Data Management Plan for Colombia's Climate Variables

Gloria Carrascal

2025-02-14

Table of contents

1 Data Management Plan (DMP)

1.1 Project Overview

Project: ETL for Maximum Daily Temperature in the Atlántico region, Colombia
Date: February 14, 2025
Description: This project involves extracting and processing daily maximum temperature data from the IDEAM website for the Atlántico department in Colombia. The data is collected using web scraping with Selenium and organized through a structured ETL pipeline.

1.2 Data Sources and Collection

The primary data source is the IDEAM website: IDEAM DHIME.
Daily temperature data is extracted through automated web scraping using Selenium. The data is saved in different stages:

1. **Bronze Layer**: Raw data from IDEAM (`descargaDhime.csv` and zipped reports).
2. **Silver Layer**: Cleaned and processed data (`dailymaxtemperature.csv`, `temperature_processed.csv`, `temperature_stationary.csv`).

1.3 Data Processing and Analysis

The ETL pipeline processes the data through several steps:

1. **Data Extraction**: Automates data retrieval using Selenium (`data_extraction.py`).
2. **Data Preprocessing**: Cleans and transforms the data, ensuring consistency and handling missing values.
3. **Exploratory Analysis and Reporting**: Uses Jupyter notebooks for initial analysis (`exploratory_data_analysis.ipynb`) and Quarto for generating documents reports (`main.qmd` and `main.pdf`).

1.4 Project Structure

The project is organized into the following directory structure:

```
README.md
data
   bronze (Raw data)
   silver (Processed data)
dockerfile (Docker configuration)
notebooks (Exploratory analysis)
src (Source code for ETL pipeline)
tests (Unit tests for quality assurance)
```

1.5 Storage and Access

Data is stored locally and managed using version control (Git). A Docker container ensures reproducibility.
Access to the project is restricted to authorized users. Processed data can be shared for academic and research purposes, following IDEAM's data sharing policies.

1.6 Reproducibility and Quality Assurance

The project adheres to best practices for reproducible research, including:

1. **Version Control**: Tracking changes with Git.

2. **Containerization**: Ensuring consistent environments with Docker.

3. **Testing**: Comprehensive unit tests for data integrity and accuracy (`tests/` directory).

1.7 Long-Term Preservation

Processed datasets and documentation will be archived for future research. The project will undergo regular updates and maintenance to ensure long-term usability.

1.8 Creating and Setting Up a Script: A Terminal Command Adventure

I began my journey by creating a new directory called `missing` in `/tmp`. This directory would hold the script I was about to write. Here's how I did it:

```
mkdir /tmp/missing
```

With the directory ready, I needed to create a new file called `semester` inside `missing`. After consulting the `man` pages, I learned about the `touch` command, which creates an empty file. I used it to create `semester`:

```
touch /tmp/missing/semester
```

Now, it was time to write the script. I opened the file with a text editor (`nano`) and added the following lines:

```
#!/bin/sh
curl --head --silent https://missing.csail.mit.edu
```

- The first line (`#!/bin/sh`) is called a **shebang**. It tells the system to use `/bin/sh` to interpret the script.

- The second line runs `curl` to fetch the headers of the webpage at `https://missing.csail.mit.edu` without displaying additional information (`--silent`).

1.9 Running the Script (The First Attempt)

I tried to execute the script by typing:

```
./semester
```

However, it didn't work. The shell returned a **permission denied** error. I quickly checked the file permissions using `ls -l`:

```
ls -l /tmp/missing/semester
```

The output revealed that the script did not have execute permissions (`-rw-r--r--`). This meant the file could be read and written, but not executed as a program.

1.10 Running the Script with `sh`

To verify that the script was correct, I explicitly invoked the `sh` interpreter and provided the script as an argument:

```
sh /tmp/missing/semester
```

This worked perfectly! The script executed, and the headers for the webpage were displayed. The reason this worked is that I was directly telling the `sh` interpreter to execute the file, bypassing the need for the execute permission.

1.11 Granting Execute Permission

To make the script executable, I used the `chmod` command:

```
chmod +x /tmp/missing/semester
```

Now, when I checked the permissions again with `ls -l`, I saw that the file had execute permissions (`-rwxr-xr-x`).

1.12 Running the Script Again

With execute permissions granted, I tried running the script again:

```
./semester
```

This time, it worked perfectly without needing to invoke `sh` manually. The shell knew to use `sh` to interpret the script because of the **shebang** (`#!/bin/sh`) at the top of the file.

1.13 Redirecting and Writing the Output

The final task was to capture the "Last-Modified" date from the script's output and write it into a file called `last-modified.txt` in my home directory. I used a combination of the pipe (`|`) and redirection (`>`) operators:

```
./semester | grep "Last-Modified" > ~/last-modified.txt
```

- `|` pipes the output of `./semester` into the `grep` command, which filters for the "Last-Modified" line.
- `>` redirects the filtered output to the file `last-modified.txt` in my home directory (`~`).

1.14 Conclusion

By the end of this exercise, I had:

1. Created a directory and a script.
2. Learned how to use `touch` to create a file.
3. Understood file permissions and how to use `chmod` to make a file executable.
4. Used the `sh` interpreter to run scripts.
5. Combined `|` and `>` to extract specific output and redirect it to a file.