
University of Chicago · Financial Mathematics Program

FINM 32800

Data Pipelines for Quantitative Research

Syllabus — Autumn 2026

By Jeremiah Bejarano



Updated October 1, 2026.

Course Description

Data Pipelines for Quantitative Research is a hands-on course centered on building reproducible analytical pipelines: automated and fully reproducible workflows that carry a quantitative analysis from raw data to published result. This course examines every stage of the pipeline, from data extraction and cleaning (extract, transform, load, or ETL), through data validation, exploratory analysis, visualization, modeling, and finally to publication and deployment. In industry, this work is sometimes called *productionizing* research, pairing developers with researchers to translate ideas into practice. The course teaches the core set of tools used to build such pipelines, tools which are common across computing and data science: build automation and CI/CD, dependency management, SQL, unit testing and automated data-quality checks, the Linux command line, SSH and working with remote machines, Git for version control, peer review through GitHub pull requests, and experiment tracking and model monitoring (basic MLOps).

These skills are taught through a series of case studies, each of which introduces students to a new set of tools and a new key financial data set: pricing and fundamentals from CRSP and Compustat, options data from OptionMetrics, corporate bond transactions from FINRA TRACE, intraday trades and quotes from NYSE TAQ, and order book data from CME Globex.

Prior experience at an intermediate level with Python and the PyData stack is assumed.

1 Course Information

- **Instructor:** Jeremy Bejarano, jbejarano@uchicago.edu
- **Class meetings:** One session per week, in person. The day, time, and room are listed on Canvas.
- **Instructor office hours:** By appointment. Book a one-on-one slot through the link posted on Canvas.
- **Teaching assistant:** Announced on Canvas, along with TA office hours.
- **Textbook:** All lecture notes and case studies are published on the course website, <https://finm-32800.github.io/>. Chapters are posted incrementally as the quarter proceeds.
- **Canvas:** Used for grades, announcements, and links only. Homework and notes live on the course website.
- **Code:** All code for the course, including the code that generates the textbook and the homework assignments, is posted in repositories under the [finm-32800 GitHub organization](#). Many lectures are accompanied by small runnable examples collected in the [in-class examples repository](#). Clone it and follow along during class.
- **Questions and discussion:** Post questions on the [GitHub Discussions page](#) rather than emailing, so that the whole class can benefit from the answers.

Prerequisites

- Python at an intermediate level, including pandas and the broader PyData stack (*Python for Financial Data Science*, or equivalent experience)
- Basic probability and statistics
- Familiarity with financial data and markets (*helpful*)

2 What This Course Teaches

Consider the following excerpt from a job posting for the data engineers who work alongside quantitative researchers:

“Partner with researchers to produce high-value datasets”

“Translate high-level market research concepts into scalable processes that further transform the data”

—Citadel Securities, Senior Research Engineer (Data) ([full archived posting on the course website](#))

This is the job that this course trains students to do. Different companies assign different titles to this role. Commonly used titles include research engineer, data engineer, and quantitative developer on a data team, but the work is the same: take research that lives in a researcher’s head or in a published paper and turn it into a production data pipeline that runs reliably every day. In short, this course teaches students to *productionize* research, to borrow industry’s word for it.

The course’s central exercise, which is to replicate results from a well-known or classic peer-reviewed academic finance paper, exists for three reasons. First, replicating and productionizing the paper’s main analysis mimics the task described in the job posting above: the paper plays the role of the researcher in that it hands the student a high-level research concept and the student translates it into a scalable, automated process that carries raw data to a finished, reproducible result. Second, the course spends time replicating several papers throughout the quarter. This gives students a chance to gain real-world experience with several canonical finance datasets, which is also a skill employers list in their postings. Third, reproducibility is fundamental to good science. A result that cannot be rebuilt from its raw data cannot be trusted, in a journal or on a trading desk. Every case study in the course works this way (e.g., Week 2 rebuilds the Fama–French 1993 portfolios from raw CRSP and Compustat data), and the final project does it end to end.

