# Implementation Manual – Conflict in Ethiopia Before and After Abiy Ahmed

## 0. Purpose & Overview

This manual tells Cursor AI **exactly how to build a full analysis pipeline** for the blog post:

> **"Conflict in Ethiopia: Before and After Abiy Ahmed"**

We will:

- Use **ACLED** as the primary quantitative dataset.
- Define the **cut-off date** as the day Abiy Ahmed was sworn in as Prime Minister: **2018-04-02** (configurable).
- Build a **reproducible Python analysis stack** (data ingestion → cleaning → feature engineering → spatio-temporal analysis → figures & tables → blog-ready outputs).
- Generate **publication-quality figures and tables** suitable for EthiopiaConflictAnalytics.org.

> **Instruction to Cursor AI (global mindset):** - Always write **clean, modular, well-documented Python code**. - Prefer **functions and small modules** over giant scripts. - Add **docstrings**, **type hints**, and **basic tests** where helpful. - Use **sensible defaults**, but make the analysis configurable via a central `config.py`.

---

## 1. Project Structure & Environment

### 1.1 Create project folder & structure

**Goal:** Set up a clean repo where all code, data, and outputs are organized.

Suggested structure:

```
ethiopia_conflict_pre_post_abiy/
  ├─ data/
  │   ├─ raw/
  │   ├─ interim/
  │   └─ processed/
  ├─ geo/
  │   └─ ethiopia_admin_boundaries/   # shapefiles / GeoJSON
  ├─ notebooks/
  ├─ src/
  │   ├─ __init__.py
```

```
│   ├─ config.py
│   ├─ acled_client.py
│   ├─ data_prep.py
│   ├─ features.py
│   ├─ analysis_time_series.py
│   ├─ analysis_spatiotemporal.py
│   ├─ analysis_statistical.py
│   ├─ visualization.py
│   └─ utils_logging.py
├─ reports/
│   ├─ figures/
│   └─ tables/
├─ .env
├─ pyproject.toml or requirements.txt
└─ README.md
```

**Atomic Prompt 1.1 – Create project skeleton**

> *"You are an assistant helping me set up a Python project for conflict analysis using ACLED data. Create a folder structure (you can describe it in text and generate any starter files) with the following layout:* `data/raw`, `data/interim`, `data/processed`, `geo/ethiopia_admin_boundaries`, `notebooks/`, `src/` *with modules:* `config.py`, `acled_client.py`, `data_prep.py`, `features.py`, `analysis_time_series.py`, `analysis_spatiotemporal.py`, `analysis_statistical.py`, `visualization.py`, `utils_logging.py`, *plus* `reports/figures`, `reports/tables`, `.env`, `README.md`, *and a dependency file (*`pyproject.toml` *or* `requirements.txt`*). For each Python module, add a minimal docstring explaining its purpose in the pre/post-Abiy conflict analysis pipeline."*

---

## 1.2 Set up Python environment & dependencies

**Recommended stack:**

- `python` (3.10+)
- Core: `pandas`, `numpy`, `python-dotenv`, `requests` (or `httpx`), `tqdm`
- Geo: `geopandas`, `shapely`, `pyproj`
- Plots: `matplotlib`, `seaborn` (for convenience)
- Stats: `statsmodels`, `scipy`

**Atomic Prompt 1.2 – Define dependencies**

> *"Generate a* `requirements.txt` *(or* `pyproject.toml` *section) for this project. It should include: pandas, numpy, python-dotenv, requests, tqdm, matplotlib, seaborn, geopandas, shapely, pyproj, statsmodels, scipy. Also update the* `README.md` *with step-by-step instructions for creating a virtual environment and installing these dependencies, assuming the project is called* `ethiopia_conflict_pre_post_abiy`*."*

## 2. Configuration & Secrets

### 2.1 Central configuration file

**Goal:** Make the analysis configurable from one place.

Add `src/config.py` with variables like:

```
from datetime import date
from pathlib import Path

PROJECT_ROOT = Path(__file__).resolve().parents[1]

AB IY_CUTOFF_DATE = date(2018, 4, 2)
COUNTRY = "Ethiopia"

ACLED_BASE_URL = "https://api.acleddata.com/acled/read"
DATA_DIR = PROJECT_ROOT / "data"
GEO_DIR = PROJECT_ROOT / "geo"
REPORTS_DIR = PROJECT_ROOT / "reports"

START_YEAR = AB IY_CUTOFF_DATE.year - 5
# five calendar years before the cutoff (e.g., 2013)
END_YEAR = 2025    # configurable
END_YEAR = 2025    # configurable
```

(**Note:** fix any whitespace issue above when coding.)

