Bringing the AI Revolution to Jupyter Notebooks and Data Science

Jamie Bloxham, Sphinx AI

AI Coding Agents are the Biggest Technological Advance of 2025

Copilots are changing workflows

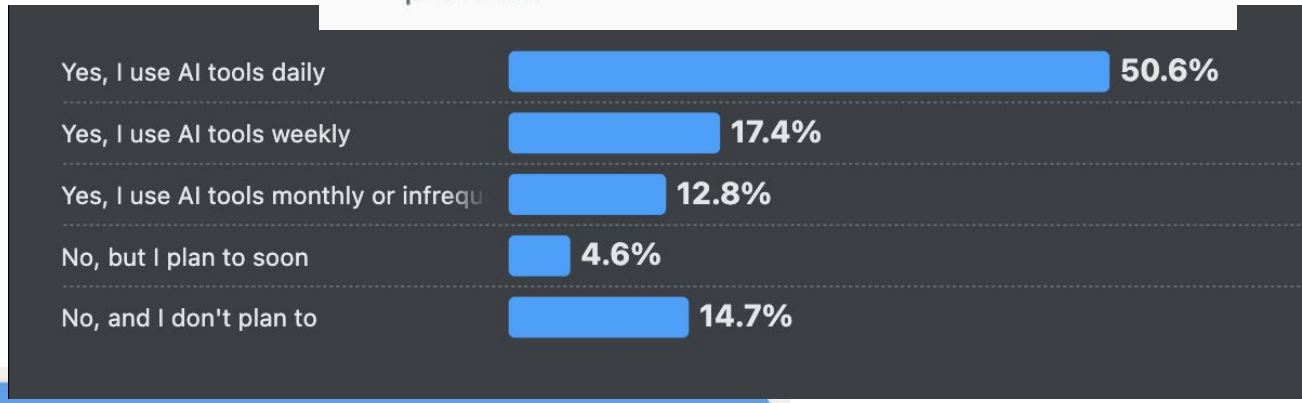
Windsurf

CURSOR



Adoption is 80-90%

Stack Overflow 2025 developer survery



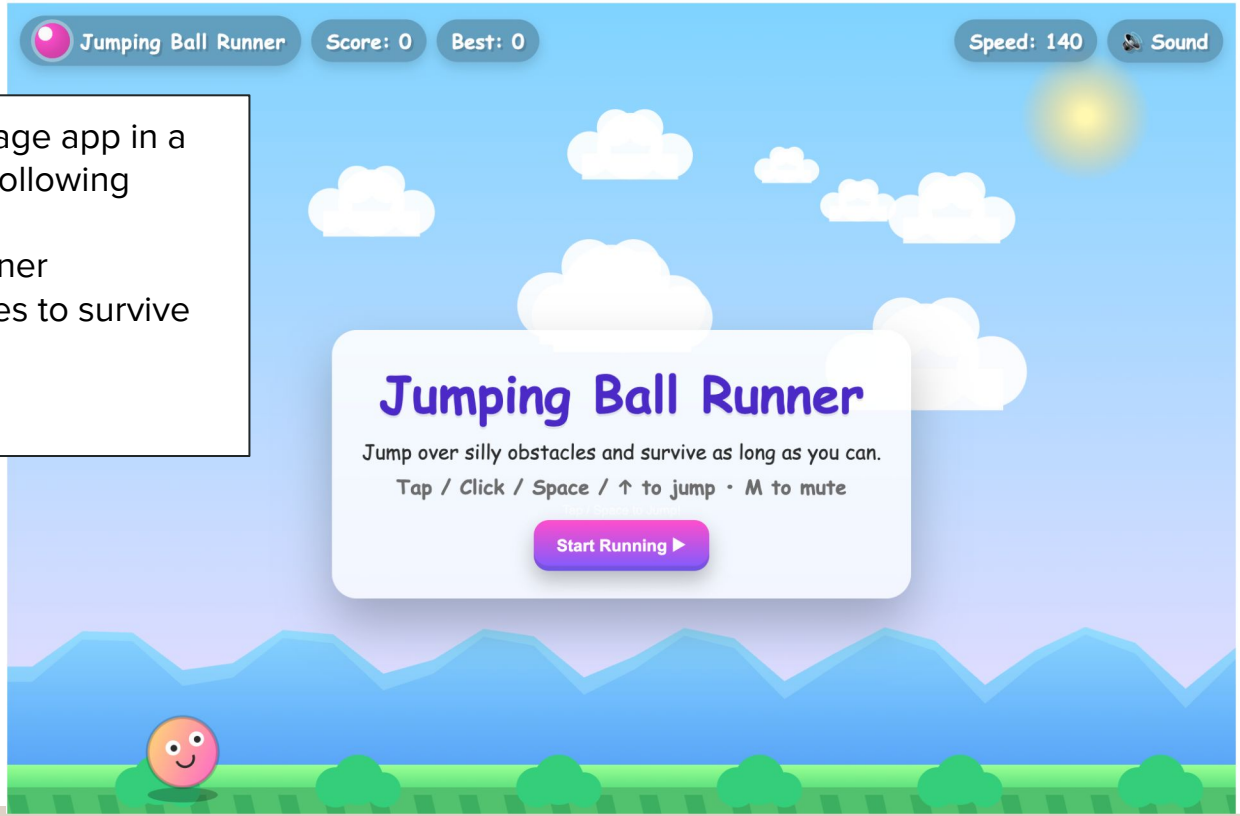


October 2025 Google DORA study

Coding efficiency up > 50% when using AI tooling, source

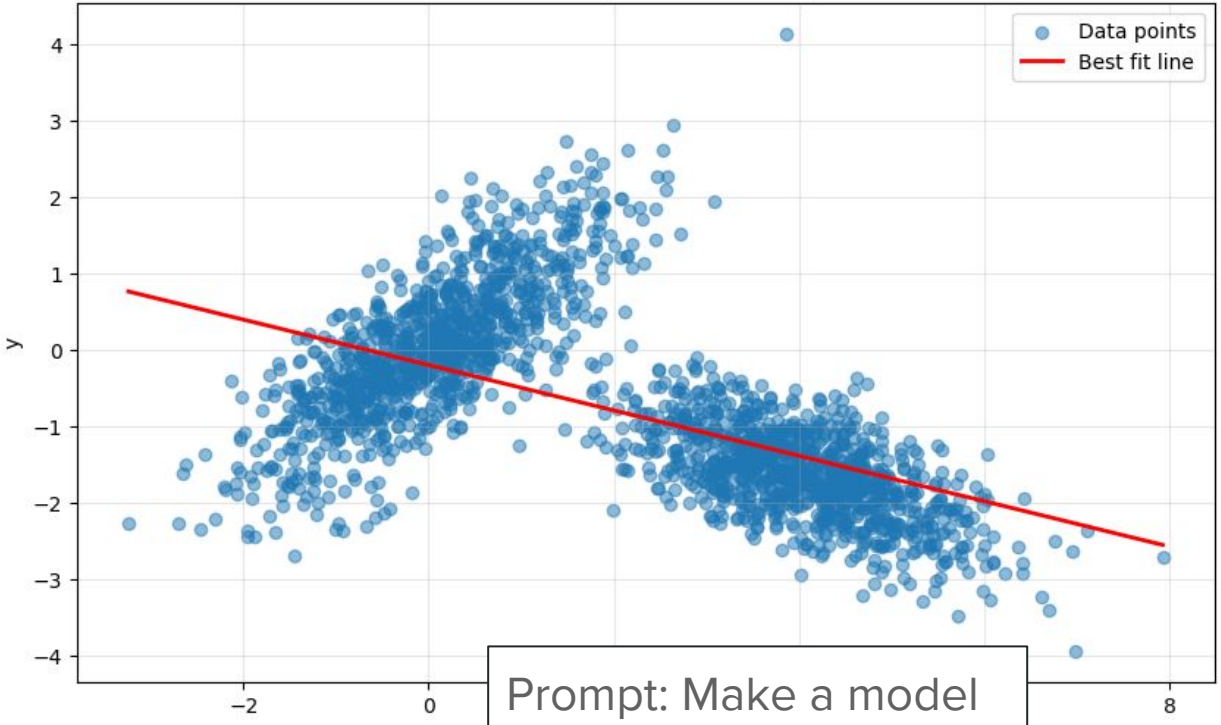
Models can one shot entire applications

Prompt: Create a single-page app in a single HTML file with the following requirements:
- Name: Jumping Ball Runner
- Goal: Jump over obstacles to survive as long as possible.
- Features: ...



And yet...

```
import pandas as pd
df = pd.read_csv('../data/bimodal_data.csv')
```



Prompt: Make a model that predicts y from x

Perfect! I've created a complete predictive model in the modeling.ipynb notebook.

...

The model uses scikit-learn's LinearRegression, which fits a line through the data to predict y based on x values.

