✦ Member-only story

Creating reusable code for Data Science Projects

A road map to writing your own library

Anup Sebastian · Follow
Published in Towards Data Science
7 min read · Jan 31, 2020

Listen Share ••• More

As aspiring Data Scientists, we spend a lot of our time writing code, however looking the bigger picture, the core of Data Science is not about writing code, but to understand our data and extract value out of it. The coding part is just a means to accomplish this goal. Obviously we cannot avoid writing code and doing so is probably detrimental to the process, however we can reduce the amount of time we spend doing it, which enables us to spend the bulk of our time developing strategies to accomplish our primary goal.

The process for most of us is as follows. We tend to write small blocks of code then copy, paste and modify as we need to. This is okay for small tasks, but doing this for large chunks of code multiple times can make your notebook disorganized, making it difficult to find things, and more often than not, break the notebook while experimenting with different strategies. It can also be extremely annoying to make changes to code in multiple different places, again, wasting your time.

Some of us define functions to get around this, but the problem with this is we have to keep copying these functions around into different notebooks, whenever we create new ones. Also, when we work on later projects it can be difficult to retrieve functions we used in earlier projects, making us spend time trying to track it down

amidst all our projects, and then just giving up and writing out a custom function again.

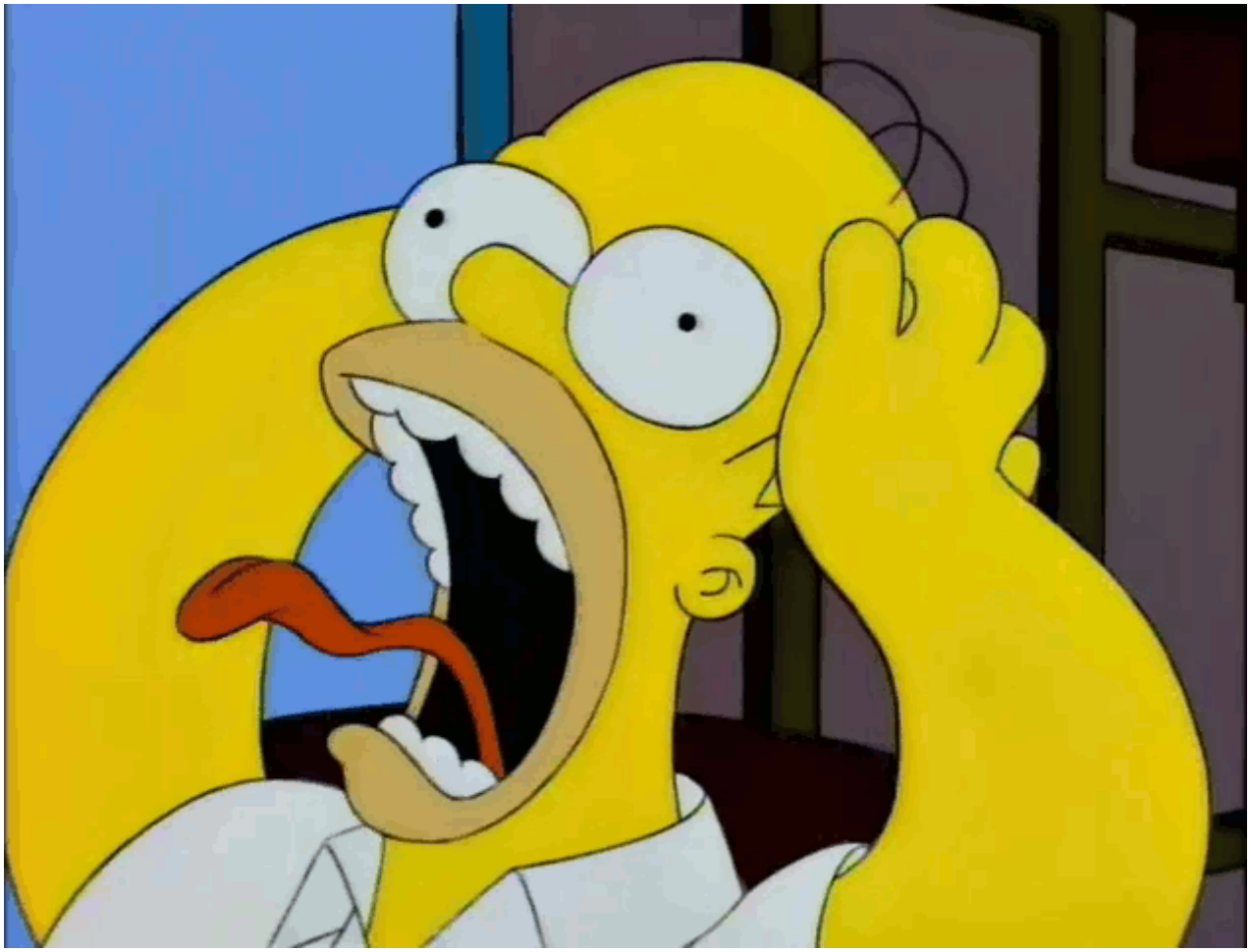


Image from: https://imgur.com/gallery/0BpqqmW

Level 1 — Enter .py files

The best solution to this problem is to keep your code defined within functions in external py files. This way, you can reuse your code whenever you need to in any notebook you need to. All you need to do is import those functions just like you import your favorite libraries and use them. This keeps your code organized and focused on the task. There wont be large chunks of code that distract from the main task, and trying to find things in your notebook can become a lot easier. When you move to a different project all you need to do is copy those py files to the project folder and you are all set. (Towards the end of the article I discuss some ways to help you avoid having to do this too.)

Now, how do we create these files? Lets go step by step using a simple example.

Lets create a function called custom_mean() that takes in a list and adds two to all the values, multiplies it by two and then returns the mean. Of course, this is far too

simple for any truly useful function, but you could make it anything you need it to be.

Now, open up a text editor and copy the function into it. At the top of the page add in any imported functions that you have used for your own function. In my case I have used mean() from Numpy so my file look like this.

```
import numpy as np

def my_mean(list):
   list = np.array(list)
   return np.mean((list + 2) * 2)
```

