# How to Write Code Like a Pro

## Approaches and techniques that help you to write code to professional standards

### Background

I have been writing code for 20 years and over that time I have established a set of 10 principles that I believe data scientists and software developers can adopt to help them to write their programme code to professional standards.

These approaches are generally applicable to any software development environment but as all of my coding these days is in Python in VS Code I have focused on those tools for specific examples.

## Principles, Tools and Techniques

### Linting

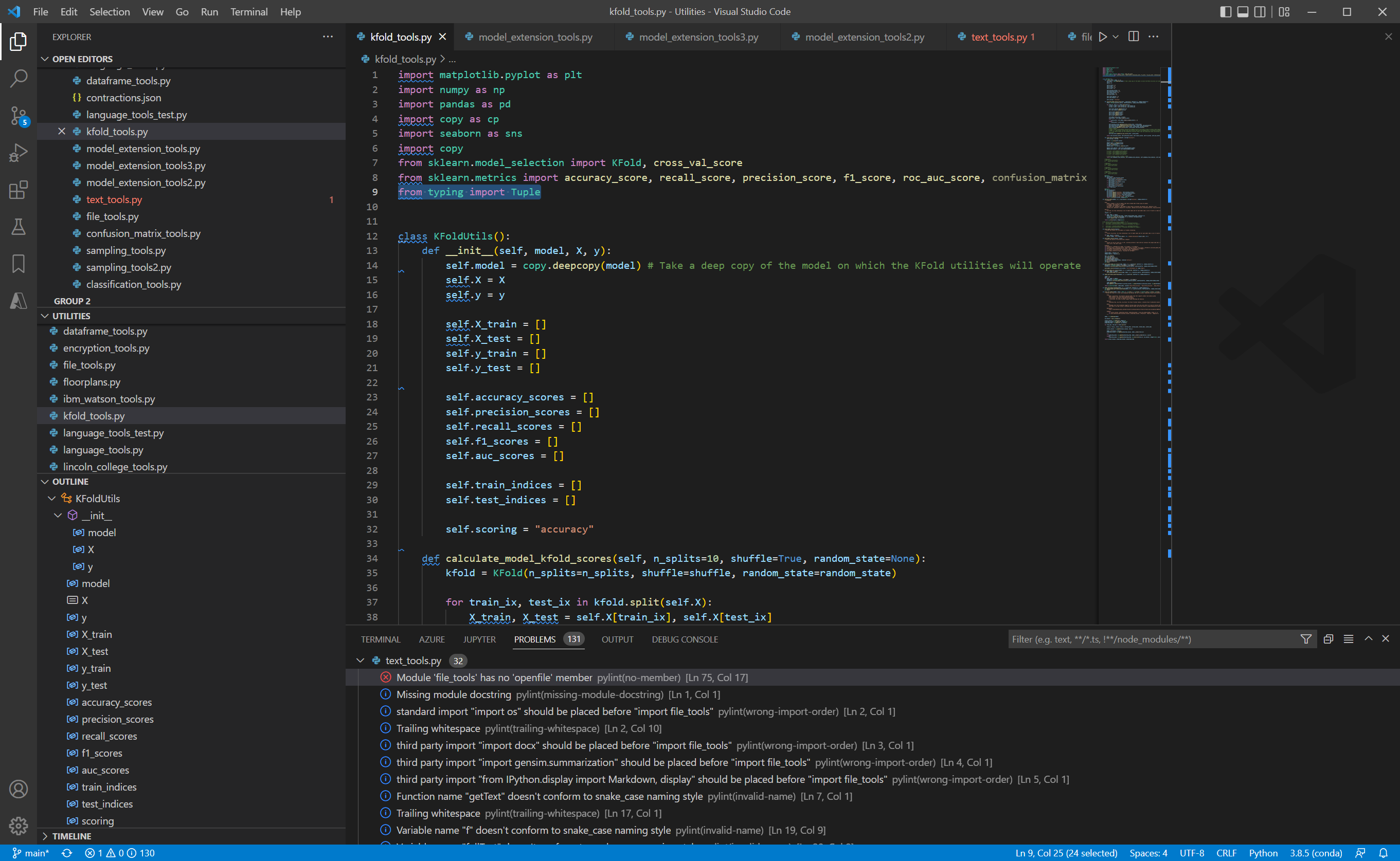
The first easy win to help in writing professional code is to use linting.

