Team members:

* Andre Juarez
* *Nicole Barrientos*
* *Marvin Quispe*

**Executive Summary**

***GreenVida Farms - Connecting Supply and Demand with Artificial Intelligence***

GreenVida Farms is an application that uses artificial intelligence (AI) to process and obtain information on agricultural products in Lima. It is designed to transform food production and consumption in Metropolitan Lima, addressing barriers to accessing fresh and nutritious food in vulnerable communities. This system connects local farmers with urban consumers, improving price negotiation, reducing food waste and strengthening food security. Leveraging historical data and predictive models, GreenVida Farms anticipates price and consumption trends, promoting a sustainable and equitable food system.

**Problem Context**

In Metropolitan Lima, approximately 3.7 million people faced food insecurity (INEI, 2024). Barriers to accessing fresh and nutritious foods, especially in urban peripheries, exacerbate malnutrition. Key challenges included:

1. Limited information on the availability of fresh products in markets, supermarkets, and communal pots (ollas comunes) as well as short-term food price trends. This complicates planning for both consumers and families facing food insecurity.
2. Inefficiencies in using data analysis tools to optimize prices and volumes of fresh products, impacting both farmers and consumers.
3. Lack of effective nutritional education programs to help consumers make healthy dietary decisions and better utilize available resources.

**Achieved Objectives**

* **Mapping Food Resources:** Using geospatial data to identify markets, supermarkets, "ollas comunes," and product availability in urban areas.
* **Predictive Models for Access to Fresh Products:** Forecasting prices, availability, and delivery frequency of fresh products in different neighborhoods.
* **Intelligent Household Chatbot:** Developing a virtual assistant to help and guide households in using artificial intelligence to create recipes based on available market products, promoting healthy and sustainable dietary decisions.

**Results**

1. **Identification of Food Resources:** GreenVida Farms mapped markets, supermarkets, and "ollas comunes," providing key information on the availability of fresh products in urban areas.
2. **Predictive Models for Price Management:** The system generated accurate predictions of fresh food prices and volumes, helping farmers and consumers better plan.
3. **Intelligent Consumer Chatbot:** Implemented a virtual assistant that recommended recipes based on available products and provided personalized nutritional information, fostering healthy and sustainable eating habits.

**Social and Environmental Impact**

* **Vulnerable Communities:** GreenVida Farms will improve access to fresh and nutritious foods for families facing food insecurity, thanks to precise information on product availability and prices in markets and "ollas comunes."
* **Housewives:** The intelligent chatbot will facilitate planning of healthy meals based on available products, empowering households with personalized and accessible recommendations.
* **Farmers:** They will enhance their negotiation capacity by accessing accurate predictions of prices and production volumes, reducing losses and optimizing planning.
* **Policymakers:** It will provide geospatial data and predictive models to serve as a foundation for designing more inclusive food policies tailored to local needs.
* **Environment:** It will reduce food waste through optimized distribution and promote a more sustainable food system by directly connecting producers and consumers.

**Recognition** GreenVida Farms received third place in the Global Urban Agriculture Hackathon “YouthforUA” in November 2024, standing out for its ability to transform data into concrete actions to improve nutrition.

**1. Technical Details**

**1.1 Overview**

This PoC demonstrates a **data-driven web application** built with **Shiny for Python**. The main objectives are to:

* Scrape daily market data from a public website (e.g., an agricultural market site).
* Perform forecasting (daily and monthly) via **Prophet**.
* Provide interactive visualizations (time series, bubble charts, bar charts) using **Plotly**.
* Enable chat-driven analytics using **OpenAI’s GPT-4** model (via langchain) and a custom “case study” prompt.
* Containerize the entire solution using **Docker** for portability.