A data pipeline is the path from raw source data to a finished analytical product: extraction, cleaning, transformation, validation, analysis, documentation, and publication, automated end to end. Every case study in this course is one of these pipelines, and the final project is one built from scratch. Section 3 shows the hiring evidence behind this focus, in the firms’ own words.

The tools involved are rarely taught in a traditional computer science curriculum. A CS degree covers algorithms, operating systems, and programming languages, while the practical skills of the computing ecosystem, such as the command line, shell scripting, data wrangling, and version control, are usually left for the individual to pick up alone. (This gap was the motivation for a supplementary course at MIT, “[The Missing Semester of Your CS Education](#).”) This course provides a structured overview of these tools and demonstrates, by example, how they are used to collect, clean, and analyze financial data, to collaborate within a team of researchers, and to create analyses that outside users can easily reproduce.

Objectives. The work described in postings like the one above has two halves, building the pipeline and knowing the data it carries. The course takes its two objectives from them directly:

- **Build the pipeline.** Teach a core set of software tools and the principles behind assembling them into analytical workflows that are reproducible and scalable from end to end. This is Citadel’s “design, create, automate, and maintain custom data pipelines,” taught one tool at a time. In production, an analysis creates value only when it is repeatable and its pipeline runs reliably and predictably, and the same conviction underlies the whole toolkit: reproducibility is fundamental to good science. This toolkit serves as a technical foundation for whatever subfield of quantitative finance or data science a student goes on to pursue.
- **Know the data.** Give students hands-on experience with the financial data sets quantitative researchers use every day. Some job [postings](#) explicitly ask for candidates to demonstrate an “understanding of the financial data vendor landscape.” Each data set requires domain-specific knowledge to clean and interpret properly, and each poses its own computational challenges to use effectively: NYSE TAQ means working with intraday records at a scale where pandas begins to struggle, and CME Globex’s intraday order book feeds mean confronting exchange-specific quirks head on. Each case study is designed to give students the basic knowledge and tools to use these data sets well.

3 The Jobs This Course Targets

As evidence of the importance of these skills, consider three recent postings from some of the most selective firms in quantitative finance (Figures 1–3). The highlighted skills—pipelines that ingest and transform data, SQL and DataFrame libraries, Python on Linux, automated data-quality checks—are the skills this course teaches. These three are not outliers. The same language appears at many other firms. Postings from ten firms in all are archived on the [course website](#), each with its complete text, a screenshot, a PDF snapshot, and a Wayback Machine link, and all ten are quoted in the week-by-week outline below.

4 Course Outline

The course meets for nine lecture weeks, one per chapter of the textbook, followed by the final project presentations in week ten. Lecture 0 is a preview of the course; its chapter and Homework 0 can be done on your own. Each week below opens with a short description of its tools and why they matter, followed by quotes from the postings of those ten firms. Each quote links to the archived posting on the course website.

Week 1: Git, GitHub, and virtual environments

Git is the version control system used essentially everywhere in industry: it records every change ever made to a codebase, lets a team work on the same code without overwriting each other, and makes any past version recoverable. GitHub is the platform where those repositories live and where teammates review each other’s changes. A virtual environment (we use conda) pins the exact package versions a project depends on, so an analysis runs identically on every machine. This is the vocabulary behind the postings’ phrases below: “GitOps” means driving deployments entirely through Git, and “peer code reviews” happen through the GitHub workflow you will use every week of this course. Our first case study is the CAPM: we connect to WRDS for the first time and rebuild the CRSP market portfolio and the S&P 500 from their constituents. Notice that the postings also treat knowledge of the data vendors themselves—who sells which dataset and what its quirks are—as a skill in its own right; the course’s case studies (CRSP, Compustat, OptionMetrics, Databento, FINRA TRACE, NYSE TAQ) are chosen to build exactly that.

