Computational & AI-Assisted Methods for Social Sciences

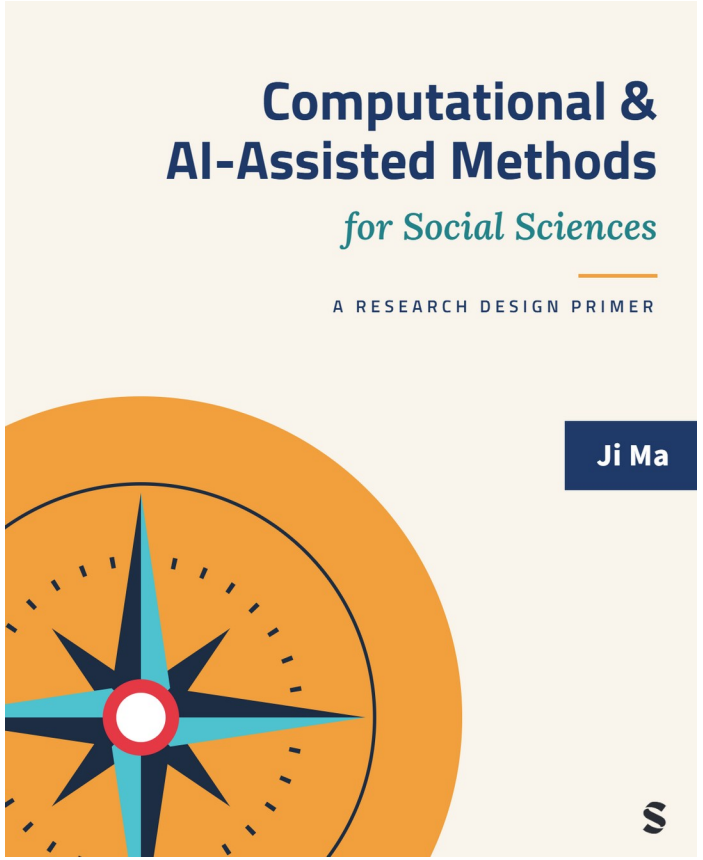
A Research Design Primer

Chapter 6

Automated Coding: Tools, Tips, and Exercises

Ji Ma

cssprimer.org



Learning objectives

By the end of this chapter, you can:

- Build a coding-ready corpus with documented boundaries and provenance
- Construct and compare three deductive instruments: dictionary, LLM prompt, supervised classifier
- Fit, refine, and validate a topic model as an inductive measurement instrument
- Triangulate deductive and inductive outputs and diagnose representation effects
- Reduce empirical codes to theoretical constructs and audit them with class-level metrics

One corpus, one transparent pipeline

- Reuse the OpenAlex corpus you built in Chapter 4
- The goal is an adaptable coding pipeline, not one fixed solution
- Arc: deductive → inductive → integrative → featurization → reduction → audit
- Deliverables are cumulative; each exercise feeds the next

§6.1

Deductive coding

Three instruments for predefined categories: dictionary, LLM, classifier.

Before any coding: a stable analysis table

- One OpenAlex work record = one observation: doc_id, text, metadata
- Define corpus boundaries: topic filters, year range, venues, language
- Deduplicate; flag missing or very short text
- Reconstruct abstracts from the inverted index once, then save

Takeaway: If the corpus table is unstable, every downstream measure is unstable too.

Exercise: building a coding-ready corpus

- Write a corpus memo: topic scope, inclusion criteria, exclusion rules
- Reflect on one difficult boundary decision and how you resolved it
- Construct the document-level table with stable IDs, text, and metadata
- Add a provenance note on how raw records became this table

You will need

- The OpenAlex records you retrieved in Chapter 4
- A programming environment with a dataframe library
- New to JSON, tokenization, or parquet? The text-data primer on the companion site

Deliverable

The saved corpus file, plus a one-page corpus memo: boundaries, rules, the difficult boundary decision, and a provenance note.

↗ **Chapter 6 hub** · cssprimer.org/ch06/

Dictionary coding: transparent, rigid, revisable

- Map predefined terms to conceptual categories
- Every decision is inspectable — but synonyms, negation, and context are missed
- Validated lists (LIWC, Moral Foundations) versus custom project lists
- First drafts always need revision after reading actual documents

Takeaway: Transparency is both the dictionary's strength and its limitation.

What dictionary failures teach you

- Equity terms: equity, disparity, justice, marginalized …
- Efficiency terms: optimization, scalable, cost-effective …
- “Scalable equity interventions” scores high on both — clarify the category logic
- “Access” and “inclusion” missed entirely — expand on the next iteration

Takeaway: False positives and false negatives are the revision engine, not embarrassments.