**1.2 Core Technologies and Components**

1. **Programming Languages & Frameworks**
   * **Python 3.10**: The main language for the backend (data scraping, forecasting, web server).
   * **Shiny for Python**: Renders the user interface, manages reactivity, and serves the web application.
   * **Plotly** and **Seaborn**: For data visualization.
   * **Leafmap**: For web mapping.
   * **Selenium**: Used (optionally) for web scraping if needed to gather daily prices from external websites.
2. **Machine Learning Models**
   * **Prophet** (a forecasting library from Meta):
     + Used for daily price forecasting per category.
     + Used for monthly price forecasting for farmers’ data (monthly “chakra” data).
3. **Databases and Storage**
   * **SQLite** (optional) or **Excel** (.xlsx) files:
     + We showed two approaches: Some setups rely on local Excel files in a data/ folder; others rely on a SQLite database data\_finalv3.db.
     + The PoC has code for loading and merging daily and monthly data from Excel, while also referencing an optional SQL database engine.
4. **Cloud / Infrastructure**
   * Docker-based deployment: A single container that holds both the Shiny for Python server and the data.
   * Cloud service: The Docker container can be deployed on AWS, Azure, GCP, or any container-based environment.
5. **APIs & External Services**
   * **OpenAI GPT-4** via langchain\_openai: For conversation logic and text analysis.
   * **TavilySearchAPIWrapper** from langchain\_community: For external web search capabilities.

**2. Code Modules**

**2.1 Summary of Code Files**

|  |  |
| --- | --- |
| **Module/File** | **Purpose** |
| app.py | Main entry point for the Shiny for Python app. Sets up UI, server logic, chat agent. |
| modules/price\_data\_scrapping.py) | Scrapes daily market prices using Selenium, merges with historical data. |
| modules/volumen\_data\_scrapping.py | Analogous approach for volume data (scraping or loading from CSV/Excel). |
| modules/train\_forecast.py | Trains Prophet models (daily and monthly) on merged data, saves forecast outputs. |
| price\_analysis\_page.py | Contains utility functions to generate specific Plotly charts (bubble chart, bar chart, etc.). Also houses data merging logic or “set\_data(...)” to unify loaded data. |
| requirements.txt | Python dependencies (Shiny, Plotly, Prophet, Selenium, etc.). |
| Dockerfile | Docker build instructions (installs Python, dependencies, optionally Chrome + chromedriver). |
| data/ folder | Contains all the tabular data necessary for analysis and visualizations (Excel, CSV) |

**2.2 High-Level Module Functions**

1. **app.py**
   * **UI Definition**: Uses ui.page\_navbar(...) with multiple “pages” (Price Analysis, Chatbot).
   * **Server Logic**:
     + Loads final data from precio\_final\_merged\_latest.xlsx, volumen\_final\_merged\_latest.xlsx, and forecast\_latest.xlsx (or the SQLite database).
     + Merges data, runs reactive UI code, calls chart-rendering functions, and sets up a GPT-4 chat agent.
2. **modules/daily\_update.py** (Scraper)
   * update\_price\_data() function:
     + Loads existing aggregated data from precio\_final\_merged\_latest.xlsx.
     + Scrapes new daily data via Selenium (iterates from the “last known date” up to “yesterday”).
     + Merges scraped data back into the final Excel file.
3. **modules/volumen\_data\_scrapping.py** (Optional Scraper)
   * update\_volume\_data() function:
     + Similar approach to daily\_update but focusing on volume data from another source.
   * Merges new volumes to produce volumen\_final\_merged\_latest.xlsx.
4. **modules/train\_forecast.py**
   * forecast\_prod() function:
     + Loads the latest merged price & volume data.
     + Trains Prophet models for daily and monthly forecasts, then writes predictions to forecast\_latest.xlsx.
5. **price\_analysis\_page.py**
   * set\_data(...): Accepts DataFrames (aggregated\_precio, volumen, forecast, chakra) for a consistent “in-memory” reference.
   * Plotting Functions:
     + price\_forecast\_for\_categories(...)
     + plot\_price\_vs\_volume\_with\_monthly\_line(...)
     + plot\_category\_group\_with\_dual\_axes(...)
   * Utility: calculate\_relative\_change\_with\_flag(...) to find top-5 categories with downward trend, etc.
6. **modules/img.py**
   * Helper for managing images (e.g., saves plot outputs to tmp/images/, ensures they can be displayed in the Shiny app).
7. **Dockerfile**
   * Installs Python 3.10, Shiny for Python, Chrome/Chromedriver if scraping is done in Docker.
   * Copies the source code into /app, sets EXPOSE 3838, and runs shiny run app.py.
8. **Other relevant files**
   * .env: Contains environment variables for OPENAI\_API\_KEY and TAVILY\_API\_KEY.
   * docker-compose.yml: (Optional) If you want a simpler Docker Compose approach.
   * data/merged\_categoria\_hortalizas.xlsx: If used, ensures valid category matching for scraping.