- “Demonstrated experience working on an Agile team employing software engineering best practices, such as GitOps and CI/CD, to deliver complex software projects” — Akuna Capital
- “An understanding of the financial data vendor landscape and product offerings” — DRW

Week 2: ETL and pipeline automation with PyDoit; SQL

ETL—extract, transform, load—is the industry’s name for the core pattern of data work: pull raw data from its source, clean and reshape it, and store the result where analysis can use it. A build-automation tool (we use PyDoit) chains those steps into a dependency graph of tasks, so the entire pipeline reruns in the correct order with a single command and skips any step whose inputs haven’t changed. Our case study rebuilds the Fama–French (1993) portfolios from raw CRSP and Compustat data this way. When Citadel asks for “ETL dev” and DRW asks for “ingestion pipelines,” this pattern—not any single tool—is what they mean.

SQL is the standard language for pulling data out of databases, and pandas and Polars are the Python “DataFrame libraries” for reshaping that data once you have it—when a posting names


CITADEL | Securities

[Who We Are](#)
[What We Do](#)
[News & Insights](#)
[Careers](#)



Senior Research Engineer (Data)

Miami

Job Description

Role Overview

Data Research Engineers in Citadel Securities are responsible for the curation of large-scale structure and unstructured data and for data assimilation across a multitude of sources. They focus on the translation of high-level conceptual queries to efficient back-end queries on extremely large-scale data collections. Our data science is supported by state-of-the-art cloud-based services that scale out. The advanced queries will be crucial in enriching alpha signals for our multi-billion-dollar Systematic Business and research function, enabling us to deliver a competitive edge against competitors.

Our team of talented Data Research Engineers works hand-in-hand with the QR teams to design the most insightful queries. In addition, they closely work with business operations and QRs in order to identify and evaluate the most promising datasets from anywhere in the world and available in numerous modalities. We are looking for individuals passionate about driving business impact through data!

Responsibilities

- Act as data owner of several datasets each with its own unique schema and delivery frequency
- Partner with researchers to produce high-value datasets
- Design, create, automate, and maintain custom data pipelines
- Extract the raw data
- Clean and normalize the data
- Translate high-level market research concepts into scalable processes that further transform the data
- Handle all edge cases
- Develop high-performance schemas and queries
- Setup "data checks" and alerts to determine when the data is "bad"
- Load the processed dataset to a format easily digested by quantitative researchers and models
- Develop tooling to facilitate the development and monitoring of custom data pipelines
- Setup analytical infrastructure to facilitate exploration and visualization of datasets
- Large scale cloud development of applications as a service

Skills and Qualifications

Required:

- Demonstrated experience driving meaningful business outcomes through data engineering
- Prior ownership of a data project (or at least a large part of it)
- Strong attention to detail
- Good analytical and quantitative abilities
- Demonstrated ability to quickly learn new technologies and skills
- Experience with ETL dev

Preferred:

- Experience with data manipulation and query design and optimization
- Experience in time series forecasting and machine learning techniques
- Strong data science skills
- Strong coding skills: proficiency in Python, SQL DBs, Cloud, schedulers, containers, CI/CD, software packaging
- Strong ability to communicate with other stakeholders (e.g., data vendors, QRs, etc.)
- Data Build Tool (DBT)
- Education: BS degree in Computer Science, Mathematics, or related Computer Engineering or Science curriculum

Figure 1: **Citadel Securities — Senior Research Engineer (Data), Miami.** “Partner with researchers to produce high-value datasets”; “Design, create, automate, and maintain custom data pipelines”; “Experience with ETL dev”; “proficiency in Python, SQL DBs, Cloud, schedulers, containers, CI/CD, software packaging”; “Data Build Tool (DBT).” ([Full archived posting on the course website.](#))



WHO WE ARE WHAT WE DO ▾ THE LATEST CULTURE ▾ JOIN JANE STREET ▾ 🔍

[Join Jane Street](#) > [Open roles](#) > [Experienced candidates](#) > Job description

Data Engineer

LOCATION
London

DEPARTMENT
Technology

TEAM
Software Engineering

[Apply](#)

[Share this job](#)

About the Position

We are looking for a Data Engineer to turn messy, real-world datasets into reliable inputs for research and trading. Jane Street depends on data from a wide variety of sources to understand markets and the global economy—think world news, decades of weather patterns, deidentified credit card spending, or packet captures of stock exchange market data feeds. Clean, well-structured data is the difference between noise and a profitable trading signal.

