

AI-assisted coding

Russ Poldrack

Department of Psychology

Stanford University

<https://bit.ly/3H1PfMc>



Why do we need code?

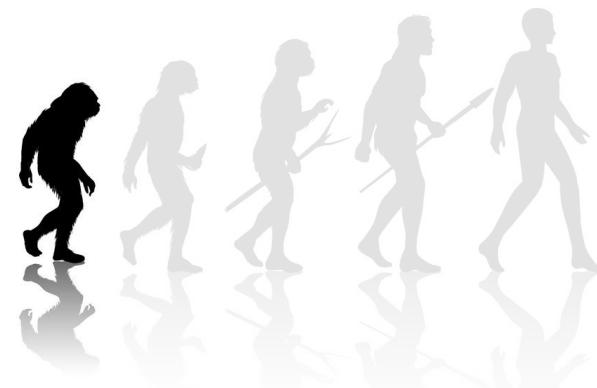
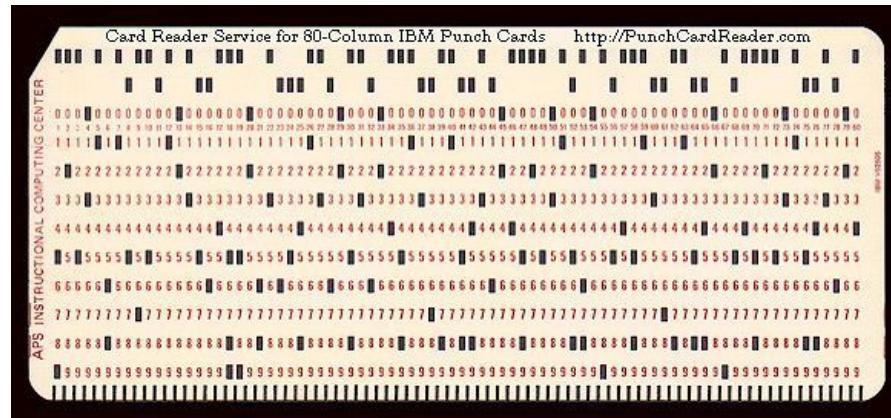
Why do we need code?

Nobody has responded yet.

Hang tight! Responses are coming in.



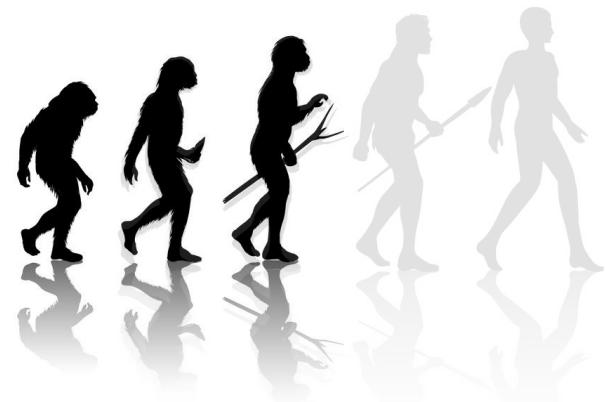
The evolution of coding



```
section          .text
global           _start
_start:
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int 0x80
    mov eax, 1
    int 0x80
section          .data
    msg      db
"Hello world!", 0xa
    len      equ $-msg
```



```
print('hello world')
```



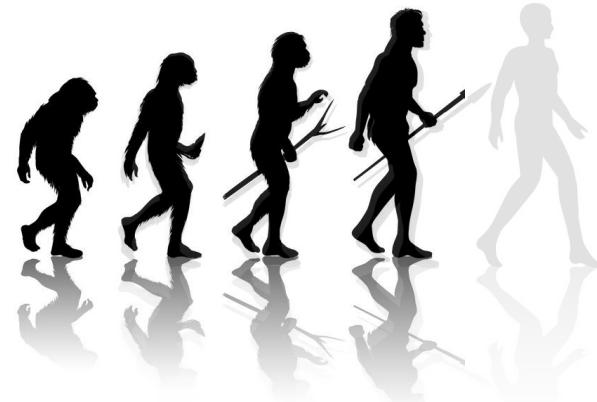
write a hello world program in rust

Here's a simple "Hello, world!" program in Rust:

rust

 Copy  Edit

```
fn main() {  
    println!("Hello, world!");  
}
```



write a hello world program in rust

Here's a simple "Hello, world!" program in Rust:

rust

 Copy  Edit

```
fn main() {  
    println!("Hello, world!");  
}
```

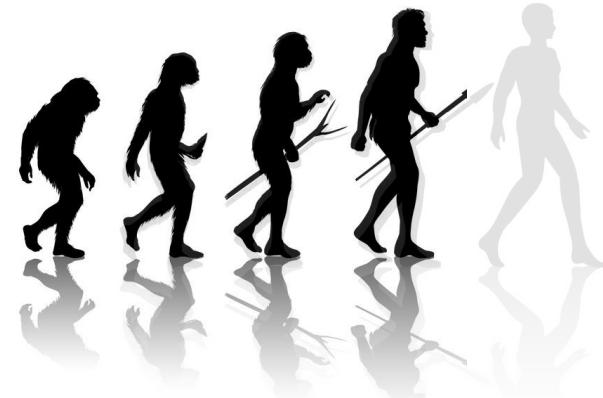


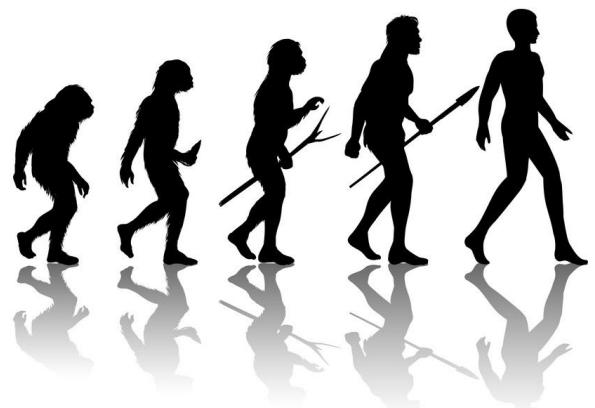
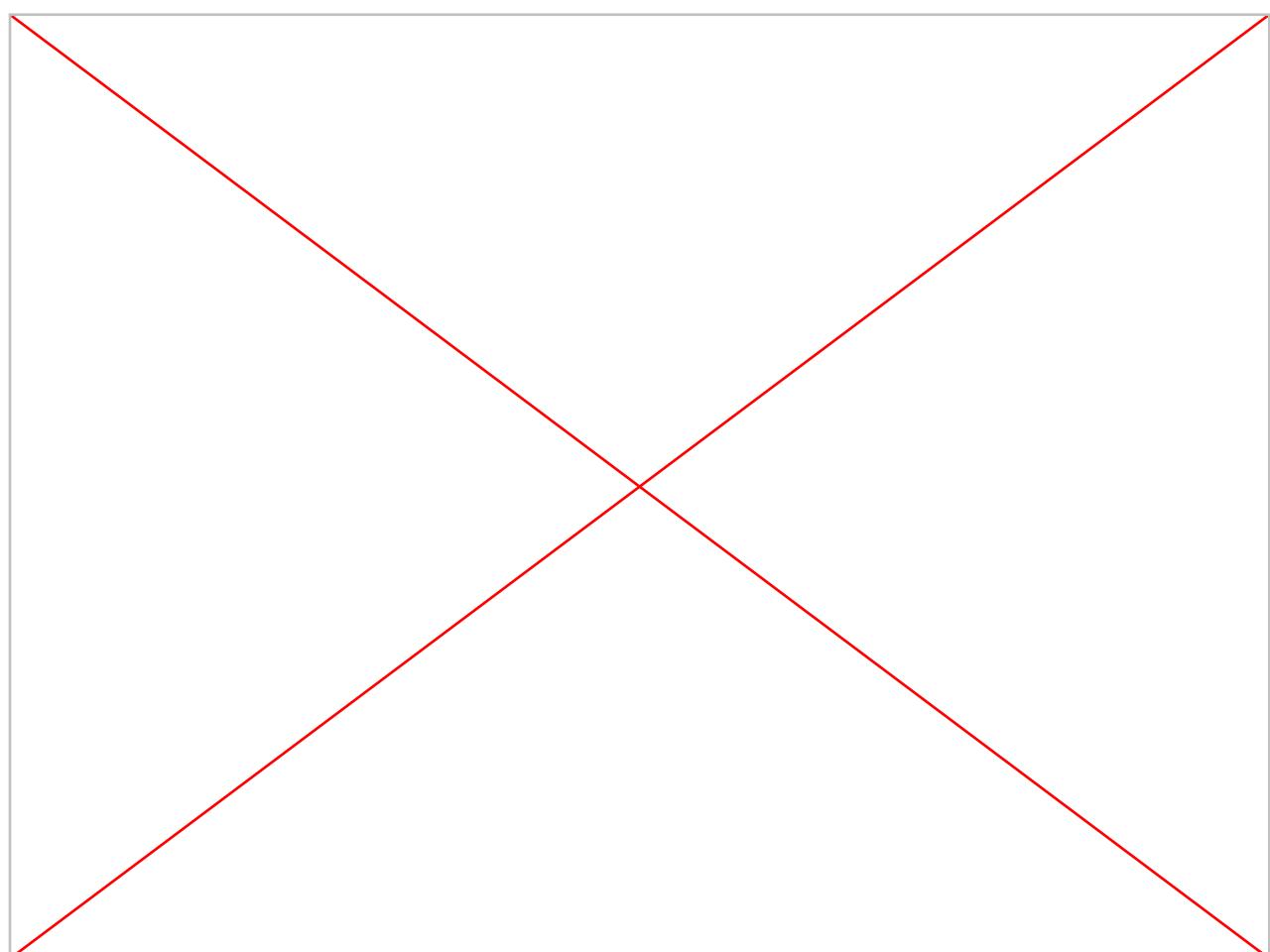
Andrej Karpathy ✅