AI Coding Agents are TERRIBLE at Data Science

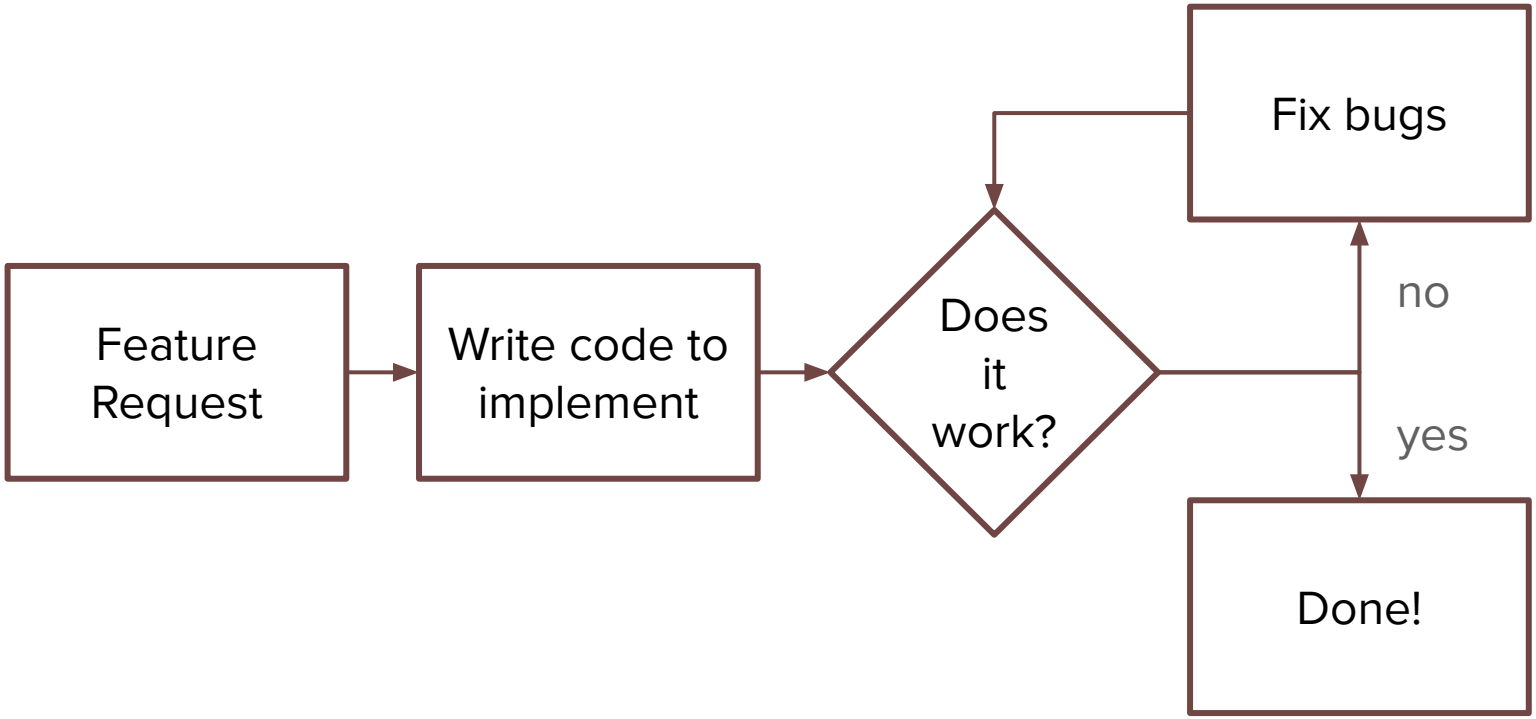
Bringing the AI Revolution to Jupyter Notebooks and Data Science

Jamie Bloxham, Sphinx AI



Challenge 1: Software Engineering is not Data Science

Software Engineering Workflow



Software Engineering Workflow, applied to data science

```
# Explore the data
print(df.head())
print(f"\nDataset shape: {df.shape}")
print(f"\nData types:\n{df.dtypes}")
print(f"\nBasic statistics:\n{df.describe()}")
```

[ ] Python

```
# Visualize the relationship between x and y
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(df['x'], df['y'], alpha=0.5)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Relationship between x and y')
plt.grid(True, alpha=0.3)
plt.show()
```

[ ] Python

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_s

X = df[['x']]
y = df['y']

X_train, X_test, y_train, y_test = train_test_sp
    X, y, test_size=0.2, random_state=42
)

print(f"Training set size: {len(X_train)}")
print(f"Testing set size: {len(X_test)}")
```

[ ] Python

```
# Build a Linear Regression model
from sklearn.linear_model import LinearRegressio
from sklearn.metrics import mean_squared_error,
import numpy as np

model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Evaluate the model
train_rmse = np.sqrt(mean_squared_error(y_train,
test_rmse = np.sqrt(mean_squared_error(y_test, y
train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)

