The Motivations Behind This Blog

§

Made with Obsidian

Type blog | Category learning-resources

Technologies Python, Julia, R, SQL, Markdown, LaTeX | Website Post Link

Hello, & welcome!

A couple of months ago, I had an idea; to create a place where people could learn Data Science & Computer Science-related content for free by reading quality and approachable publications. This idea is far from unique; there is already a sea of infinite web pages and online courses doing an excellent job. Still, I wanted to make it unique in many ways; I wanted to make it my own and, of course, return the favour for all that I've learned over the last couple of years.

I'm Pablo, a Chemical Engineering, full-time Data Scientist and Linux & Open Source enthusiast. I'm passionate about learning things and transmitting them to others in the most transparent and rigorous way possible. This, for me, is a crucial step in owning knowledge and is called The Feynman Technique[1]:

- I. Study
- II. Teach
- III. Fill the Gaps
- IV. Simplify

In this article, I'll introduce the scope of this blog, the purpose and motivations behind its inception, the added value I'm planning to provide, the overall blog structure, a more detailed explanation of what to expect from each section, and ways in which to collaborate.

I'll be using scripts which can be found in the Blog Article Repo.

§

Table of Contents

§

Why learn to program?

There are multiple answers depending on who you ask, but for me, it boils down to 3 core concepts:

1. It's an exciting time to learn

The last couple of years have proven to be just the beginning of an entirely new way of understanding our world and creating technology to improve it; code helps us abstract complex problems into simpler ones and solve them using what we already know.

Additionally, there has been increasing pressure for private companies to release their code as open-source; this creates more transparency and security and increases collaborative efforts in creating better and more secure code.

2. It creates a feedback loop

Learning to program does not exclusively result in learning to write code. It's just a skill like any other, and learning a new challenging skill has many side effects apart from learning the actual thing.

As we understand it, our brain is wired by a reward system. When exposed to a rewarding stimulus, our brain's reward system responds by increasing the release of the dopamine neurotransmitter, which causes pleasure.

Learning something new activates the release of dopamine, making us crave more while at the same time providing confidence to tackle other challenging tasks. If managed correctly, this mechanism can be sustainable and become a loop of positive feedback: the more you learn, the more confident you become, and the easier it is to wake up the following day and do the same, but better.

3. It provides a sense of independence

Not just because code can be written anywhere or because programming has unlimited applications, but also because it can be learnt entirely for free thanks to the increasing community.

§

What does it take?

There's a toxic cloud of stigma and prejudices surrounding what it takes to learn to program. People often think that programmers are born programmers and that everyone else is cancelled out of the equation by defacto; that's simply not true (*at least for 99% of the cases*), and millions of converted developers are out there to prove it.

Apart from a computer, a power outlet, and caffeine, I believe there are six skills required to start coding and make it a sustainable activity:

- **Curiosity:** If there's no curiosity, getting hooked will be almost impossible, even if the intellectual challenge is up there. This is true especially for programming because it's a vast world. The journey never really ends, no matter how good of a programmer you are (*Wikipedia claims there are 700 + programming languages, and that's without considering all the frameworks, utilities, services and other technologies available*)[2]. Still, while curiosity is a must, unmoderated curiosity will most certainly lead to oversaturation and loss of impetus; programming can become a rabbit hole if done wrong.
- **Selectivity:** It's a great thing to want to learn, but without selectivity, curiosity turns into a spiral of chaos and confusion. Having criteria as to whether what you're learning is valuable is considered part of the sanity check.
- **Tolerance to frustration:** When learning to code, there can and will be multiple things that will go wrong, potentially simultaneously, and more probably when you're already having a bad day. Still trying to figure out how, but this happens more often than not. Tolerance is not just learning to endure but also to back up, get some air, and return the next day to find out you already figured it out while in your sleep.
- **Continuity & consistency:** As with any learning activity, steadiness is key. This creates familiarity between concepts; it helps us learn to connect the dots faster, and things suddenly start making more sense.
- **Modesty:** Getting overconfident is easy, especially when starting. In fact, I would say it's natural. But if programming has taught me something, it's that without modesty, you'll never own your mistakes, and whenever a bug appears, you'll go into severe denial; this will make you either quit because "*the interpreter hates me and made a fowl play*", because "*if I couldn't handle this one, how am I going to keep up?*"
- **Managing expectations:** This one is closely related to the latter; if the expectations are too high, you will feel pressured to keep up and start comparing with other people, lose confidence, abuse coffee, and eventually enter into kernel panic. If the expectations are too low, then you might always feel you're doing perfectly fine, except when that technical interview comes, and well, we all know the rest of the story; coding needs to push the boundaries of what we think we are capable of, while at the same time knowing we are capable of doing it.

