>) before other websites with coding information. In this case, the first hit is [this Stack Overflow post](#) which shows me the `skiprows` parameter, but tailors the advice to this particular individual's use case which requires keeping the first row and skipping rows 2 through 11, something not relevant to me in this case. I can get to the right answer this way, but it is slower than going directly to the docstring.

Stack Overflow is by far the most widely used and informative repository of coding help and knowledge on the internet. But before we discuss Stack Overflow in more detail, there are other useful and credible websites that often appear when using a Google search to solve a coding problem:

- More detailed documentation for specific Python modules, such as:
 - <https://scikit-learn.org/>
 - <https://pandas.pydata.org/>
 - <https://matplotlib.org/>
- High-quality blogs:
 - <https://towardsdatascience.com/>
 - <https://medium.com/>
- Free content from tutorial websites with a lot more paid content:
 - <https://realpython.com/>
 - <https://www.geeksforgeeks.org>
 - <https://www.w3schools.com>
 - <https://www.datacamp.com>
 - <https://www.dataquest.io/>

There are also many, many more resources and new ones are created all the time. Find the resources you like best: going directly to the most credible links will save you a great deal of time in this effort to get yourself unstuck.

1.7. How to Avoid Toxicity in Online Communities

The next options involve becoming a **responsible, respectful** member of the worldwide community of Python users. Open-source platforms like Python and R depend on a community of volunteers who

develop and maintain the tools that we use. All of these people are doing volunteer work for the common good, and that's a beautiful thing.

In addition to Python itself, Stack Overflow, Slack, Discord, the Freenode IRC, and mailing lists are online communities for Python users. But, like any online community, there's the potential for a toxic culture to destroy everything.

What is a toxic culture? How do you know one when you see one?

Toxic cultures are more likely when

- members are allowed to be anonymous (see [Lapidot-Lefler and Barak 2011](#))
- and members up-vote and down-vote and comment on each other's contributions (see [Massanari 2015](#)).

A culture can be either actively or passively toxic. Actively toxic communities are easy to identify. They encourage and are characterized by overt sexism, racism, bigotry, and calls for violence or other aggression against individuals. Most of the toxicity you will encounter in online programming and data science communities is not actively, but passively toxic. Passive toxicity is characterized by **gate-keeping**: Subtle behaviors that discourage people with less experience, or with some social anxiety, from participating.

Stack overflow, IRCs, and mailing lists are notorious for passive toxic behavior. Passive toxicity is a bigger problem for us than active toxicity because actively toxic behavior is usually explicitly banned by codes of conduct, and individuals are often unaware of when they are acting in a passively toxic way.

Examples of passive toxic behavior:

- **Condescending language**: often people who are trying to earnestly answer a question unthinkingly use language that makes someone feel dumb for asking the question. If you are answering a question on an online programming forum, avoid using words such as "obviously", "clearly", "actually", "just", or "that should be easy".
- **Shaming**: making an implication, even a very slight one, on someone's intelligence or work ethic instead of giving people the benefit of assuming they've tried other avenues before posting. Some subtle examples include saying "google it", or "read the manual", implying that the questioner must not have tried these methods first. Sometimes people shame others more aggressively on these forums by saying things like "Learn to debug your own code" and "If you don't get this, you have no business being a data scientist", but hopefully that's becoming more rare.
- **Downvotes without explanation**: this can be both confusing and very upsetting to anyone, especially to people with less experience.
- **Virtue signaling**: implying that people are superior/inferior because of the language, software, or

so many memes:



Source: ['Statswars' by Kieran Healy](#)

- **Authoritarianism:** Abusing people for failing to follow all of a community's rules for asking questions. For example, a comment that entirely ignores the content of the question but comments "all questions must provide an example," or editing a user's post to remove where they wrote "Hi everyone" and "thanks."
- **Overzealous curation:** Being very quick to tag a question as a "duplicate" without checking to see nuanced ways in which the question comes from a new situation.

The result of passive toxicity is that many potential community members choose not to participate in the community because their initial experiences made them feel ashamed, confused, or belittled. In addition, other potential new members observe these negative interactions involving other members, and choose to disengage.

Passive toxicity shrinks the community and makes it more homogeneous. Across society, small, homogeneous communities are much more likely to exclude or discriminate against people based on sex, race, class, language and other factors. And that leads to many ethical problems.

Remember, most of time when someone is exhibiting passively toxic behavior on a forum, they are well-intentioned but are not thinking about how their responses will be perceived. Very few people deliberately contribute to a toxic online culture, but many people do contribute by failing to empathize with the questioner and by not choosing words carefully.

Please keep the behaviors described here in mind when you engage in online communities, and avoid

submitting a response, and make the edits that you think would lead to a better online interaction. Don't be afraid to call out other people who behave in these ways. And if you are yourself the person who posts, be aware that these behaviors still happen much too frequently, but know that despite this, most people on these forums mean well.

1.8. Method 4: Stack Overflow

[Stack Overflow](#) is the most popular and most useful website for help with programming of all kinds. Google searching a Python problem will usually lead to a Stack Overflow post on the same issue. Python is now the most frequent tag for posts on Stack Overflow, and shown in this video:

```
from IPython.display import IFrame
IFrame(src="https://www.youtube.com/embed/cKzP61Gjf00", width="560", height="315")
```

Most Popular Programming Languages on Stack Overflow ...



