From Analysis to Deployment

Building & Publishing Data-Driven Software

Atilio Barreda II | Sample syllabus | 14-week software studio | Revised November 2026

What the course builds

A notebook that runs on your laptop is not yet a tool another person can use. In this studio, students take one small analysis through a reproducible data pipeline, an interactive interface, and a release that someone else can operate. The project begins in week one and changes as its assumptions meet actual data and users.

The technical work is also work on interpretation. What does a filter leave out? Does a blank value mean zero, unknown, or withheld? Does a chart invite a conclusion the query cannot support? We return to these questions when defining the data model, designing the interface, and deciding what to publish.

Starting point and tools

Students should have a small analysis or some experience with Python or JavaScript. Early labs include starter code and a common dataset; this is not a complete introduction to programming. The core path uses Python and uv, SQL in DuckDB, HTML/CSS, JavaScript, Vue, and Git. FastAPI and PostgreSQL are introduced for projects that need a service or shared persistent data. A static, browser-only project is a valid final result. No paid cloud account is required; service exercises can run locally or on an instructor-provided host.

By the end

- Reproduce a result from documented inputs, a locked environment, and a repeatable command.
- Write and test data transformations separately from file access, network calls, and interface state.
- Build an accessible interface whose labels, counts, and exports agree with its underlying queries.
- Decide whether a project needs a backend, and explain the cost of that decision.
- Release, document, and hand off a project with a rollback and a plan for its data.

Assessment

Reading notes, project logs, and discussion: **20%**. Labs and small releases: **30%**. Final project, including proposal and progress reviews: **40%**. Peer testing and code review: **10%**. A well-scoped, tested static project can earn full credit. Additional services or frameworks do not earn credit simply by being present.

Weeks 1-5 | From a question to a working page

1. Define the claim and the audience

Choose one question your analysis can support and one person who might use it. Inspect the source, license, collection method, and missing values. Start a Git repository, README, and project log. Read selected passages from Haraway's "Situated Knowledges" alongside the dataset's own documentation. **Ship:** a one-page brief stating what the project can and cannot establish.

2. Make the analysis repeatable

Use uv to define the Python environment. Turn notebook cells into small functions with explicit inputs and outputs; keep source files unchanged. Add a repeatable CSV or Parquet export and a first test. **Ship:** one command that rebuilds a documented result, plus a source note recording when and how the inputs were obtained.

3. Query before you draw

Use DuckDB to inspect types, duplicates, missing values, joins, and grouped counts. Compare row counts before and after each operation. Choose a stable identifier and write down what one row represents. **Ship:** a small set of named queries, a data dictionary, and a regression test for a misleading join or aggregation.

4. Build the first usable view

Use semantic HTML, CSS, and SVG to show a result with its source, units, and limitations. Provide a table or text alternative to the chart. Test narrow screens and keyboard navigation before adding elaborate controls. **Ship:** a page that a peer can read without opening your notebook or guessing what a number means.

5. Add interaction without a second source of truth

Load a versioned JSON export with JavaScript. Keep filtering, sorting, and summary calculations in small functions; derive the table and chart from the same selected records. Handle loading, empty results, and failures separately. **Ship:** a working filter and a test showing that the displayed count, chart, and downloaded result agree.

Weeks 6-10 | Interfaces, services, responsibilities

6. Organize the interface with Vue

Separate reusable components from data transformations. Use props, events, and computed values to make state changes traceable. Keep controls labeled and preserve focus when results change. **Ship:** a Vue version of the working page, with a short explanation of what the component boundary makes easier to understand or test.

7. Release something small

Build and publish a static version on an available host, such as GitHub Pages, or demonstrate it on a provided local server. A peer starts from the README and checks one result against the input. **Ship:** a tagged first release, a recorded issue from peer testing, and a scoped plan for the final project. Discuss the difference between access to a URL and access to the reasoning behind it.

8. Decide whether an API earns its place

Build a small FastAPI exercise with validated query parameters and a documented response model. Test invalid input, empty results, and an unavailable service. Compare it with the static export from week seven. **Ship:** an API contract and an architecture decision. Keep the final project static if a service adds no necessary capability; never pass arbitrary user SQL to the database.