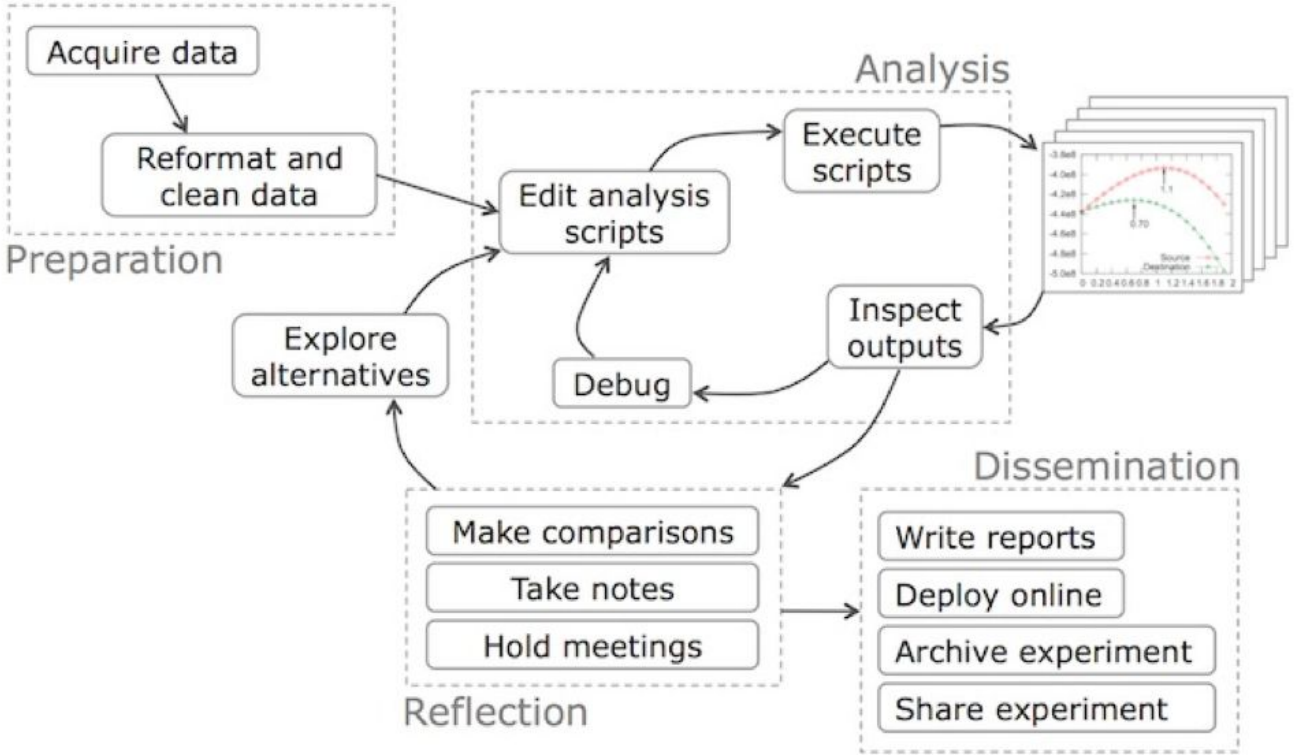
print("Linear Regression Model Results:")
print(f"Coefficient (slope): {model.coef_[0]:.4f
print(f"Intercept: {model.intercept_:.4f}")
print(f"\nTraining RMSE: {train_rmse:.4f}")
print(f"Testing RMSE: {test_rmse:.4f}")
print(f"Training R²: {train_r2:.4f}")
print(f"Testing R²: {test_r2:.4f}")
```

[ ]

Coding copilots produce great code, but they don't even bother to execute the cells

No iteration
No reasoning
No science

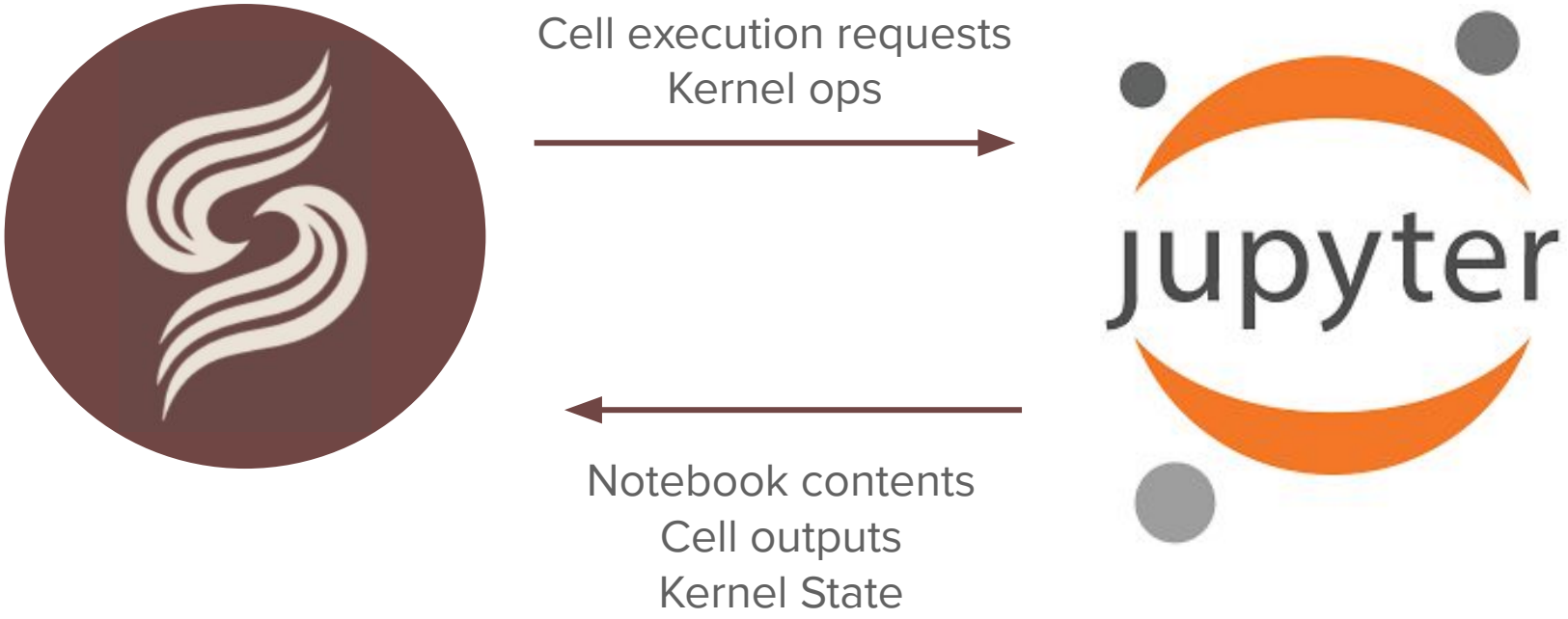
Data Science Workflows are far more complex



The objective of data science is not to write code, it's to learn something about your data

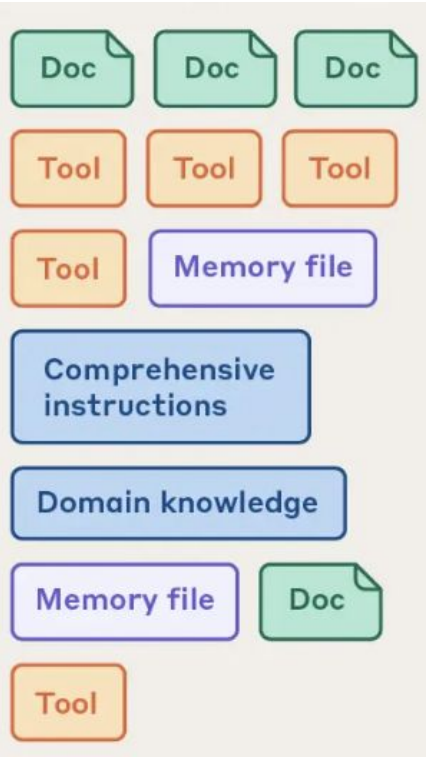
jupyter

A data science agent needs first class Jupyter integration



Challenge 2:
Data Fluency

Agents and Context Windows