Finding a Stack Overflow post that's relevant to your problem can give you both the code and underlying intuition to solve your problem. Or maybe not! Small differences in the situation can make the solution irrelevant to you. Be cautious and don't treat a Stack Overflow post as automatically a definitive answer.

1.8.1. How Stack Overflow Works

1. Someone asks a question
2. Other people comment on and provide answers to the question
3. The person who asked the question replies to the comments, and can choose an answer to mark as "accepted"

4. People with reputation scores higher than 15 can upvote or downvote questions and answers.
5. Reputation points are awarded for asking questions or giving answers that other people upvote, or for having an answer accepted. Points are taken away for downvotes or spam or offensive posts.

Going for reputation is an entirely optional activity. If you don't want to worry about it, don't.

1.8.2. Asking a Question on Stack Overflow

Okay, so you're stuck. You've combed through the Python documentation, Google, and old Stack Overflow posts, but you haven't found a solution. It's time to consider writing a new question on Stack Overflow.



Source: ['10 Ways to Learn WordPress Hooks', RachieVee: Rachel's Blog.](#)

This can be frightening. A lot of the time, people answering questions on Stack Overflow can be, well ... huge assholes that cause [real suffering](#). You might choose to avoid posting to Stack Overflow, so as not to support a website that has harbored abuse. That's completely fair. If you do post to Stack Overflow, you are likely to get some very useful responses if you follow some guidelines. There's [a strategy for getting good responses](#). You are more likely to get a good response if you follow these steps:

1. Search Stack Overflow and Google to see if the question has already been answered.

Commenters dislike if the same question is asked repeatedly. This [poor guy](#) got roasted for posing a "duplicate" question. (An aside: *Why?* It's not like Stack Overflow is running out of space on their website. There's an idea that Stack Overflow should be a central repository of knowledge. That means there should be one canonical answer to one question. But people often take this much too far. There are kinder ways to point to an existing answer.) So spend a significant amount of time digging through the internet. If there's something similar, but not quite what you need, you

2. **Write a good title for your post.** A good title is specific about the problem, and also succinct:

- Bad: Problem with matplotlib (not specific)
- Also Bad: How do I place the labels of cars in a scatterplot of the weight and miles per gallon of cars onto the points in the scatterplot using matplotlib 3.3.1 on Python 3.7.4 on Mac OSX 10.14.5? (not succinct)
- Good: How to place labels on top of points in a matplotlib scatterplot?

3. **Start the post with a paragraph describing the problem in more detail.** Some good things to include in this paragraph:

- The context of the problem: how did you come across the problem? Describe the overall goal, not the just the buggy step.
- What you've already tried to solve the problem, and what happened.
- What is the expected output? What do you see instead?
- You can write the version of Python you are using, the version of the modules, and the operating system on your computer, in case the problem turns out to be specific to an old version of one of those things.

4. If possible, **include code that reproduces the problem.** The code should not simply be the code in your script that isn't working. It needs to be able to work on someone else's computer. That means the code should not depend on any specific data files, and should not contain file addresses that refer to a location on your computer. If possible, only use modules that are easy to get. If the code needs to run on data, can you use something pre-loaded in Python that everyone can access? (There are example datasets included with `scikit-learn`, for example.) Make the code as short and stripped down as possible while still producing the behavior or error that needs to be fixed, and use comments to help people understand the code more quickly.

A few additional things to keep in mind:

- Be courteous and respectful. Respond to and thank everyone who comments.
- Post a follow-up once the problem is solved so that people who come across this page in the future with the same problem know the solution.
- Don't ask people to write code for you. It's better to request help with code you provide.
- Don't claim you found a bug in Python or in a module. It's a bit rude to the people who programmed the code (who don't get paid).
- Don't ask about homework problems. ([Here's an example](#) of someone getting called out on this.)

1.9. Method 5: Using a Large-Language Model (a ChatBot) to Generate and Debug Code

Since the release of Open AI's [ChatGPT](#) in March 2023, generative AI and especially chatbots have occupied a great deal of the attention of data scientists in industry and academia. There is far too much to describe about these chatbots in terms of their construction, use, and ethics to fit as a concise discussion in a text like this one. But the main idea you should keep in mind about chatbots is that they are both a necessary part of a data scientist's toolkit, and that they are not any more intelligent than what can be found with a careful search on Google or Stack Overflow. Chatbots are trained on massive amounts of data, largely scrapped from the internet, and with respect to coding the major sources of data are Stack Overflow and GitHub. It's useful to think of a chatbot as a more efficient way to sift through the information that already exists on those websites.

Schools are grappling with the question of whether using a chatbot constitutes cheating. But professional software engineering and data science has largely adopted the view that chatbots are fair to use. That said, there is an approach to using chatbots for coding effectively, and knowing the right way to prompt a chatbot to solve coding problems is itself an important skill. So if you are going to use a chatbot, use it correctly. The following discussion outlines some ways to access a chatbot as part of your own workflow and some strategies for writing prompts that can generate the code you need or debug code.