@karpathy

The hottest new programming language is English

12:14 PM · Jan 24, 2023 · 3.8M Views





Stanford University

- Created `hello_world.py` with a simple print statement.

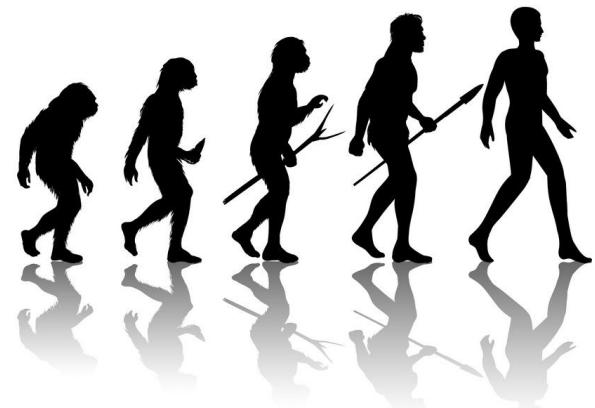
```
> /exit
└ (no content)

> █
▶ auto-accept edits on (shift+tab to cycle)

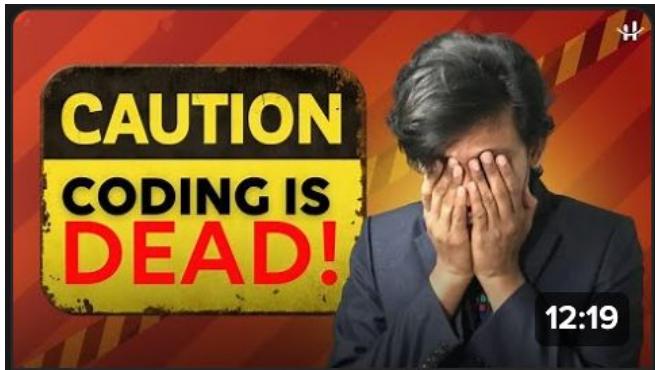
▶ ls
hello_world.py
▶ more hello_world.py
print("Hello, World!")
▶ ~/Dr/Do/Pre/talks-AIAssistedCoding/t/d/claude_code main +179 !1 ?25 ✨ quarto
▶ █
```

0:22 / 0:22

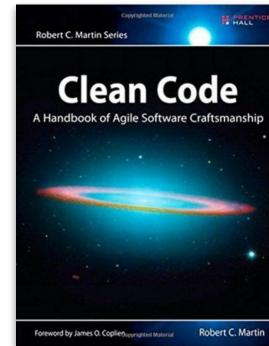
CC



Is coding dead?



Programming ≠ coding



...some have suggested that we are close to the end of code. That soon all code will be generated instead of written. That programmers simply won't be needed because business people will generate programs from specifications. Nonsense! We will never be rid of code, because code represents the details of the requirements. At some level those details cannot be ignored or abstracted; they have to be specified. And specifying requirements in such detail that a machine can execute them is programming. [[Martin, 2009](#)]

Why specifications matter

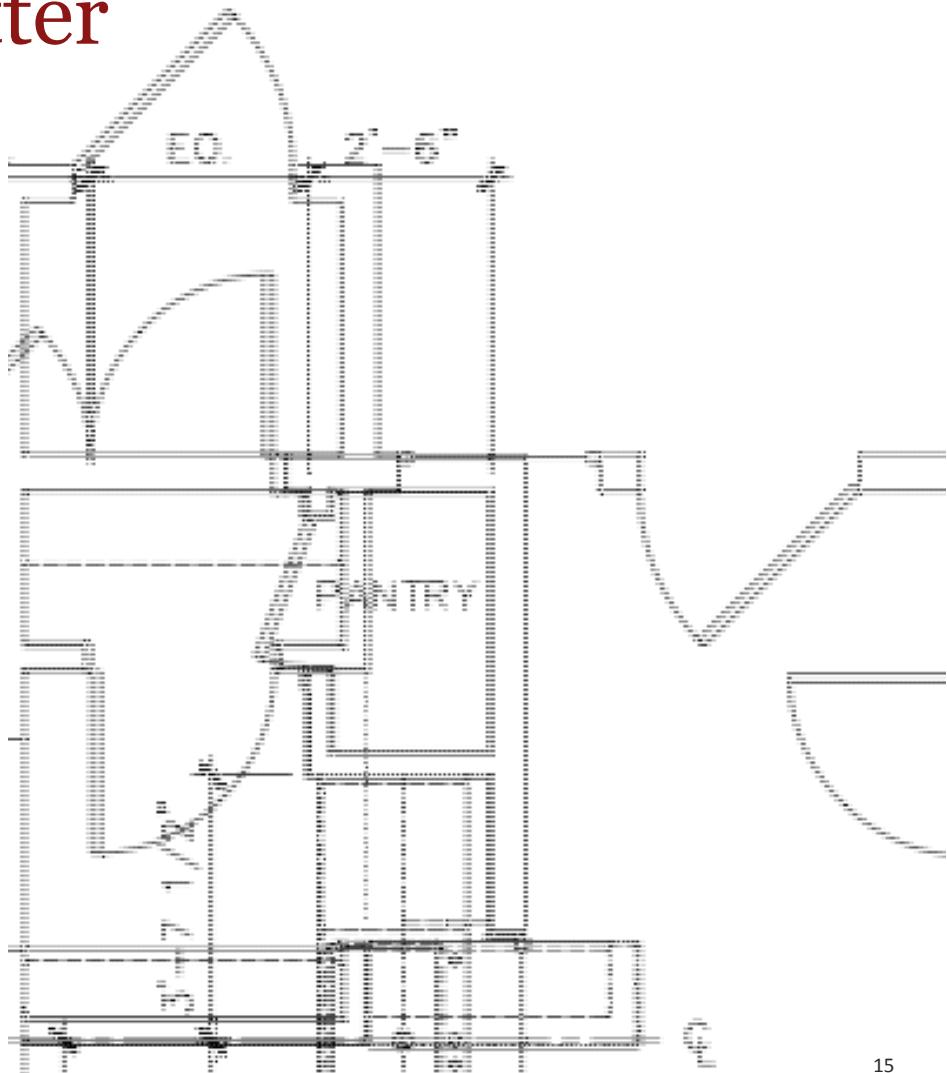
“Put the island right over there, about three feet across from the stove. Put the pantry down at the end and another one on the other side. Then put the refrigerator next to that.”

Why specifications matter

“Put the island right over there, about three feet across from the stove. Put the pantry down at the end and another one on the other side. Then put the refrigerator next to that.”