9. Model persistent data

Use a supplied PostgreSQL exercise to work with keys, constraints, migrations, and transactions. Distinguish an analytical snapshot from a database that accepts shared writes. **Ship:** a tested schema and one failed operation handled safely. For the final project, document who can write data and whether persistence is needed at all.

10. Review what should leave the machine

Read Glissant's "For Opacity" alongside a publication review of the project. Separate public fixtures from restricted source material. Check browser bundles, repository history, exports, logs, and error messages for unintended disclosure. **Ship:** a data-access plan and a demonstration using permitted or synthetic inputs. Keep secrets out of client code; explain the limits of redaction and later deletion.

Weeks 11-14 | Testing, release, handoff

11. Test the claim, not just the happy path

Use pytest for the data/service code and Vitest for JavaScript transformations where applicable. Add checks for missing values, duplicate identifiers, failed requests, and disagreement between views. Have a peer try the interface on a narrow screen and without a mouse. **Ship:** a short test report tying each check to a possible wrong result or unusable interaction.

12. Make deployment reversible

Separate the build from runtime configuration. Static projects deploy their generated assets; service projects use a documented container or managed runtime. Check HTTPS, environment variables, dependency versions, and a minimal health check. **Ship:** release instructions and a demonstrated rollback to the previous working version. Use the provided environment rather than opening a paid account for the assignment.

13. Hand it to someone else

Ask a peer to install or open the project without verbal coaching. Record what they expected and where the interface misled them. Estimate hosting and maintenance needs, and specify how to export or remove the data. **Ship:** a revision, a handoff note, and a list of limitations that remain. Keep a backup and restore procedure if the project stores mutable data.

14. Final demonstration and review

Demonstrate the project from a fresh checkout or a documented build. Trace one visible result back to the query and source records. Show an error case and explain one decision changed through testing. **Submit:** the working release or local demonstration, source, tests, data documentation, and a 1,000-1,200-word engineering and interpretation note.

What the final project must explain

The handoff identifies the intended audience, source and license, data transformations, interface defaults, and publication boundary. It includes setup commands, sample configuration without secrets, a release/rollback procedure, and a named maintenance responsibility. If the work should not be public, demonstrate it locally with safe fixtures and explain why.

Suitable projects include an image-collection browser, a small public-records explorer, or a classroom tool for comparing clustering results. The assessment follows one question through the system: can another person understand where a result came from, use it without avoidable confusion, and recognize what it does not establish?

References used at the point of need

Technical documentation is read alongside the operation it supports. We use one workable release route, not a tour of hosting vendors. The conceptual readings are short selections tied to source interpretation and disclosure decisions; students do not need prior coursework in critical theory.

Data and services

- uv: working on projects: environments, project metadata, lockfiles, and repeatable commands.
- DuckDB documentation: SQL over analytical data and documented import/export operations.
- PostgreSQL tutorial: relational tables, constraints, and transactions for the persistence exercise.
- FastAPI tutorial: validated inputs, response models, errors, and testing.
- pytest documentation: small regression tests for the pipeline and service.

Interfaces and publishing

- MDN Web Docs: HTML, CSS, JavaScript, HTTP, and browser behavior.
- Vue guide and Vitest guide: components, state, and JavaScript tests.
- W3C accessibility tutorials: labels, structure, images, tables, and usable controls.
- Pro Git and GitHub Pages documentation: version history, collaboration, and the static-release exercise.

Reading the work's assumptions

- **Donna Haraway.** "Situated Knowledges". *Feminist Studies* 14, no. 3 (1988). Used when specifying what a source and a result can support.
- **Edouard Glissant.** *Poetics of Relation*, translated by Betsy Wing. "For Opacity." Used when deciding what information a public tool should make available.

Library access would be arranged for assigned book excerpts. Public documentation is not treated as a substitute for testing the project's actual behavior. Students keep brief decision notes: what they chose, what evidence informed it, and what would make them reconsider.