+34 684 047 366 www.esposito.com.es francesco@esposito.com.es Linkedin

Francesco Esposito

Data Scientist

Soft Skills

- Problem Solving
- Creativity
- Communication
- Critical Thinking
- Adaptability
- Collaboration
- Continuous Learning
- Curiosity

Hard Skills

- Python
- Pandas
- NumPy
- PyTorch
- SQL
- Machine & Deep Learning
- Data Visualization
- HTML
- CSS
- JavaScript
- Git
- Django
- Flask
- Web Scraping
- Data Cleaning
- Deployment

About Me

Hi, I'm Francesco. I'm a tech professional with a strong background in electronics and networking, and a growing passion for data. Over the years, I've evolved from programming embedded systems to building end-to-end data workflows — from scraping and cleaning to analysis, visualization, and web deployment. I specialize in Python, using libraries like Pandas, Requests, and Seaborn to explore data, and frameworks like Django and Streamlit to build data-driven tools. I'm meticulous, creative, and always focused on practical, real-world solutions. For me, data isn't just numbers — it's the raw material for insight, action, and storytelling.

I started my career in IT and Telecommunications, and that's where I first got close to the world of programming. Later, I moved into the electronic security industry, where I worked with CCTV systems, embedded devices, and network infrastructure. After relocating to Madrid, I took on a department lead role in a distribution company within the same field. A few years ago, I reignited my passion for programming with a Raspberry Pi project, which led me to discover Python. That moment shifted everything. I began automating tasks, experimenting with data, and eventually dove deep into the world of Data Science — a field I formally trained in through a Master's program in Data Science and Machine Learning. Since then, I've been applying what I learn in real-world scenarios — including my current job — and constantly expanding my skills. I use AI tools daily, not just as a productivity booster but as a natural extension of my workflow. I believe AI is shaping the future, and I want to be part of that transformation, not watch it from the sidelines.

Languages

Over the years, I've learned and used multiple languages in personal and professional contexts. Here you'll find a detailed view of how confident I feel in each skill area for every language I speak. Based on the CEFR levels.

	COMPREHENSION		SPEAKING	
	Listening	Speaking	Reading	Writing
English	A2	A2	B1	B1
Spanish	C2	C2	C2	C2
Italian	Native	Native	Native	Native
Neapolitan	Native	Native	Native	Native

Education

Sep 2024 - Dec 2024

Django Course: Backend Web Development

Udemy – Remote

An intensive, hands-on course focused on mastering Django for real-world web backend development with Python. Through three progressively complex projects — a personal portfolio site, a dynamic business website, and a user-authenticated web app — I built 10 reusable Django apps and learned to manage authentication, user profiles, and private messaging systems. The course emphasized practical application over theory, covering key concepts like CBVs (Class-Based Views), the MVT architecture, asynchronous requests with JavaScript's Fetch API, and deployment to cloud servers. By the end of the course, I had deployed production-ready Django projects and strengthened my backend skills with experience in testing, modular design, and scalable web architecture.

Oct 2021 - Jul 2022

Data Science & Machine Learning

ID Bootcamps – Madrid, Spain

A highly practical, industry-focused training program totaling over 450 hours, covering the entire data science pipeline — from Python programming and mathematics to machine learning, deep learning, NLP, data visualization, APIs, and web scraping. The program was designed to simulate a real work environment, emphasizing hands-on learning through continuous projects, peer collaboration, and live mentorship. It concluded with a final project where I applied the skills learned to solve a real-world problem using a complete data science workflow — from data extraction and modeling to presentation and deployment.

1999 - 2004

IT and Telecommunications

IIS G.Marconi – Torre Annunziata, Naples, Italy

This technical diploma provided a solid foundation in computer systems, software development, databases, web technologies, and telecommunications. I acquired practical skills in configuring and managing IT systems and networks, alongside theoretical knowledge in mathematics, science, and economics. The program emphasized project collaboration, data privacy, and professional use of English in a highly international context.