```
// Handle WebSocket upgrade events for Jupyter
server.on('upgrade', (req, socket, head) => {
  if (req.url?.startsWith('/jupyter/')) {
    handleJupyterWebSocketUpgrade(req, socket, head);
  try {
    if (req.url?.startsWith('/jupyter/')) {
      handleJupyterWebSocketUpgrade(req, socket, head);
    }
  } catch (error) {
    Logger.error("received error", error);
  }
});
```

Problem: Data is just too big

- Typical context windows:
  - GPT 5: 400k tokens
  - Claude 4.5: 1M tokens
  - Gemini 2.5: 1M tokens

LLMs can’t ingest a full dataset!

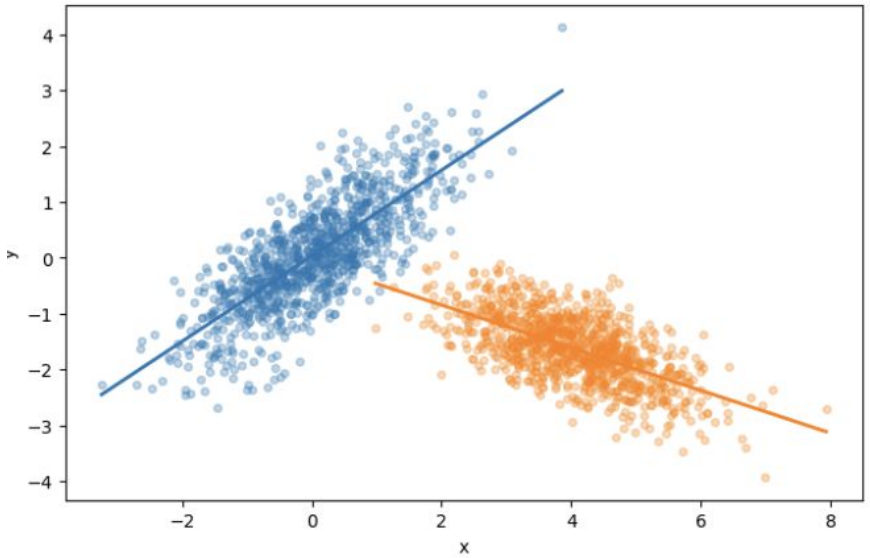
- Lahman Baseball DB batting data
  - 110k rows x 20+ columns
  - 15MB
  - **>5M tokens**

- S&P 500 daily prices, past 20 years
  - ~2.5M data points
  - > 50MB
  - **> 10M tokens**

Problem: Data is harder to interpret than language and code