§

A Journey Through Data Science: A space where currents converge

Programming is not just about programming; it's more about curiosity and becoming obsessed with something so badly you cannot let go, tricking your reward system into a feedback loop. **A Journey Through Data Science** was created as an enabler for that feedback loop to happen.

1. On the scope

The blog is mainly about Data Science and Computer Science-related topics with a hands-on, business approach, but it really covers whatever life throws in my path; this curiosity thing I mentioned is genuine. I like to research new languages and technologies, adopt them, crack them and write about them if I think they provide value.

2. On the Technologies

I'll mainly be covering the following languages:

Python

- **What is it?**
  Python is a high-level, general-purpose, object-oriented programming language. It's currently the basis for Data Science and Machine Learning and the second most used language as of 2022, with 15.7 million developers.

- **Why learn it?**
  It's easy yet flexible, has an outstanding community, 200,000+ packages, and can be used for virtually anything.

Python's syntax has been often referred to as pseudo-code-like, and that's because it really reads as such.

Let us define a decoder that will accept an encrypted message and return a decrypted one:

Code

```python
import string

def decodeString(**kwargs):
    '''
    Parameters
    ----------
    **kwargs : list, list
        An encrypted message.
        A list of uppercase alphabet letters.

    Returns
    -------
    decoded : string
        A decoded message.
    '''

    decoded = ''.join([my_alphabet[x] if x >=0 else ' ' for x in my_list])

    print(decoded)

    return decoded

# Declare message and alphabet
my_list = [0, -1,
           9, 14, 20, 17, 13, 4, 24, -1,
           19, 7, 17, 14, 20, 6, 7, -1,
           3, 0, 19, 0, -1,
           18, 2, 8, 4, 13, 2, 4]

my_alphabet = list(string.ascii_uppercase)

decodeString()
```

Output

```
A JOURNEY THROUGH DATA SCIENCE
```

Julia

- **What is it?**
  Julia is a high-level, dynamically and statically typed programming language, but most importantly, Julia is extremely fast; it's like Python on steroids.

- **Why learn it?**
  Well, let us refer to the creators themselves:

> We want a language that's open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that's homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, and as good at gluing programs together as the shell. Something that is dirt simple to learn yet keeps the most serious hackers happy. We want it interactive, and we want it compiled. (Did we mention it should be as fast as C?)

— *Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman*[3]

Julia is a new love of mine, and I'll be using it extensively in this blog, maybe even eventually replacing Python.

Let us replicate the decoder function now in Julia:

Code

```julia
function decodeString(my_arr, my_alphabet)
    #=
    Parameters
    ----------
    my_arr : array
        An encrypted message.

    my_alphabet : vector
        Uppercase letters of the alphabet

    Returns
    -------
    decoded : string
        A decoded message.
    =#

    decoded = join([if x ≥ 0 my_alphabet[x+1] else ' ' end for x in my_arr])

    println(decoded)

    return decoded

end

# Declare message and alphabet
my_arr = [0 -1 [
         9] 14 20 17 13 4 24 -1 [
         19] 7 17 14 20 6 7 -1 [
         3] 0 19 0 -1 [
         18] 2 8 4 13 2 4]

my_alphabet = collect('A':'Z')

decodeString(my_arr, my_alphabet)
```

Output

```
A JOURNEY THROUGH DATA SCIENCE
```

R

- **What is it?**
  R is a programming language for statistical analysis, graphics representation and reporting. It was created by statisticians to statisticians; it specializes in statistical modelling and computing and offers a wide variety of packages for doing so.

- **Why learn it?**
  Apart from statistical applications, R can be used to tackle a variety of tasks: from web applications to mathematical and scientific computing to econometrics to quantitative analysis.

Let us define our decoder function yet again, now in R:

Code

```r
library(comprehenr)

decodeString <- function(my_list, my_alphabet) {

  paste(
    to_list(
      for(x in my_list) ifelse(x >= 0, my_alphabet[x+1], ' ')
      ),
    collapse = '')

}

my_list = list(0, -1,
           9, 14, 20, 17, 13, 4, 24, -1,
           19, 7, 17, 14, 20, 6, 7, -1,
           3, 0, 19, 0, -1,
           18, 2, 8, 4, 13, 2, 4)

my_alphabet = LETTERS

decodeString(my_list, my_alphabet)
```

Output

```
A JOURNEY THROUGH DATA SCIENCE
```

SQL

- **What is it?**
  SQL is a domain-specific language used to manage data in relational databases. Although multiple SQL systems are available, we'll concentrate efforts on MySQL.