“Linting highlights syntactical and stylistic problems in your Python source code, which often helps you identify and correct subtle programming errors or unconventional coding practices that can lead to errors.” ([https://code.visualstudio.com/docs/python/linting#](https://code.visualstudio.com/docs/python/linting))

To turn linting on in VS Code go to the command palette with Ctrl+Shift+P and type “Select Linter”, then chose the linter you want to use.

There are a range of linters available and the choice will depend on individual and team preference but my personal favourite is “pylint” because I like way in which it presents issues in the code and because it is easy to configure.

Once linting is on VS Code will display its suggestions in the problems window which will update dynamically as you write new code and improve existing code –



If you ever come across a pylint error you do not understand they are all detailed here - <https://vald-phoenix.github.io/pylint-errors/>.

You should try to resolve all of the pylint errors and warnings but occasionally you will come across some that you would like to turn off. In this instance edit the settings.json file and exclude them as follows –

{

    "python.linting.pylintArgs": [

        "--max-line-length=300", "--disable", "raise-missing-from", "--disable", "missing-module-docstring", "--disable", "eval-used", "--disable", "too-few-public-methods", "--disable", "no-self-use"

    ]

}

### Comments, Type Hints and Documentation

Let’s start with type hints. Python is a “dynamically typed” language. The type of a variable is not required and most online examples omit the type of variables …

    def train\_test\_from\_median\_fold(scoring):

        self.scoring = scoring

        scores = np.array(self.scores)

The drawback is that the client or caller cannot easily know what type the function is expecting and may either need to know or do some research. If the type is added to the code for the parameter and return types the readability and understandability increase …

    def train\_test\_from\_median\_fold(self, scoring : str = "accuracy") -> pd.DataFrame:

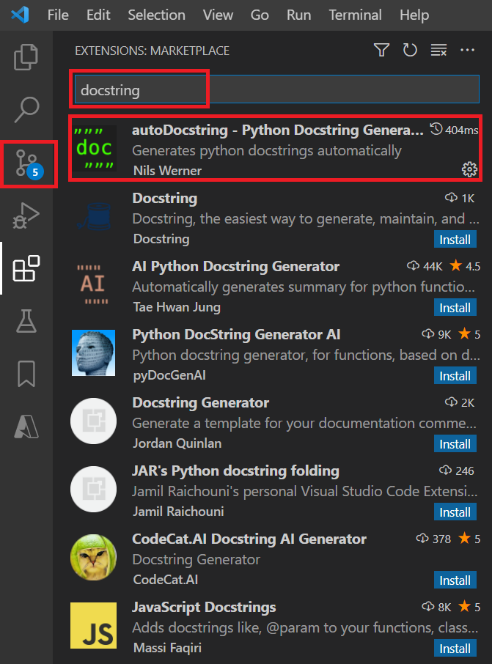
        self.scoring = scoring

        scores = np.array(self.scores)

Once type hints are included the next stage is well commented and documented code which is one of the things that sets professional code apart.

In addition to improving readability and maintainability it is also the case that as you add comments, type hints and documentation it makes you think about your code from a different angle which leads to automatic incremental improvements.

To begin with comments start by adding the autoDocstring extension into VS Code –



This means that when you start a new line underneath a def statement, and enter three double quotes a docstring will automatically be created leaving just the descriptions to be completed -

Text

Description automatically generated

In addition to a well written and informative docString it is also important to leave lots of comments in the code to explain it to other developers but also to explain tricky code to yourself when you come back to it in several months’ time needing to make changes. It is also worth wrapping big comment blocks in a region that can be folded to keep the main code base clean and readable.