```
,x,y
0,0.4967141530112327,1.2369845843605873
1,-0.13826430117118466,0.44416876881071343
2,0.6476885381006925,0.5539290524326584
3,1.5230298564080254,0.8302618185030761
4,-0.23415337472333597,0.23161128838948508
5,-0.23413695694918055,0.04878166569370526
6,1.5792128155073915,1.8004861844225524
7,0.7674347291529088,0.9950508643315086
8,-0.4694743859349521,0.2541521204436393
9,0.5425600435859647,0.11290690793243102
10,-0.46341769281246226,0.41970228513062546
11,-0.46572975357025687,-0.25402404004076573
12,0.24196227156603412,1.4387263408279862
13,-1.913280244657798,-1.944136886579979
14,-1.7249178325130328,-0.3383559841112769
15,-0.5622875292409727,-0.3310835533151895
16,-1.0128311203344238,-1.201115698436208
17,0.3142473325952739,-0.03893363435637376
18,-0.9080240755212109,-0.9186276453335609
19,-1.4123037013352915,-0.8753433932270834
```

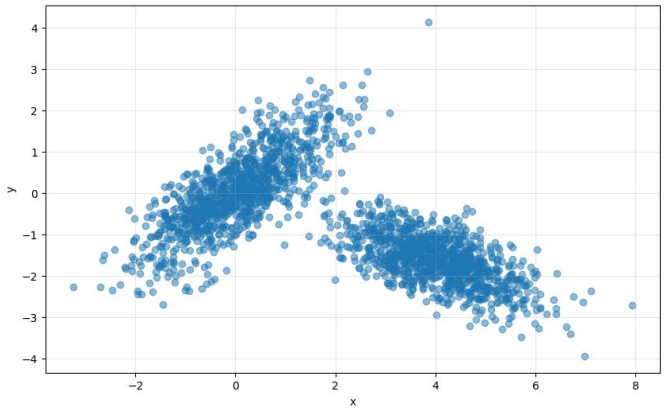
Prompt: Make a model that predicts y from x

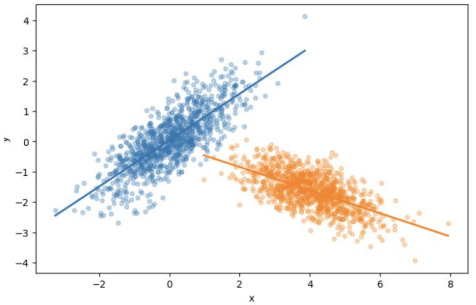


Problem: Data is harder to interpret than language and code

```
,x,y
0,0.4967141530112327,1.2369845843605873
1,-0.13826430117118466,0.44416876881071343
2,0.6476885381006925,0.5539290524326584
3,1.5230298564080254,0.8302618185030761
4,-0.23415337472333597,0.23161128838948508
5,-0.23413695694918055,0.04878166569370526
6,1.5792128155073915,1.8004861844225524
7,0.7674347291529088,0.9950508643315086
8,-0.4694743859349521,0.2541521204436393
9,0.5425600435859647,0.11290690793243102
10,-0.46341769281246226,0.41970228513062546
11,-0.46572975357025687,-0.25402404004076573
12,0.24196227156603412,1.4387263408279862
13,-1.913280244657798,-1.944136886579979
14,-1.7249178325130328,-0.3383559841112769
15,-0.5622875292409727,-0.3310835533151895
16,-1.0128311203344238,-1.201115698436208
17,0.3142473325952739,-0.03893363435637376
18,-0.9080240755212109,-0.9186276453335609
19,-1.4123037013352915,-0.8753433932270834
```

Prompt: Make a model that predicts y from x





This intermediate step is not intuitive to LLMs!

Exploratory Data Analysis and Representation Learning

To understand data, an agent needs to employ a wide range of techniques:

- Basic charting, graphing, sampling
- Clustering algorithms
- Data gathering, cleaning, etc.
- Dimensionality reductions (e.g. PCA, t-SNE, manifold learning)
- Graph/network construction and analyses
- Sentiment analysis and other row-wise AI-powered methods

This is really hard!

Challenge 3: Verifiability and Explainability

Verifying AI-generated software is (relatively) easy

Software is ultimately defined by its functionality more than anything

- Vibe coding a new app => manually test the app’s features
- Fixing a bug => check that the bug is gone
- Doing a refactor => check that the test suite is passed
- Performance optimizations => measure the speed gain

None of these strictly involve looking at the code

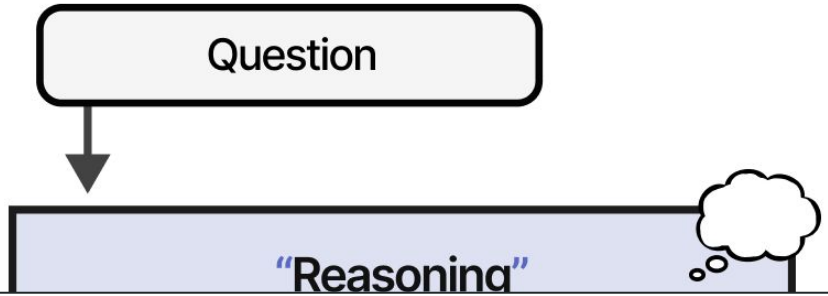
Verifying data science is hard

Imagine if a tool only gave you direct answers to data questions:

- “Make a model to predict y from x” => ***Here’s a function `f(x): y`***
- “How much revenue is this new feature driving?” => ***$0.5M/yr***
- “Who is going to win the Superbowl this year?” => ***Green Bay Packers***
- “Should I short Nvidia?” => ***Yes***

Without explainability, these outputs don’t mean anything

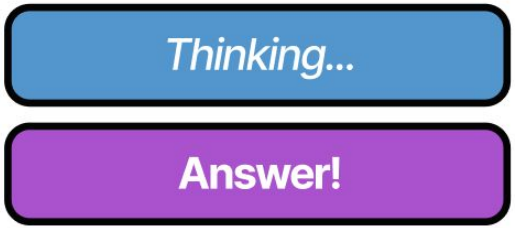
Reasoning models threaten explainability



Reasoning Models Don't Always Say What They Think

Chen et al.

Solution: Use Non-Reasoning Models with Explicit Thoughts



> "Therefore, after weighing multiple factors including user experience, competitive advantage, and the option to pursue the chain of thought monitoring, we have decided not to show the raw chains of thought to users.

Data scientists deserve AI that’s built for data, not code



[Extension Development Host] examples

SPHINX

bimodal

EDA and modeling for y ~ x

make a model that predicts y from x

Before building a model to predict y from x, I need to understand the data structure, check for missing values, and visualize the relationship between x and y. This will inform preprocessing and appropriate modeling steps. I'll start by inspecting and visualizing the data.

Sphinx added a new cell