- **Why learn it?**
  SQL is key in almost any data-related job. It's the language used to extract & transform data from relational servers.

Suppose you're already using SQL in your job for data extraction. In that case, you can rely more on it for data transformation processes since it's faster than other languages when querying and aggregating. And even if you're not currently using it, learning and experimenting with SQL can provide valuable insight into your company's data structure.

SQL is a declarative language, meaning it's easy to read and write. The code is, for the most part, self-explanatory.

Let us create a simple table and calculate the lifespan of fiction writers:

Code

```sql
CREATE TABLE WRITERS (
  FirstName varchar(255),
  LastName varchar(255),
  Birth varchar(255),
  Death varchar(255)
);

INSERT INTO WRITERS (FirstName, LastName, Birth, Death)
VALUES ('Charles', 'Dickens', 1812, 1870),
    ('Mark', 'Twain', 1835, 1910),
    ('Fyodor', 'Dostoevsky', 1821, 1881),
    ('Leo', 'Tolstoy', 1828, 1910),
    ('Joris-Karl', 'Huysmans', 1848, 1907),
    ('Dante', 'Alighieri', 1265, 1321);

SELECT *, Death-Birth
AS Lived
FROM WRITERS
ORDER BY Lived DESC
```

Output

FirstName	LastName	Birth	Death	Lived
Leo	Tolstoy	1828	1910	82
Mark	Twain	1835	1910	75
Fyodor	Dostoevsky	1821	1881	60
Joris-Karl	Huysmans	1848	1907	59
Charles	Dickens	1812	1870	58
Dante	Alighieri	1265	1321	56

Table 1. Some Fiction Writers And Their Lifespan On This Earth

Even though life in the 13th century was for sure though, Dante surpassed the average life expectancy of the time by almost double his age.[4]

Markdown

- **What is it?**
  Markdown is a simple yet powerful markup language supporting multiple syntax elements such as headers, ordered and unordered lists, images, tables, diagrams, LaTeX and even HTML & CSS code.

- **Why learn it?**
  It provides a distraction-free typing experience, enjoys compatibility with a multitude of platforms and applications, is extremely easy to learn, and is natively supported by GitHub & GitHub Gists. Also, dozens of fully-featured Markdown editors exist, such as Obsidian, Draft.js, Ulysses, and even VS Code.

We can create a simple document containing a LaTeX expression, and an unordered list.

Code

```
The Normal Distribution, also known as the Gaussian distribution, is the most widely known and
used of all distributions and is used to approximate a variety of real-world phenomena. It's
symmetric around the mean and has two parameters: $(\sigma,\mu)$

A given random variable $X$ is normally distributed with mean $\mu$ and standard variance
$\sigma^{2}$:

$$X \sim \mathcal{N}(\mu,\sigma^{2})$$

Where:
- $\sigma$ is the standard deviation.
- $\mu$ is the mean or expected value.
```

Output

The Normal Distribution, also known as the Gaussian distribution, is the most widely known and used of all distributions and is used to approximate a variety of real-world phenomena. It's symmetric around the mean and has two parameters: (σ,μ)

A given random variable X is normally distributed with mean μ and standard variance σ^{2}:

$$X \sim \mathcal{N}(\mu,\sigma^{2})$$

Where:

- σ is the standard deviation.
- μ is the mean or expected value.

LaTeX

- **What is it?**
  LaTeX is not a language per se but a software system for document preparation using the TeX typesetting system. It's tailored explicitly to scientific writing, supports a variety of syntactic objects, and can be further expanded using a vast collection of available packages. Unlike Markdown, it has a learning curve associated.

- **Why learn it?**
  It makes mathematical equations beautiful to stare at. Also, it gives the user excellent control over the formatting of documents. We can define templates and classes that, once customized, we can reuse by simply importing them into our environment.