**Atomic Prompt 2.1 – Implement** `config.py`

> *"Create a* `src/config.py` *module that centralizes configuration for the Ethiopia conflict pre/post Abiy analysis. Include:* `PROJECT_ROOT`, `DATA_DIR`, `GEO_DIR`, `REPORTS_DIR`, `ACLED_BASE_URL`, `COUNTRY`, `AB IY_CUTOFF_DATE` *(2018-04-02 as a Python* `date`*), and dynamic* `START_YEAR` / `END_YEAR` *parameters for ACLED pulls. By default, set* `START_YEAR = AB IY_CUTOFF_DATE.year - 5` *so the analysis window begins five calendar years before the cutoff date (e.g., 2013 for a 2018 cutoff), and choose* `END_YEAR` *as the latest year with stable ACLED coverage. Use* `pathlib.Path` *and add type hints and docstrings so other modules can import from* `config` *cleanly."*

### 2.2 Environment variables and secrets

Use a `.env` file to store ACLED credentials:

```
ACLED_EMAIL=your_registered_email@example.org
ACLED_API_KEY=your_acled_api_key_here
```

**Atomic Prompt 2.2 – Load secrets via dotenv**

> *"In* `src/config.py`*, integrate* `python-dotenv` *to load environment variables from a* `.env` *file in the project root. Add helper functions or constants:* `ACLED_EMAIL`*,* `ACLED_API_KEY`*. If they are missing, raise a clear error message guiding the user to create the* `.env` *file. Do not print secrets to logs."*

---

# 3. Connecting to the ACLED API

## 3.1 Read the ACLED API documentation (conceptual)

Cursor doesn't need to fetch the docs, but code must follow the pattern:

- Base endpoint: `https://api.acleddata.com/acled/read`
- Important query parameters: `key`, `email`, `country`, `year`, `limit`, `page`, potentially `event_date`, `region`, etc.
- Responses are paginated.

## 3.2 Build `acled_client.py`

**Responsibilities:**

- Provide functions to **download ACLED events for Ethiopia** for a given set of years.
- Handle **pagination** and **rate-limiting** politely.
- Cache raw responses as CSV in `data/raw/` for reproducibility.

Suggested functions:

```
fetch_acled_page(params: dict) -> dict
fetch_acled_for_year(year: int) -> pd.DataFrame
fetch_acled_range(start_year: int, end_year: int) -> pd.DataFrame
save_raw(df, path)
```

**Atomic Prompt 3.2 – Implement ACLED client**

> *"In* `src/acled_client.py`*, implement a robust ACLED API client. Requirements: - Use* `requests` *to call* `config.ACLED_BASE_URL` *with* `key` *and* `email` *from* `config`*. - Implement* `fetch_acled_for_year(year: int) -> pd.DataFrame` *that retrieves all ACLED records for Ethiopia for that year, handling pagination (e.g., using* `page` *and* `limit` *parameters). - Implement* `fetch_acled_range(start_year: int, end_year: int) ->`

`pd.DataFrame` *that loops through the years and concatenates results. - Save each year's raw data as CSV in* `data/raw/acled_ethiopia_{year}.csv`*. - Add logging and basic error handling (HTTP errors, unexpected schema). Provide docstrings and type hints."*

### 3.3 Command-line entry point or notebook for data download

Create a simple script/notebook to download all years in one go.