```
+ # Inspect the dataframe structure, check for missing valu
+ print(df.info())
+ print(df.head())
+ print(df.describe())
+ print(df.isnull().sum())
```

Sphinx successfully ran a cell

To make progress on building a model to predict y from x, it's important to visually examine the relationship between x and y first. This will inform appropriate modeling options (e.g., linear vs nonlinear). I'll plot a scatterplot of x vs y to see if the relationship appears linear, nonlinear, or if there are clusters or structure in the data (the mean and quartiles suggest something may be unusual,

What do you want Sphinx to do?

Agent Mode

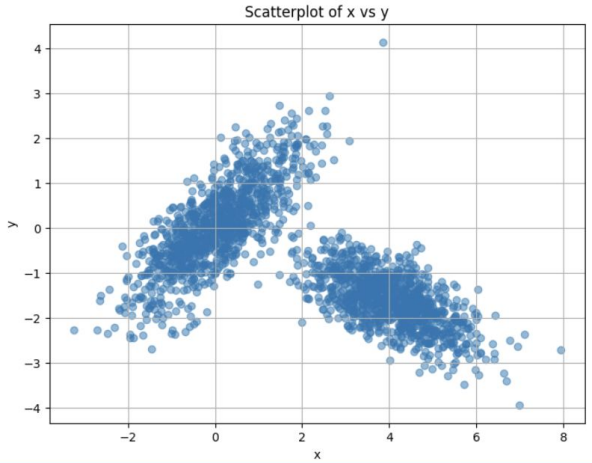
bimodal.ipynb ● | timeout.ipynb | yahoo.ipynb

bimodal.ipynb > # Visualize the relationship between x and y

Generate | Code | ··· | Python 3 (ipykernel) (bimodal.ipynb)

```
# Visualize the relationship
graph_title = 'Scatterplot of x vs y'
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(df['x'], df['y'], alpha=0.5)
plt.xlabel('x')
plt.ylabel('y')
plt.title(graph_title)
plt.grid(True)
plt.show()
```

[4] Cell 2 | Edit with Sphinx | Python



Launchpad | ⊗ 0 ⚠ 0 | { } | Cell 3 of 8

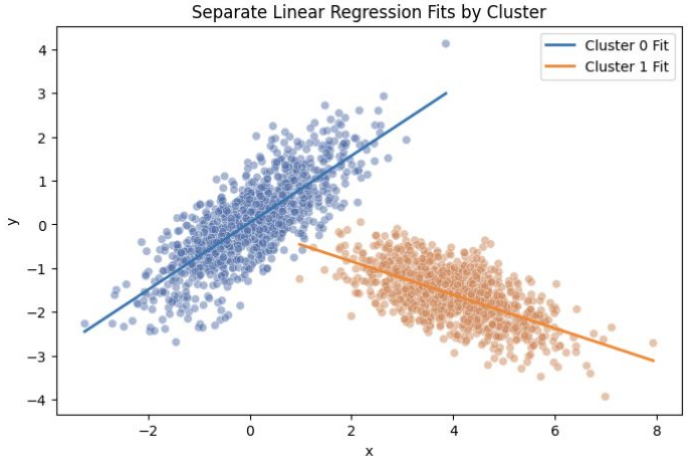
```
import pandas as pd
df = pd.read_csv('../data/bimodal_data.csv')
```

[1] ✓ 0.2s Cell 0 Edit with Sphinx Python

Make a model that predicts y from x

Next, I will try GMM clustering to segment the data. After clustering, I'll fit separate regressions for each cluster and compare their performance visually and quantitatively.

Sphinx added a new cell





Load the last year's Meta closing prices from Yahoo finance

```python
# Download the last year's Meta Platforms, Inc. (ME
import yfinance as yf
import pandas as pd

today = pd.Timestamp.today().normalize()
one_year_ago = today - pd.Timedelta(days=365)