Experience

2022 - Present

Data Scientist

Visiotech – Madrid, Spain

While maintaining my responsibilities as a manager, I began applying machine learning models and data analysis techniques to solve real business challenges within the company. This hybrid role allowed me to bring automation and data-driven solutions into operational workflows, increasing efficiency and insight in key processes.

2020 - 2022

Support Manager

Visiotech – Madrid, Spain

As the head of the technical support department, I lead a team ensuring smooth operations and high client satisfaction in the electronic security sector. I've introduced structured processes, improved team coordination, and actively supported the implementation of new technologies.

2015 - 2020

Support Specialist

Visiotech – Madrid, Spain

I provided high-level technical support for CCTV and security systems, working closely with customers and installers to ensure product performance and integration. This hands-on experience deepened my knowledge in embedded systems, networking, and troubleshooting.

2008 - 2015

IT and Network Administrator

starSistemi – Torre del Greco, Naples, Italy

In parallel to my support role, I oversaw the IT infrastructure of the company, managing servers, networks, and systems maintenance. This position shaped my understanding of IT environments and gave me the foundation that later supported my transition into data and development.

2005 - 2015

Pre and Post-Sales / Support Agent

starSistemi – Torre del Greco, Naples, Italy

I managed both pre-sales consultancy and post-sales technical support, advising clients on system configuration and ensuring smooth deployment. My role demanded a mix of communication skills and technical knowledge, bridging the gap between commercial and technical departments.

User Onboarding Portal

🏷️ **Internal Tools** 🗓️ **April 2024**

In 2024, we started working with a new manufacturer whose devices rely on their own cloud platform to function properly. As distributors, we were given administrator access to this cloud portal, which allowed us to manage both devices and installer accounts.

However, the system had a major limitation: there was **no option for users to register themselves**. Every request had to go through our team, who had to manually create each account. At first, with just a few requests per day, the process was manageable. But as the brand became more popular, the requests grew into **dozens per day**.

This meant assigning team members to a **repetitive and inefficient task**, causing delays for customers who needed immediate access to the platform. Frustration increased, as many arrived on site without having registered and had to **wait—sometimes for hours—**for someone from our team to complete the process.

I decided to fully automate the user registration process. The idea was to create a simple web page, accessible to our customers, where they could register themselves without any support.

Beyond solving the operational issue, I saw the opportunity to turn this tool into a reference portal for installers, by integrating the complete product documentation as well.

The technical solution was based on **Flask**, which is ideal for simple and fast-to-develop applications.

Frontend: I built a clean interface using **HTML, CSS,** and **JavaScript**, with client-side validation to improve the user experience.

Backend: Flask handles form processing and makes requests to the manufacturer's cloud portal, which I had previously reverse-engineered by inspecting the network traffic between their frontend and backend.

Technical documentation: I integrated content from our knowledge base using **Zendesk APIs**, converting articles to **Markdown** automatically. I used **Flask-FlatPages** to render static pages written in Markdown directly on the site.

The architecture is minimalist: a single Flask app deployed on a server, with no database needed, as data is sent directly to the manufacturer's platform.

Technical Challenges

The main challenge was understanding how the cloud portal's API worked, since no official documentation was provided. I analyzed the original frontend's network requests, and although I had no support from the manufacturer, the structure of the portal was solid enough that I could reproduce the necessary requests from my backend without major issues.

Results or Impact

Hundreds of hours saved on repetitive tasks for our technical team

Zero waiting time for our clients, who can now register anytime, on their own

Centralized and accessible documentation, which has significantly reduced common support questions

Over 2,000 registrations processed automatically in the first year

The solution has proven to be stable, useful, and scalable, requiring almost no maintenance.

Lessons Learned

Technically, I strengthened my skills with **Flask** and learned how to work with **undocumented APIs**, which is a very valuable skill in real-world scenarios.

Professionally, I gained a deeper understanding of the importance of looking at processes from the customer's point of view and finding ways to reduce friction.