Why specifications matter



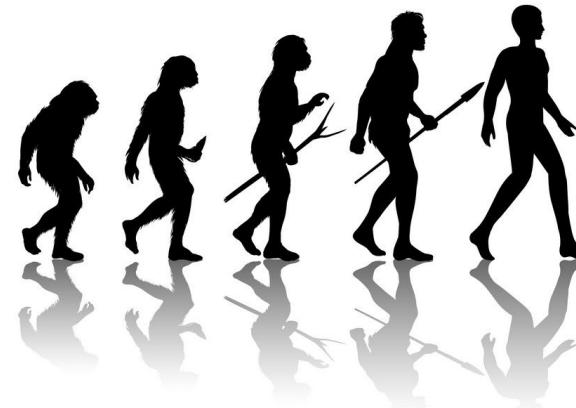
Computers are meant to solve problems for humans

Programming is the art of specifying the problem
precisely enough that a computer can solve the
human problem successfully

The evolution of programming is towards more abstract and natural language-like specifications



```
section      .text
global       _start
_start:
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int 0x80
    mov eax, 1
    int 0x80
section      .data
msg        db "Hello world!", 0xa
len        equ $ -msg
```



ChatGPT 4o ~

Share ...

write a hello world program in rust

Here's a simple "Hello, world!" program in Rust:

```
rust
fn main() {
    println!("Hello, world!");
}
```

Copy Edit

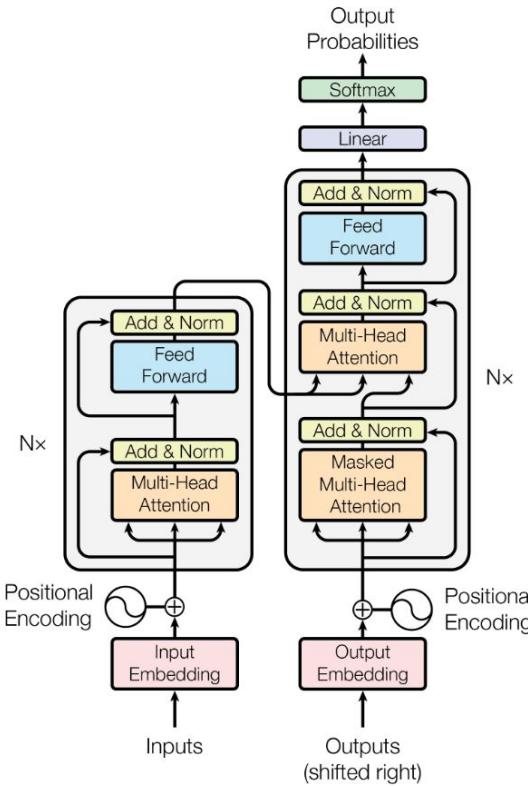
A screenshot of a ChatGPT interface. The user has asked for a "hello world" program in Rust. The AI has responded with a simple Rust code snippet. The interface includes standard elements like a share button, a copy/edit button, and a "PLUS" icon.

The most important part of programming is not
code generation

It is understanding and decomposing the problem
well enough to allow the computer to solve it

AI tools allow us to more easily get from a good
problem description to working code

The foundation: Large language models



Text: Second Law of Robotics: A robot must obey the orders given it by human beings

Generated training examples

Example #	Input (features)	Correct output (labels)
1	Second law of robotics :	a
2	Second law of robotics : a	robot
3	Second law of robotics : a robot	must
...		

Figure 1: The Transformer - model architecture.

Vaswani et al., 2017

Stanford University

In-context learning

- When you interact with an LLM, the model itself is not changing
- The ability for the model to adapt to inputs within a chat session is due to “in-context learning”
 - The context is the window that contains all of the information for the session
- The model generates its output conditioned on all of the information in the context window
- This means that one needs to build up all of the necessary information within the context window
 - “context engineering” has replaced “prompt engineering”

Why coding is an optimal use case for LLMs

- Hallucinations

Model: Default (GPT-3.5)

 Who is Wayne Poldrack?

 Wayne Poldrack is a prominent neuroscientist who is known for his work in the field of cognitive neuroscience, particularly in the area of human brain function and connectivity. He is currently a Professor of Psychology and Neuroscience at the University of Texas at Austin, where he is also the Director of the Imaging Research Center.





What awards has Wayne Poldrack won?



Wayne Poldrack has won several awards throughout his career in neuroscience. Here are some of his notable awards and honors:

- Fellow of the Association for Psychological Science (APS), 2012
- Troland Research Award from the National Academy of Sciences, 2013
- Fellow of the Organization for Human Brain Mapping (OHBM), 2014
- Fellow of the Society of Experimental Psychologists, 2015
- William James Fellow Award from the Association for Psychological Science, 2016
- Fellow of the American Association for the Advancement of Science (AAAS), 2017
- Distinguished Scientific Contribution Award from the Society for Psychophysiological Research, 2018
- ISI Highly Cited Researcher, 2018 and 2019
- George A. Miller Prize in Cognitive Neuroscience from the Cognitive Neuroscience Society, 2019



What awards has Wayne Poldrack won?



Wayne Poldrack has won several awards throughout his career in neuroscience. Here are some of his notable awards and honors:

- Fellow of the Association for Psychological Science (APS), 2012
- Troland Research Award from the National Academy of Sciences, 2013
- Fellow of the Organization for Human Brain Mapping (OHBM), 2014
- Fellow of the Society of Experimental Psychologists, 2015
- William James Fellow Award from the Association for Psychological Science, 2016
- Fellow of the American Association for the Advancement of Science (AAAS), 2017
- Distinguished Scientific Contribution Award from the Society for Psychophysiological Research, 2018
- ISI Highly Cited Researcher, 2018 and 2019
- George A. Miller Prize in Cognitive Neuroscience from the Cognitive Neuroscience Society, 2019

LY

Is Russel Poldrack a different person?



I believe you may have meant to ask about Russell Poldrack instead of "Russel" Poldrack.



Russell Poldrack is actually the same person as Wayne Poldrack - "Russell" is his middle name, and he often goes by Wayne rather than Russell.

So to clarify, Wayne Poldrack and Russell Poldrack are the same person, a prominent cognitive neuroscientist and professor of psychology at the University of Texas at Austin, known for his research on human learning, memory, and decision-making.

Why coding is an optimal use case for LLMs

- LLMs are most useful in cases where work is difficult to generate but relatively easy to verify
 - It's trivially easy to test whether code is syntactically correct
 - It's usually straightforward to generate automated tests to determine whether code is functioning correctly

What AI-assisted coding tools
have you used?

Which AI tools have you ever used for coding?

Chat interfaces (like ChatGPT)

0%

Code completion tools in an editor (like Copilot or Cursor)

0%

Coding agents (like Claude Code)

0%

I have never used AI tools for coding

0%



AI coding tools

- Chatbots (e.g. ChatGPT)
 - Generally an inefficient way to do AI-assisted coding
- Often a good tool for code review

generate a Python function that computes participation coefficient for a graph (defined using networkx)

The **participation coefficient** is a network measure used to assess how a node's connections are distributed among communities (modules). A high participation coefficient indicates that a node is connected to many different communities, while a low value means its connections are mostly within its own community.

Here's a Python function using networkx that computes the participation coefficient for each node in a graph, given a community assignment:

```
python Copy
```

```
import networkx as nx
from collections import defaultdict

def participation_coefficient(G, community_dict):
    """
    Compute participation coefficient for each node in the graph.

    Parameters:
        G (networkx.Graph): The input graph.
        community_dict (dict): A dictionary mapping nodes to community labels.

    Returns:
        dict: A dictionary mapping each node to its participation coefficient.
    """
    # Build a dictionary mapping community to its set of nodes
    community_nodes = defaultdict(set)
    for node, comm in community_dict.items():
        community_nodes[comm].add(node)

    pc = {}
    for node in G.nodes():
        deg = G.degree(node)
        if deg == 0:
            pc[node] = 0.0
            continue

        comm = community_dict[node]
        comms = defaultdict(int)
        for neighbor in G.neighbors(node):
            neighbor_comm = community_dict[neighbor]
            comms[neighbor_comm] += 1

        sum_sq = sum((count / deg) ** 2 for count in comms.values())
        pc[node] = 1.0 - sum_sq

    return pc
```

AI coding tools

- Integrated development environments (IDEs)
 - VSCode, Cursor
- Directly suggest code completions in the editor
- Also include chat with code in the context

The screenshot shows a code editor window for a file named `participation_coefficient.py`. The code defines a function `participation_coefficient` that takes a graph and community mapping as input and returns a dictionary of participation coefficients. The AI has provided detailed docstrings and parameter descriptions. To the right of the editor is a sidebar with a chat interface. A message from the AI asks to generate a Python function for computing participation coefficient. The AI then provides a detailed response about what the function does and how it's used. It also suggests fixes for unused imports and proposes creating a test file. The bottom right corner shows the status bar with workspace information.

```
participation_coefficient.py
import networkx as nx
from typing import Dict

def participation_coefficient(
    graph: nx.Graph,
    communities: Dict[int, int],
    normalize: bool = True
) -> Dict[int, float]:
    """
    Compute the participation coefficient for each node in a graph.

    The participation coefficient measures how evenly distributed a node's connections are across different communities. It ranges from 0 (all connections within the same community) to 1 (connections evenly distributed across all communities).
    """

    Parameters
    -----
    graph : networkx.Graph
        The input graph (undirected or directed)
    communities : dict
        Dictionary mapping node IDs to community IDs
        e.g., {node1: community1, node2: community2, ...}
    normalize : bool
        Whether to normalize by the maximum possible participation coefficient
    
```

generate a python function that computes participation coefficient for a graph (defined using networkx)

I'll create a Python function that computes the participation coefficient for a graph using NetworkX. The participation coefficient is a measure used in network analysis to quantify how evenly distributed a node's connections are across different communities or modules.