Text

Description automatically generated

When type hints and docStrings have been completed throughout a module it is very easy to create a web page of professional looking documentation. Simply invoke pydoc as follows …

python -m pydoc -w "..\lib\crypto\_tools.py"

… and the documentation will be automatically created …

Graphical user interface, text, application

Description automatically generated

The key thing to take away from type hints, comments and documentation is not to treat it as a chore – the boring bit you do at the end of the project. Rather it is an opportunity to constantly review, reappraise and professionalise your code.

### Project Structure

Many smaller projects can get away with being created in a single folder that contains all of the code and configuration necessary to run the project.

However small projects tend to grow into medium and large projects and it does not take long before using a single folder as a dumping ground for all project files becomes messy and unprofessional.

I have seen several recommendations for a standard folder layout for Python projects in various articles and concluded that it does not matter which one is chosen so long as it provides a sensible, logical, intuitive, discrete split of project resources.

This is not a particular standard, but it is the standard I have developed and adopted for my projects …

Text

Description automatically generated

The “data” sub-folder contains any data files relating to the project and I commonly add “in” and “out” folders if my project cleans or transforms data.

“docs” is where I store the documentation created by pydoc from the docStrings and a batch to invoke pydoc to enable one-click documentation production.

“keys” was a special project for this project which created security keys demonstrating that I am not averse to extending my standard approach based on the needs of the project.

“lib” is where I store any re-usable code libraries. In this case I moved all the code that had potential future re-use value into crypto\_tools.py and refactored it to maximise usability and maintainability.

“notebooks” is where all Jupyter Notebooks are separated and stored. I usually use Notebooks to create a sample user interface and to demonstrated and show off how a project works and can be used.

“src” is where I store any other Python code that is not a reusable library.

“unittests” is where all of the pytest unit tests are stored. In a medium-large project there could be a lot of unit test files and these really need moving to a discrete location to maintain tidiness. This requires a bit of additional configuration that I will document in a future article.

Lastly, I include “main.py” in the root and include the following code –

if \_\_name\_\_ == "\_\_main\_\_":

to ensure that the whole project can be run from the command line.

### Unit tests

Spending time developing unit tests is critical to coding like a professional. The two main frameworks used for unit testing Python in the VS Code environment are pytest and unittest.

My personal preference and standard is pytest; you can read more about the pros and cons here - <https://code.visualstudio.com/docs/python/testing>.

Once everything is configured and set up simply use the flask icon inside VS Code to discover all the unit tests and then click play to execute them all –

Text

Description automatically generated

They will light up green if everything works OK or red if any unit tests fail.

The power of using pytest comprehensively is that future changes and updates can be made to the project in the confidence that if anything is inadvertently broken a single-click test run will instantly highlight the problems.

In this way code can be developed and maintained professionally and current and future code quality will be high.

### Object Oriented Programming

The 4 main concepts in Object Oriented Programming (OOP) are

* Inheritance - a class inherits the properties and methods of another class)
* Encapsulation - hides the data of a class through private properties)
* Polymorphism - methods that have the same name with different implementation - think len(int) vs. len(list)
* Abstraction – enforcing standard interfaces – think .fit() and .fit\_transform() in scikit-learn

A detailed explanation with some Python examples can be found here - <https://www.analyticsvidhya.com/blog/2020/09/object-oriented-programming/>.

Moving beyond the core concepts there are many benefits to moving beyond procedural (writing functions) into object oriented preprograming (writing classes). If you Google the benefits of OOP you will find many advantages including re-useability, ease-of-maintenance, improved security, improved design, increased productivity, easier trouble shooting etc.

Balanced against that the Python implementation of OOP is much weaker than languages like C++, C# and even Java and it can be seen as difficult to learn and more time consuming for programmers.