We can write the normal distribution density function, along with a plot of the actual distribution:

Code

```latex
\documentclass{article}
\usepackage{pgfplots}
\usepackage{mathtools,amssymb}
\usepackage{tikz}
\usepackage{xcolor}
\pgfplotsset{compat=1.7}
\begin{document}
\pgfmathdeclarefunction{gauss}{2}{\pgfmathparse{1/(#2*sqrt(2*pi))*exp(-((x-#1)^2)/(2*#2^2))}%
}

\begin{tikzpicture}

\begin{axis}[no markers, domain=0:10, samples=100,
axis lines*=left, xlabel=$x$, ylabel=$f(x)$,
height=6cm, width=10cm,
xticklabels={-4, -3, -2, -1, 0, 1, 2, 3, 4}, ytick=\empty,
enlargelimits=false, clip=false, axis on top,
grid = major]
\addplot [fill=black!20, draw=none, domain=-3:3] {gauss(0,1)} \closedcycle;
\addplot [fill=gray!20, draw=none, domain=-3:-2] {gauss(0,1)} \closedcycle;
\addplot [fill=gray!20, draw=none, domain=2:3] {gauss(0,1)} \closedcycle;
\addplot [fill=blue!20, draw=none, domain=-2:-1] {gauss(0,1)} \closedcycle;
\addplot [fill=blue!20, draw=none, domain=1:2] {gauss(0,1)} \closedcycle;
\end{axis}
\end{tikzpicture}

\begin{displaymath}
f(x)=\frac{1}{\sqrt{2\pi\sigma^{2}}}e^{-\frac{(x-\mu)^{2}}{2\sigma^{2}}}
\end{displaymath}

\end{document}
```

Output

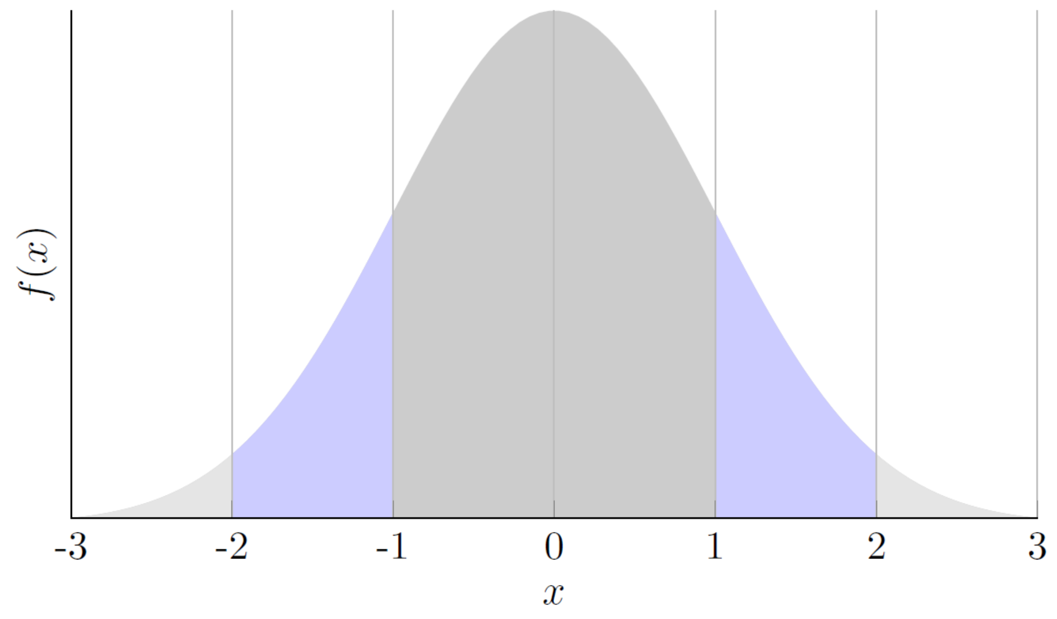

Figure 1: The PDF Of The Normal Distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Other languages & technologies

Apart from the languages above, I'll also be including the following:

- **Spark:** A multi-language engine specialized for parallel computing tasks. Apache Spark has integrations with Java, Scala, Python & R.
- **Scala:** A highly scalable language, useful for building fault-tolerant, highly concurrent systems, and an ideal companion when supported by tools like Apache Spark.
- **DAX:** The functional programming language that everybody loves (*nobody said ever*). This one is a headache but extremely useful when working with Data Analysis since it constitutes the basis of PowerBI calculations along with Power Query M.
- **Rust:** The most loved language for several consecutive years. It's used for software applications such as game engines, operating systems, file systems, browser components, simulation engines for virtual reality, and of course, Machine Learning. I'm just getting started on this one, but we will crack it open eventually.
- **C++:** The defacto language for high-performance. I'm planning on including it for guided projects in the future.