**Atomic Prompt 3.3 – Data download driver**

> *"Create either a small script (e.g.,* `scripts/download_acled_data.py`*) or a Jupyter notebook (*`notebooks/01_download_acled.ipynb`*) that: - Imports* `fetch_acled_range` *from* `acled_client`*. - Uses* `config.START_YEAR` *and* `config.END_YEAR`*. - Downloads all ACLED events for Ethiopia and saves a combined CSV at* `data/raw/acled_ethiopia_all_years.csv`*. Add progress reporting with* `tqdm`*. Document how to run it in the README."*

---

## 4. Geographic Data for Ethiopia

### 4.1 Acquire administrative boundaries

We need **admin boundaries** to map events by region/zone/woreda.

**Steps (manual, then codify):**

1. Download an Ethiopia administrative boundary dataset (e.g., regions, zones, woredas) as Shapefile or GeoJSON.
2. Save it under `geo/ethiopia_admin_boundaries/`.

**Atomic Prompt 4.1 – Geo data loading module**

> *"In* `src/data_prep.py`*, add functions to load Ethiopia administrative boundaries using* `geopandas`*. Example functions: -* `load_admin1_boundaries() -> gpd.GeoDataFrame` *-* `load_admin2_boundaries() -> gpd.GeoDataFrame` *Assume the shapefiles/GeoJSONs are stored under* `geo/ethiopia_admin_boundaries/` *with clear names (we can adjust once files are placed). Include docstrings explaining that these boundaries will be used for mapping pre/post-Abiy conflict patterns."*

### 4.2 Decide admin level for main maps

For core blog visuals, we likely want **Admin 1 (regions)**; optionally Admin 2 (zones) for a deeper dive.

**Atomic Prompt 4.2 – Config for map admin level**

*"Extend* `config.py` *with a setting for preferred mapping admin level (e.g.,* `MAP_ADMIN_LEVEL = 1` *for regions). In* `data_prep.py`*, ensure that the loading functions can be switched or extended to admin2 if we later decide to map at zone level."*

---

# 5. Cleaning & Harmonizing ACLED Data

## 5.1 Standardize columns and types

Key ACLED fields:

- `event_id_cnty`, `event_date`, `year`, `event_type`, `sub_event_type`
- `actor1`, `actor2`, `admin1`, `admin2`, `location`
- `latitude`, `longitude`, `fatalities`

**Atomic Prompt 5.1 – Implement** `clean_acled_data`

*"In* `src/data_prep.py`*, implement a function* `clean_acled_data(df: pd.DataFrame) -> pd.DataFrame` *for ACLED Ethiopia data. Steps: - Parse* `event_date` *as* `datetime`*. - Ensure numeric types for* `year` *and* `fatalities` *(coerce errors to NaN, then fill with 0 for fatalities). - Standardize column names to snake_case (e.g.,* `event_type`*,* `sub_event_type`*,* `admin1`*,* `admin2`*). - Drop duplicate rows. - Filter to* `country == "Ethiopia"` *just in case. Return a clean DataFrame ready for feature engineering. Document assumptions in the docstring."*

## 5.2 Save cleaned dataset

**Atomic Prompt 5.2 – Save cleaned CSV & Parquet**

*"Add a small script or notebook (e.g.,* `notebooks/02_clean_acled.ipynb`*) that: - Loads* `data/raw/acled_ethiopia_all_years.csv`*. - Applies* `clean_acled_data`*. - Saves the result as both* `data/interim/acled_ethiopia_clean.csv` *and* `data/interim/acled_ethiopia_clean.parquet`*. Use logging or print statements to show number of rows before/after cleaning."*

---

# 6. Feature Engineering – Pre vs Post Abiy & Conflict Metrics

## 6.1 Period indicators (pre vs post)

We need to tag each event as **pre-Abiy** or **post-Abiy** based on `event_date` and `AB IY_CUTOFF_DATE`.

