AI in Data Science: From Assistant to Partner

ISEA Training Program

Wei Ai

February 20, 2026

Table of contents

Introduction

- We all know there is an elephant in the classroom.
- AI tools are ubiquitous in modern learning and coding environments.
- But using them *existentially* is different from using them *rigorously*.

What is “Vibe Coding”?

- Using natural language to direct AI to write, modify, and debug code.
- **You** describe what you want; the **AI** figures out how to do it.
  - → The human focuses on the intent, logic, and verification (the “vibe”).
  - → The AI handles the implementation details, syntax, and boilerplate.
- Vibe coding is not just ‘vibing’ and doing nothing; it is **managing a junior developer** (the AI).

Vibe Coding is a Spectrum

Approach	You Do	AI Does
Traditional coding	Write all code	Autocomplete, syntax help
AI-assisted coding	Write structure, use AI for functions	Generate specific functions, debug
Vibe coding (light)	Describe task, review/edit code	Write initial code, iterate
Vibe coding (heavy)	Describe goal, validate outputs	Everything else

Vibe Coding Tools

- **Chat interfaces with context:**
  - → Chatbot: ChatGPT, Claude, …
  - → Chatbot +: ChatGPT canvas, Claude.ai artifacts
  - → Chatbot + projects (project knowledge, upload files, maintain context)
  - → Chatbot + external app (Google Doc, etc.)
- **IDE-integrated:**
  - → GitHub Copilot (auto completion, tab completion, etc.)
  - → Cursor/Claude Code (terminal/chatting functionality)
- **Agentic Application:**
  - → Claude Code on Remote Code Repository

Three Levels of AI in Data Science

In this course, we should embrace AI across different workflows:

1. **Level 1:** AI as a coding assistant.
2. **Level 2:** AI as a data processor (via LLM APIs).
3. **Level 3:** AI as an analysis partner (for brainstorming, planning, and auditing).

Level 1: AI as a Coding Assistant

Level 1: AI as a Coding Assistant

This is where most of you already are.

- Focuses on generating, debugging, and explaining code.
- **Example:** “How could I change the name of this column?”
- **Example:** “How could I pivot the DataFrame into this format?”
- AI is highly reliable at syntax, recognizing patterns, and debugging at this level.

This is your comfort zone. Now let’s see what’s beyond it.

Level 2: AI as a Data Processor

Level 2: AI as a Data Processor

Moving from analyzing static files to processing dynamic streams of unstructured data via LLM APIs.

Table Caption

student_id	course_code	feedback_text	sentiment	main_topic	mentioned_issues
1001	INST878	The lectures were engaging but assignments felt rushed	Negative	Teaching	deadlines, workload
1002	INST878	Loved the hands-on labs, really helped me understand APIs	Positive	Curriculum	-
1003	EDUC201	Classroom was too cold, hard to focus on content	Negative	Facilities	temperature
1004	CMSC320	Great professor but textbook was outdated	Mixed	Teaching	materials
1005	INST878	Project requirements were unclear at first	Negative	Curriculum	instructions
1006	EDUC201	Excellent discussions, learned a lot from peers	Positive	Teaching	-

Why can’t traditional ML do this well? Traditional ML requires labeled training data for each new task. LLMs perform *in-context learning* — you describe the task in natural language.

What Does Calling an LLM API Look Like?

```python
from openai import OpenAI
client = OpenAI(api_key="...")

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "Extract sentiment from student feedback."},
        {"role": "user", "content": feedback_text}
    ]
)
df['sentiment'] = response.choices[0].message.content
```

But what if the task is ambiguous? “Extract sentiment” could mean many things. We need to design the right prompt.

Designing Effective Prompts: Industry Consensus

For simple tasks, a one-sentence prompt works. For complex tasks, we need structure.

- **Persona Assignment:** Narrow the model's latent space (e.g., *"Act as a senior educational researcher"*).
- **Explicit Task Definition:** Use strong, unambiguous action verbs.
- **Data Delimiters:** Fence off user input from system instructions.
- **In-Context Examples:** Demonstrate the exact expected input-to-output format.