Additionally, we'll be working mostly on the following IDEs and environments:

- VS Code
- JupyterLab
- Pluto
- Spyder
- RStudio
- IntelliJ Idea

We'll also be using some additional technologies:

- GitHub & GitHub Gists
- WinSCP
- DBeaver
- Obsidian
- CodePen
- Datapane
- PowerBI
- Texmaker

§

The Motivations: Winds are changing

For better or for worse, things are changing fast; open-source is getting an insane amount of traction, ultra-high-performance languages are being cooked, obsolete social media platforms are being left behind, targeted add companies are shooting themselves in the foot, and the list goes on and on; everything is being laid out for what seems to be an exciting series of years ahead. Technology is becoming demonopolized, democratized, and fully accessible to the public, and with that come advantages but also responsibilities.

1. On the lack of rigorous mathematical knowledge

Machine Learning is an exciting discipline that years ago was reserved for academics in research environments. Now, thanks to approachable libraries and frameworks, it has become accessible to the public. This is a considerable breakthrough no doubt, but it also has its drawbacks.

Because of the increase in ML popularity, accessibility, and all the cool stuff that has been achieved with Machine Learning, there's a huge misconception that ML is the solution to every problem; someone once told me that "*the simpler solution is almost always the best solution*", and I've come to realize this is especially important to remember in an era where overcomplication and sophistication appear to be a rule of thumb.

There's a generalized rush to learn ML concepts as fast as possible; watching as many tutorials on `scikit-learn` & `TensorFlow` as possible, reading volumes of Medium articles on how to fine-tune hyperparameters and perform feature engineering, asking the same "*is this approach convenient, and why*" Stack Overflow question on ten different threads, and in the end, deploying a solution based on the 100 iterations of a Miss. Kate Connolly surviving the Titanic crash while realizing that an ML model is actually a terrible idea for the target problem and that it could be solved with a simpler approach sounds too familiar.

This recipe-style strategy has become far too common, and I firmly believe that mathematical theory should be a key part of ML adoption. Otherwise, who becomes responsible for fine-tuning the racist ML chatbot that John, the intern, deployed? Not Miss. Kate Connolly, I suppose.

2. On free education

I've always been passionate about open-source technologies and non-profit learning resources. I learned to program for free, and I know for sure it had a lot to do with what I've achieved so far.

Especially in learning how to code, I strongly believe that free learning resources provide extreme value, and I intend to do just that.

3. On cutting-edge technologies

There's too much to learn in 2023, but there are some clear patterns on future trends: it seems that high performance is the new direction, and it makes sense; we cannot deploy all those ambitious models if we don't have the workhorses to sustain them.

Also, ML is at its peak: it's disrupting how we use and write code. GitHub Copilot and transformer models are good examples.

One of the motivations for this blog is to exhaustively look for cutting-edge languages, technologies, libraries and methods that offer the highest performance and the most advanced capabilities.

4. On sustainable & green code

As ML approaches take place in this new way of solving problems, energy consumption also skyrockets. Let us look at what the ChatGPT training algorithm involved:

OpenAI trained its GPT-3 model on 45 terabytes of data. To train the final version of MegatronLM, a language model similar to but smaller than GPT-3, Nvidia ran 512 V100 GPUs over nine days.

A single V100 GPU can consume between 250 and 300 watts. If we assume 250 watts, then 512 V100 GPUS consumes 128,000 watts, or 128 kilowatts (kW). Running for nine days means the MegatronLM's training costs 27,648-kilowatt hours (kWh).

The average household uses 10,649 kWh annually, according to the U.S. Energy Information Administration. Therefore, training the final version of MegatronLM used almost the amount of energy three homes use in a year.[5]

Sustainable code is an entirely new and exciting concept to me and will have its secure place in this space.

5. On returning the favour

I remember very well the first script I wrote some years back. I also remember that without the help of that vibrant university colleague of mine who introduced me to this rabbit hole, I would not have been able to do it. My preconception of Linux was that of an obscure technology used exclusively by Mr. Elliot Alderson to cause mayhem inside EvilCorp. Then, I realized most servers out there run on Linux systems, and yes, Kali Linux does actually exist.

Help can arrive in the form of a colleague, a YouTube Channel, or a passive-aggressive Stack Overflow moderator. It really makes no difference; what really matters is that help is out there, one ChatGPT query away.

§

The Boat: Traversing through a sea of infinitude

Why create a new blog when so many free resources are already available?