Personally, I proved to myself that it's possible to deliver real value in a short time by combining observation, decisiveness, and execution.

Used technologies:

Python, Requests, Flask, HTML, CSS, Javascript, Flask-FlatPages

Password Reset Portal

Internal Tools **March 2024** **February 2025**

The technical support team was spending a significant amount of time manually handling password reset requests for devices frequently used by customers. This process was repetitive, error-prone, and slowed down customer service. It also placed an unnecessary burden on specialized technical staff. There was a clear need to automate this task to make it **faster**, **more accessible**, and **secure**.

A public web platform was created, accessible from any browser, allowing authorized staff to generate unlock keys independently. The main goal was to simplify the process, prevent common errors, reduce waiting times, and free up technical resources to focus on more critical tasks.

The platform was built using **modern web technologies** with a modular architecture:

Languages and tools: Python for the backend, HTML/CSS/JS for the frontend, and a relational database to store logs and user data.

Architecture: Server-based web app with email-based authentication and role-based access levels.

Key features:

Unlock key generation using data like serial number, date, MAC address, or QR codes.

Input validation to ensure correct data formats and avoid mistakes.

Image upload support via drag-and-drop, file picker, or paste from clipboard.

Internal algorithms for decoding QR codes and **AI-based tools** to process data captured from user submissions.

A **multilingual help section** with tutorials for each method.

Additional tools were also included:

Logs: Detailed record of every action performed, with filters by user, date, IP address, result, etc.

Statistics (in development): Visual graphs and usage reports, to be added once enough data is collected.

User management: Create, edit, or remove user accounts with permission control. Admins cannot delete their own account to avoid critical errors.

Settings: Configurable options like email notifications, log visibility, and external platform credentials.

Results or Impact

The portal has brought clear improvements:

Faster processing: Team members can complete tasks in seconds without needing technical support.

Increased autonomy: Operators can act immediately, improving response times to customer requests.

Fewer errors: Input checks and automated processing have significantly reduced mistakes.

Better service: Customers receive faster and more accurate support, improving overall satisfaction.

Lessons Learned

Technically: I learned how to integrate different reset methods into a **modular system**, and how to develop **custom algorithms** for image and data processing.

Professionally: I deepened my understanding of **user-centered design** and the importance of **security** and **role management** in internal tools.

Personally: This project strengthened my ability to **make strategic decisions** with a direct impact on team workflows, and helped me balance **simplicity with functionality**.

Used technologies:

Python, Django, Yolo, PyTorch, Requests, API, Flask, pyzbar

VEXPA - AI Knowledge Bot

LLM September 2023 November 2023

As in many companies, in ours the most relevant information is spread across many different sources: internal documentation, technical articles, databases, external websites, and more. This dispersion turns simple tasks — like finding a precise technical answer — into slow and inefficient processes.

In the specific case of the Technical Support Service, this meant a significant waste of time searching for information that already existed but was poorly organized. There was a clear need: to access relevant answers quickly, without having to browse through multiple systems or rely on informal knowledge shared by colleagues.

The idea behind **VEXPA** was simple but powerful: to build a tool based on **large language models (LLMs)** that could answer natural language questions using both internal and external sources — going beyond the data the model was originally trained on.

In essence, the goal was to expand the capabilities of **ChatGPT** so it could interact with information not included in its training: internal documents, specific technical articles, and support websites. This would allow anyone on the team to get quick, relevant, and context-aware answers — in **any language**.

The tool was developed in **Python**, using several key components:

LLM Model: The language engine is a large language model instance, set up to respond in a conversational way.

LlamaIndex: Used to connect the LLM with external data sources, indexing content and building semantic structures for reasoning.

Requests + **Beautiful Soup**: Used to scrape content from our partner's support website and our internal Knowledge Center, extracting both structured and unstructured data.

Streamlit: Chosen to quickly build a simple internal web interface, allowing any team member to use VEXPA without technical knowledge.