Optimizing Prompts Like Fitting a Model

Think of prompt optimization like linear regression:

$$\text{Output} = \text{LLM}(\underbrace{\text{Instructions}}_{w} + \underbrace{\text{Examples}}_{b})$$

You don't guess w and b by hand — you fit the model.

Same with prompts: packages like DSPy can automatically optimize instructions and examples to maximize accuracy.

The Bridge: Understanding AI Context

LLM APIs are Stateless

The API is **not** a chat interface; it is a stateless HTTP endpoint.

- The server retains **zero memory** between requests.
- You are responsible for sending the *entire* conversation history with every new request.
- Because LLMs have zero memory, the **context** you engineer shapes the entire environment.

LLMs have zero memory

You are responsible for the full history on every call.

Turn 1

```
Request:
  [system: "You are a helpful assistant."]
  [user: "My name is Alex."]

Response:
  [assistant: "Nice to meet you, Alex!"]
```

Turn 2 — resend everything

```
Request:
  [system: "You are a helpful assistant."]
  [user: "My name is Alex."]
  [assistant: "Nice to meet you, Alex!"]
  [user: "What is my name?"]

Response:
  [assistant: "Your name is Alex."]
```

Protecting Logic with Delimiters

Models view your instructions and the input data as one continuous stream of text.

❌ Without delimiters

```
Summarize the text and classify
its sentiment. The new curriculum
exceeded expectations. Scores
grew 23%. Classify: positive or
negative.
```

Where does the task end and the data begin?

✅ With XML tags

```
Summarize the text in <document>
and classify its sentiment.

<document>
The new curriculum exceeded
expectations. Scores grew 23%.
</document>
```

From API Calls to Heavy Vibe Coding

The same rationale of **“construct the right context for LLM API calls”** also applies to **heavy vibe coding**.

- For **Level 2 (API calls):** Context = system prompts, delimiters, few-shot examples
- For **Level 3 (vibe coding):** Context = your entire project environment

To make AI act as an **Analysis Partner**, you need: **Context + Workflow + Review**

Level 3: AI as Analysis Partner (Critical Considerations)

Making Vibe Coding Helpful: Context + Workflow + Review

- **Context:** What the AI “knows”
  - → Your data files (schema, sample rows)
  - → Your stated goals and constraints
  - → Documentation you’ve shared
  - → ...
- **Workflow**
  - → How the AI approaches the task: Does it plan first? Does it ask clarifying questions? Does it make and validate assumptions?
  - → How AI validates its work: Does it test its own code?
- **Review**
  - → How do you claim success.

Principle 1: Managing Context

- What data do you have
  - → not just description, actual rows
- Your analysis goal, in plain language
- Resources and Constraints
  - → Available tools
  - → Time and computing resources
- Preference, history, and documentation
  - → What you've already tried that works or does not work
  - → What previous vibe-coding sessions have achieved
  - → Additional documentation that AI should refer to.

Example

```
I am working with [dataset] to answer [question].
Here is the output of df.info() and the sampled rows
[paste 5–10 rows, more if additional dataframes are involved.]
I have pandas and matplotlib in my environment, but we have limited
memory.
I've already tried: [what didn't work]
Please suggest an approach before coding.
```

Principle 2: The “Plan First” Rule

- AI loves to rush. It will write code immediately, often leading to hallucinated column names or wrong logic.
- The Solution: Force it to think before it codes.
- Most AI tools now have “planning modes” - use them! > “I have a CSV about X. Before writing any code, help me: Understand what questions this data can answer. Identify potential data quality issues. Outline an analysis approach. Then we’ll implement step by step.”
- Allow you to review the plan and catch logic errors before debugging the code.

Principle 3: Iterative Execution

- **Risky:**
  - → "Clean this data, handle missing values, merge with the other file, create features, and build a model."
- **Safer:**
  - → "Step 1: Show me summary statistics and missing value counts."
  - → [Review output]
  - → "Step 2: Let's discuss how to handle the missing values in column X."
  - → [Make decision together]
  - → "Step 3: Now implement that approach and show me the result."
- **Treat AI like a junior analyst — check their work at each stage.**

Principle 4: Documentation as Code

“I have made all required changes and this code can successfully handle XYZ.”