You'll work closely with trading desks to identify Jane Street's data needs, evaluate datasets, and build pipelines that ingest and transform external data. You'll figure out what data means and how best to represent it. We build our own tools and software when needed, so you'll also help shape the systems you use every day.

This is a collaborative role, but you'll own your work from end to end. It requires a mix of engineering, data analysis, and product sense. You'll develop deep expertise in your data domains and use that understanding to decide what's worth building next.

About You

- Have strong programming skills in any language (Python a plus); you write clear, correct, and maintainable code
- Proficient with SQL or DataFrame libraries like pandas or Polars
- Intellectually curious with a meticulous approach to data investigation; you enjoy the detective work required to make sense of unfamiliar datasets
- A clear and concise communicator
- Fluency in English required

If you're a recruiting agency and want to partner with us, please reach out to agency-partnerships@janestreet.com.

Figure 2: **Jane Street — Data Engineer.** “...build pipelines that ingest and transform external data”; “Proficient with SQL or DataFrame libraries like pandas or Polars.” ([Full archived posting on the course website.](#))

 hudson river trading
[< Back to jobs](#)

Data Production Engineer

📍 London, United Kingdom; New York, NY, United States; Singapore

Apply

Hudson River Trading (HRT) is looking for a Data Production Engineer to join our Data team. Data is at the core of everything we do at HRT; we excel at deriving deep insights from all types of data, allowing us to achieve consistent success in a dynamic market.

This role is an opportunity to work directly with live trading teams to support one of the largest automated trading systems in the world. You will write automation, explore data, work closely with our research and trading teams, and interact with a variety of external partners such as data providers, brokers, and exchanges. In addition to being a critical part of the trading process, you will have the opportunity to acquire, analyze, and prepare data for quantitative research.

Responsibilities

- **Data Engineering:** Write tools to classify, onboard, and reconcile data. Onboard datasets, explore data, and automate tasks using a modern Python data stack
- **Data Analysis:** Parse, analyze, and understand data sets. Perform data reconciliations, validations, and quality checks. Identify and develop new processes within the data request process to enrich data. Assist our researchers in cleaning and featurizing data
- **Data Debugging:** Find anomalies in derived datasets and trace the issues back to their source. This can include using a mix of deductive reasoning, technical analysis, and communicating with multiple stakeholders in a data pipeline
- **Production Support:** Provide proactive oversight of our data pipeline, handle inquiries from internal customers, and resolve issues under efficient turnaround times

Profile

- Track record of being detail-oriented and thorough
- You excel in problem solving and researching large datasets to resolve complex issues
- You have a collaborative attitude that lends itself to cross-team customers and projects
- You thrive in the fast-paced environment of a daily live trading operation

Qualifications

- 2+ years of experience in a data engineering/science role OR a degree in data science or a similar discipline
- Experience in Python strongly preferred
- Experience managing ETL pipelines is a plus
- Experience with financial datasets (e.g. Refinitiv, S&P, Bloomberg) is a big plus
- Comfortable with the Linux command line
- Experienced in at least one SQL dialect (PostgreSQL, MSSQL, MYSQL) and able to use others as needed
- Able to provide technical support in a production trading environment

Figure 3: **Hudson River Trading — Data Production Engineer.** “...automate tasks using a modern Python data stack”; “Perform data reconciliations, validations, and quality checks”; “Experience managing ETL pipelines is a plus”; “Experienced in at least one SQL dialect.” ([Full archived posting on the course website.](#))

any of these, it is describing the first mile of a pipeline. We practice on the real thing: WRDS (Wharton Research Data Services), the gateway to the databases used across the industry, which you first connected to in week 1. This week we pull CRSP and Compustat with SQL.