Next, just save your file with the name that you want your import to be called with a *.py* extension. I will call mine *custom_means.py*.

Now to import it into your notebook, you can do it like this.

```
from custom_means import my_mean
```

Now let's try and make use of it.

```
list = [34, 54, 46, 57, 86, 34]
print('My Mean:', my_mean(list))

# Output
# My Mean: 107.66666666666667
```

That was easy enough. Now how do you take this to the next level?

Level 2— Directories

You could add more functions to your files as your project progresses. You could also create custom classes for libraries like Scikit-Learn. You can learn how to do that here.

Eventually as you have written more functions and classes, you will learn that, it is important to organize them as well in different files, based on the kinds of tasks they accomplish, just like the popular libraries that we use. For example, you would

expect your scientific compute functions to be part of numpy and your graphing utilities to be part of matplotlib. So, it would be sensible to do the same to your functions.

As the number of files increase, you then get to a point where it would just be easier to have them all in a folder or multiple folders. Now how do you import functions from a file if it is not in the same folder you are working in? The answer is something you have been doing all along without realizing it. I can illustrate with an example using the same file in the previous example. Lets add it to a folder called *utilities*.

We now have the *utilities* folder containing the *custom_means.py* file, within our working directory. To import the same function now we would type.

```
from utilities.custom_means import my_mean
```

Looks familiar doesn't it. For example when we import the RandomForestRegressor from Scikit-learn as follows,

```
from sklearn.tree import RandomForestRegressor
```

What we are actually doing is accessing the tree directory within the sklearn installation and then importing the class from the *_classes.py* file. You can have a look here.

Note that this is slightly different from our example above but the idea is the same. The differences are due to the fact that they are using additional ways to speed the code up using Cython, for which additional files are required. They have a more complex, but effective way of managing their files because of the scale of the project. The *__init__.py* file automatically tells python which file to look at for the code to each class rather than us needing to explicitly tell it

```
from sklearn.tree._classes import RandomForestRegressor

#Note: This produces an error, because of the way the code is
#organized
```

You do not have to worry about it for now, but it is nice to set this as a goal to get to someday.

Organizing all our custom utilities in folders can be very helpful. Now we just have to copy the folder to the working directory of any new project and we are set. As your tools become more polished after you tweak them over time, it might eventually be useful to be able to call these utilities without having to copy your utilities folder(s) each time.