Weighing up the pros and cons the key benefit of OOP is that the consumers and clients of the classes and objects you write have a much more intuitive and usable interface leading to front-end code that is more compact, maintainable and readable.

Consider the following code snippet –

one\_way\_hash = OneWayHash(hashing\_algorithm="SHA256")

hash\_value\_1 = one\_way\_hash.hash\_data("Hello World")

hash\_value\_2 = one\_way\_hash.hash\_data("Hello Universe")

The author of the OneWayHash class has clearly gone to the trouble of designing an easy interface, supporting multiple underlying algorithms including SHA256 and provided a way for the client to instantiate an object (one\_way\_hash) and then execute multiple hashes.

That is the power of OOP and why I always write my libraries that have re-use value as classes and objects and if this is one of the key approaches in writing professional code.

### Code Duplication and Refactoring

The duplication of the same or very similar code leads to projects that are prone to error. If a similar piece of code has been repeated 10 times in a project and then a bug needs fixing or an enhancement adding it needs doing 10 times which is inefficient and laborious and provides 10 opportunities to make a mistake.

Consider these simple functions –

STRING\_ENCODING : str = "ascii"

def bytes\_to\_str(data : bytes) -> str:

    return data.decode(STRING\_ENCODING)

def str\_to\_bytes(data : str) -> bytes:

    return bytes(data, STRING\_ENCODING)

Each one is only replacing a single line of code so why bother? Well, this code snippet is taken from a project that contained 100s of instances of the decode and encode and at one point in the project the encoding had to be changed from “utf-8” to “ascii”.

This required many changes and somehow a couple of them were missed leading to bugs and errors that were not spotted until after the code went into production.

By making the change to pulling the instances out into two simple functions the code looked cleaner, the duplication was eradicated and any future changes to the encoding can be made quickly and confidently.

In programming parlance this is known as the “DRY” method – “Don’t Repeat Yourself” (see <https://en.wikipedia.org/wiki/Don%27t_repeat_yourself> for more details).

Put simply if you ever see a piece of code that is duplicated, refactor it. Remove the repeated code into a function, class or library and replace the code with a call to the new function.

This leads onto the second part of this section. Always refactor your code and refactor it multiple times, and then refactor it again!

This is the process of reviewing your code with a critical eye, constantly asking yourself the question “How can I improve it?”. For example …

* Can you replace several lines of code with one line?
* Conversely does it need to be a little bit more verbose and hence more readable?
* Have you written something bespoke where an existing library already exists?
* Could you have moved that repeating code into a function?
* Could you have removed several functions into a class?
* Could you have added a bit more code to enhance the future proofing and re-usability?
* Have you considered improper use cases and added the appropriate exceptions and error checking?
* Have you adopted “pythonic” approaches like using lambda functions, list comprehension etc.?
* Is there a way to make the code run faster?
* Is the code re-runnable and re-usable?

Considering these and other questions and being semi-obsessive about iteratively refactoring your code until it is close to being perfect is one of the key techniques that will help you to code like a pro.

### Building Libraries

There is a wealth of code available online using resources like <https://stackoverflow.com/> and whenever I get stuck it is common to be able to Google the problem and find an answer.

Also, there is the pressure to write code fast to get the job done for our employers, our customers and ourselves.

Those two factors encourage quick, and sometimes sloppy working, but you can have the best of both worlds and spend time improving your coding and code more quickly. Here’s how …

Build libraries! My personal approach is to maintain two libraries.

The first is called “Sample Code”. It is a dumping ground for all the useful code snippets I come across online and in books that I know I will want to refer back to in future and then not be able to find!

Taking the time to copy potentially useful code into my “Sample Code” project helps me to code faster and better in future.

The second library I maintain is my “Utilities” library. I also keep useful code in here but for code to qualify for my utilities library it has to be re-factored, tidied, tested, documented and structured in such a way that it will be generically useful.