General Architecture

Extraction of external and internal sources (HTML, PDFs, etc.).

Content indexing with LlamaIndex.

User interface built with Streamlit.

Conversational back-end powered by the LLM, responding based on the indexed context.

Technical Challenges

Hallucination control: We added traceability mechanisms to provide links to the sources, allowing users to verify the accuracy of the system's answers.

Real-time performance: The indexing was optimized to return results in just a few seconds, even with multiple sources.

Multilingual support: The tool was tested in several languages to ensure consistent answers beyond Spanish.

Results or Impact

Although still in beta, VEXPA has proven useful from day one:

Time savings: Technicians now get answers in seconds, without having to manually search through dozens of documents or websites.

Smarter access to internal knowledge: The tool gives new life to existing documentation, making it **easier** to reuse.

Multilingual support: Team members can ask questions in their native language, improving **accessibility**.

Foundation for future improvements: By logging questions and answers, the system helps identify gaps in current documentation and opportunities for improvement.

Lessons Learned

Technically:

I learned how to connect LLMs with external data using LlamaIndex, which introduced me to the world of Retrieval-Augmented Generation (**RAG**).

I also improved my skills in **web scraping** with Python and in **semantic content indexing**.

Professionally:

This project showed me how important it is to build solutions with the end user in mind, not just to meet technical goals.

Collecting ongoing feedback from the team was essential for quick iterations and early error detection.

Personally:

Managing the project from start to finish helped me improve my planning, task prioritization, and technical decision-making.

I also confirmed that a project can be useful even in an early stage, as long as you launch quickly and listen to your users.

Used technologies:

Python, Streamlit, LlamaIndex, Requests, Beautifulsoup, API

Newsletter Index

Internal Tools **May 2023** **June 2023**

For a while, I was handling a task that wasn't part of my official role but belonged to another department: presenting weekly product updates during the Italian team's meetings. These presentations were mostly based on the newsletters sent to customers the week before.

The problem was that there was no easy way to check if a newsletter had been published for that market. The information was scattered across inboxes with no clear index, which meant wasting time every week—both for me and for other colleagues in the same situation.

I noticed that the browser version of the newsletters followed a **consistent URL pattern**, based on language and date. This made it possible to automatically retrieve the information. I started with a small script for personal use, but once I realized that others were facing the same issue, I turned it into a **shared tool**: a web page that updates automatically and lists all published newsletters, with date filters included.

A simple, practical, and accessible solution for everyone on the team.

Languages and Libraries: I used **Python** with **Requests** and **BeautifulSoup** to scrape the company website, extracting titles, dates, and URLs of each published newsletter.

Web Interface: The information is presented in an **HTML** template, styled with **CSS**, and includes date filtering using vanilla **JavaScript**.

Automatic Updates: The **script runs periodically** to keep the index up to date.

Deployment: The project was hosted on a **VPS** with an **Apache server**, making it accessible internally through any browser within the corporate network.

Results or Impact

Weekly time savings for multiple team members

Fast, centralized access to all published newsletters

Improved preparation for internal meetings

A simple and maintainable tool that other departments found useful too

Lessons Learned

Technically, I strengthened my skills with **BeautifulSoup** and **Python** automation.

Professionally, I learned the value of sharing solutions—how a small personal improvement can turn into something **helpful for many others**.

Used technologies:

Python, Requests, Beautifulsoup, HTML, CSS, Javascript

License Plate Recognition App

Computer Vision **July 2022**

In the field of video surveillance (CCTV), artificial intelligence has been used for years to automate tasks like detecting people, vehicles, fires, and license plates. I had been supporting the development of these systems—including LPR (License Plate Recognition)—for some time, but I was always curious to understand in depth how the algorithm works to detect and extract license plate characters.

When the end of the Data Science and Machine Learning Bootcamp I was attending approached, I decided to turn that curiosity into my final project: an **Automatic Number Plate Recognition (ANPR)** app. Although the course had only briefly covered Computer Vision, I wanted to go deeper and explore the topic on my own.