**Atomic Prompt 6.1 – Add period features**

*"In* `src/features.py`*, implement a function* `add_abiy_period_features(df: pd.DataFrame) -> pd.DataFrame` *that: - Imports* `AB IY_CUTOFF_DATE` *from* `config`*. -*

*Adds a boolean column* `is_post_abiy` *(True if* `event_date` *>= cutoff). - Adds a categorical column* `period` *with values* `"pre_abiy"` *and* `"post_abiy"`*. - Optionally add* `years_since_cutoff` *(float) with years relative to cutoff (negative before, positive after). Return an updated DataFrame."*

### 6.2 Aggregate metrics per month and per region

We want time-series and region-level metrics:

- Events per month
- Fatalities per month
- Events/fatalities by `event_type` and `sub_event_type`
- Events per month per `admin1`

**Atomic Prompt 6.2 – Monthly & regional aggregates**

*"In* `src/features.py`*, implement: -* `aggregate_monthly(df: pd.DataFrame) -> pd.DataFrame` *that groups by* `event_month = event_date.to_period('M')` *(or year/month columns) and computes total events, total fatalities, plus separate counts for key* `event_type` *categories. -* `aggregate_monthly_by_admin1(df: pd.DataFrame) -> pd.DataFrame` *that groups by* `admin1` *and month. Ensure both functions carry a* `period` *column ('pre_abiy' / 'post_abiy') and return tidy DataFrames ready for plotting."*

### 6.3 Actor categorization (optional but useful)

Define broad actor categories (e.g., state forces, non-state armed groups, protesters, civilians).

**Atomic Prompt 6.3 – Actor classification skeleton**

*"In* `src/features.py`*, add a helper* `classify_actor(actor: str) -> str` *that maps ACLED actors into high-level categories: e.g.,* `state_forces`*,* `non_state_armed_group`*,* `protesters`*,* `civilians`*,* `other`*. Then add a function* `add_actor_categories(df: pd.DataFrame) -> pd.DataFrame` *that applies this to* `actor1` *and* `actor2` *(e.g.,* `actor1_type`*,* `actor2_type`*). Start with simple keyword rules and keep it easy to extend later."*

---

## 7. Time-Series Analysis (National Level)

### 7.1 Core time-series plots

Key figures (national level):

1. **Monthly events over time (all types)** with a vertical line at 2018-04-02.
2. **Monthly fatalities over time** with the same vertical line.
3. **Events by event_type** (stacked or faceted) comparing pre/post visually.

**Atomic Prompt 7.1 – Implement time-series analysis module**

> *"In* `src/analysis_time_series.py`*, implement functions that: - Take the monthly aggregate DataFrame and produce summary statistics comparing pre and post periods (mean events/month, mean fatalities/month, etc.). - Return small DataFrames suitable for tables (e.g.,* `summary_pre_post_overall`*,* `summary_pre_post_by_event_type`*). Include docstrings clearly stating that these functions support the 'Conflict in Ethiopia: Before and After Abiy Ahmed' blog."*

### 7.2 Publication-quality plots

Use `src/visualization.py` for styling.

**Atomic Prompt 7.2 – Visualization style utilities**

> *"In* `src/visualization.py`*, create helper functions to enforce a consistent, publication-quality style using Matplotlib and optionally Seaborn. Requirements: - Set a readable font size and aspect ratio. - Ensure axis labels and titles are clear and informative. - Provide a function* `set_publication_style()` *that can be called at the start of each plotting function. - Implement a function* `add_abiy_cutoff_vline(ax)` *that draws a vertical line at the cutoff date on time-series plots and labels it. Use vector-friendly formats (e.g., saving as PNG and SVG/PDF)."*

**Atomic Prompt 7.3 – Implement time-series plotting functions**

> *"In* `src/visualization.py`*, implement plotting functions: -* `plot_monthly_events(df_monthly, output_path)` *-* `plot_monthly_fatalities(df_monthly, output_path)` *Each should: - Produce a national time-series plot with* `event_month` *on the x-axis and counts on the y-axis. - Call* `add_abiy_cutoff_vline(ax)` *to show the pre/post boundary. - Save the figure to* `reports/figures/` *with meaningful filenames (e.g.,* `fig_monthly_events_pre_post_abiy.png`*). Ensure the plots are publication-quality (good labels, legend, grid, etc.)."*

---

## 8. Spatio-Temporal Analysis

### 8.1 Link events to admin regions

We need to spatially join ACLED points to admin boundaries to compute **events per region**.