- “Design, create, automate, and maintain custom data pipelines” — Citadel Securities
- “3+ years of demonstrated experience designing and implementing ingestion pipelines” — DRW
- “Experience with ETL dev” — Citadel Securities
- “Proficient with SQL or DataFrame libraries like pandas or Polars” — Jane Street
- “Experienced in at least one SQL dialect (PostgreSQL, MSSQL, MYSQL) and able to use others as needed” — Hudson River Trading

Week 3: Reproducible reports — ChartBook, GitHub Pages, and LaTeX

A pipeline’s output has to land somewhere people can actually see it. GitHub Pages publishes a website directly from a repository, and ChartBook organizes a project’s charts and tables into a browsable, always-current catalog. This is what the postings call “dashboards” and “analytical infrastructure.” For written results, we use LaTeX, the typesetting system behind professional-quality quantitative reports and nearly every academic paper in finance. The habit this week builds is treating a report not as a one-off document but as a product the pipeline regenerates automatically every time the data updates. The course also leans on the pull request, which is how real teams change code: you propose an edit, a teammate reads it line by line and comments, and only after review does it merge. That is the “peer code review” in AQR’s posting. The postings below make a further point worth taking seriously: firms ask for communication skills in the same breath as technical ones, because a pipeline’s results only matter if the researcher, trader, or client on the other end understands them. Our case studies are two replications: the U.S. Treasury yield curve of Gürkaynak, Sack, and Wright (2007), estimated from CRSP Treasury quotes, and the expected path of the policy rate read from fed funds futures, the way the CME FedWatch tool does.

- “Setup analytical infrastructure to facilitate exploration and visualization of datasets” — Citadel Securities
- “Clear written and verbal communication skills to translate complex technical work for business stakeholders and collaborate in agile, cross functional teams.” — JPMorganChase
- “Strong command of engineering best practices, including code quality, design documentation, peer code reviews, automated testing, and test coverage” — AQR

Week 4: Python packaging and documentation with Sphinx

A Python package is code organized for reuse and installation—the difference between a folder of scripts only you can run and a library a teammate can `pip install` and build on. Packaging forces the habits employers list under “software packaging”: a clean project layout, explicitly declared dependencies, and versioned releases. Documentation completes the package. Sphinx, the same tool that builds the documentation for pandas and the course’s own textbook, generates browsable reference pages from the code itself, so the docs live next to the code and never fall behind it. This

is also the week that turns your coursework into something you can hand a hiring manager: an installable, documented library, with your final project as the natural first candidate. Our case study is an installable package that recovers option-implied crash probabilities from the volatility smile, following Martin and Shi (2025), using the OptionMetrics volatility surface and E-mini S&P 500 options from CME Globex.

- “Strong coding skills: proficiency in Python, SQL DBs, Cloud, schedulers, containers, CI/CD, software packaging” — Citadel Securities
- “gain experience with our full-cycle process for development, testing, and release” — Jump Trading
- “Produce clean, well-tested, and documented code with a clear design to support mission critical applications” — Akuna Capital

Week 5: Unit tests and data validation with pytest

A unit test is a small program that checks your code automatically: for a data pipeline, tests answer questions like *did every date parse? are there duplicate tickers? do the portfolio weights sum to one?* This is precisely what the postings call “data checks,” “data validation,” and “data quality control”—the difference between noticing bad data yourself and having a customer notice it for you. We write these tests with pytest and wire them into the pipeline, so every rebuild checks its own data before it publishes anything. The opening example is the cleaning of FINRA TRACE corporate bond transactions, where every filter is a convention. The homework then moves to a case with a ground truth: rebuilding the E-mini S&P 500 order book from Databento’s message feed and validating it against the vendor’s own book, before measuring order-flow toxicity following Easley, López de Prado, and O’Hara (2012).

- “Setup ‘data checks’ and alerts to determine when the data is ‘bad’” — Citadel Securities
- “Proven expertise in developing data quality control processes to detect gaps or inaccuracies” — DRW
- “Perform data reconciliations, validations, and quality checks” — Hudson River Trading

Week 6: SQL at scale, remote machines, and job schedulers

When data outgrows a laptop, the toolkit changes: Polars is a modern DataFrame library built for speed, Parquet is a compressed columnar file format designed for analytics, and the computation moves to remote Linux servers that you reach over SSH and operate entirely from the command line—which is why so many postings insist on Linux and “Unix scripting.” We practice on two platforms: NYSE TAQ, the intraday trades and quotes of US equities, on the WRDS Cloud and its Grid Engine scheduler, following Holden and Jacobsen (2014); and the CME order book on UChicago’s Midway HPC cluster with SLURM, at a scale where these tools genuinely matter. Batch schedulers like these are the “schedulers” in Citadel’s list.