Here is an example. I needed some synthetic data to test a classification algorithm. Google soon helped me track the code down but it was a bit messy and undocumented. After a bit of extra work my utilities library gained a useful new method as follows –

def make\_classification\_dataframe(n\_samples : int = 10000, n\_features : int = 25, n\_classes : int = 2, n\_clusters\_per\_class : int = 2, feature\_name\_prefix : str = "feature\_", target\_name : str = "target", random\_state : int = 42) -> pd.DataFrame:

    """ Creates a data frame of sample data suitable as input to a classification machine learning algorithm consisting of the specified number of features and a target with the specified numbeer of classes.

        Args:

            n\_samples (int, optional): The number of data points to generate. Defaults to 10000.

            n\_features (int, optional): The number of features in the data. Defaults to 25.

            n\_classes (int, optional): The number of classes in the target. Defaults to 2.

            n\_clusters\_per\_class (int, optional): The number of clusters. Defaults to 2.

            feature\_name\_prefix (str, optional): The prefix for the feature names in the data. Defaults to "Feature" which generates "Feature 1", "Feature 2", ... , "Feature n".

            target\_name (str, optional): The prefix for the name of the target. Defaults to "Target".

            random\_state (int, optional): The seed for the random state. Defaults to 42.

        Returns:

            pd.DataFrame: The generated DataFrame.

        Example:

            >>> df\_data = make\_classification\_dataframe(n\_samples=N\_SAMPLES, n\_features=N\_FEATURES, n\_classes=N\_CLASSES, n\_clusters\_per\_class=N\_CLUSTERS\_PER\_CLASS, feature\_name\_prefix=FEATURE\_NAME\_PREFIX, target\_name=TARGET\_NAME, random\_state=RANDOM\_STATE)

            >>> X = df\_data.drop([TARGET\_NAME], axis=1).to\_numpy()

            >>> y = df\_data[TARGET\_NAME].to\_numpy()

            >>> df\_data.head()

        """

    X, y = make\_classification\_data(n\_samples=n\_samples, n\_features=n\_features, n\_classes=n\_classes, n\_informative = n\_classes \* n\_clusters\_per\_class, random\_state=random\_state)

    feature\_names = [feature\_name\_prefix + str(v) for v in np.arange(1, n\_features+1)]

    return pd.concat([pd.DataFrame(X, columns=feature\_names), pd.DataFrame(y, columns=[target\_name])], axis=1)

The only other thing you need to do is create a pointer to your utilities library in future project so the methods and objects can be used without needing to copy code or modules (which would break our DRY – Don’t Repeat Yourself rule). Here’s how …

import sys

sys.path.insert(1, r'C:\Users\GHarrison\Python Projects\ Utilities')

from synth\_data\_tools import make\_classification\_data

sys.path.insert adds the utilities folder to the Python path for the project and then the make\_classification\_data is imported and can be used.

Building libraries will help you to code like a pro. The act of refactoring code to make it useful will improve your knowledge and understanding and in future projects you will be able to pull that library code to solve the double conundrum of coding very quickly without sacrificing quality.

### Write Blogs

The protégé effect is a psychological phenomenon where teaching … or preparing to teach information to others helps a person learn that information (<https://effectiviology.com/protege-effect-learn-by-teaching/>).

That is just one of the benefits of blogging about code you have written and other data-science associated topics regularly.

Over the last 2 years I have written regularly and published my articles on medium.com (<https://medium.com/>)and Towards Data Science(<https://towardsdatascience.com/>).

Here are my articles - <https://grahamharrison-86487.medium.com/>.

In the act of preparing your code to include on a blog you will review it with a renewed critical eye; after all you do not want any mistakes to make it onto a public article!

Also, the act of explaining your code to others will help you to understand it completely and to improve your knowledge and expertise in the process.