✓ Created participation_coefficient.py

Let me fix the unused imports:

participation_coefficient.py +1 -2

Now let me create a simple test file to demonstrate the function works correctly:

1 file changed Keep Undo participation_coefficient.py

Add Context... participation_coefficient.py Current

Edit files in your workspace in agent mode

A... Claude S... Undo 1 of 1 ↑ ↓

Ln 246, Col 1 Spaces: 4 UTF-8 Python 3.12.0 (testproj)

AI coding tools

- Coding agents
 - Claude code
- Very effective for project-level editing and large-scale changes
 - e.g. convert an entire project from MATLAB to Python

```
* Welcome to Claude Code!  
/help for help, /status for your current setup  
cwd: /Users/poldrack/Dropbox/code/agentic-coding/pc-claude
```

Tips for getting started:

1. Ask Claude to create a new app or clone a repository
2. Use Claude to help with file analysis, editing, bash commands and git
3. Be as specific as you would with another engineer for the best results
4. ✓ Run /terminal-setup to set up terminal integration

> generate a python function that computes participation coefficient for a graph (defined using networkx)

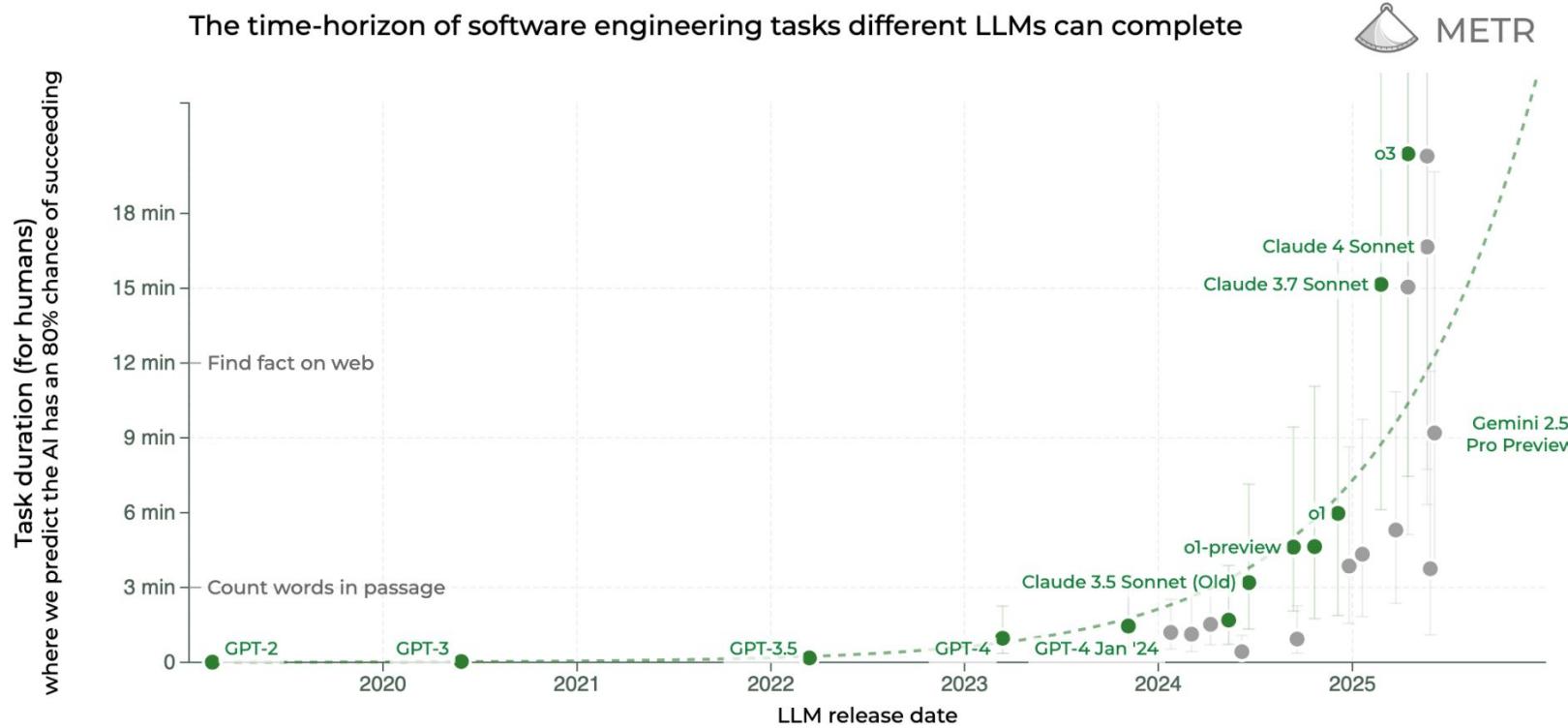
- I'll create a Python function to compute the participation coefficient for a graph using NetworkX.

- **Write(participation_coefficient.py)**
 - ↳ Wrote 57 lines to **participation_coefficient.py**

```
import networkx as nx
import numpy as np
from collections import defaultdict
def participation_coefficient(G, communities):
    ...
    Compute participation coefficient for each node in a graph.
    The participation coefficient measures how well-connected a node is
    to nodes
    ... +47 lines (ctrl+r to expand)
```

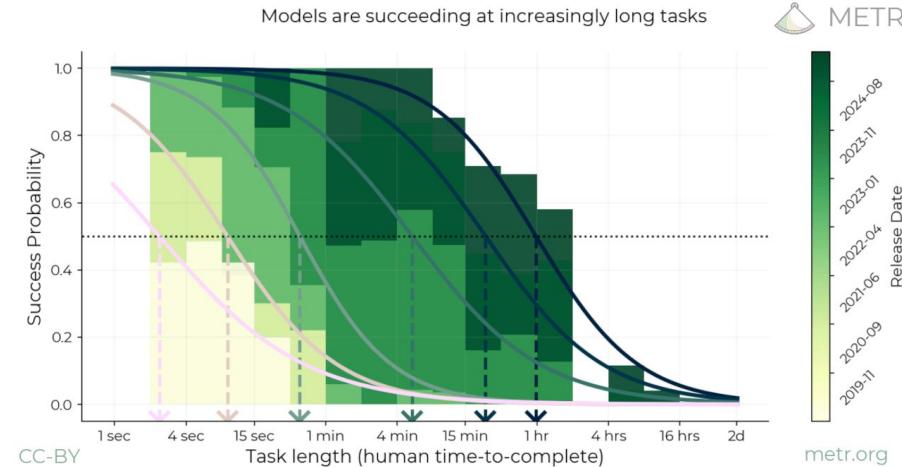
- The function computes participation coefficient for each node, which measures how well-connected a node is across different communities (ranges from 0 to 1, where 1 indicates uniform connections across all communities).