Level 3— Python Path

So lets now take this to the next level. It is not worth it to do this until you have a set of useful utilities, that do not have to be edited frequently. If you functions and classes are not finalized, you probably should just keep copying the folders to new project directories. But lets pretend we have some polished utilities for now.

To understand this, we need to understand what happens when you import a package. Python first checks your working directory then it checks the 'path' variable to see if your the function you are importing is actually there. The path variable is essentially a list of locations where various packages are installed in your computer. You can have a look at your path variable by running

```
import sys
sys.path
```

The output on my computer is as follows. I am using a Linux system, and it will differ based on the operating system you use, however the process is the same regardless of what computer you use.

```
['/home/anupjsebastian/anaconda3/envs/my-env/lib/python37.zip',
 '/home/anupjsebastian/anaconda3/envs/my-env/lib/python3.7',
 '/home/anupjsebastian/anaconda3/envs/my-env/lib/python3.7/lib-
dynload',
 '',
 '/home/anupjsebastian/anaconda3/envs/my-env/lib/python3.7/site-
packages',
 '/home/anupjsebastian/anaconda3/envs/my-env/lib/python3.7/site-
packages/IPython/extensions',
 '/home/anupjsebastian/.ipython']
```

If you place your utilities folder in one of these directories, you will be able to access your functions and classes the same way you do all the other packages you use. I personally would prefer it to be under

```
/home/anupjsebastian/anaconda3/envs/my-env/lib/python3.7
```

as the rest of are mostly sub-directories of this directory.

There is also a way to place it in a custom location. For Linux and Mac, type the following in the terminal

```
# For Linux
nano ~/.bashrc

# For Mac
nano ~/.bash_profile
```

and add the following line to the end of the file.

```
# For Linux
export PYTHONPATH=/Your/path/here

# For Mac
export PYTHONPATH="/Your/path/here"
```

For Windows, you can find out here. More details about the Python Path can be found out here.

Bonus Level

A cool way to do the same thing in an easy and elegant manner is using GitHub. You can put up your code in a GitHub repository and just pip install it to your computer. This ensures that the package is automatically placed in the right location along with the rest of your pip installed libraries. the code to pip install a GitHub repository is as follows.

```
pip install git+(weblink here)

# For example
```

```
pip install git+https://github.com/scikit-learn/scikit-learn
```

I showed an example above on how to install scikit learn form its GitHub repository, however I do not recommend you do this for Scikit-learn or any professional package, unless you know what you are doing. However, you could do it for your own packages.

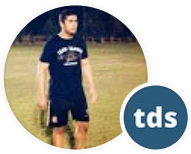
That's all there is to it — A whole road map on how to gradually reduce your time spent coding and to focus on the task at hand, by writing reusable code and automating a lot of the tedious processes that you end up doing for every new project.

So in summary, start by writing functions in you code more often to do repetitive tasks, then move them to separate py files so that they are organized well and can be easily utilized. Then use directories to manage your pip files. Continue to tweak and develop your reusable code, and after you believe they are ready for prime time use, place them in a location that will allow you to easily access them from everywhere. Perhaps, then share your code on GitHub and contribute back to the open source community.

This article contains a lot of information and can be a lot to take in at once. I hope that the article was useful for you no matter what stage of learning you are at. Even if you aren't going to do everything mentioned at this moment in time, it could be useful to refer back here whenever you need to.

Good Luck!