Trust me on this one – one of your future readers and students will be yourself! In 6 months or a year’s time you will have forgotten all the details of that really useful coding technique you discovered and you will go back to your own articles to help you remember.

Lastly, blogging on a reputable platform like medium and towards data science will help build your professional persona online which help your peers, the programming community and potential future employers to have an awareness of your professional standards and capability.

### Reading and Challenges

Read as much relevant material as you can get your hands to help you improve your professional coding skills and subject knowledge

Sign up for mailing lists like Real Python (<https://realpython.com/>) and make sure you join medium.com …

… then put the app front-and-centre on your smart phone so you can be reading about your subject every time you are stood in a shopping queue or are waiting for the kettle to boil …

Graphical user interface

Description automatically generated

Also sign up to the “Towards Data Science” podcast so you can be improving your professional skills whilst you are driving or sat on the sofa - <https://open.spotify.com/show/63diy2DtpHzQfeNVxAPZgU?si=224fbdb47b6f4c1f>.

So, there is a universe of free or low cost resources out there to help you to code like a pro but every now and then it is also worth paying real money for a good, old fashioned book. Here are a selection of the books that I have used recently to improve my skills –

<https://www.amazon.co.uk/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1098125975>

<https://www.amazon.co.uk/Python-Machine-Learning-Example-scikit-learn/dp/1800209711>

<https://www.amazon.co.uk/Deep-Learning-Coders-fastai-PyTorch/dp/1492045527>

<https://www.amazon.co.uk/Practical-Blockchains-Cryptocurrencies-Application-Applications/dp/1484258924>

The last recommendation of this section is about challenging yourself using the wealth of online tools and resources.

If you want general Python coding challenges you can use a resource like “Python Principles” - <https://pythonprinciples.com/challenges/>.

If you want a bigger data science-type challenge and to measure your performance against your peers why not pick a dataset you like the look of on Kaggle and see how far up the leader board you can make it. Here is a competition I entered where I am currently 4th on the leaderboard; why not have a go and see if you can overtake me! –

<https://www.kaggle.com/competitions/credit-default-prediction-ai-big-data/leaderboard>

### Practice

“It takes 10,000 hours of intensive practice to … [be] as good as Bill Gates at computer programming.” (Malcolm Gladwell).

Malcom Gladwell’s oft-quoted rule-of-thumb would mean that at 8 hours a day, 5 days a week it would still take 5 years to become a professional, expert coder, but I don’t believe it will take that long.

Rather it is how you practice that can fast-track you to becoming a coding pro which has been the core theme of this article.

And whilst you don’t need to lock yourself in a room banging out code every day for 5 years I do wholeheartedly believe that practice makes perfect and hence I try to find some time every day to code, even if it is just a few minutes at lunchtime or before I go to bed (where I often code in my sleep anyway!).

## Conclusion

Becoming a pro coder does involve some hard work and dedication but if, like me, coding excites and enthuses you and if you love your subject you will get there and in this article I have explore 10 tools and techniques that can accelerate that journey.

As well as writing I have coached and mentored many programmers and developers and if this is something that interests you why not get in touch with me at [GHarrison@lincolncollege.ac.uk](mailto:GHarrison@lincolncollege.ac.uk).

## Thank you for reading!

If you would like to support the author and 1000's of others who contribute to article writing world-wide by subscribing, please use this link - <https://grahamharrison-86487.medium.com/membership>. Note: the author will receive a proportion of the fees if you sign up using this link.

And if you enjoyed reading this article, why not check out my other articles at <https://grahamharrison-86487.medium.com/>?

Also, I would love to hear from you to get your thoughts on this piece, any of my other articles or anything else related to data science and data analytics.

If you would like to get in touch to discuss any of these topics please look me up on LinkedIn — <https://www.linkedin.com/in/grahamharrison1> or feel free to e-mail me at [GHarrison@lincolncollege.ac.uk](mailto:GHarrison@lincolncollege.ac.uk).