1.9.1. How to Access a ChatBot for Coding

The most common way that people are interacting with a chatbot currently is the OpenAI website for chatting with ChatGPT: <https://chat.openai.com/>. This platform is also free. But there are alternative methods, some of which cost money, that are better suited for an efficient data science workflow. One of the most popular chatbot interfaces in industry is [GitHub Copilot](#), which integrates with Visual Studio Code. A primary advantage of GitHub Copilot is that it can access your kernel to see and work with the objects you've already created, and it can read the code you've already written. So for example, if you load a dataset into your environment as a `pandas` dataframe and call it `mydata`, GitHub Copilot will already know what `mydata` contains without having to specify it within your prompt. It can save time once you are comfortable using it. There is a cost to use GitHub Copilot, however, and depending on your use case the cost may or may not be worth it.

One important point about using chatbots to help you code: it is not a fast method. Generally, getting a useful result from a chatbot requires having a "conversation" with the chatbot to narrow in on the desired product. That also involves taking time to study and run the code that the chatbot produces to see what is correct and what is still lacking. The time needed to build to a final output and assess it

prepare data, using the chatbot only for particular tasks that slow or stop the progress you can make on your own.

Chatbots respond to user-supplied prompts, which can be anything from “write me a description of different types of fungi in the style of a Jay-Z song” to specific tasks related to coding and data science. As part of a data science workflow, you might use a chatbot to help you write original code to accomplish a specific task, or to help you debug code you’ve already written that is generating an error or an unexpected behavior.

1.9.2. Prompt Engineering for Generating New Code

The dream is that we will be able offload the heavy-lift of data wrangling entirely, so that if we simply prompt the chatbot with “please clean this data” it will instantaneously generate code that gives us the clean data we want. But that’s not how chatbots work at present.

We can use a chatbot to do all the work prior to analysis, but it tends to be a slow process, involving a long conversation with a chatbot to flush out the details of what “clean data” means for your specific application. Think of this process as similar to supervising a hard-working but inexperienced intern at work. You must provide clear enough instructions to enable the intern to give you what you want and you need to check the intern’s work through every iteration of the project. That’s not a fast method! So please think of chatbots as a tool, not a cheat-code, and specifically as a tool that is slower and more difficult to use than knowing the code yourself, reading documentation and error messages, and using Google and Stack Overflow.

The quickest and most effective method for using a chatbot for data wrangling is to engineer a prompt using these general principles: **Data, Packages, Chunking, Streamline, Knowledge** (or to remember, “Don’t play chess with [Stephen King](#)”).

- **Data:** Provide “just enough” data within the prompt to get the chatbot to write code that is more specific to the data at hand. That often means copy-and-pasting just a few rows in CSV or JSON format, and including it in the prompt while directly writing how you are providing data.
- **Packages:** Directly ask for specific coding languages and packages if you know the right ones to use.
- **Chunking:** Ask for elaboration and chunk the task. Chatbots like ChatGPT work in reference to previous queries, so it is appropriate to issue a second prompt that references the first to add complexity to the task.
- **Streamline:** Chatbots are not guaranteed to generate efficient code in terms of length, even if the code works. Sometimes one line of code using a method in a package like `pandas` accomplishes what 50 lines of code can do in base python. Using the complicated version of the code makes

to a chatbot to ask for code that accomplishes the same task with in a more streamlined way:

“Provide code that works the same as the previous response but uses fewer lines.”

- **Knowledge:** The main challenge, however, is knowing what to ask for and understanding whether or not the provided code genuinely accomplishes the task. If you aren’t specific with how you ask the question and don’t supply data, the chatbot will usually provide code that looks sort-of right but can’t be run to confirm whether it works as needed, and if you aren’t sure what the output ought to look like it will be impossible to know whether the code is correct or not.

1.9.3. Example: Data from the World Bank

Let’s take as an example the task of wrangling data from the [World Bank’s World Development Indicators \(WDI\)](#). The following data contain information from every country tracked by the World Bank on CO2 emissions, gross domestic product per capita, and population for every year from 2012 through 2022.

```
import pandas as pd
worldbank = pd.read_csv('worldbank_data.csv')
worldbank.head(10)
```

	Country Name	Country Code	Series Name	Series Code	2012 [YR2012]	2013 [YR2013]
0	Afghanistan	AFG	CO2 emissions (kt)	EN.ATM.CO2E.KT	10208.13	9402.05
1	Afghanistan	AFG	GDP per capita (constant 2015 US\$)	NY.GDP.PCAP.KD	570.676064337005	582.103978902877
2	Afghanistan	AFG	Population, total	SP.POP.TOTL	30466479	31541209
3	Albania	ALB	CO2 emissions (kt)	EN.ATM.CO2E.KT	4541.8	4795.4
4	Albania	ALB	GDP per capita (constant 2015 US\$)	NY.GDP.PCAP.KD	3736.34007957648	3780.69919254096
5	Albania	ALB	Population, total	SP.POP.TOTL	2900401	2895092
6	Algeria	DZA	CO2 emissions (kt)	EN.ATM.CO2E.KT	134934.2	139024.1
7	Algeria	DZA	GDP per capita (constant 2015 US\$)	NY.GDP.PCAP.KD	4025.64148367191	4057.76480695524
8	Algeria	DZA	Population, total	SP.POP.TOTL	37260563	38000626
9	American Samoa	ASM	CO2 emissions (kt)	EN.ATM.CO2E.KT

The particular way in which this dataset has been formatted by the World Bank places the three features in separate rows, and the years in separate columns. In order to conduct an analysis (regressing CO2 emissions on GDP per capita and population, for example), I need years in the rows and the features in the columns. I also want to drop Series Code, to give the features in Series Name better code-friendly column names without parentheses or dollar signs, and I want to remove the

bracketted labels such as [YR 2012] from the years. These tasks are just a few of the steps necessary to create “clean data” for a statistical model.