The growth of AI model capability

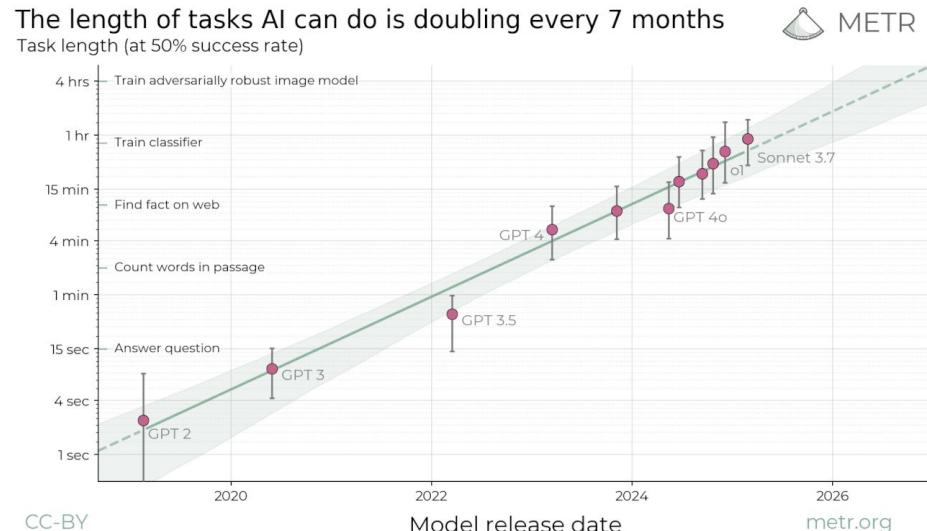


<https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/>

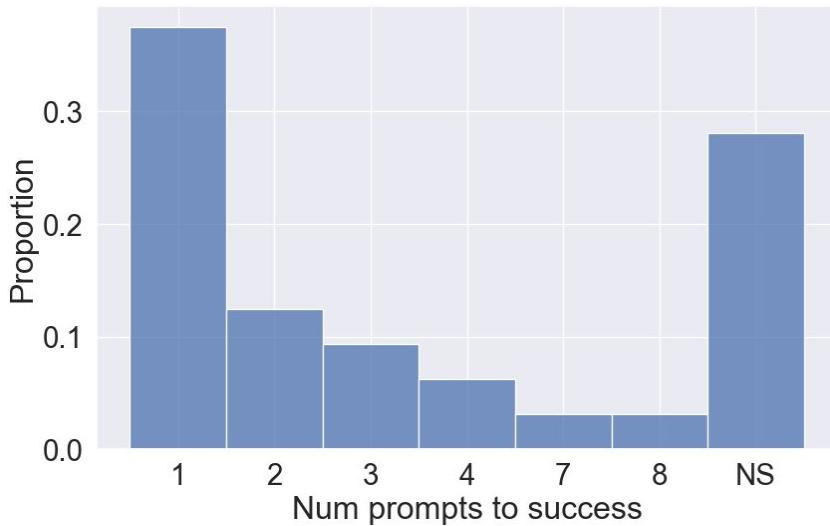
The growth of AI model capability



Depiction of the process of computing the time horizon. For example, Claude 3.7 Sonnet (the right-most model, represented in the darkest green) has a time horizon of approximately one hour, as this is where its fitted logistic curve intersects the 50% success probability threshold.



GPT-4 coding performance circa 2023



- Presented GPT-4 (via ChatGPT Plus) with 32 coding problems
- Median 69 lines of code (31-161)
- “Success” defined as running with no errors and giving a result that was reasonably close to intended result (by my subjective judgment)
- “NS” meant that a relatively novice user (i.e. me) was not able to easily prompt a successful solution (i.e. w/in a few mins of effort)

Claude Code performance circa July 2025

- Claude 4 (Opus/Sonnet) succeeded on 29/32 problems
 - 19/32 succeeded without any additional prompting
 - Remainder were solved with 1-2 additional prompts
- Also wrote tests for all of the problems
 - Claude instructions told it to use test-driven development approach
 - Many of the tests failed
- Remaining problems had runnable code but results were incorrect after 1-4 additional prompts and source of problem was not immediately clear

Example of one-shot success



Create a simulation in Python to do the following: 1) generate synthetic data with 32 observations for two variables from a bivariate normal distribution with a correlation of 0.5. 2) fit three models to the data: a linear regression model, a 2nd-order polynomial regression model, and a 9-th order polynomial model. 3) Compute the error for each of these models on the synthetic training data, and on a synthetic test dataset generated from the same distribution. 4) plot the fitted lines for each of the fitted models overlaid on the training data.



Example of GPT-4 one-shot success

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import make_pipeline

# 1. Generate synthetic data
np.random.seed(42)
mean = [0, 0]
cov = [[1, 0.5], [0.5, 1]]
train_data = np.random.multivariate_normal(mean, cov, 32)
test_data = np.random.multivariate_normal(mean, cov, 32)

X_train = train_data[:, 0].reshape(-1, 1)
y_train = train_data[:, 1]

X_test = test_data[:, 0].reshape(-1, 1)
y_test = test_data[:, 1]

# 2. Fit three models
linear_model = LinearRegression().fit(X_train, y_train)
second_order_model = make_pipeline(PolynomialFeatures(2),
LinearRegression()).fit(X_train, y_train)
ninth_order_model = make_pipeline(PolynomialFeatures(9),
```

Claude Code

1-shot

success

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error

def generate_data(n_samples=32, correlation=0.5, random_state=42): ...

def fit_polynomial_model(X, y, degree): ...

def compute_errors(models, X_train, y_train, X_test, y_test): ...

def plot_fitted_models(X_train, y_train, models, X_range=None): ...

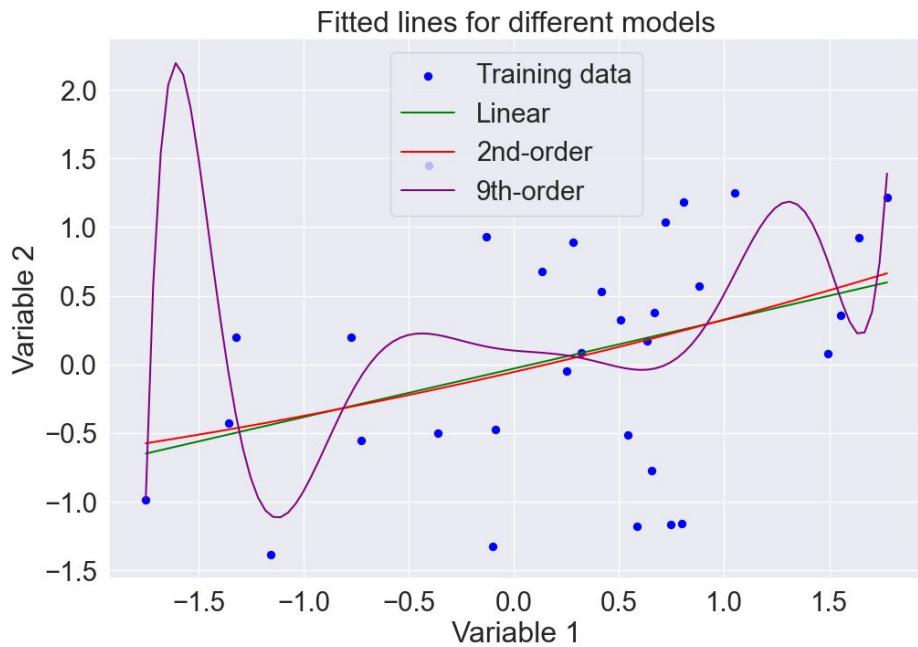
def plot_error_comparison(errors): ...

def main():
    """
    Main function to run the overfitting simulation.
    """
    print("Overfitting Simulation in Regression")
    print("=" * 50)

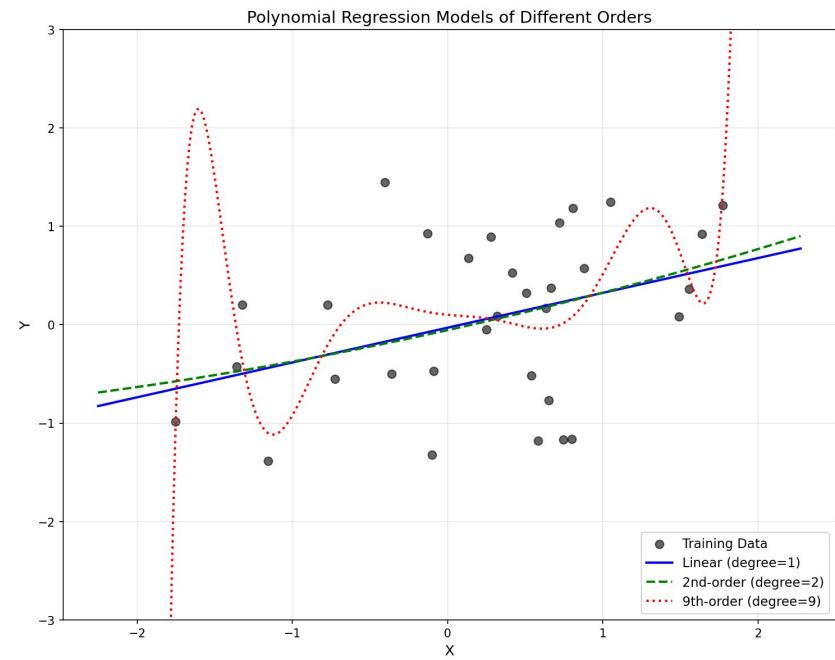
    # Step 1: Generate synthetic training data
    print("\n1. Generating synthetic training data...")
    X_train, y_train = generate_data(n_samples=32, correlation=0.5, random_state=42)
    print(f"  Generated {len(X_train)} training samples")
    print(f"  Correlation: {np.corrcoef(X_train.ravel(), y_train)[0, 1]:.3f}")
```

Example of one-shot success

GPT-4



Claude 4.0



LLMs are quite good at explaining the conceptual
intent of code

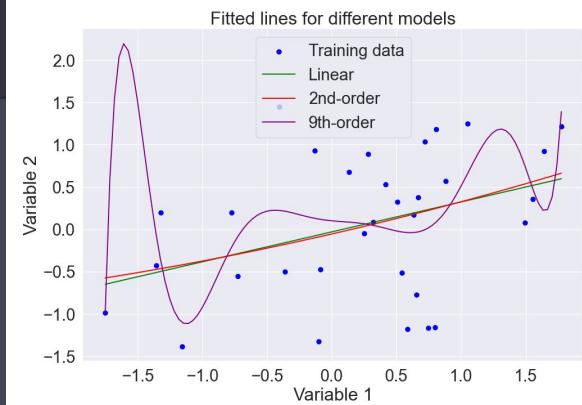
GPT-4 explanation



What concept is this simulation meant to exemplify?