**Atomic Prompt 8.1 – Spatial join**

> *"In* `src/analysis_spatiotemporal.py`*, implement a function* `assign_admin_units(df_events: pd.DataFrame, gdf_admin: gpd.GeoDataFrame) -> gpd.GeoDataFrame` *that: - Converts the events DataFrame to a GeoDataFrame using*

*latitude/longitude and CRS EPSG:4326. - Performs a spatial join with the admin boundaries to assign each event to an admin1 (and optionally admin2). - Returns a GeoDataFrame of events with added admin attributes. Document assumptions and how to handle events that fall outside boundaries (e.g., drop or mark as unknown).”*

## 8.2 Regional intensity maps (pre vs post)

Construct maps showing **events per admin1** in pre vs post periods.

Key figures:

1. **Choropleth of total events per region (pre-Abiy)**.
2. **Choropleth of total events per region (post-Abiy)**.
3. **Choropleth of change (post – pre) in events per region**, normalized by years in each period or population (if available).

**Atomic Prompt 8.2 – Aggregations for mapping**

*“In* `src/analysis_spatiotemporal.py`*, implement: -* `aggregate_admin1_period(gdf_events: gpd.GeoDataFrame) -> gpd.GeoDataFrame` *that groups by* `admin1` *and* `period` *to compute counts of events and fatalities. - Optionally compute per-year rates to account for different lengths of pre/post periods. Return a GeoDataFrame merged with the admin1 geometry, suitable for choropleth mapping.”*

**Atomic Prompt 8.3 – Choropleth mapping functions**

*“In* `src/visualization.py`*, add functions to create Ethiopia choropleth maps: -* `plot_admin1_events_pre_post(gdf_admin_agg, output_prefix)` *that: - Produces side-by-side maps for* `pre_abiy` *and* `post_abiy` *showing events per admin1. - Uses a colorbar, consistent color scale, and clear titles. - Saves maps as high-resolution PNG and SVG/PDF in* `reports/figures/`*. - Optionally implement* `plot_admin1_change_map(gdf_admin_agg, output_path)` *to map the change in event counts between periods. Ensure maps are legible and publication-ready (legend, north arrow optional, scale of colors suitable for print).”*

## 8.3 Simple spatio-temporal animation (optional)

Optionally, create a **GIF** or **MP4** showing the evolution of monthly events across Ethiopia.

**Atomic Prompt 8.4 – (Optional) animated map**

*“If feasible, implement an optional function in* `analysis_spatiotemporal.py` *(or a notebook) that creates a simple spatio-temporal animation: - For each month, map events as points or admin1 counts. - Combine frames into a GIF (using* `imageio` *or* `matplotlib.animation`*). - Save the animation to* `reports/figures/anim_ethiopia_conflict_timeline.gif`*. This is optional but should be coded in a way that can be skipped if dependencies are missing.”*

---

# 9. Statistical Comparison – Pre vs Post

## 9.1 Descriptive comparisons

We want clear descriptive statistics:

- Average events per month (pre vs post)
- Average fatalities per month (pre vs post)
- Distribution across event types and regions.

**Atomic Prompt 9.1 – Summary tables**

> *"In* `src/analysis_statistical.py`*, implement functions to compute summary statistics: -* `summarize_pre_post_overall(df_monthly) -> pd.DataFrame` *-* `summarize_pre_post_by_event_type(df_monthly) -> pd.DataFrame` *that calculate mean, median, standard deviation of events and fatalities per month in pre vs post periods. Save these as CSV in* `reports/tables/` *for direct inclusion in the blog (e.g.,* `tbl_pre_post_overall.csv`*,* `tbl_pre_post_by_event_type.csv`*)."*

## 9.2 Interrupted time series (simple model)

Estimate a simple **interrupted time-series model** at the national level:

- Outcome: events per month.
- Predictors: time trend, post-Abiy dummy, optional interaction.