I can use the `pandas` package to do this quickly. But for this example, I will use chatbot prompt engineering exclusively to try to accomplish the same result. I use the free ChatGPT website.

First, I need to supply data. ChatGPT limits prompts to [4096 characters](#), and that includes data included within the prompt. So the best practice is to narrow the data down to a small size that still illustrates the format of the data. Usually the first 5 rows of a dataframe, obtained with the `df.head()` method, is sufficient. The data still need to be pasted into the prompt in some text-based way, and comma-separated values (CSV) works well. A `pandas` dataframe can be converted to CSV-style text with the `df.to_csv(index=False)` method. To get a CSV version of the first 5 rows of the World Bank data, we can type

```
worldbank.head(5).to_csv(index=False)
```

```
'Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014]
```

We copy the data and use it in the following prompt:

Prompt:

Using the following data in CSV format:

```
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
```

Write code using python and pandas that represents the year columns as rows, represents the values of “Series Name” as columns, and drops “Series Code”.

Response:

Certainly! You can achieve this transformation using the pandas library in Python. Here’s an example code:

```
import pandas as pd
from io import StringIO
```

```
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
Afghanistan,AFG,CO2 emissions (kt),EN.ATM.CO2E.KT,10208.13,9402.05,9281.34,10057.59,9294.93,
Afghanistan,AFG,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,570.676064337005,582.10397,
Afghanistan,AFG,"Population, total",SP.POP.TOTL,30466479,31541209,32716210,33753499,34636207
Albania,ALB,CO2 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4,
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254,
"""

# Read CSV data into a pandas DataFrame
df = pd.read_csv(StringIO(csv_data))

# Melt the DataFrame to transform year columns as rows
df_melted = pd.melt(df, id_vars=['Country Name', 'Country Code', 'Series Name'], var_name='Year')

# Pivot the DataFrame to represent Series Name as columns
df_pivoted = df_melted.pivot_table(index=['Country Name', 'Country Code'], columns='Series Name',
                                     values='Year')

# Drop the 'Series Code' column
df_pivoted = df_pivoted.drop(columns='Series Code')

# Display the result
print(df_pivoted)
```

This code reads the CSV data, melts the DataFrame to transform the year columns as rows, pivots the DataFrame to represent “Series Name” as columns, and finally drops the “Series Code” column.

Although the code supplied by the chatbot looks like it should work, I ran the code and it fails with an error that reads: “TypeError: Could not convert string ‘.’ to numeric”. Here I have to use my knowledge of the data. Further down the rows, some values use “.” as a missing value. The error indicates that these symbols are not recognized by python as a missing value and cause problems when converting columns to rows. So I issue a new prompt.

Prompt:

Revise the previous code to replace all occurrences of “.” with missing values prior to melting or pivoting the dataframe.

Response:

Certainly! You can use the `replace` function to replace all occurrences of “.” with missing values (NaN) before melting or pivoting the DataFrame. Here’s the revised code:

```
import pandas as pd
from io import StringIO
```

```

Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
Afghanistan,AFG,CO2 emissions (kt),EN.ATM.CO2E.KT,10208.13,9402.05,9281.34,10057.59,9294.93,
Afghanistan,AFG,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,570.676064337005,582.10397,
Afghanistan,AFG,"Population, total",SP.POP.TOTL,30466479,31541209,32716210,33753499,34636207
Albania,ALB,CO2 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4,
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254,
"""

# Read CSV data into a pandas DataFrame
df = pd.read_csv(StringIO(csv_data))

# Replace all occurrences of ".." with NaN
df.replace("..", pd.NA, inplace=True)

# Melt the DataFrame to transform year columns as rows
df_melted = pd.melt(df, id_vars=['Country Name', 'Country Code', 'Series Name'], var_name='Year')

# Pivot the DataFrame to represent Series Name as columns
df_pivoted = df_melted.pivot_table(index=['Country Name', 'Country Code', 'Year'], columns='Series Name')

# Display the result
print(df_pivoted)

```

This code first replaces all occurrences of “.” with NaN using the `replace` function before performing the melting and pivoting operations.

This code still produces an error that some values in the data cannot be converted to numeric format. So I try again by asking ChatGPT to convert all the columns that begin with a year to numeric after replacing the missing values.

Prompt:

Revise the previous code to convert all columns that begin with “20” to numeric prior to melting or pivoting the dataframe.