My goal was to develop a complete application that could take an image or video frame, detect a license plate, and then recognize its characters. To do this, I split the problem into three independent phases:

1. License plate detection in images (Object Detection)

2. Character recognition inside the plate (custom OCR)

3. Correct ordering of the detected characters

I selected tools and algorithms that offered a good balance between accuracy, performance, and ease of deployment, since I didn’t have access to high-end hardware to run heavy models.

Phase 1: License Plate Detection

I used **YOLOv5** for plate detection, due to its good performance and low computational cost. I started by searching for public datasets, but the ones I found had poor-quality images. I tried to create my own dataset by taking photos in the street, but this approach was not realistic due to time and geographical limitations (most plates were Spanish).

After some in-depth searching, I discovered a website for license plate enthusiasts. I scraped it and collected **thousands of images**, which I then manually labeled using tools like **Roboflow**.

Model training was done iteratively. The first versions were decent, and I used those initial detections to **speed up the labeling of the rest of the dataset**.

Phase 2: Character Recognition

I initially tried generic OCR libraries (like Tesseract and EasyOCR), but they didn’t work well for plates and had poor accuracy. So I decided to train a second custom neural network, again using **YOLOv5**, this time to detect each character individually.

This part was even more time-consuming: I had to label every character, considering there were **at least 35 different classes** (letters and numbers).

Phase 3: Character Ordering

Once the model was trained, a new challenge appeared: correctly ordering the detected characters so the plate could be read like a human would. This turned out to be more complex than expected, because plates vary greatly between countries and vehicle types (cars, motorcycles, police vehicles, etc.).

User Interface

To present the results, I built a simple interface using Streamlit. While it’s limited in terms of customization, it allowed me to create a functional and easily deployable demo in a short amount of time.

Results or Impact

- Full end-to-end ANPR system developed using open-source tools

- High accuracy in both plate detection and character recognition

- Functional interface to test with images and display results

- Full automation of the process: from image input to text output

This project showed that it’s possible to build a working ANPR system without advanced hardware, and it gave me the opportunity to explore the entire Computer Vision and Machine Learning pipeline in a real-world use case.

Lessons Learned

Technical:

Custom neural network training with YOLOv5

Hands-on experience with PyTorch and OpenCV

Deep understanding of the importance of good dataset labeling in model performance

Professional:

Ability to manage a full project independently, from research to delivery

Making smart technical decisions under hardware and time limitations

Personal:

Intensive time management, especially under pressure (two months of focused work)

Validation that I can successfully handle complex, multi-step projects

Used technologies:

Python, OpenCV, Yolo, PyTorch, Streamlit, Roboflow

Serial Number Format Converter

Scripting & Automation **June 2022**

In the access control industry, authentication using cards or tags remains one of the most common solutions, even though more advanced technologies like fingerprint scanning or facial recognition have become available.

These cards usually contain a UUID (Universally Unique Identifier) stored in a 4-byte memory block. In theory, this number should be unique and easy to read, but in practice, each manufacturer interprets the byte order differently. This means that the same card can produce different UUIDs depending on the reading system.

This issue becomes especially critical during **migrations between access control systems**: previously stored serial numbers often do not match the ones used by the new system, which in most cases forces teams to physically re-read all the cards.

In environments with hundreds or thousands of users, this process is highly impractical and costly in terms of time and resources.

Since this issue appeared from time to time with our clients, I decided to develop a tool that could automatically convert UUIDs from the format used by the old system to the format required by the new one.

The idea was to create a desktop application that, starting from an Excel file exported from the original system, could batch-convert UUIDs and generate a new file compatible with the destination system.

The goal was to deliver a simple, efficient, and user-friendly solution that could be used even by non-technical users, making the migration process much smoother.

The project was developed using Python, along with the following technologies and libraries:

Tkinter and **TtkBootstrap**: to create a clean and usable GUI for Windows environments.