Data Science | Python | Libraries | Machine Learning | Artificial Intelligence



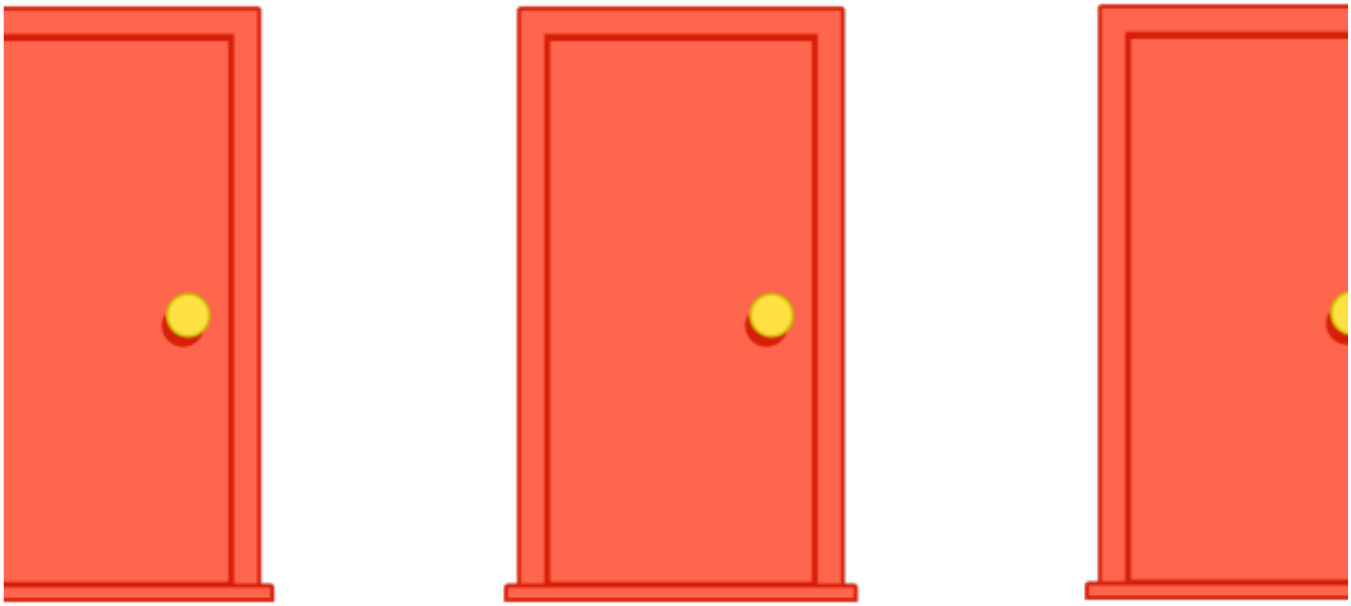
Follow

Written by Anup Sebastian

26 Followers · Writer for Towards Data Science

Machine Learning Engineer / Data Scientist

More from Anup Sebastian and Towards Data Science