- “Comfortable with the Linux command line” — Hudson River Trading
- “Unix scripting experience (bash, python, etc.)” — IMC Trading
- “Strong coding skills: proficiency in Python, SQL DBs, Cloud, schedulers, containers, CI/CD, software packaging” — Citadel Securities

Week 7: Orchestration across projects with Apache Airflow

A task runner like PyDoit orders the steps inside one project. A workflow orchestrator orders many projects: it runs each pipeline on a schedule or when the data it depends on lands, retries what fails, records every run, and can backfill a gap after an outage. Apache Airflow is the most widely used of these tools. This week we generate its workflow definitions directly from the manifests that already document the course’s pipelines, so that the class report rebuilds itself whenever one of its inputs updates. Then we operate it: read a failure from the scheduler’s interface and its logs, fix it, and backfill. Our case study is the class’s collaborative report on the equity premium predictors of Goyal, Welch, and Zafirov (2024).

- “Strong coding skills: proficiency in Python, SQL DBs, Cloud, schedulers, containers, CI/CD, software packaging” — Citadel Securities
- “Experience with monitoring, observability, and alerting systems for data pipelines” — DRW
- “Build, deploy, and monitor our data processing pipelines (Java, Python, Spark, Flink)” — IMC Trading

Week 8: CI/CD with GitHub Actions

CI/CD stands for continuous integration and continuous deployment: every time code is pushed, an automated service checks out the change, runs the full test suite, and—if everything passes—rebuilds and republishes the output, with no human in the loop. GitHub Actions is GitHub’s built-in CI/CD service, and it is what we use to run each case study’s tests and publish its results automatically. When the postings below say “CI/CD”—and notice how many of them do—this machinery is exactly what they mean. Our case study is a live monitor of Fed rate-move probabilities that rebuilds and republishes itself every morning, computing the measure of monetary policy surprises that Bernanke and Kuttner (2005) read from fed funds futures. The contrast between a CI service and the scheduler of the previous week is the point of the week.

- “Demonstrated experience working on an Agile team employing software engineering best practices, such as GitOps and CI/CD, to deliver complex software projects” — Akuna Capital
- “Ability to work with a team in a fast-paced environment, deploying new software daily” — Jump Trading
- “Build, deploy, and monitor our data processing pipelines (Java, Python, Spark, Flink)” — IMC Trading

Week 9: Basic MLOps — experiment tracking and monitoring models

MLOps is the discipline of keeping a model healthy after it ships. A model in production is a pipeline that never stops running: new data arrives on a schedule, the model recomputes its outputs, and someone has to notice when the data goes bad or the model’s behavior drifts. This week applies the course’s toolkit to that problem. We add experiment tracking with MLflow, so every run’s inputs, parameters, and outputs are recorded and comparable. We extend the course’s data validation habits with drift checks, so the pipeline refuses quietly corrupted inputs instead of publishing them. And we close the loop with the scheduling and CI/CD machinery from earlier weeks, so the model retrains and republishes with no human in the loop. Our case study is the open forecasting benchmark of Bejarano et al. (2026): we fit its forecasting models to the class’s

own predictor dataset on a schedule, track every run, and watch forecast error drift as new data arrives.