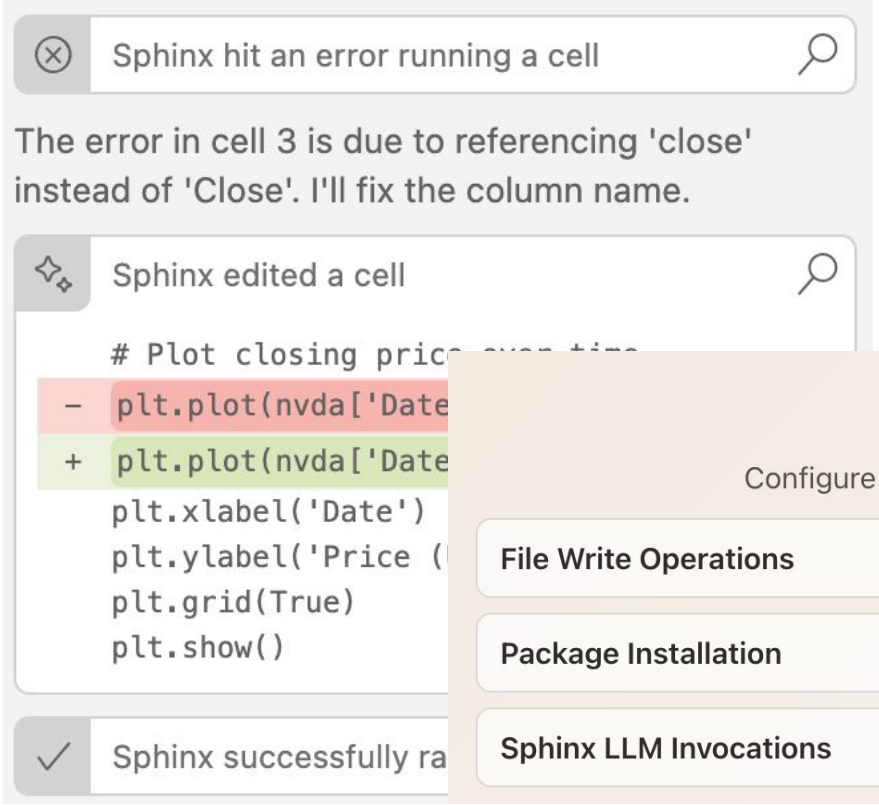
data = yf.download('META', start=one_year_ago, end=

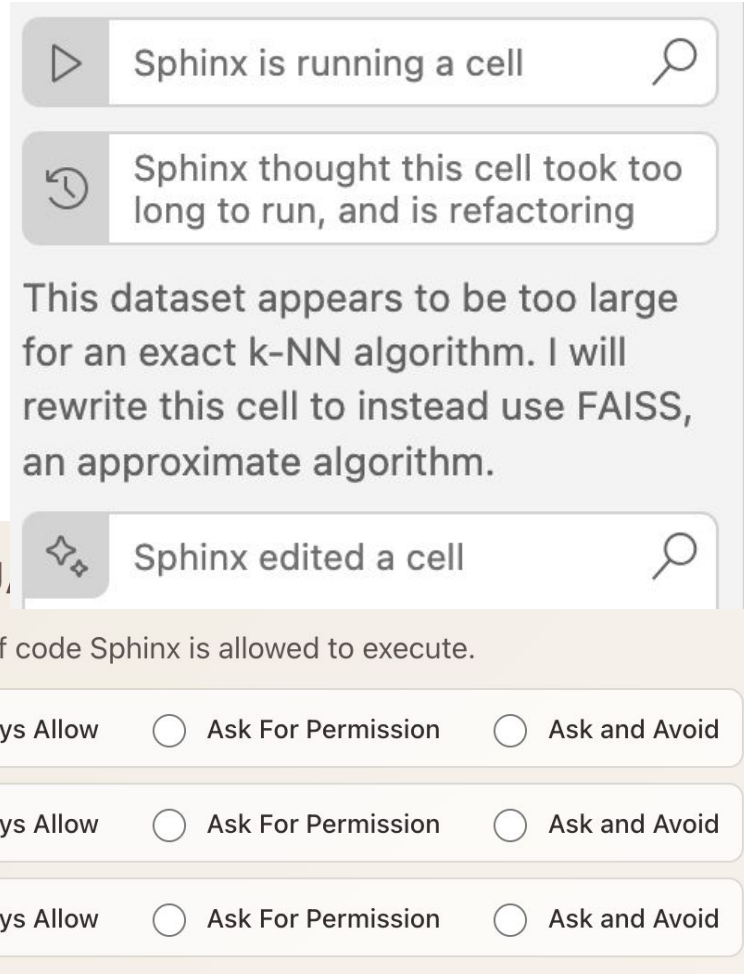
# Show the first few rows to confirm structure
print(data.head())
```

✓ 6.1s Cell 0 Edit with Sphinx Python

Price	Close	High	Low	
Ticker	META	META	META	
Date				
2024-10-29	591.420410	591.809141	573.596450	578.33
2024-10-30	589.944946	599.315506	587.532549	599.09

Execution-Centric Features

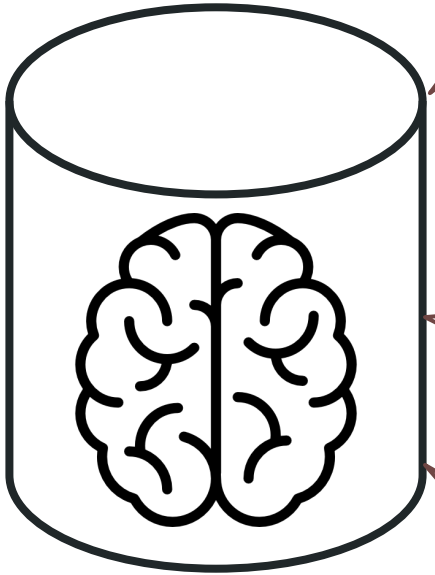




Semantic indexing's time to shine



Index databases, notebooks, saved queries



What table holds our revenue data?

What are the known gaps in our usage data?

Is there a previous notebook I can base this analysis off of?

Questions