**Atomic Prompt 9.2 – ITS regression**

> *"In* `src/analysis_statistical.py`*, implement a function* `fit_interrupted_time_series(df_monthly) -> dict` *that: - Constructs a time index (e.g., months since start). - Creates a* `post_abiy` *dummy. - Fits a simple OLS regression using* `statsmodels`*: events_per_month ~ time + post_abiy (+ time*post_abiy if desired). - Returns a dictionary with key coefficients and p-values. - Optionally, export a formatted table (e.g., as LaTeX or markdown) summarizing the model to `reports/tables/its_results.md`. Include comments noting limitations and that this is descriptive, not a causal identification strategy."*

---

# 10. Central Orchestration Script / Notebook

To make it easy to re-run everything, create an orchestration script or notebook that calls each step in order.

**Atomic Prompt 10.1 – End-to-end pipeline notebook**

*"Create a notebook* `notebooks/99_pipeline_run.ipynb` *(or a* `scripts/run_pipeline.py`*) that: - Downloads (or loads cached) ACLED Ethiopia data. - Cleans the data. - Adds features (period indicators, aggregates). - Runs time-series, spatio-temporal, and statistical analyses. - Generates all key figures and tables and saves them into* `reports/figures/` *and* `reports/tables/`*. Use clear markdown cells to describe each step so a future reader (for EthiopiaConflictAnalytics.org) can understand the pipeline."*

---

# 11. Figure & Table Specification for the Blog

This section tells Cursor **exactly which outputs** to generate for the blog.

## 11.1 Figures

1. **Figure 1 – Monthly conflict events in Ethiopia, from five years before the cutoff date to the most recent available year**
2. Data: `df_monthly` national series.
3. Visual: line plot, vertical line at 2018-04-02, shaded bands or annotation for pre/post.
4. **Figure 2 – Monthly conflict fatalities in Ethiopia, from five years before the cutoff date to the most recent available year**
5. Similar to Figure 1, but fatalities.
6. **Figure 3 – Composition of conflict event types, pre vs post Abiy (national)**
7. Bar charts (or stacked bar) comparing share of protests, battles, violence against civilians, etc. pre vs post.
8. **Figure 4 – Regional distribution of conflict events, pre vs post Abiy (Admin 1)**
9. Two choropleth maps side by side.
10. **Figure 5 – Change in conflict intensity by region (post minus pre events per year)**
11. Single choropleth map with diverging color scale.
12. **Optional Figure 6 – Simple ITS model fit**
13. Overlay fitted values from the ITS regression on the monthly events plot.

**Atomic Prompt 11.1 – Implement figure generation wrapper**

*"In* `src/visualization.py`*, add a function* `generate_all_blog_figures(df_monthly, gdf_admin_events, its_results)` *that:*

*- Calls the individual plotting functions (time-series, composition, maps). - Saves figures using consistent filenames matching the blog numbering (e.g., `fig01_monthly_events.png`, `fig02_monthly_fatalities.png`, etc.). - Returns a dictionary mapping figure labels (e.g., `"FIG1"`) to file paths. This will make it easy to reference the figures when drafting the blog post."*

### 11.2 Tables

Core tables to export to `reports/tables/`:

1. **Table 1 – Summary of monthly conflict events and fatalities, pre vs post Abiy**
2. **Table 2 – Event type distribution pre vs post Abiy (share of total events)**
3. **Table 3 – Regional averages of events per month, pre vs post Abiy (Admin 1)**
4. **Table 4 – Interrupted time-series regression coefficients (national events per month)**

**Atomic Prompt 11.2 – Table export utilities**

*"In `src/analysis_statistical.py` (or `src/visualization.py` if more convenient), implement helper functions to: - Save key summary DataFrames (pre/post overall, by event type, by admin1) as both CSV and markdown tables. - Name files consistently (e.g., `tbl01_pre_post_overall.csv`, `tbl01_pre_post_overall.md`, etc.). Ensure the markdown tables have clean column names and rounded numeric values (2–3 decimal places) so they can be pasted directly into the blog or a static site generator."*

---

## 12. Blog Draft Support (Quantitative Sections)

Once the pipeline runs, we want to help draft the quantitative sections of the blog using the outputs.