Response:

Certainly! You can convert all columns that begin with “20” to numeric before melting or pivoting the DataFrame. Here’s the revised code:

```

import pandas as pd
from io import StringIO

# Input CSV data
csv_data = '''
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],

```

```
Afghanistan,AFG,"Population, total",SP.POP.TOTL,30466479,31541209,32716210,33753499,34636207
Albania,ALB,C02 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254
...
```

```
# Read CSV data into a DataFrame
df = pd.read_csv(StringIO(csv_data))

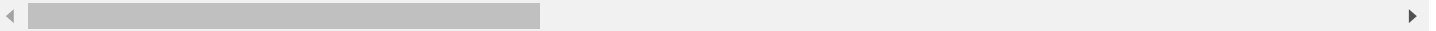
# Replace "." with missing values
df.replace('..', pd.NA, inplace=True)

# Convert columns that begin with "20" to numeric
df.iloc[:, 4:] = df.iloc[:, 4:].apply(pd.to_numeric, errors='coerce')

# Pivot the DataFrame
df_pivoted = df.melt(id_vars=['Country Name', 'Country Code', 'Series Name'], var_name='Year')

# Drop 'Series Code' column
df_pivoted = df_pivoted.drop(columns=['Series Name'])

# Display the result
df_pivoted
```



	Country Name	Country Code	Year		Value
0	Afghanistan	AFG	Series Code	EN.ATM.CO2E.KT	
1	Afghanistan	AFG	Series Code	NY.GDP.PCAP.KD	
2	Afghanistan	AFG	Series Code	SP.POP.TOTL	
3	Albania	ALB	Series Code	EN.ATM.CO2E.KT	
4	Albania	ALB	Series Code	NY.GDP.PCAP.KD	
5	Afghanistan	AFG	2012 [YR2012]		10208.13
6	Afghanistan	AFG	2012 [YR2012]		570.676064
7	Afghanistan	AFG	2012 [YR2012]		30466479.0
8	Albania	ALB	2012 [YR2012]		4541.8
9	Albania	ALB	2012 [YR2012]		3736.34008
10	Afghanistan	AFG	2013 [YR2013]		9402.05
11	Afghanistan	AFG	2013 [YR2013]		582.103979
12	Afghanistan	AFG	2013 [YR2013]		31541209.0
13	Albania	ALB	2013 [YR2013]		4795.4
14	Albania	ALB	2013 [YR2013]		3780.699193
15	Afghanistan	AFG	2014 [YR2014]		9281.34
16	Afghanistan	AFG	2014 [YR2014]		576.48782
17	Afghanistan	AFG	2014 [YR2014]		32716210.0
18	Albania	ALB	2014 [YR2014]		5188.0
19	Albania	ALB	2014 [YR2014]		3855.760744
20	Afghanistan	AFG	2015 [YR2015]		10057.59
21	Afghanistan	AFG	2015 [YR2015]		566.881133
22	Afghanistan	AFG	2015 [YR2015]		33753499.0
23	Albania	ALB	2015 [YR2015]		4797.0
24	Albania	ALB	2015 [YR2015]		3952.803574
25	Afghanistan	AFG	2016 [YR2016]		9294.93
26	Afghanistan	AFG	2016 [YR2016]		564.920844

	Country Name	Country Code	Year	Value
27	Afghanistan	AFG	2016 [YR2016]	34636207.0
28	Albania	ALB	2016 [YR2016]	4573.2
29	Albania	ALB	2016 [YR2016]	4090.372728
30	Afghanistan	AFG	2017 [YR2017]	10022.78
31	Afghanistan	AFG	2017 [YR2017]	563.488239
32	Afghanistan	AFG	2017 [YR2017]	35643418.0
33	Albania	ALB	2017 [YR2017]	5403.7
34	Albania	ALB	2017 [YR2017]	4249.820049
35	Afghanistan	AFG	2018 [YR2018]	10972.38
36	Afghanistan	AFG	2018 [YR2018]	553.973309
37	Afghanistan	AFG	2018 [YR2018]	36686784.0
38	Albania	ALB	2018 [YR2018]	5316.1
39	Albania	ALB	2018 [YR2018]	4431.555595
40	Afghanistan	AFG	2019 [YR2019]	11238.83
41	Afghanistan	AFG	2019 [YR2019]	559.140957
42	Afghanistan	AFG	2019 [YR2019]	37769499.0
43	Albania	ALB	2019 [YR2019]	4993.3
44	Albania	ALB	2019 [YR2019]	4543.38771
45	Afghanistan	AFG	2020 [YR2020]	8709.47
46	Afghanistan	AFG	2020 [YR2020]	529.144912
47	Afghanistan	AFG	2020 [YR2020]	38972230.0
48	Albania	ALB	2020 [YR2020]	4383.2
49	Albania	ALB	2020 [YR2020]	4418.660874
50	Afghanistan	AFG	2021 [YR2021]	NaN
51	Afghanistan	AFG	2021 [YR2021]	407.616507
52	Afghanistan	AFG	2021 [YR2021]	40099462.0
53	Albania	ALB	2021 [YR2021]	NaN

	Country Name	Country Code	Year	Value
54	Albania	ALB	2021 [YR2021]	4857.111942
55	Afghanistan	AFG	2022 [YR2022]	NaN
56	Afghanistan	AFG	2022 [YR2022]	NaN
57	Afghanistan	AFG	2022 [YR2022]	41128771.0
58	Albania	ALB	2022 [YR2022]	NaN
59	Albania	ALB	2022 [YR2022]	5155.29086

This code includes a line to convert all columns that begin with "20" to numeric using `df.iloc[:, 4:] = df.iloc[:, 4:].apply(pd.to_numeric, errors='coerce')` before pivoting the DataFrame.

The previous code runs without error, but it does not generate the correct data. It drops "Series Name" instead of "Series Code", and it did not place the different values of "Series Name" in new columns. Let's see if we can get ChatGPT to fix these mistakes.

Prompt:

Alter the previous code by dropping "Series Code" instead of "Series Name", and by representing the values of "Series Name" in separate columns.