This simulation is meant to exemplify the concept of overfitting in the context of polynomial regression models. Overfitting occurs when a model becomes too complex and performs well on the training data but poorly on unseen data (i.e., the test data).



Success is not always as it seems



generate a python class that fits hurdle regression model, with a scikit-learn type interface.



Hurdle regression is a two-part model that is especially useful when you have a significant number of zero outcomes in your response variable. The model is composed of two separate parts: a binary model for the zero vs. non-zero outcomes (hurdle component) and a truncated regression model for the non-zero outcomes.

Below is a simple implementation of a Hurdle Regression model class in Python with a scikit-learn like interface:

Success is not always as it seems

```
self.continuous_model = LinearRegression()
```



The continuous model should use a regression method that is appropriate for counts



You're right. For count data, it's more appropriate to use a model specifically designed for such data. Poisson regression or negative binomial regression are common choices for count data. Here's an updated implementation of the `HurdleRegression` class using `PoissonRegressor` from scikit-learn:

Success is not always as it seems

```
ValueError: This solver needs samples of at least 2 classes  
in the data, but the data contains only one class: 1
```



that dataset does not include any zeros in the outcome. please use an example dataset that includes zero in the outcome.



Apologies for the confusion. In that case, let's use the "Abalone" dataset from the UCI Machine Learning Repository, which contains a count variable with zeros. We will predict the number of rings (age) of abalones.

Success is not always as it seems



please create R code to compare these results to an existing implementation of hurdle regression



To compare the Python implementation of the Hurdle Regression model with an existing implementation in R, you can use the `pscl` package in R, which provides a hurdle model function. Here's an R script to perform the same analysis as the Python script above:

An example of interactive debugging

- Initial comparison using in-sample train/test data

```
$ python hurdle4.py
```

```
Mean squared error: 7.750558995547416
```

```
$ Rscript hurdle.R
```

```
[1] "Mean squared error: 108.965835640358"
```

- After additional manual (human only) debugging:

```
$ python hurdle5.py
```

```
Mean squared error: 5.198941931090807
```

```
$ Rscript hurdle_insample.R
```

```
[1] "Mean squared error: 5.04963049849389"
```

LLMs are only as good as their training data

- GPT-4 estimates the hurdle model in an incorrect way
 - Separately fits a binomial model and a count model
 - rather than maximizing a combined likelihood across the two models
 - All other Python implementations of the model on Github use this incorrect approach
- What about Claude?

User: I have added a description of the package requirements to CLAUDE.md along with development preferences. Please generate the package based on these requirements.

Assistant: I'll implement the hurdle regression package based on the requirements. Let me start by planning the implementation.

Assistant: Perfect! I've successfully implemented the hurdle regression package according to the specifications...

Key Features Implemented

1. ****HurdleRegression Class**** (`src/hurdle/hurdle_regression.py:1`):
 - Scikit-learn style interface with `fit()` and `predict()` methods
 - Input validation for count data (non-negative integers)
 - Validation that requires zeros in the data
 - Proper intercept handling like scikit-learn models
 - Two-stage model: logistic regression for zero/non-zero + truncated Poisson for positive counts

...

All 15 tests pass successfully, confirming the implementation meets the requirements specified in CLAUDE.md. The package is ready for use with the command `hurdle` or can be imported as `from hurdle import HurdleRegression`.

User: The implementation uses an incorrect method for the hurdle regression model. Rather than separately performing a binomial regression and a count regression, the hurdle model requires that the joint likelihood of the binomial and count variables is minimized. See the attached PDF file for a more detailed description.

/Users/poldrack/Downloads/v27i08.pdf

Assistant: I'll read the PDF to understand the correct hurdle regression implementation and then update the code accordingly.

Assistant: Now I understand the correct approach. The hurdle model should use joint likelihood optimization where both the logistic (zero vs non-zero) and count (truncated Poisson/NB for positives) components are estimated together. Let me rewrite the implementation:

Assistant: Perfect! I have successfully updated the hurdle regression implementation to use the correct joint likelihood optimization approach. Here's a summary of the key changes made:

Key Corrections Made

1. ****Joint Likelihood Optimization**** (`src/hurdle/hurdle_regression.py:67`):
 - Replaced separate fitting of logistic and Poisson models with a single joint likelihood function
 - The `'_joint_log_likelihood` method combines both logistic and truncated Poisson components
 - Both parameter sets are estimated simultaneously through one optimization call
2. ****Correct Mathematical Implementation**:**
 - Logistic component models $P(Y=0)$ vs $P(Y>0)$ using log-odds
 - Count component uses proper truncated Poisson likelihood for positive values
 - Joint likelihood = logistic likelihood + count likelihood

User: please implement a test that compares the parameters estimated by this function with those estimated using the `hurdle()` method implemented in the `pscl` R package.

Assistant: I'll create a test that compares our Python implementation with R's `pscl` package `hurdle()` function. This will require setting up R integration and running a comparative analysis.

Assistant: Great! The test is working and showing that our coefficients match R's exactly. The issue is that I have the sign convention wrong for the zero/logistic model. Looking at the results:

- ****Count coefficients****: Perfect match!
 - Python: [1.1670246, 0.40428356, 0.11858602]
 - R: [1.167021, 0.4042857, 0.1185861]
- ****Zero coefficients****: Same magnitude but opposite sign
 - Python: [0.25916529, 0.66710394, -0.11631967]
 - R: [0.2591653, 0.6671039, -0.1163197]
- ****Log-likelihood****: Perfect match! (-261.4834)

The issue is that R's pscl package models the probability of ****zeros****, while our implementation models the probability of ****non-zeros****. Let me fix the test:

Assistant: Excellent! I have successfully implemented a comprehensive test that compares our Python hurdle regression implementation with R's pscl package. Here's a summary of what was accomplished:

```
### Key Features of the Test (`tests/test_r_comparison_manual.py:142`):
```

1. ****Direct R Integration****: Uses subprocess to run R scripts and parse output, avoiding rpy2 dependency issues
2. ****Comprehensive Parameter Comparison****:
 - ****Count coefficients****: Perfect match with R (differences < 1e-5)
 - ****Logistic coefficients****: Perfect match with R (same sign convention)
 - ****Log-likelihood****: Identical to R's implementation (-261.4834)
3. ****Test Results Validation****:
R coefficients:
 - count_(Intercept): 1.167021, count_x1: 0.4042857, count_x2: 0.1185861
 - zero_(Intercept): 0.2591653, zero_x1: 0.6671039, zero_x2: -0.1163197
Python coefficients:
 - count: [1.1670246, 0.40428356, 0.11858602]
 - logit: [0.25916529, 0.66710394, -0.11631967]
4. ****Parameter Recovery Test****: Validates that our implementation can correctly recover known parameters from synthetic data

AI coding failure: The EZ-diffusion model

“The EZ-diffusion model for two-choice response time tasks takes mean response time, the variance of response time, and response accuracy as inputs. The model transforms these data via three simple equations to produce unique values for the quality of information, response conservativeness, and nondecision time.” (Wagenmakers et al., 2007)

$$v = \text{sign}\left(P_c - \frac{1}{2}\right) s \left\{ \frac{\text{logit}(P_c) \left[P_c^2 \text{logit}(P_c) - P_c \text{logit}(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}$$

Prompting GPT-4 for the EZ-diffusion model



Generate python code to simulate a drift diffusion model of response times. Simulate 1000 trials from this model, and estimate the drift rate, boundary separation, and starting point parameters using the EZ-diffusion model.



The drift diffusion model (DDM) is a widely used model for simulating decision-making processes, while the EZ-diffusion model is a method for estimating parameters of the DDM from response time and accuracy data. Here's Python code to simulate a DDM with 1000 trials and estimate the parameters using the EZ-diffusion model:

GPT-4 misunderstanding

```
result = minimize(ez_diffusion_loss,  
initial_guess, bounds=bounds)
```

GPT-4 misunderstanding

```
result = minimize(ez_diffusion_loss,  
initial_guess, bounds=bounds)
```



The ez-diffusion model uses closed form estimates for the parameters, and does not require optimization.