Exercise: dictionary-based coding

Exercise 6.1.2

- Define at least two concept categories with theoretical meanings
- Draft include and exclude terms; consider a validated starting list
- Apply to the corpus; read high- and low-scoring documents
- Revise at least once, documenting what changed and why

You will need

- The coding corpus from Exercise 6.1.1
- A written definition of the concept you intend to measure

Deliverable

The dictionary file (both versions), the scored corpus, and a half-page revision note on what reading the documents changed, and why.

Prompt-based coding: the prompt is the instrument

- Captures context-dependent meaning that static word lists miss
- A promptbook entry mirrors a codebook entry: rules, schema, worked examples, guardrails
- Guardrails: quote the evidence span; allow “insufficient evidence”
- Ordering is pedagogy: these labels bootstrap the classifier next

Takeaway: LLM labels can start a workflow; they are not a substitute for rigorous annotation.

LLM coding errors are non-random

- Confident answers can be ungrounded in the source text
- Stuhler et al.: religion imputed from names, despite explicit instructions
- Plausible case-level errors become systematic downstream bias
- Audit where errors concentrate: period, topic, group

Takeaway: Bias in labels propagates into every model trained on them.

Exercise: prompt-based coding with LLMs

- Design the promptbook entry; log model version and settings
- Run batch inference; parse responses into structured variables
- Stress-test with a deliberately bad prompt on a subsample
- Revise, rerun, and compare runs for agreement and drift
- Write an audit note on recurring error patterns

You will need

- The coding corpus from Exercise 6.1.1
- Access to a language model (free and local options on the companion setup page)
- A written construct definition

Deliverable

The promptbook entry, both label sets (careful and deliberately biased), a label-distribution comparison table, and a short audit note on error patterns and the fixes.

↗ **Prompting and its failure modes** · cssprimer.org/primers/prompting

Supervised learning in three ideas

- The split: train, validate, test — grade the model on unseen documents
- The feature matrix: every document becomes a row of numbers (TF-IDF)
- The output: probabilities you can threshold, rank, and reuse
- Start simple: logistic regression before anything complex

Takeaway: Transparent errors are often worth more than marginal accuracy gains.

Why aggregate accuracy misleads: a worked confusion matrix

§6.1

	Predicted: Methodological	Predicted: Substantive
Actual: Methodological (50)	42	8
Actual: Substantive (150)	23	127

Takeaway: Accuracy 84.5% — but precision for the “methodological” minority class is only 64.6%.

When AI writes the training script

- Generated code can be plausible and wrong in ways that never crash
- Check the split happens before feature fitting — no leakage
- Check class-level metrics, a logged random seed, the stable corpus file
- Spot-check a handful of predictions by hand

Takeaway: The use-and-validate discipline applies to code exactly as it applies to labels.

Exercise: supervised coding with labeled data

- Build the training set; manually verify a sample of LLM labels
- Train a baseline and one alternative model; compare
- Close-read misclassifications: label problems or genuine difficulty?
- Retrain on the biased-prompt labels and compare the two classifiers

You will need

- Labeled documents (hand-coded, or the LLM labels from Exercise 6.1.3)
- The biased-prompt labels you saved in that exercise
- A machine-learning library; never trained a model? See the ML primer

Deliverable

Class-level performance tables (precision and recall per class, against a majority-class baseline) for both classifiers, an error-analysis note, and a memo on when supervised coding beats your other instruments.

↗ **Supervised learning, gently** · cssprimer.org/primers/ml

§6.2

Inductive coding: topic analysis

Let patterns emerge — then hold them to measurement standards.

Choose the model that fits the question, not the fashion

§6.2

Model	Best for	Mechanism
STM	Estimating how topics vary by covariates (year, venue, party)	LDA extended with document-level covariates
Top2Vec	Exploring an unfamiliar corpus with no preset topic count	Dense clusters in a joint document-word embedding space
BERTopic	Fine-grained distinctions in short or specialized text	Transformer embeddings + HDBSCAN + class-based TF-IDF

Takeaway: The choice follows research purpose and corpus characteristics, not library popularity.

Label topics with terms plus representative documents

- Top terms alone mislead: “diffusion, spread, cascade” could be epidemiology
- Representative documents disambiguate what a cluster actually is
- Fix hyperparameters and a seed; export the document-topic table
- Expect at least one adjustment round — first fits are drafts

Takeaway: There is rarely one true topic count; pick the granularity your theory needs.