Response:

Certainly! Here's the revised code to drop "Series Code" instead of "Series Name" and represent the values of "Series Name" in separate columns:

```
import pandas as pd
from io import StringIO

# Input CSV data
csv_data = '''
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
Afghanistan,AFG,CO2 emissions (kt),EN.ATM.CO2E.KT,10208.13,9402.05,9281.34,10057.59,9294.93,
Afghanistan,AFG,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,570.676064337005,582.10397
Afghanistan,AFG,"Population, total",SP.POP.TOTL,30466479,31541209,32716210,33753499,34636207
Albania,ALB,CO2 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254
'''

# Read CSV data into a DataFrame
df = pd.read_csv(StringIO(csv_data))
```



```
df.replace('..', pd.NA, inplace=True)

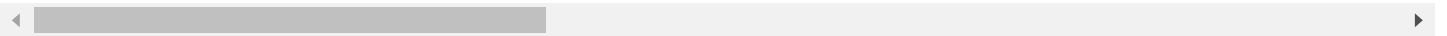
# Convert columns that begin with "20" to numeric
df.iloc[:, 4:] = df.iloc[:, 4:].apply(pd.to_numeric, errors='coerce')

# Pivot the DataFrame
df_pivoted = df.melt(id_vars=['Country Name', 'Country Code', 'Series Name', 'Series Code'],

# Drop 'Series Code' column
df_pivoted = df_pivoted.drop(columns=['Series Code'])

# Pivot the DataFrame again to have separate columns for each 'Series Name'
df_pivoted = df_pivoted.pivot_table(index=['Country Name', 'Country Code', 'Year'], columns=

# Display the result
df_pivoted
```



Series Name	Country Name	Country Code	Year	CO2 emissions (kt)	GDP per capita (constant 2015 US\$)	Population, total
0	Afghanistan	AFG	2012 [YR2012]	10208.13	570.676064	30466479.0
1	Afghanistan	AFG	2013 [YR2013]	9402.05	582.103979	31541209.0
2	Afghanistan	AFG	2014 [YR2014]	9281.34	576.48782	32716210.0
3	Afghanistan	AFG	2015 [YR2015]	10057.59	566.881133	33753499.0
4	Afghanistan	AFG	2016 [YR2016]	9294.93	564.920844	34636207.0
5	Afghanistan	AFG	2017 [YR2017]	10022.78	563.488239	35643418.0
6	Afghanistan	AFG	2018 [YR2018]	10972.38	553.973309	36686784.0
7	Afghanistan	AFG	2019 [YR2019]	11238.83	559.140957	37769499.0
8	Afghanistan	AFG	2020 [YR2020]	8709.47	529.144912	38972230.0
9	Afghanistan	AFG	2021 [YR2021]	NaN	407.616507	40099462.0
10	Afghanistan	AFG	2022 [YR2022]	NaN	NaN	41128771.0
11	Albania	ALB	2012 [YR2012]	4541.8	3736.34008	NaN
12	Albania	ALB	2013 [YR2013]	4795.4	3780.699193	NaN
13	Albania	ALB	2014 [YR2014]	5188.0	3855.760744	NaN
14	Albania	ALB	2015 [YR2015]	4797.0	3952.803574	NaN
15	Albania	ALB	2016 [YR2016]	4573.2	4090.372728	NaN

Series Name	Country Name	Country Code	Year	CO2 emissions (kt)	GDP per capita (constant 2015 US\$)	Population, total
			[YR2017]			
17	Albania	ALB	2018 [YR2018]	5316.1	4431.555595	NaN
18	Albania	ALB	2019 [YR2019]	4993.3	4543.38771	NaN
19	Albania	ALB	2020 [YR2020]	4383.2	4418.660874	NaN
20	Albania	ALB	2021 [YR2021]	NaN	4857.111942	NaN
21	Albania	ALB	2022 [YR2022]	NaN	5155.29086	NaN

This code first melts the DataFrame and then pivots it again to have separate columns for each 'Series Name'. The 'Series Code' column is dropped before melting.

Now the data look very close to what we need. There are just a few more details to polish: I want to drop the labels like [YR 2012] next to each year, and I want to rename the columns. If I try to get ChatGPT to include these tasks in the prior code, it tends to forget to preserve the structure of the data we tried hard to build. A better approach would be to reset the conversation by passing a CSV-formatted string from the new data to ChatGPT.

In short, the process of using the chatbot exclusively for data wrangling is slow and prone to error. But chatbots will improve in their coding ability in the future. Effective prompting is a useful skill worth practicing.

[1.9.4. Prompt Engineering for Debugging or Explaining Existing Code](#)

Another use for chatbots in a data wrangling workflow is to debug or explain code. For debugging, the key is to isolate the code that generates the error and to supply the error message. That said, the chatbot's interpretation may be right or wrong for your specific case, the same as any post on Stack Overflow. The purpose of using a chatbot to work through errors is not to fix the code, exactly, but rather to shed some light on what is causing the error so that you can fix it yourself.