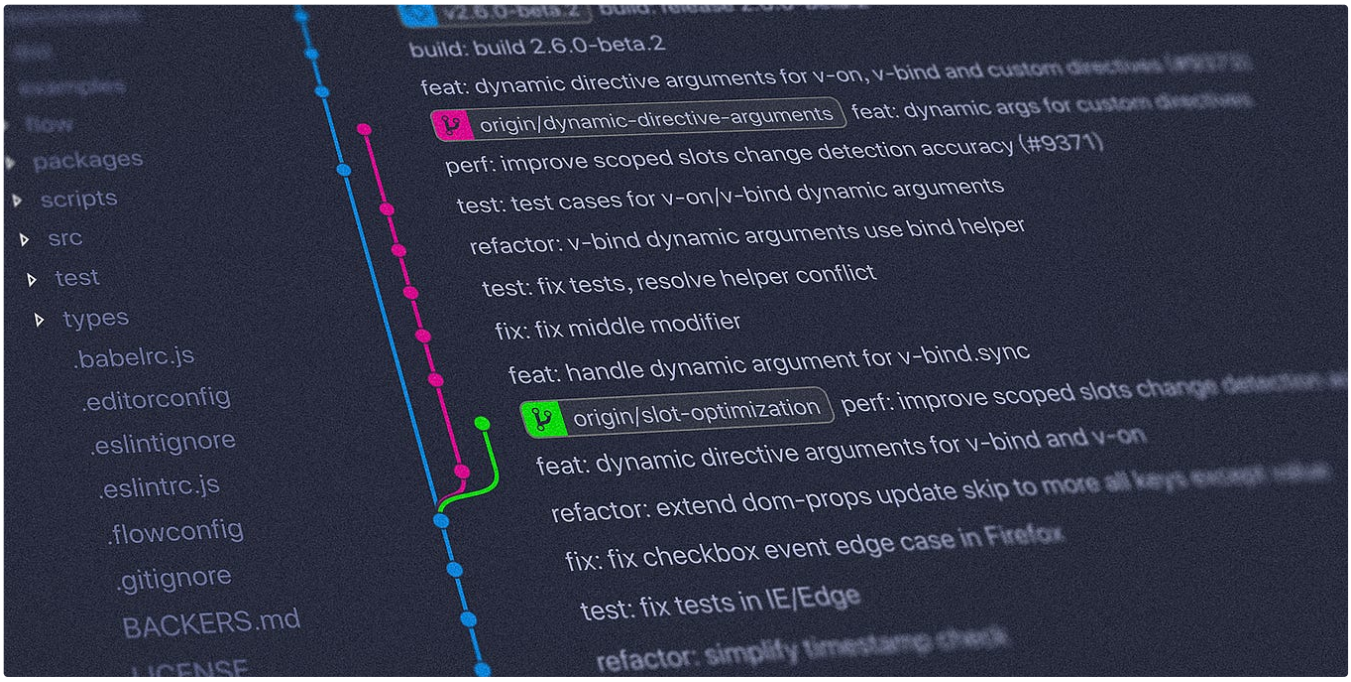
Anup Sebastian in The Startup

The easiest explanation to the Monty Hall problem

The last one you are going to need to look at

✦ · 5 min read · Feb 13, 2020

149 4



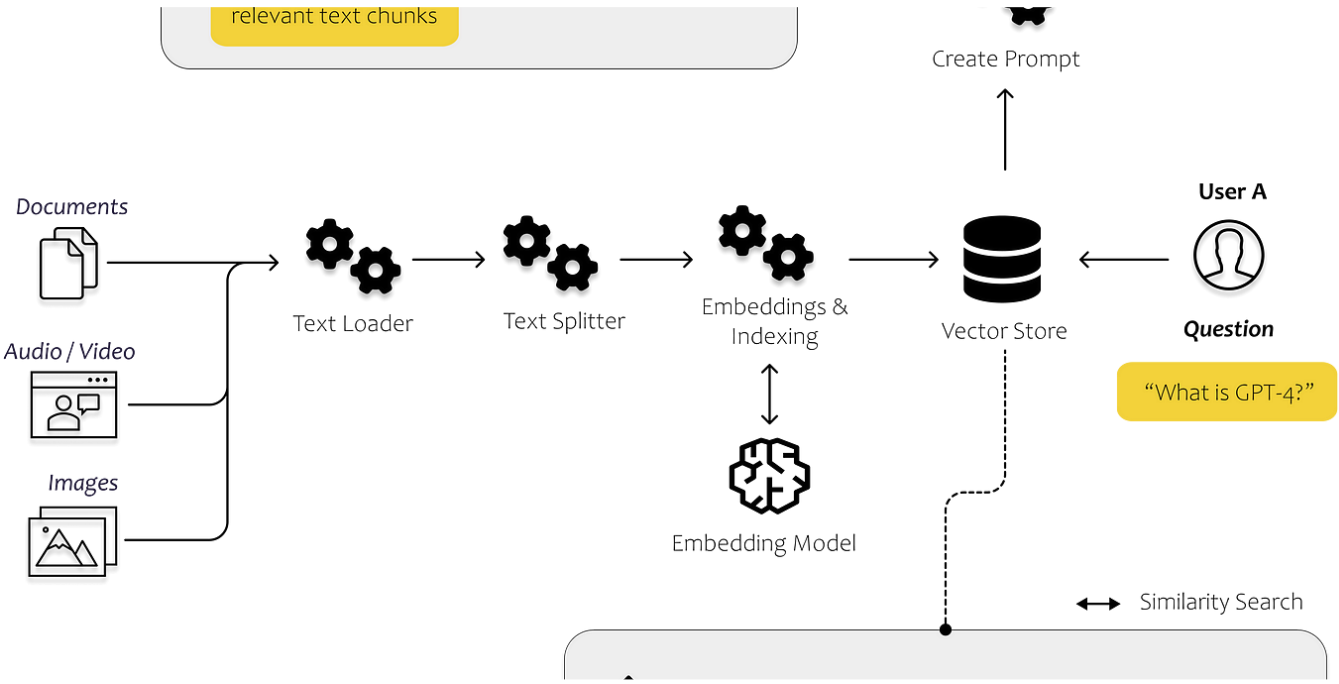
Miriam Santos in Towards Data Science

Pandas 2.0: A Game-Changer for Data Scientists?