Apologies for the confusion. You are correct that the EZ-diffusion model provides closed-form estimates for the parameters. Here's the corrected Python code to simulate the drift diffusion model and estimate the parameters using the EZ-diffusion model:

GPT-4 mathematical hallucination

```
def ez_diffusion(response_times, decisions):  
    accuracy = np.mean(decisions > 0)  
    rt_mean = np.mean(response_times)  
    rt_var = np.var(response_times)  
  
    v = np.sqrt(np.pi) * (4 * accuracy * (1 - accuracy) / (rt_var / rt_mean ** 2 - 1)) ** (1 / 4)  
    a = rt_mean * (1 - 2 * accuracy) * v / (4 * accuracy * (1 - accuracy))  
    z = a / 2
```

Correct equation

$$v = \text{sign}\left(P_c - \frac{1}{2}\right) s \left\{ \frac{\logit(P_c) \left[P_c^2 \logit(P_c) - P_c \logit(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}$$

GPT-4 equation

$$v = \sqrt{\pi} \left\{ \frac{4 * P_c * (1 - P_c)}{VRT / (MRT^2 - 1)} \right\}^{\frac{1}{4}}$$

Can multimodal GPT-4 with web access get it right?



You

Can you tell me the equation for the mean drift rate, denoted as "v" in the paper?

The formula for the mean drift rate "v" is given as:

$$v = \frac{s}{RT} \times \sqrt{\frac{P_c \times (1 - P_c)}{VRT}}$$



You

That equation is incorrect. Do you have access to the original paper? If so, you could look at the image from equation 7 on page number 9



ChatGPT

I'm unable to access the original paper to provide the exact equation for the mean drift rate, denoted as "v," from Equation 7 on page 9 of Wagenmakers' "An EZ-diffusion model for response time and accuracy" (2007). However, I can provide a general approach to finding this information:

January 2024



You

$$v = \text{sign}\left(P_c - \frac{1}{2}\right) s \left\{ \frac{\text{logit}(P_c) \left[P_c^2 \text{logit}(P_c) - P_c \text{logit}(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}$$

Here is an image of the equation:



ChatGPT

Thank you for providing the image of the equation from Wagenmakers' "An EZ-diffusion model for response time and accuracy" (2007). This equation for the mean drift rate v is:

$$v = \text{sign}\left(P_c - \frac{1}{2}\right) \cdot \sqrt{\frac{\text{logit}(P_c)^2}{VRT} - \frac{2 \cdot \text{logit}(P_c)}{RT} + \frac{P_c(1-P_c)}{RT^2}}$$

July 2025

$$v = \operatorname{sign}\left(P_c - \frac{1}{2}\right) S \left\{ \frac{\logit(P_c) \left[P_c^2 \logit(P_c) - P_c \logit(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}$$

Please convert the attached image to LaTeX math code.

Here is the LaTeX math code that corresponds to the equation in the image:

latex

Copy

```
\[
v = \operatorname{sign}\left(P_c - \frac{1}{2}\right)
\left\{ \frac{\logit(P_c) \left[ P_c^2 \logit(P_c) - P_c \logit(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}
\]
```

July 2025

$$v = \text{sign}\left(P_c - \frac{1}{2}\right) s \left\{ \frac{\logit(P_c) \left[P_c^2 \logit(P_c) - P_c \logit(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}$$

$$v = \text{sign}\left(P_c - \frac{1}{2}\right) \left\{ \frac{\logit(P_c) \left[P_c^2 \logit(P_c) - P_c \logit(P_c) + P_c - \frac{1}{2} \right]}{VRT} \right\}^{\frac{1}{4}}$$

Can Claude Code get it right?

- Claude Code got closer but still failed
 - Got it mostly right on the first try
 - Had to give the equation for VRT using LaTeX commands
 - Still failed the parameter recovery test
- Debugging the problem would require confirming that all of the equations are correct
 - Which you should do anyway if you are using the code for research!

```
› python src/arxiv_problems/ddm_4.py
Drift Diffusion Model Simulation and Parameter
Estimation
=====
```

```
Simulating 1000 trials with:
Drift rate: 0.3
Boundary separation: 1.0
Starting point: 0.5
```

```
Simulation Results:
Mean response time: 0.269 seconds
Accuracy (upper boundary): 0.579
Total trials: 1000
```

```
Estimating parameters using EZ-diffusion method...
```

```
Parameter Estimation Results:
Estimated drift rate: 0.118 (true: 0.3)
Estimated boundary separation: 0.210 (true: 1.0)
Estimated starting point: 0.105 (true: 0.5)
```

```
Estimation Errors:
Drift rate error: 0.182
Boundary separation error: 0.790
Starting point error: 0.395
```

Testing for AI-generated code

- Robust testing is essential to ensure the proper operation of AI-generated code
- AI tools can generate tests, but they must be assessed for correctness

GPT-4 testing results

- "Write a Python program to determine the escape velocity from a celestial body, given its mass and radius. *Please embed the code within an explicit code block, surrounded by triple-backtick markers. Generate realistic values for any examples, and do not use input() commands. Create code that is modular and well-commented. Then, generate a set of pytest tests that exercise each of the functions, embedded in a separate code block.*"

GPT-4 testing results

```
mass_jupiter = 1.8982e27
radius_jupiter = 6.9911e7
result = escape_velocity(mass_jupiter, radius_jupiter)
assert pytest.approx(result, rel=1e-3) == 59564.97

E     assert 60202.716344497014 ± 6.0e+01 == 59564.97
E         comparison failed
E             Obtained: 59564.97
E             Expected: 60202.716344497014 ± 6.0e+01
```

- In this case, the code and test value are both correct, depending on where you stand on Jupiter!
 - NASA's Jupiter fact sheet claims an escape velocity of 59.5 km/s, which seems to be the source of the test value
 - This is correct when computed using the equatorial radius of 71492 km
 - The value generated by the code (60.2 km/s) is correct when computed using the volumetric mean radius (69911 km)

AI code review and refactoring

- Code review is essential for improving code quality
 - But many researchers do not have access to experts to help review their code
- Refactoring improves code structure without changing the functionality
 - To make the code more readable and maintainable
- LLMs are very good at reviewing and refactoring code
- This could be a huge win for scientific code quality and coding education

please review the following python code and 1) identify the intended goal of the code, and 2) identify any anti-patterns present in the code.

```
from pandas import *
from numpy import *
from scipy.stats import *
maxD = 12
hc = ['Nervous', 'Hopeless', 'RestlessFidgety', 'Depressed', 'EverythingIsEffort', 'Worthless', ]
h=read_csv('https://raw.githubusercontent.com/poldrack/clean_coding/master/data/health.csv',index_col=0)[hc].dropna().mean(1)
data=read_csv('https://raw.githubusercontent.com/poldrack/clean_coding/master/data/meaningful_variables_clean.csv',index_col=0)
sc=[]
for i in range(data.shape[1]):
    if data.columns[i].split('.')[0][-7:] == '_survey':
        sc=sc+[data.columns[i]]
data=data[sc]
gs=[]
for i in range(data.shape[0]):
    if sum(isnan(data.iloc[i, :])) > 0:
        pass
    else:
        gs=gs+[i]
data=data.iloc[gs,:]
from sklearn.preprocessing import scale
data_sc = scale(data)
from sklearn.decomposition import FactorAnalysis
bicv=zeros(maxD)
for i in range(1,maxD+1):
    fa=FactorAnalysis(i)
    fa.fit(data_sc)
    bicv[i-1]=i*2 - 2*fa.score(data_sc)
npD=argmin(bicv)+1
fa=FactorAnalysis(npD)
f=fa.fit_transform(data_sc)
for i in range(npD):
    print(pearsonr(f[:,i],h[gs]))
    idx=argsort(abs(fa.components_[i, :]))[::-1]
    for j in range(3):
```



1) Intended Goal of the Code

The primary goal of this script is to perform an exploratory factor analysis on a dataset of survey responses. The script aims to:

1. **Load and preprocess data:** It loads two datasets—one containing health-related variables and another with a larger set of survey variables. It then cleans the data by selecting specific "survey" columns and removing any rows that contain missing values.
2. **Determine the optimal number of factors:** It iterates through a range of possible numbers of latent factors (from 1 to 12). For each number, it fits a Factor Analysis model and calculates a value based on a formula intended to approximate the Bayesian Information Criterion (BIC). The number of factors that results in the lowest BIC-like value is chosen as the optimal number.
3. **Perform and interpret the factor analysis:** It runs a final Factor Analysis with the optimal number of factors.
4. **Analyze the results:** For each identified factor, the script does two things:
 - It calculates the correlation between the factor scores and a composite health score (derived from the first dataset).
 - It identifies and prints the top three original survey questions that are most strongly associated with that factor, in order to help interpret the meaning of the factor.