Refine one decision at a time

- Change a single modeling decision per run; log settings and seeds
- Compare coherence, overlap, and interpretability across runs
- Corpus-specific stop words: drop terms that dominate without discriminating
- Upweight list: protect theory-laden terms the statistics underweight

Takeaway: A run log turns tinkering into reproducible refinement.

Validate the model as you would any instrument

§6.2

- Validity: face checks, coherence and exclusivity metrics, blind LLM labeling, construct checks
- Reliability: reruns across seeds and preprocessing variants
- Qualitative trustworthiness: credibility, transferability, dependability, confirmability
- 789 studies reviewed: validation practice has never converged

Takeaway: Choose validation that fits what the model is for, and report which strategies you used and why.

Exercise: topic modeling analysis

- Fit an initial model; label topics with terms plus representative documents
- Change one modeling decision; compare runs
- Run at least one validity check and one reliability check
- Write an interpretation memo: final topics, ambiguities, scope conditions

You will need

- The model and pipeline you selected in Exercise 6.2.1
- A run log recording each run's settings and seed

Deliverable

The complete run log (settings, seeds, outputs), final labeled topics with representative documents, and an interpretation memo stating unresolved ambiguities and scope conditions.

↗ **Topic-modeling notebook** · cssprimer.org/notebooks/ch06-topic-modeling.ipynb

§6.3

Integrating deductive and inductive approaches

Triangulate the two streams; divergence is data.

Divergence is data, not failure

- First ask: are both instruments even coding the same conceptual dimension?
- A dictionary may capture policy stance; topics capture thematic domain
- Sample three groups: high agreement, high disagreement, boundary cases
- Forcing unification can obscure what each instrument does well

Takeaway: Sometimes the honest outcome is keeping both dimensions, documented.

Computational grounded theory names this workflow

§6.3

- Step 1, pattern detection: computational surfacing — your instruments
- Step 2, pattern refinement: close interpretive reading — your reconciliation
- Step 3, pattern confirmation: computational tests across the corpus — your rerun

Takeaway: An established methodological tradition, not ad hoc reconciliation (Nelson 2020).

Exercise: triangulating deductive and inductive coding

- Build a document-level comparison table for both measures
- Close-read divergences: coding error, boundary case, or different dimension?
- Revise rules — or deliberately retain both dimensions
- Rerun the revised pipeline; summarize what changed

You will need

- One deductive measure (dictionary, LLM, or classifier)
- Your inductive topics, keyed to the same doc_id

Deliverable

The reconciliation table, a close-reading note classifying each major divergence, and a statement of what you revised or deliberately retained.

§6.4

Featurization and auxiliary instruments

The representation silently decides what the instrument can see.

Same two abstracts, four representations

§6.4

Representation	What it emphasizes	Equity vs. efficiency abstracts
Bag of words	Word presence; order and context discarded	Placed apart — few shared words
TF-IDF	Rare, distinctive vocabulary	Contrast sharpened further
Dense embedding	Semantic similarity in a meaning space	Pulled together by shared topic
Citation network	Relational position, intellectual community	Regrouped into two separate communities

↗ **Text-representation notebook** · cssprimer.org/notebooks/ch06-text-representation.ipynb

Takeaway: The representation, not the text, decides whether the instrument reads them as similar or different.

Exercise: diagnosing representation effects

- Select 15–20 documents with notable deductive-inductive disagreement
- Re-represent each in at least two new ways
- Compare pairwise similarities against your earlier coding
- Diagnose each divergence: representation effect or coding-rule effect?

You will need

- The divergent cases identified in Exercise 6.3
- The means to build two alternative representations (e.g., TF-IDF and dense embeddings)

Deliverable

The pairwise similarity comparison, a one-line diagnosis per divergence, and a memo naming the representation that best supports your conceptual goals — and what it misses.

§6.5

Theorization and theoretical reduction

From coding outputs to constructs that can carry claims.

Three ways to justify a theoretical linkage

§6.5

- Existing theory: a named mechanism predicts the pattern (agenda-setting)
- Existing literature: empirical precedent for a similar result
- Researcher logic: your own argued case for the mapping
- No articulable reason? The mapping is premature

Takeaway: Inductive discoveries especially need documented linkage evidence, constructed after the fact.