- But, it changes 10 other scripts to make this new functionality work…
- You have no idea what the code does and why the code works, in 30 minutes.

The solution: keep the memory.

- “We just successfully cleaned the data. Please summarize all the transformation steps we took into a bulleted list so I can put it in my documentation.”
- Keep a readme or a scratchpad open. Ask AI to add comments explaining why, not just what. Keep a decision log: “We chose X because Y”
- Chat context can get too long and AI struggles with long-context (needle in the hay)

Principle 5: Version Control & Project Structure

Treat each iteration as a product change.

- What goal this specific iteration is addressing, how is it connect to the product.
- What changes have we made, and why
- What tests did we conducted.
- Track file changes with version control.

Maintain a clear project folder structure.

- Data folder,
- Source code,
- Scripts,
- Documentation, …

Why AI Fails at Level 3 Tasks

Let's think about how AI is trained:

Supervised Fine-Tuning (SFT):

- Trained on finished code, project folders, documentation
- Sees the outcome, not the process
- Doesn't see: debates, rejected approaches, "why we didn't do X"

RLHF (Reinforcement Learning from Human Feedback):

- Optimized to be helpful, harmless, honest
- Rewards completing tasks, not questioning tasks
- May not represent real data science decision-making

Level 3: AI as an Analysis Partner

Using AI to design the analytical workflow itself — the most powerful but least straightforward application.

- Asking the AI to help design an analysis plan for a dataset.
- Asking the AI what questions the data could answer.
- Asking the AI what biases you should watch for in the analysis.
- **The Value:** AI excels at brainstorming structure, enumerating options, and catching obvious issues.
- **The Caveat:** AI is weak at reasoning about what is *conceptually appropriate* for your specific domain and research question.

Understanding AI Limitations: Confidence ≠ Correctness

- AI can generate a 10-step data analysis plan in seconds.
- A **detailed** plan does not mean it is a **correct** plan.
- **Step-by-step** formatting does not mean the approach is **thoughtful**.
- The AI sounding **confident** does not mean it is **right**.

> **Note**
>
> **How do we make AI-assisted data analysis both efficient AND correct?**
>
> We need to understand where AI coding suggestions can lead us astray, even when the code executes perfectly.

When “Correct” Code Yields “Wrong” Answers

Code can execute perfectly without errors, but still produce analytically invalid results.

```
# This code runs perfectly. But is it right?
df.dropna().groupby('group').mean()
```

To understand why standard coding fixes fail, we must look deeply at the **data generation process**.

Example 1: UC Berkeley Gender Bias

In the early 1970s, UC Berkeley was sued for gender discrimination over graduate school admissions.

- **Male applicants:** 8,442 applied, 44% admitted.
- **Female applicants:** 4,351 applied, 35% admitted.
- If we ask an AI to check for bias, it will likely write:

```
df.groupby('gender').agg({'admitted': 'mean'})
```

UC Berkeley Gender Bias (Department-Level Breakdown)

When you look department by department, most departments admitted women at *equal or higher* rates.

Department	All		Men		Women	
	Applicants	Admitted	Applicants	Admitted	Applicants	Admitted
A	933	64%	**825**	62%	108	**82%**
B	585	63%	**560**	63%	25	**68%**
C	918	35%	325	37%	**593**	34%
D	792	34%	417	33%	375	35%
E	584	25%	191	28%	**393**	24%
F	714	6%	373	6%	341	7%
Total	4526	39%	2691	**45%**	1835	**30%**

Simpson’s Paradox: Aggregation Matters

- Women applied disproportionately to highly competitive departments (English, History).
- Men applied disproportionately to less competitive departments (Engineering, Chemistry at the time).
- **Simpson’s Paradox:** The direction of an association reverses when the data is disaggregated by a third variable.
- Practically, this is a **Join/Groupby** issue: aggregating at the wrong level (university vs. department) inverts your results.
- A special case of **omitted variable bias**: Simpson’s reversal arises because **a confounding variable** (department selectivity) is omitted from the analysis.

```
# The necessary correction the AI usually misses:
df.groupby(['department', 'gender']).agg({'admitted': 'mean'})
```

Example 2: Women's Labor Market Participation

Imagine a dataset where we want to compare the potential average wages of men versus women.