**3. Data Dictionary**

Below is a brief overview of the main columns and tables. In the PoC, data can be stored either in **Excel** files or **SQLite**. We’ll assume the data approach to illustrate a dictionary.

**3.1. Tables and Columns**

1. **PRECIO\_VOL\_MERCADO\_DIARIO**
   * **CATEGORIA** (*str, uppercase*): The product category (e.g. “PAPA”, “TOMATE”)
   * **Extraction Date** (*date*): The day of data extraction or the day the price was recorded.
   * **precio\_prom** (*float*): The average daily price (S/.) on that date.
   * **Volumen** (*float*): The daily volume sold (in tons, or relevant unit).
   * **FORECAST\_TYPE** (*str, nullable*): If empty/null, real/historical data. If “daily”, it’s forecasted daily data.
2. **PRECIO\_AGRICULTORES\_MENSUAL**
   * **CATEGORIA** (*str, uppercase*): The product name or category.
   * **AÑO** (*int*): Year of the monthly record.
   * **MES** (*int*): Month of the monthly record.
   * **DEPARTAMENTO** (*str*): Region or department in Peru.
   * **SIEMBRA** (*date or str*) – optional: Planting start date.
   * **COSECHA** (*date or str*) – optional: Harvest date.
   * **precio\_prom** (*float*): The monthly average price that farmers receive (S/.)
   * **Extraction Date** (*date*): The date or month the data was collected.
   * **FORECAST\_TYPE** (*str, nullable*): If “monthly”, it’s forecast data. Otherwise, historical.
3. **RECETARIO\_EXCEL**
   * **platillo** (*str*): Name of the dish/meal.
   * **ingredientes** (*str*): Full text describing ingredients and quantities.
   * **preparacion** (*str*): Step-by-step instructions.
   * **aporte\_nutricional** (*str*): Description of nutritional benefits.

**Notes**:

* Any “PRESENCE\_COUNT,” “PRECIO\_MIN,” or similar columns appear in aggregated data files used by the scrapers.
* Additional or helper columns may exist for internal logic (e.g., upper/lower forecast bounds, day-of-week).

**4. Setup Instructions**

Below is a step-by-step guide to running the PoC from scratch, from source code and data:

* 1. **Prerequisites**

1. **Docker Installed**: It is recommended to run the application using Docker to avoid library dependency problems among other configurations. You can find the download link and installation manual here: <https://docs.docker.com/desktop/setup/install/windows-install/>
2. **Python 3.10**: If you want to run the application locally you need to install python 3.10. and then install the libraries required to run the app. The required libraries can be found in ‘requeriments.txt’. You can find the download link and installation manual here: <https://www.python.org/downloads/>
3. **Chrome & Chromedriver** (if you want to do Selenium scraping locally).
   1. **Initial setup**
4. **Clone the repository**:

Clone git <https://github.com/GreenVidaFarms/green-vida-farms-app.git>

1. **Set your API’s**

**You need to create a .env file with your credentials**

**gh repo clone GreenVidaFarms/green-vida-farms-app**

**4.2 Docker Setup**

1. **Build the Docker Image**:

bash

Copiar código

docker build -t my-shiny-app .

The Dockerfile installs Python + Shiny for Python and all dependencies.

1. **Run the Docker Container**:

bash

Copiar código

docker run -p 3838:3838 \

-v $(pwd)/data:/app/data \

--env-file=.env \

my-shiny-app

Exposes port 3838, maps local data folder to /app/data, and sets environment variables from .env.

1. **Browse the App**:
   * Visit http://localhost:3838 to view the Shiny application.

This documentation outlines the PoC’s technical components (programming languages, frameworks, Docker usage), code modules (their functionality and how they interact), the data dictionary (defining core schemas), setup instructions (covering local and Docker runs), and extra references (API keys, environment variables).