Codes to constructs: progressive abstraction

§6.5

- Inventory outputs; group them into pattern categories (Saldaña's pattern codes)
- Example: equity rhetoric + access barriers + inclusion policy → Equity Framing
- Test boundary and negative cases; frequency is not importance
- End point: an auditable linkage table, and a key assertion in provisional language

Takeaway: “It is good when a researcher develops a theory, but it is all right if it does not happen.”

Exercise: from empirical to theoretical

- Inventory outputs; group them into pattern categories
- Document two of the three linkage types per category
- Include at least one negative or ambiguous case
- Draft a key assertion; assemble the handoff package

You will need

- All coding outputs so far, with representative documents for each
- The reconciliation table from Exercise 6.3

Deliverable

The completed linkage table (output → category → construct → evidence and scope note), your drafted key assertion, and the handoff package for Chapter 7.

§6.6

Validity and reliability audit

Pull the threads together — and check who your pipeline fails.

97.6% accurate, missing half the cases that matter

§6.6

	Predicted: Equity	Predicted: Non-equity
Actual: Equity (80)	44	36
Actual: Non-equity (1,920)	12	1,908

Takeaway: Recall 55%, precision 78.6%: the equity category is both too small and too noisy.

The marginal-cases audit checklist

- Report precision and recall per class, never accuracy alone
- Close-read minority false negatives: are central cases being suppressed?
- Close-read false positives: shared surface vocabulary, different concept?
- Test whether errors concentrate in particular subgroups
- Mitigate: oversample, cost-sensitive loss, sharper coding rules

Exercise: validity and reliability audit

- Evaluate one construct against three of the eight goodness criteria
- Apply at least two of the six validation strategies
- Run the marginal-cases audit on your minority class
- Write an honest assessment of remaining threats

You will need

- Your linkage table from Exercise 6.5
- Your strongest coding measure and its class-level error counts

Deliverable

The criteria assessment, results of two validation strategies, the marginal-cases audit with class-level metrics, and an honest assessment of where the measure is least trustworthy.

A tool for every step of the pipeline

Task	Tools
Dictionary coding	Quanteda, LIWC, custom scripts
Supervised classification	scikit-learn, HuggingFace Transformers
Topic modeling	STM, BERTopic, Top2Vec
LLM-based coding	OpenAI API, OpenRouter, Ollama, LangChain
Text embeddings	Sentence-Transformers, OpenAI Embeddings
Validation and audit	scikit-learn metrics, seaborn
Documentation	Jupyter, Quarto, Git

↗ **Automated-coding tools** · cssprimer.org/tools/coding

Takeaway: Pick per task, document versions, and keep the pipeline auditable end to end.

Discussion

- Which worldview best describes the pipeline you actually built — and did it shift along the way?
- Where did your three deductive instruments diverge, and what did that teach you about your concept?
- Which representation choice had the largest effect on your results, and what does that imply?
- What single change would most improve minority-class recall without flooding the category with false positives?

Key terms from Chapter 6

Tokenization: Splitting raw text into words or subword units; the first preprocessing step

Stemming vs. lemmatization: Rule-based chopping vs. dictionary-based base forms; lemmatization preserves concept-relevant distinctions

TF-IDF: Term weights reflecting how characteristic a word is of a document within the corpus

Promptbook: A codebook for machine coders: rules, schema, examples, guardrails, logged settings

Hyperparameter: A model setting chosen before fitting, such as topic count or minimum topic size

Stop-word / upweight lists: Terms removed for dominating without discriminating; terms boosted for theoretical weight

Confusion matrix: Actual vs. predicted counts; the source of class-level precision and recall

Trustworthiness criteria: Credibility, transferability, dependability, confirmability — qualitative validation for inductive results

Pattern code: A meta-code naming what similarly coded data share; the unit of theoretical reduction

Key assertion: A summative, provisional claim your data support when a full theory is out of reach

Materials on the companion site

Chapter 6 hub cssprimer.org/ch06/
Exercise materials plus the pre-executed notebooks for this chapter

Worked notebooks cssprimer.org/notebooks
Text-representation and topic-modeling notebooks, runnable end to end

Prompting and its failure modes cssprimer.org/primers/prompting
The primer behind the LLM-coding exercises

Supervised learning, gently cssprimer.org/primers/ml
The classifier background, with a runnable demo script

Automated-coding tools cssprimer.org/tools/coding
The maintained tool list for dictionaries, classifiers, and topic models

Where to go next

- Read: Chapter 7, computational data analysis — your coded measures become its inputs
- Bring: the handoff package — coding table, data dictionary, linkage table, candidate questions
- Compare: your pipeline against a published study in the CSS Empirical Studies Database