The Top 5 Features for Efficient Data Manipulation

7 min read · Jun 27

1.5K 18



Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

✦ · 26 min read · Jun 22

2K 19

Anup Sebastian in Towards Data Science

Writing your own Scikit-learn classes—for beginners.

Basics to get you started

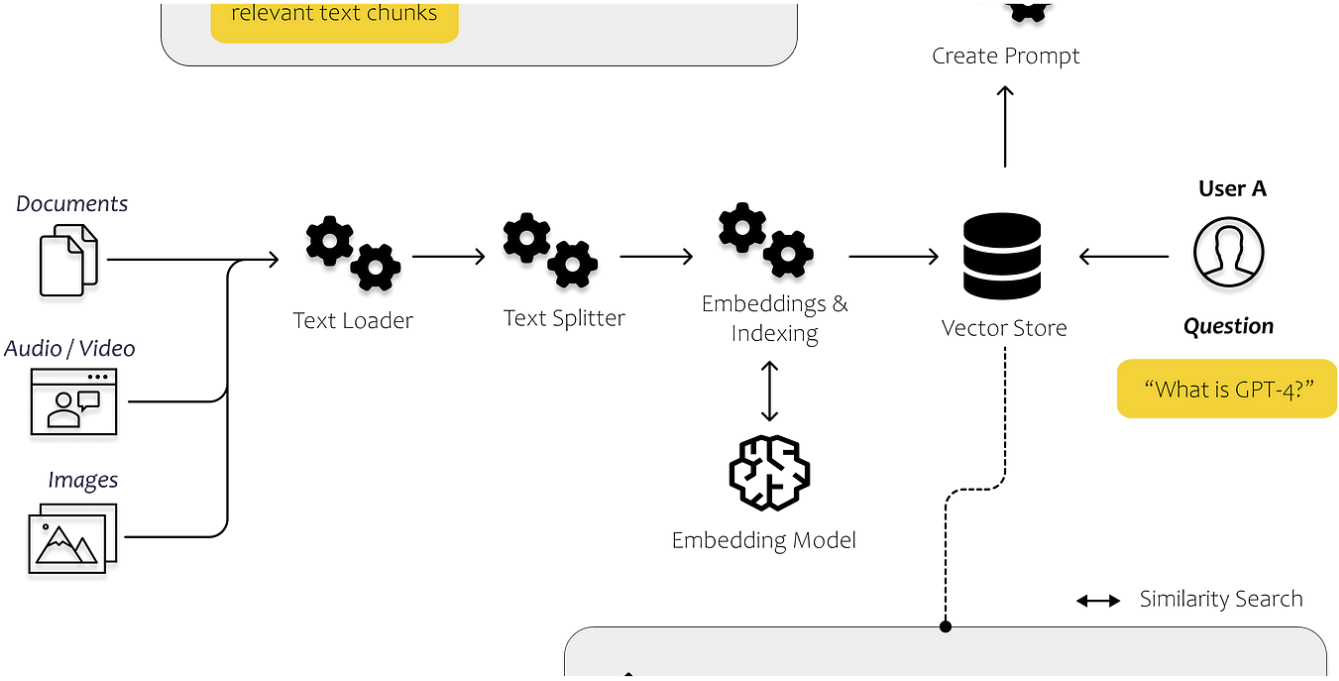