The idea is to leverage two main concepts:

- **The Feynman Technique:** Helping others learn to code results in knowledge ownership, making this a win-win approach.
- **Provide added value:** By making this a responsive, structured, well-organized, distraction-free and minimalist Blog without ads, sponsorships, annoying cookies or referral fees while keeping the entire source code open and providing a business-oriented approach whenever possible.

§

The Cabins: A place to unwind

1. On the structure

This website is divided into five main pillars:

- Blog: Technical articles or essays, and can be single or serialized.
- Deep Dives: Specialized articles exploring modules, libraries, extensions and plugins in detail.
- Guided Projects: Hands-on, step-by-step projects containing difficulty level & suggested prerequisites.
- Portfolio: Single or collaborative projects containing author, date started & completed, current status, version & license.
- Documentation: Markdown documents for programming languages & technologies.

Each article contains the following:

- One associated category.
- One or more associated technologies.
- Created & updated timestamps.
- Approximate reading time.
- A comprehensive index.
- A code-output structure.
- Syntax-highlighted code blocks displaying the programming language used, with the capacity to copy and paste code.
- LaTeX expressions rendered as `.svg` objects.
- Article PDF download links.
- Embed responsive objects such as Gists, Datapanes, and more to support content.
- Dark mode toggle (*only supported in the desktop version*).
- GitHub repository links for quick access.

Each GitHub repository contains the following:

- The article's Markdown source code, including all LaTeX expressions if applicable.
- The article's PDF document.
- The source code for all scripts used.
- Plots, charts and other visual objects supporting the article.

§

The Lighthouse: How to get involved?

1. Contact me

You can reach out anytime by heading to my website's Contact form and saying hi. I'm currently based in Mexico City, so kindly expect a delay if you're in a different time zone.

2. Fork & make pull requests

All my code is open source; you're free to fork and create pull requests for any GitHub repository. I will be sure to review your commits and merge them if they add value to the project.

3. Become a Patreon

The publicity-free compromise is genuine, and I intend to enforce it. I believe monetary contributions should be voluntary and based exclusively on each one's decision.

This is why I've set up a Patron account; it will allow me to maintain this blog's associated costs without incurring other monetization techniques.

3.1 How Patron works?

Patreon is a membership platform that provides business tools for content creators to run a subscription service. Patreon for **A Journey Through Data Science** has 3 tiers:

- Enthusiast
- Inquisitive Programmer
- Mentee

3.2 Enthusiast

Aimed at helping support the creation of weekly Data Science content while getting access to discussions in the official Discord Server.

3.3 Inquisitive Programmer

Aimed to get the extra mile and learn the nuts and bolts of Data Science by getting monthly bonus material, early access to article drafts, and access to a private channel in the official Discord Server.

3.4 Mentee

Aimed at receiving guidance for your next project, orientation for working in the industry, deciding which branch of Data Science is right for you, creating your portfolio, or even debugging some code by providing access to the Patron's community, exclusive voting power & personalized requests, personalized 1-on-1s for any Data Science-related task you plan to tackle next, and access to a private channel in the official Discord Server.

If you're interested in becoming a Patreon, you can head to the Membership section, where you'll be presented with the three options. Just select the one that suits you best.

4. Join the Discord Server

I've created a Discord Server containing general and exclusive channels. The general channels are completely free, while the exclusive ones are reserved for Patreons, and include the following:

- A Mentorship channel for collaborative and 1-on-1 sessions.
- A Suggestions channel where proposals are directly translated into pipelined posts & projects. Pipeline review and adjustment will take place once per month.
- An Early Access channel where I'll release post drafts and source code before committing to the corresponding repository.

§

Conclusions

There's much to cover in the following months, and I'm genuinely excited to begin this new journey with you. It will be challenging for you as well as for myself, but in the end, I know it'll be worth the sweat.

Thank you for giving yourself the time to read all this craziness, and hope you enjoy the content as much as I enjoyed designing this Blog.

Happy coding, and until next time.

§

References

- University of Colorado Boulder, The Feynman Technique
- Wikipedia, List of Programming Languages
- Julia, Why We Created Julia
- AgeUp, A Brief History of Human Longevity
- TechTarget, Energy consumption of AI poses environmental problems

§

Copyright

§

1. University of Colorado Boulder, The Feynman Technique↩
2. Wikipedia, List of Programming Languages↩
3. Julia, Why We Created Julia↩
4. AgeUp, A Brief History of Human Longevity↩
5. TechTarget, Energy consumption of AI poses environmental problems↩