### 12.1 Narrative stubs for Cursor to generate text

**Atomic Prompt 12.1 – Generate quantitative narrative**

*"After the analysis is complete and all figures/tables exist, read the summary tables and, based strictly on those quantitative results, draft a narrative for the blog section titled `Quantitative Trends Before and After Abiy Ahmed`. The narrative should: - Describe the overall change in conflict events and fatalities per month (pre vs post) with approximate percentages. - Highlight which event types grew or declined most. - Identify which regions saw the largest increases or decreases in conflict activity. - Reference figures and tables by their labels (e.g., "Figure 1", "Table 1") but do not invent findings not supported by the data. Keep the tone analytical, neutral, and concise (800–1,000 words)."*

### 12.2 Uncertainty and limitations paragraph

**Atomic Prompt 12.2 – Limitations section**

*"Using knowledge of the ACLED dataset and the results produced, draft a short section titled* `Data Limitations and Caveats` *for the blog. Cover at least: - Reporting biases and under-reporting. - Differences in period length (pre vs post) and how we adjusted (e.g., per-month or per-year rates). - The descriptive (non-causal) nature of pre/post comparisons. Limit this section to 2–4 paragraphs, in neutral language."*

---

# 13. Quality Control & Reproducibility

## 13.1 Basic tests

**Atomic Prompt 13.1 – Add lightweight tests**

*"Create a* `tests/` *folder with a few simple tests using* `pytest` *(or a minimal custom test script) that: - Check that ACLED data loads and has expected columns. - Verify that* `add_abiy_period_features` *correctly classifies dates before and after the cutoff. - Ensure monthly aggregation functions return non-empty DataFrames with expected columns. These tests can use small synthetic DataFrames to avoid hitting the API."*

## 13.2 Reproducible run description

**Atomic Prompt 13.2 – README pipeline section**

*"Update the* `README.md` *with a* `How to Reproduce the Analysis` *section that lists, in order: 1. Create* `.env` *with ACLED credentials. 2. Download or place Ethiopia admin boundary data in* `geo/` *. 3. Run the data download step. 4. Run cleaning & feature engineering. 5. Run the analysis pipeline (time-series, spatio-temporal, statistics). 6. Inspect figures and tables under* `reports/` *. Provide exact commands or notebook names where relevant."*

---

# 14. Optional Extensions (If Time Allows)

These are extras Cursor can implement later, but the core pipeline should work without them.

1. **Population-adjusted rates** using regional population to compute events per 100,000.
2. **Alternative cutoffs** (e.g., first full year of Abiy's government) for robustness.
3. **Sub-period analysis** (e.g., Tigray war, Amhara conflict) nested inside the post-Abiy era.

**Atomic Prompt 14.1 – Configuration hooks for extensions**

*"In* `config.py` *, add optional configuration flags (e.g.,* `USE_POPULATION_RATES` *,* `ALT_CUTOFF_DATES` *) and ensure the analysis modules are written in a way that these can be activated later without major refactoring. You do not need to fully implement the extensions now, but add TODO comments showing where they would plug in."*

---

## 15. Final Checklist for Cursor AI

Before considering the pipeline complete, confirm that:

1. **Data ingestion** from ACLED works and is cached in `data/raw/`.
2. **Cleaning** produces a consistent schema in `data/interim/`.
3. **Features** correctly classify pre/post Abiy and build monthly & regional aggregates.
4. **Time-series analysis** runs and saves figures (events & fatalities) with the cutoff annotated.
5. **Spatio-temporal analysis** generates at least two regional choropleths (pre vs post) and optionally a change map.
6. **Statistical analysis** outputs summary tables and a simple ITS regression table.
7. **All key figures and tables** are saved under `reports/figures/` and `reports/tables/` with clear filenames.
8. **README** explains how to reproduce everything.

   **Final Atomic Prompt – Self-check for Cursor AI**

   *"Review the entire project and confirm that: - The ACLED client, cleaning, feature engineering, time-series, spatio-temporal, and statistical analysis modules are implemented with docstrings and type hints. - Key figures and tables required for the* `Conflict in Ethiopia: Before and After Abiy Ahmed` *blog are generated and stored under* `reports/`*. - The README provides clear reproduction steps. If any piece is missing or fragile (e.g., hard-coded paths, magic numbers), propose and implement improvements to make the pipeline robust and maintainable."*