- “they engineer data, build and deploy models, and monitor them in production” — JPMorganChase
- “Experience with monitoring, observability, and alerting systems for data pipelines” — DRW
- “Build, deploy, and monitor our data processing pipelines (Java, Python, Spark, Flink)” — IMC Trading

What the course does not claim

The course does not cover everything in these postings. Kafka, Spark, Kubernetes, and warehouse platforms like Snowflake and Databricks appear in several and are beyond our scope, though the course’s progression (pandas, then Polars and Parquet, then remote machines) is deliberately pointed in that direction. The postings also reflect how fast expectations move—Jump Trading’s internship posting lists “Comfortability with Claude Code” ([archived posting](#)) among its required skills. What the course does claim is the layer beneath all of it: the habits of building pipelines that are versioned, tested, documented, automated, and reproducible. Those habits are what every posting above is asking for.

5 Assignments and Final Project

Coursework at a glance

The graded work has three parts: homework, one exam, and a group project. From week 4 on, one homework or exam is due each week, and never two.

Week	Date	Due
4	Tuesday, October 20	Homework 1: CRSP, the CAPM, and Fama–French
5	Tuesday, October 27	Homework 2: the yield curve and the policy path
6	Tuesday, November 3	Homework 3: option-implied crash probabilities
7	Tuesday, November 10	Homework 4: order book validation and flow toxicity
8	Tuesday, November 17	Homework 5: your figure for the class report
	Tuesday, November 24	No class: Thanksgiving break
9	Tuesday, December 1	Exam, in class
10	Week of December 8	Final project presentation

The homework comes early on purpose. All of it is finished before Thanksgiving. What remains after the break is the exam and your project.

Homework

- There are five homework assignments: four coding assignments, and your contribution to the class report, which is one figure and one paragraph. Homework 0 is ungraded setup; complete it as soon as possible.

- Homework is distributed and collected through GitHub, and it is due at the start of class. The assignments launch in class in weeks 1, 3, 4, 5, and 6, each in the week its material is taught, and the lectures keep teaching the pieces of an open assignment after it launches.
- Homework 1 through 4 are graded automatically by an autograder that runs on GitHub Actions, and solutions are released shortly after the deadline. The due date is therefore strict, and late assignments are not accepted.
- Each student submits individually unless otherwise specified. Students are encouraged to discuss the problems in groups, but copying code is not allowed. Each student must write their own solutions.
- After grading, solutions are posted in separate repositories under the [finm-32800 GitHub organization](#).

Exam

There is one exam, held in class in week 9. It is in-person, closed-book, multiple choice, and answered on a bubble sheet. Each question has four options and one or more may be correct; a question earns its point only if every bubble is right. It covers the material through week 8, including Apache Airflow and GitHub Actions. Practice questions in the same format, with their answer key, are handed out ahead of time. There is no separate final exam; the final project is the capstone. See the [Exam Preparation](#) page on the course website for the scope and format.

Final project

Students work in groups of four to build a reproducible analytical pipeline from end to end, replicating a published result from academic finance: the team pulls data from the original sources, cleans and transforms it in tested, documented Python, and publishes the replicated result automatically. Groups choose from a list of candidate projects provided at the beginning of the quarter. The project has two steps:

1. **Consultation.** Each group schedules a one-on-one meeting with the instructor, in weeks 4 through 6, to get feedback on the plan: the data sources and the reusable “product” it intends to build on top of the replication. The consultation is ungraded. It ends with an agreed extension, which the group records in its README.
2. **Final project presentation.** The project is due 24 hours before the group’s 30-minute meeting with the instructor in week 10. The instructor runs and grades the code before the meeting, so there are no slides and no demo. The meeting is spent questioning each member individually about the paper, the data, the design choices, and the extension.

See the [Final Project Instructions and Rubric](#) on the course website for full details.

6 Assessment

Grades are based on the following components:

Homework	30%
Exam	25%
Final project	40%
Participation	5%

- **Homework (30%)** is submitted individually and graded by an autograder, which runs as a CI/CD workflow on GitHub Actions. Because you can re-run the autograder until your code passes, the assignments are best thought of as teaching vehicles. The grade primarily reflects whether you did the work to complete them. Homework 5, your contribution to the collaborative class report, is one figure and one paragraph, submitted as a pull request and graded on a rubric after review by two classmates.
- **Exam (25%)** is in-person, closed-book, and multiple choice, with one or more correct options per question and no partial credit. It tests the tools, the data sets, and the papers covered in the notes and the homework through week 8.
- **Final project (40%)** is completed in groups of four. It is graded not only on how well it accomplishes the assigned data cleaning and analysis task, but primarily on whether (1) the steps to reproduce it are fully automated and well documented, (2) the code is written in a clean and reusable fashion, and (3) the results are presented clearly and convincingly. Each member is also graded individually, on their contribution as shown by their commits and pull requests, and on their answers at the week-10 final project presentation.
- **Participation (5%)** depends on the positive impact you have on the class, including participating in in-class discussions and answering questions on the class GitHub Discussions page. Students are in no way penalized for giving wrong answers in these discussions, nor is there any penalty for asking for help. Asking for help is often the best way to learn.

7 Schedule

The course runs for ten weeks: nine weeks of lectures followed by the final project presentations in week ten. There is no class on November 24, during Thanksgiving break, so week 9 meets on December 1. Each lecture corresponds to one chapter of the course textbook, in the order given in Section 4, and each chapter opens with that week’s agenda.

Week	Date	Topic	Launches	Due
0	Preview	Portfolio selection and a first look at the course	HW 0 (ungraded)	
1	Sep 29	Git, GitHub, and virtual environments	HW 1	
2	Oct 6	Task runners, notebooks and ChartBook, SQL; Fama–French (1993)		
3	Oct 13	Publishing: GitHub Pages, L ^A T _E X, pull requests	HW 2	
4	Oct 20	Python packaging, Sphinx, the ChartBook catalog, options data	HW 3	HW 1
5	Oct 27	Unit tests and data validation	HW 4	HW 2
6	Nov 3	SQL at scale, remote machines, and job schedulers	HW 5	HW 3
7	Nov 10	Orchestration with Apache Airflow; containers		HW 4
8	Nov 17	CI/CD with GitHub Actions; Ruff and pre-commit		HW 5
	Nov 24	No class: Thanksgiving break		
9	Dec 1	Exam, then MLOps		
10	Dec 8–11	Final project presentations		

The due dates are in the table under Coursework at a Glance, in Section 5, and on Canvas. Each assignment’s page on the course website gives its launch week and its due date.

8 Software and Accounts

Lectures feature live programming exercises, so bring a laptop to class. Before the first class, please install the following software. Each is free:

- [Anaconda distribution of Python](#)
- [Visual Studio Code](#) (not Visual Studio, which is a different product)
- [Git](#)
- [TeX Live](#) (or [MacTeX](#) on macOS)
- An SSH client. One is built in on macOS and Linux. On Windows, use the built-in OpenSSH client or [PuTTY](#), with [WinSCP](#) for file transfer.

Students should also sign up for free accounts with the following services:

- [GitHub](#)
- [Databento](#), for CME Globex futures and options data, used in Homework 3 and 4. The data we use is covered by the course’s license, and week 4 explains how to get your API key.
- [Wharton Research Data Services \(WRDS\)](#). Apply for access through the University of Chicago using the [registration form](#). For any issues that arise, contact the WRDS representative for UChicago’s Mathematics department, John Zekos, zekos@math.uchicago.edu.

Access to the University’s Midway3 HPC cluster (Research Computing Center) and to the WRDS Cloud, both used in Week 6, is set up in class.

9 References

Lecture notes and case studies are published on the course website, <https://finm-32800.github.io/>. As a prerequisite, you should have prior familiarity with Python and the PyData stack (NumPy, SciPy, pandas, Matplotlib). The following may serve as useful refreshers and companions:

- [Python for Data Analysis, 3rd Edition](#), by Wes McKinney
- [Python Data Science Handbook](#), by Jake VanderPlas
- [Python Programming for Economics and Finance](#), by Thomas J. Sargent and John Stachurski
- [The Missing Semester of Your CS Education](#), MIT lecture notes. A significant portion of this course is inspired by this short course, and we rely on its material in places.
- [Code and Data for the Social Sciences: A Practitioner's Guide](#), by Matthew Gentzkow and Jesse M. Shapiro