- **The Issue:** Many women in the historical dataset report no wage.
- Why? Because they are not participating in the labor force.
- How should we handle this missing wage data?

Why AI's Default Fixes Fail Here

If you ask an AI, it will usually suggest one of two things:

Naive Fix 1: `df.fillna(0)`

Treats "not participating" as "participating but earning nothing."

Result: Drastically underestimates the mean wage.

Naive Fix 2: `df.dropna()`

Drops non-participants entirely. But women who choose to work are a self-selected sample.

Result: Overestimates the average by only counting the "winners".

Missing Data Mechanisms Matter

Missingness is not an accident; it is a structural signal.

Mechanism	Definition	Example
MCAR (Missing Completely At Random)	Missingness has no pattern	Random survey non-response
MAR (Missing At Random)	Missingness depends on observed data	Older students skip optional questions
MNAR (Missing Not At Random)	Missingness depends on the unobserved value itself	Low earners don't report income

In the labor market, the probability of participation depends on the wage opportunity itself!

You cannot clean this with code alone; you need a statistical model of *why* the data is missing (e.g., Heckman correction).

Common Patterns in Data Quality Issues

Bias isn't always about demographics; it is about inference validity.

Example	Problem	Statistical Concept
Berkeley	Aggregation hides confounders	Simpson's Paradox (Omitted Variable Bias)
Labor Market	Missingness related to the value itself	MNAR (Selection Bias)

> **Note**
>
> **Every preprocessing choice encodes an assumption about the world.**
>
> Data reflects the system that generated it.

Injecting Human Judgment into Vibe Coding

To succeed at Level 3, we must explicitly inject domain knowledge into the AI's context.

- **Force the AI to consider alternatives:** "What are 3 different ways we could handle the missing wage data? What assumptions does each make?"
- **Make the AI argue against itself:** "What could go wrong with this analysis plan?"
- **Bring domain knowledge explicitly into the prompt:**

> **Bad:** "Analyze the admission rates."
>
> **Good:** "Analyze the admission rates. Context: check for Simpson's Paradox by breaking down acceptance rates by Department."

Live Activity: Testing the AI

Live Activity Setup

We are going to test how AI responds to data analysis questions **without explicit context** about the data generation process.

1. Open your preferred AI assistant (ChatGPT, Claude, Gemini, etc.).
2. Try the following two prompts.

Prompt 1: Berkeley Admissions

Note: This prompt purposefully ignores the availability of department information.

```
I have admissions data, that look like the following and I am
analyzing whether there are gender discrimination in the admission
process. Help me create a data analysis plan.

gender admitted
0   Male 0
1   Male 0
2   Male 0
3   Male 0
4   Male 1
5   Female 0
6   Female 0
7   Female 1
8   Female 0
9   Female 1
```

Prompt 2: Women’s Wages

```
I would like to study women's wage in the labor market, what is
their mean wage and what is the impact factor of their wage.
I have data that look like the following

 wage, education, children_in_family, age
0, 1000, "high-school", 0, 30
1, NA, "bachelor", 2, 35
2, 1000, "post-graduate", 1, 27

My first step is to calculate the mean wage of the population,
how should I do that.
```

Wrap-Up: The Division of Labor

Where AI Helps vs. Human Judgment

AI Can Help With…	**Human Judgment Still Needed For…**
Detecting missing data patterns	Understanding *why* data is missing
Suggesting common preprocessing steps	Knowing if those steps fit the data's true generation process
Checking code correctness	Checking conceptual correctness
Enumerating options and tradeoffs	Making the final methodological judgment call

The Division of Labor

- **Human's Job:** Define the right question, understand data generation, and make judgment calls.
- **AI's Job:** Suggest ways to operationalize, implement data processing, and enumerate options.

The tools will change, but critical thinking about data quality, bias, and human-AI collaboration will last your whole career.