For example, the chatbot's response to the first prompt we issued in the previous section yielded an

Prompt:

The following python code:

```
import pandas as pd
from io import StringIO

# Input CSV data
csv_data = """
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
Afghanistan,AFG,CO2 emissions (kt),EN.ATM.CO2E.KT,10208.13,9402.05,9281.34,10057.59,9294.93,
Afghanistan,AFG,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,570.676064337005,582.10397,
Afghanistan,AFG,"Population, total",SP.POP.TOTL,30466479,31541209,32716210,33753499,34636207
Albania,ALB,CO2 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254
"""

# Read CSV data into a pandas DataFrame
df = pd.read_csv(StringIO(csv_data))

# Melt the DataFrame to transform year columns as rows
df_melted = pd.melt(df, id_vars=['Country Name', 'Country Code', 'Series Name'], var_name='Y

# Pivot the DataFrame to represent Series Name as columns
df_pivoted = df_melted.pivot_table(index=['Country Name', 'Country Code', 'Year'], columns='

# Drop the 'Series Code' column
df_pivoted = df_pivoted.drop(columns='Series Code')

# Display the result
print(df_pivoted)
```

yields an error “TypeError: Could not convert string ‘.’ to numeric”. What is the problem and how do we fix it?

Response:

The error you are encountering is due to the presence of ‘.’ in your CSV data, which represents missing or incomplete data. When attempting to convert the data to numeric values, these ‘.’ values cause a TypeError.

To fix this issue, you can replace the ‘.’ values with a suitable representation for missing data, such as `NaN` (Not a Number). You can modify the code to handle this during the reading of the CSV data by specifying the `na_values` parameter in the `read_csv` function:

```
import pandas as pd
```

```

# Input CSV data
csv_data = """
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
Afghanistan,AFG,CO2 emissions (kt),EN.ATM.CO2E.KT,10208.13,9402.05,9281.34,10057.59,9294.93,
Afghanistan,AFG,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,570.676064337005,582.10397,
Afghanistan,AFG,"Population, total",SP.POP.TOTL,30466479,31541209,32716210,33753499,34636207
Albania,ALB,CO2 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254
"""

# Read CSV data into a pandas DataFrame and replace '..' with NaN
df = pd.read_csv(StringIO(csv_data), na_values='..')

# Melt the DataFrame to transform year columns as rows
df_melted = pd.melt(df, id_vars=['Country Name', 'Country Code', 'Series Name'], var_name='Y

# Pivot the DataFrame to represent Series Name as columns
df_pivoted = df_melted.pivot_table(index=['Country Name', 'Country Code', 'Year'], columns='

# Drop the 'Series Code' column
df_pivoted = df_pivoted.drop(columns='Series Code')

# Display the result
print(df_pivoted)

```

This modification should resolve the TypeError by treating the '..' values as NaN during the conversion process

Here ChatGPT identified the issue and suggested some code that could fix it. It so happens that the code ChatGPT supplies still results in a (different) error, but at least one hurdle has been cleared.

A second use for a chatbot is to provide an explanation for what happens line-by-line within a chunk of code. This ability is especially useful for a coder who already knows what they are doing, but needs some help tracing what someone else's code is doing. It can be a great way to learn new coding skills. For example, I can ask ChatGPT to explain the code that resulted in the best version of the data in the previous section.

Prompt:

Please explain the following Python code line by line:

```

import pandas as pd
from io import StringIO

# Input CSV data
csv_data = '''
Country Name,Country Code,Series Name,Series Code,2012 [YR2012],2013 [YR2013],2014 [YR2014],
Afghanistan,AFG,CO2 emissions (kt),EN.ATM.CO2E.KT,10208.13,9402.05,9281.34,10057.59,9294.93,

```

```

Albania,ALB,CO2 emissions (kt),EN.ATM.CO2E.KT,4541.8,4795.4,5188,4797,4573.2,5403.7,5316.1,4
Albania,ALB,GDP per capita (constant 2015 US$),NY.GDP.PCAP.KD,3736.34007957648,3780.69919254
'''

# Read CSV data into a DataFrame
df = pd.read_csv(StringIO(csv_data))

# Replace "." with missing values
df.replace('.', pd.NA, inplace=True)

# Convert columns that begin with "20" to numeric
df.iloc[:, 4:] = df.iloc[:, 4:].apply(pd.to_numeric, errors='coerce')

# Pivot the DataFrame
df_pivoted = df.melt(id_vars=['Country Name', 'Country Code', 'Series Name', 'Series Code'],

# Drop 'Series Code' column
df_pivoted = df_pivoted.drop(columns=['Series Code'])

# Pivot the DataFrame again to have separate columns for each 'Series Name'
df_pivoted = df_pivoted.pivot_table(index=['Country Name', 'Country Code', 'Year'], columns=

# Display the result
print(df_pivoted)

```

Response:

Certainly! This Python code is using the pandas library to manipulate and transform a CSV-like dataset. Let's go through it line by line:

1. Import necessary libraries:

```

import pandas as pd
from io import StringIO

```

- `pandas` is a powerful data manipulation library.
- `StringIO` from the `io` module is used to create an in-memory file-like object from a string.

2. Define CSV data as a string:

```

csv_data = '''
... (data here)
'''

```

- The CSV data is a multiline string containing information about CO2 emissions, GDP per capita, and population for different countries over several years.

3. Read CSV data into a DataFrame

```
df = pd.read_csv(StringIO(csv_data))
```

- `pd.read_csv` reads the CSV data from the provided string using `StringIO` and creates a DataFrame (`df`) from it.

