Scoping Python in a Research Project

Using Python in research can accelerate research, improve reproducibility, and reduce manual effort. But jumping into code without a clear plan often leads to frustration, rework, and scope creep. Scoping is the process of defining *what* your project will do, *how ambitious* it is, *who* is involved, and *what infrastructure and tools* are required to support it.

This chapter helps you assess your needs, estimate the required time, and align your Python work with your overall research goals. Whether you're cleaning a dataset, automating a workflow, or publishing a reusable tool, careful scoping will save you time and increase your impact.

🔍 Define Purpose

Why are you using Python in this project? Different purposes come with different levels of complexity, time, and required skills.

- **Are we exploring data?**
  Use Python for ad-hoc visualizations, summary statistics, or sanity-checking datasets. This is usually fast, flexible, and can be done in Jupyter notebooks with `pandas`, `matplotlib`, or `seaborn`.
- **Automating a repetitive task?**
  If you're repeating the same cleaning, downloading, or formatting steps every week or month, Python can automate them. This saves time but requires attention to edge cases, input formats, and long-term usability.
- **Building a long-term reusable tool?**
  Reusable tools need more structure. You'll want modular code, documentation, and tests. Think carefully about maintainability and the future users of your code.

- **Feeding simulations or other tools?**
  Your Python code might be part of a larger toolchain—generating input files, post-processing outputs, or automating scenario runs. These tasks often require careful format matching and validation.
- **Preparing publication material?**
  Generating high-quality plots, formatted tables, or reproducible outputs for papers means polishing your code and ensuring others can run it in the future.

🎯 Define Level of Ambition

Scoping also means being honest about how far you want to take things.

- **Quick task (e.g., clean one dataset)?**
  A short, focused task with minimal reusability or documentation. Best done in a single notebook or script.
- **Reusable pipeline for team use?**
  Intermediate complexity. You'll need to organize the code into functions, manage input/output files systematically, and possibly train others to run it.
- **Fully reproducible research project or publication?**
  Highest level of ambition. Expect to include version control (Git), environment management, modular code, documentation, testing, and archiving (e.g., Zenodo, GitHub).

👥 Define Roles

Projects succeed when responsibilities are clear.

- **Who is writing the code?**
  PhD students, research assistants, postdocs, or you? Estimate time based on experience level—novices will need more guidance.
- **Who reviews, documents, and maintains the code?**
  If others will reuse the project later, someone should be responsible for documenting dependencies, reviewing changes, and maintaining the codebase.
- **Are we training others?**
  If your goal includes building team capacity, set aside time for mentorship, pair programming, and code reviews. Plan for slower progress but long-term benefits. Knowledge transfer is key here.

📥 Understand Upstream Data Dependencies

- **What type of data are you receiving?**
  - Sensor or time series data (e.g. CSV, HDF5, NetCDF)
  - Excel files (often manually edited)
  - JSON/XML from API endpoints
  - Geo-spatial data (e.g., shapefiles, GeoJSON, Geopkg)
  - 3D data or BIM (e.g., OBJ, IFC, gbXML)
  - SQL databases

- **How often does the data change?**
  - Static (e.g., one-off survey)
  - Periodic (e.g., weekly log files)
  - Live/streaming (e.g., IoT sensors or APIs)
- **What are the data quality issues?**
  - Missing values, incorrect formats, inconsistent column naming, units mismatch. Plan time for exploration and cleaning—often 30–50% of the total workload.
  - If possible, refuse messy data. You can define an dummy dataset that describes the desired variable naming and shape of the data you would like to receive and work with the data sender to avoid data quality issues.

📤 Understand Downstream Outputs & Dependencies

- **What will the Python code feed into?**
  - Data visualization dashboards (e.g., Streamlit, Plotly Dash)
  - Scientific publication (tables, plots, LaTeX)
  - Simulation engines (e.g., EnergyPlus, UrbanOpt, OpenFOAM, MATSIM, Ladybugtools)
  - Web or mobile applications
  - 3D visualization tools (e.g., Rhino/Grasshopper, Revit)
  - Data archives or open repositories

- **What formats are needed for output?**
  - PDF, PNG, CSV, JSON, GeoJSON, SQL tables, IDF files, shapefiles, etc.
  - Be clear about format requirements and check compatibility early.

🧰 Toolchain and Language Integration

- **Do you need to integrate other tools or software?**
  - Excel, Rhino/Grasshopper, Revit, ArcGIS/QGIS
  - Simulators like EnergyPlus, TRNSYS, UrbanOpt
  - Web APIs or datasets
  - Front-end frameworks (if creating dashboards or visualizations)

- **Are there existing codebases in other languages?**
  - MATLAB, R, Stata, Fortran, C++
  - If so, can they be called via CLI, re-implemented, or wrapped with APIs?
- **Is there a need to define a data model or database schema?**
  - For larger datasets, especially with relational structure (e.g., households, trips, zones), you may need to define an entity-relationship (ER) model and implement it.

🧠 Additional Scoping Questions

- [ ] What is the core purpose of the Python code in this project?
- [ ] What kind of outputs will be generated?
- [ ] What types of data are coming in, and what format/quality are they?
- [ ] Are upstream sources reliable, stable, and well-documented?
- [ ] Is the code standalone, or part of a larger toolchain?
- [ ] Who are the users of the outputs?
- [ ] Will this code need to be rerun or extended later?
- [ ] Will the code integrate with databases or APIs?
- [ ] Do we need additional personnel (RA, assistant, data engineer)?
- [ ] How will outputs be shared (paper, dashboard, repo)?
- [ ] What tools and languages does it need to connect to?

🧪 Example: Building Energy Monitoring Dashboard

Goal: Monitor HVAC energy use across multiple university buildings, visualize trends, and support operational decisions.

🔧 Purpose

Create a semi-automated pipeline that:

- Fetches sensor data from a building management system (SQL)
- Aggregates daily energy use by zone
- Combines building geometry for spatial overlays
- Displays results in a web dashboard and exports summary CSVs

🚀 Level of Ambition

High-to-very high:

- Used weekly by facilities staff
- Shared on internal dashboard
- Must be robust and documented

👥 Roles

- PhD student develops data pipeline and design dashboard
- Facilities team provides weekly Excel sheets
- Project assistant supports data extraction, processing, dashboard testing and documentation
- IT staff manages SQL database credentials
- Front end developer works with IT team to take dashboard design and make it

📥 Upstream Data

- SQL database with hourly HVAC sensor data per room
- Excel files from maintenance team (room IDs, floorplans)
- 2D geometry from Scanned drawings (PDF → Rhino)
- Weather API for daily outside temperature (JSON)

📤 Downstream Outputs

- CSV with daily kWh per zone
- Streamlit dashboard with graphs and key indicators
- JSON overlays for 3D geometry visualization in Grasshopper
- Monthly report plots (PNG) for presentations

⏳ Time Estimate

- No one can give you an accurate time estimate unless you know exactly what you need.
- What you could do is describe your project to an expert in levels of ambition
- The range between the lowest and highest level of ambition should inform your time estimate

⚠️ Pitfalls to Avoid

- When budgeting almost assume that you have underestimated the project.
- Technical projects with code development will take longer because there are many unknowns with data quality and availability.
- Always plan for an MVP
- If you are building a tool that will be used by industry professionals, hire a professional developer.

✅ Summary

Scoping isn't just about writing code—it's about making sure you write the *right* code, with clear expectations, appropriate tools, and realistic timelines. Taking the time to define your inputs, outputs, team roles, toolchains, and dependencies helps avoid overwork and under-delivery. The earlier you scope well, the smoother your research project will go.