Openpyxl: to handle reading and writing of Excel (.xlsx) files.

PyInstaller: to package the script as a standalone executable for easy internal distribution.

The software architecture is straightforward: the user selects the Excel file, the sheet and column containing the serial numbers, then chooses the current UUID format and the target format. The software then performs a bulk conversion, generating a new Excel file ready to be imported into the new system.

One of the main challenges was dealing with the various byte interpretation schemes (big endian, little endian, byte swap, etc.), so the tool includes several conversion functions based on the most common reading modes in the industry.

Results or impact

The tool allowed us to offer a new service to our clients, enabling them to fully automate the UUID conversion process, without the need to physically re-read each card.

This resulted in:

Significant time and resource savings during migrations.

Fewer errors when transferring data between systems.

Faster and more reliable integration of new platforms.

The executable was published in the downloads section of the company website, and has been used by multiple clients in various migration projects.

Lessons learned

From a technical perspective, this was a great opportunity to strengthen my skills with **Tkinter** and apply **Openpyxl** in a real-world scenario, while also improving my understanding of low-level data representation models.

Professionally, it was a solid exercise in practical problem-solving in a real business context with direct impact on the company’s operations.

Personally, it reinforced the importance of deeply understanding the user's problem before jumping into coding. That initial clarity was key to delivering a truly useful solution.

Used technologies:

Python, Tkinter, Openpyxl, Pyinstaller

Automated Firmware Recovery Tool

Scripting & Automation **September 2020**

In 2020, during the rise of the COVID-19 pandemic, the company I work for began selling a large number of autonomous access control devices with temperature detection. These devices were designed to identify people with fever and automatically block access to certain areas, adding an extra layer of health safety.

However, the devices turned out to be quite fragile, and the firmware would often get corrupted, leaving them unusable. To avoid the high cost and delay of sending the devices for repair, the manufacturer (based in Asia) offered remote support. They would connect to our network, access the broken devices via Telnet, and manually restore the firmware.

This process worked, but it was very slow: only 3 or 4 devices could be recovered per day, while we were receiving over 10 faulty devices daily. As a result, the number of broken devices was quickly piling up.

After fully understanding the process the manufacturer followed, I decided to **automate the entire procedure**. The idea was to build a **Python script** that would connect to the devices via **Telnet**, upload the firmware using **TFTP**, and apply it automatically.

The main goal was to eliminate the need for external support and to scale the recovery process, so we could fix all the devices faster and more efficiently.

I developed the solution using **Python**, and used libraries to manage **Telnet connections** and **TFTP transfers**. The script automates these steps:

Telnet login: It tries different passwords automatically, since not all devices share the same credentials.

Firmware upload via TFTP: Once logged in, the script transfers the new firmware to the device.

Firmware installation: Using **Linux** shell commands, it applies the firmware to restore the system.

User options: The script allows the user to customize the device IP, and choose between a basic or advanced recovery mode (the latter includes several reboots for deeper recovery).

Technical challenges

Password variability: Solved by trying a list of known passwords.

Simple but flexible interface: I added command-line options for different recovery scenarios.

Network stability: Added automatic retries in case of temporary connection issues.

Results or Impact

Thanks to this script, we achieved:

Full independence from the manufacturer for device recovery.

An increase in recovery speed: from **3–4 to over 10 devices per day**.

A significant **reduction in downtime** and **backlog** of broken devices.

Improved service quality for our clients due to faster recovery times.

Lessons Learned

Technically

Learned to work with **Telnet** and **TFTP** protocols in Python.

Gained experience automating **low-level tasks** on embedded Linux systems.

Improved at writing **robust and customizable scripts** for real-world operations.

Professionally

Took a **proactive role** in solving urgent technical issues.

Improved in **independent decision-making** and **clear documentation** for teammates.

Personally

Gained **confidence** in identifying and solving operational bottlenecks.

Confirmed the value of **fully understanding manual processes** before automating them.

Used technologies:

Python, Telnet, TFTP, Linux Shell