4. Replace “.” with missing values:

```
df.replace('..', pd.NA, inplace=True)
```

- Replaces occurrences of “.” in the DataFrame with pandas’ `pd.NA` (a representation of missing values) in-place.

5. Convert columns starting with “20” to numeric:

```
df.iloc[:, 4:] = df.iloc[:, 4:].apply(pd.to_numeric, errors='coerce')
```

- Converts columns starting from the 5th column (index 4) to numeric values. The `errors='coerce'` parameter converts any non-numeric values to NaN.

6. Pivot the DataFrame:

```
df_pivoted = df.melt(id_vars=['Country Name', 'Country Code', 'Series Name', 'Series Code', 'Year', 'Value'])
```

- Melts the DataFrame to have a ‘Country Name’, ‘Country Code’, ‘Series Name’, ‘Series Code’, ‘Year’, and ‘Value’ columns. This is a common step for reshaping data.

7. Drop ‘Series Code’ column:

```
df_pivoted = df_pivoted.drop(columns=['Series Code'])
```

- Drops the ‘Series Code’ column as it’s no longer needed.

8. Pivot the DataFrame again:

```
df_pivoted = df_pivoted.pivot_table(index=['Country Name', 'Country Code', 'Year'], columns='Series Name', values='Value')
```

- Pivots the DataFrame again to have separate columns for each ‘Series Name’, creating a more structured format.

9. Display the result:

- Prints the final pivoted DataFrame.

In summary, the code takes raw CSV-like data, cleans it, transforms it, and finally presents it in a more structured and readable format using pandas operations.

1.10. Method 6: Interacting with the Global Community of Python Users

1.10.1. Python Slack and Discord

Slack and Discord are both platforms for groups of people to chat with each other, post to shared message boards for specific topics, share files, and hold meetings.

The main slack page for the global community of Python users is **Pyslackers**:

<https://pyslackers.com/web>. To join, just go to the URL and click "join the community". The main discussion happens on #python_, but there are many other useful channels including #data_science and #job_advice. This method is especially worthwhile if you already are used to using slack for work or for personal networking.

Python Discord (<https://www.pythondiscord.com/>) is similar to PySlackers but seems to be more focused on organizing shared Python projects and posting the code on GitHub. To join, click the "join us" link in the upper-right corner of their homepage.

One thing that helped me use Slack and Discord much more consistently is downloading the desktop app for Slack (<https://slack.com/downloads>) and for Discord (<https://discord.com/download>), and always leaving the apps open on my local computer so that I see messages right away when they are sent to me.

1.10.2. Live Chats With Python Users on Freenode

The Python user community is world-wide, and for the most part, very supportive. There are active internet relay chat (IRC) networks where you can post a question to members who are also logged in, to possibly get an answer right away. The most active Python IRC is the #python channel on Freenode: <https://webchat.freenode.net>. When I logged in while writing this notebook, there were 1,778 people logged on.

Internet chatrooms can be rough places, but the #python channel claims to enforce this [Code of](#)

1. Go to <https://webchat.freenode.net/>. Choose a nickname, and make it professional (you're a UVA student after all!) and unique.
2. Don't write anything under channel. Prove you are not a robot by selecting pictures of motorcycles or something. Then, once your humanity has been established, click "Start".
3. To use the #python channel, you need to register your nickname. To check if your nickname is unique, click on the "freenode" tab on the left-hand sidebar. A text box will appear on the bottom of the screen. Type:

```
/msg NickServ info
```

4. Step 3 will open a new tab. Switch to that tab. If no one else already has your nickname, you will see

```
NickServ: (notice) <nickname> is not registered.
```

If you see something else, it means someone already has your nickname. You can change your nickname right here by typing `/nick` followed by another nickname. Then type `/msg NickServ info` again. Repeat until you see the message listed above.

Important note: DON'T use a password here that you use for important things like email, bank accounts, etc. We shouldn't have the same faith in the security of Freenode's servers as we can have in Google's. Also, this is the kind of platform that tends to attract hackers. And for people used to a graphical user interface, it might be easy to mistype in a way that accidentally displays your password in the chat. Use a unique, throwaway password!

5. To register this nickname, type

```
/msg NickServ register <password> <email-address>
```

where `<password>` is a password you will use in the future, and `<email address>` is the email you want associated with this account.

6. Check your email for a confirmation code. Be patient, it can take up to 20 minutes for the email to go through.
7. Once you have the code, paste it and your nickname into this code, and submit it:

```
/msg NickServ VERIFY REGISTER <nickname> <secret-code>
```

8. You are now registered! Return to <https://webchat.freenode.net/> and log in with your nickname

guidelines that appear as links on the top of the screen.

1.10.3. Python Mailing Lists and Newsgroups

[Usenet](#) is a distributed discussion system (which means it has no central server). It was invented in 1979, and is still in use today. The Python Usenet message boards are at <https://mail.python.org/archives/?sort=popular>. The `comp.lang.python` board is for general discussions and questions about Python.

The tutor mailing list (<https://mail.python.org/mailman/listinfo/tutor>) is for users who want to ask questions about learning computer programming with Python.

If you have a question for the Python core development team, send an email to help@python.org. The team is pretty busy, so be sure to check other resources and lists for an answer first.

< Previous
[Surfing the Data Pipeline with Python](#)

Next >
[2. Loading Data from Electronic Data Files](#)