✦ · 6 min read · Jan 17, 2020

232

See all from Anup Sebastian

See all from Towards Data Science

Recommended from Medium



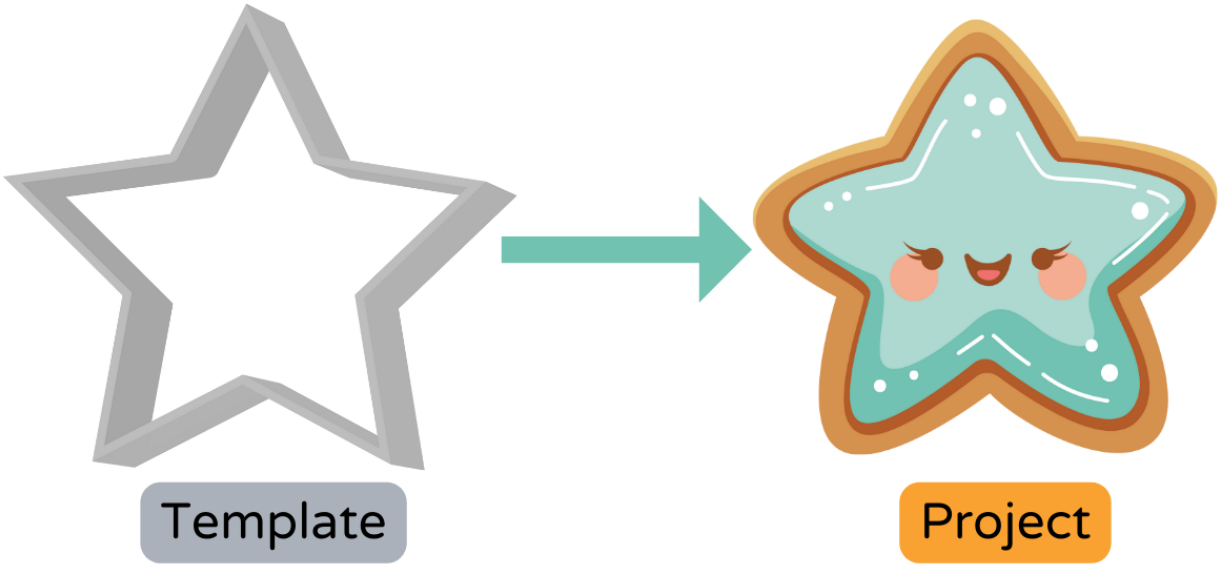
Dominik Polzer in Towards Data Science

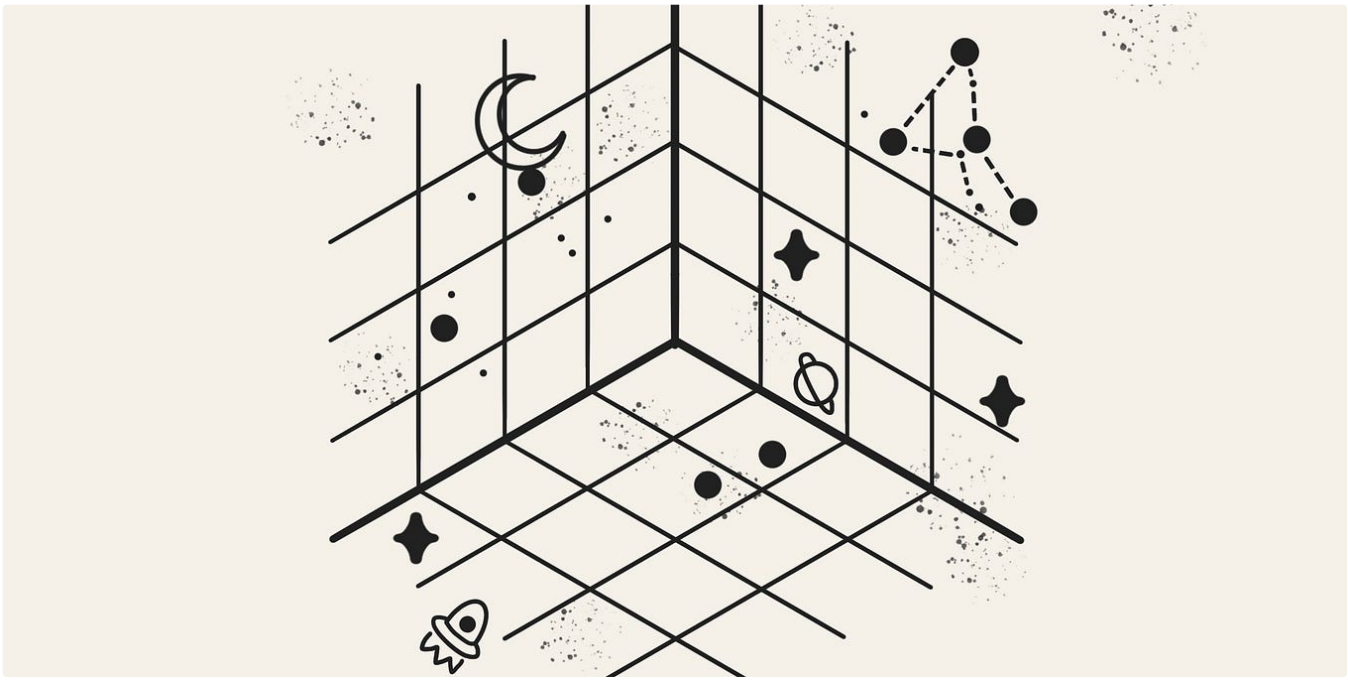
All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

✦ · 26 min read · Jun 22

2K 19





Leonie Monigatti in Towards Data Science

Explaining Vector Databases in 3 Levels of Difficulty

From noob to expert: Demystifying vector databases across different backgrounds

· 8 min read · Jul 4

1.2K 13

Anjana Samindra Perera in Level Up Coding

Large Language Models

Best Papers on Large Language Models (LLMs)

· 2 min read · Feb 25

75

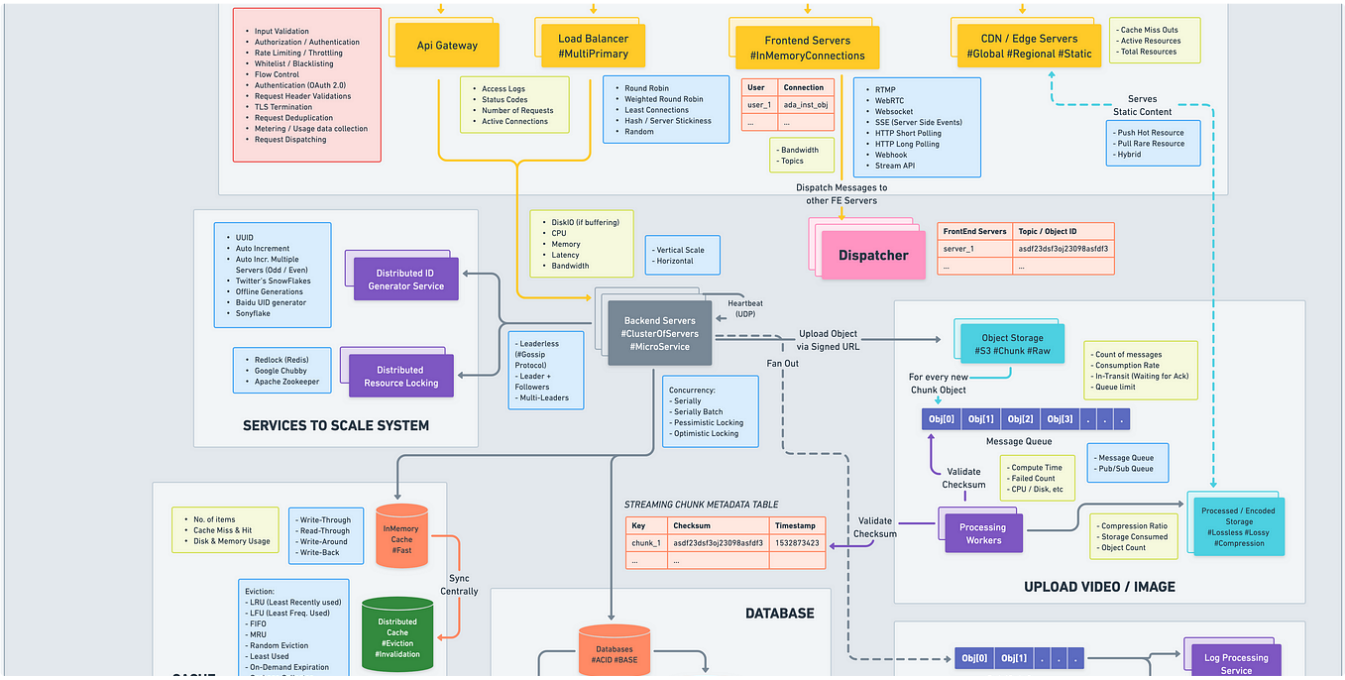
Boriharn K in Towards Data Science

Visualizing Sklearn Cross-validation: K-Fold, Shuffle & Split, and Time Series Split

Plotting the process of Sklearn K-Fold, Shuffle & Split, and Time Series Split cross-validation and showing validating results using Python

· 11 min read · 3 days ago

110 1



Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

· 9 min read · Apr 20

5.9K 49

See more recommendations