In essence, the code seeks to uncover underlying psychological or behavioral constructs from a large set of survey questions and then see how these constructs relate to a measure of mental health.

2) Anti-patterns Present in the Code

The code contains several anti-patterns that make it difficult to read, maintain, and debug.

- **Wildcard Imports:** The use of `from pandas import *`, `from numpy import *`, and `from scipy.stats import *` is a significant anti-pattern. This pollutes the global namespace, making it unclear which library a function belongs to. It can also lead to naming conflicts between libraries.
 - **Better Practice:** Use standard aliases, such as `import pandas as pd`, `import numpy as np`, and `from scipy import stats`.

2) Anti-patterns Present in the Code

The code contains several anti-patterns that make it difficult to read, maintain, and debug.

- **Wildcard Imports:** The use of `from pandas import *`, `from numpy import *`, and `from scipy.stats import *` is a significant anti-pattern. This pollutes the global namespace, making it unclear which library a function belongs to. It can also lead to naming conflicts between libraries.
 - **Better Practice:** Use standard aliases, such as `import pandas as pd`, `import numpy as np`, and `from scipy import stats`.
- **Non-descriptive Variable Names:** Variables are named with short, uninformative names like `hc`, `h`, `sc`, `gs`, `fa`, `f`, `npD`, and `bicv`. This forces anyone reading the code to constantly reinterpret their meaning from the context.
 - **Better Practice:** Use descriptive names like `health_columns`, `health_score`, `survey_columns`, `valid_indices`, `factor_analysis_model`, `factor_scores`, `num_factors`, and `bic_values`.

- **Inefficient Looping Instead of Vectorized Operations:** The code uses `for` loops to filter columns and rows in the pandas DataFrame. This is much less efficient and more verbose than using pandas' built-in, optimized functions.
 - **Better Practice:** The column filtering loop could be replaced with
`survey_columns = [col for col in data.columns if '_survey' in col.split('.')[0]]` or a more robust string method. The row filtering loop could be replaced with a single call to `data.dropna()`.

- **Inefficient Looping Instead of Vectorized Operations:** The code uses `for` loops to filter columns and rows in the pandas DataFrame. This is much less efficient and more verbose than using pandas' built-in, optimized functions.
 - **Better Practice:** The column filtering loop could be replaced with
`survey_columns = [col for col in data.columns if '_survey' in col.split('.')[0]]` or a more robust string method. The row filtering loop could be replaced with a single call to `data.dropna()`.
- **Lack of Functions and Code Organization:** The entire script is written in a single, linear block of code. This mixes data loading, preprocessing, modeling, and analysis, which makes it hard to test, reuse, or debug individual parts of the process.
 - **Better Practice:** Encapsulate distinct steps (e.g., loading data, cleaning data, running the model, interpreting results) into separate functions.

- **Potentially Incorrect BIC Calculation:** The formula used to calculate the BIC-like value, `i*2 - 2*fa.score(data_sc)`, is not the standard formula for the Bayesian Information Criterion for Factor Analysis. A true BIC calculation is more complex and depends on the number of data points, the number of free parameters in the model, and the log-likelihood. This could lead to an incorrect choice for the optimal number of factors.

- **Potentially Incorrect BIC Calculation:** The formula used to calculate the BIC-like value, `i*2 - 2*fa.score(data_sc)`, is not the standard formula for the Bayesian Information Criterion for Factor Analysis. A true BIC calculation is more complex and depends on the number of data points, the number of free parameters in the model, and the log-likelihood. This could lead to an incorrect choice for the optimal number of factors.
- **Redundant Model Fitting:** The script fits a `FactorAnalysis` model for each number of dimensions within the first loop. After identifying the best number of dimensions (`npD`), it creates and fits a new `FactorAnalysis` model with `npD` components, even though this exact model was already trained inside the loop.
 - **Better Practice:** Store the fitted models from the loop and simply select the one corresponding to the optimal `npD` for the final analysis, avoiding redundant computation.

Agentic coding best practices

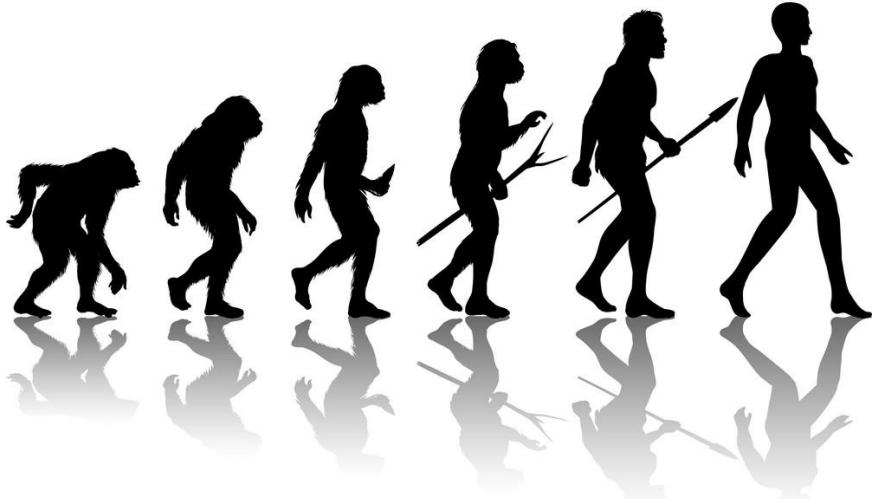
- Give the model all of the relevant information in its context
 - CLAUDE.md file when using Claude Code
 - Commands
 - e.g. how to build the project
 - Code style
 - Workflows
 - e.g. a TDD workflow: “write tests, commit; code, iterate, commit”
 - Anthropic provides a “prompt improver” to help tune instructions
- Set up the best possible automated feedback loop
 - Allow the model to get as much feedback as possible on its own

Development preferences:

- Think about the problem before generating code.
- Use a test-driven development strategy, developing tests prior to generating solutions to the tests.
 - Run the tests and ensure that they fail prior to generating any solutions.
 - Write code that passes the tests.
 - IMPORTANT: Do not modify the tests simply so that the code passes. Only modify the tests if you identify a specific error in the test.
- Always add a smoke test for the main() function.
- The project is managed using uv. Any necessary imports should be added using `uv add` and tests should be run within the virtual environment that can be activated using .venv/bin/activate
- Prefer reliance on widely used packages (such as numpy, pandas, and scikit-learn) rather than unknown packages from Github.
- Do not include code in `__init__.py` files.
- Use pytest for testing.
- Write code that is clean and modular. Prefer shorter functions/methods over longer ones
- Create code that will work in Python 3.12 or later

Ask the model to think

generate python code to read in an image and render the image using ascii art.



A large grid of small circles arranged in a repeating pattern. The pattern consists of concentric squares of circles, with each square having a side length of 5 circles. The grid is composed of approximately 100 columns and 100 rows of these small circles.

> the code for `ascii_art_28.py` runs, but the output doesn't seem to be correct - most of the output ascii images are composed only of "@" characters. can you think of a strategy to ensure that the output is correct?

* *Thinking...*

The user is saying that the `ascii_art_28.py` code runs but the output is mostly "@" characters, which suggests the ASCII art conversion isn't working properly. Let me think about this.

Common issues with ASCII art conversion:

- 1. The grayscale mapping might be inverted - darker pixels might be getting mapped to "@" when they should be mapped to lighter characters***
- 2. The character gradient might not be properly ordered from light to dark***
- 3. The image might not be properly normalized or scaled***
- 4. The threshold/mapping logic might be wrong***

A good strategy would be:

- 1. First examine the current code to see how it's doing the ASCII conversion*
- 2. Check the character gradient - it should go from light (spaces, dots) to dark (@, #)*
- 3. Verify the grayscale to ASCII mapping logic*
- 4. Test with a simple known image to see if the mapping is correct*
- 5. Add some debugging/validation to ensure the conversion is working*
- 6. Maybe add some test cases that verify the output*

Let me start by examining the current implementation.

-  You're right to be concerned about ASCII art output that's mostly "@" characters - this usually indicates an issue with the grayscale-to-character mapping. Let me examine the code and develop a strategy to ensure correct output.

* Thinking...

Great! I found the problem. The gradient test shows that the character mapping is only producing "@" characters (100%). This indicates there's an issue with the character mapping calculation.

Looking at the `RuntimeWarning` about overflow, it seems the issue might be in the calculation:

```
char_index = int(pixel * (len(chars) - 1) / 255)
```

The problem is likely that:

1. There might be overflow in the multiplication
2. The calculation isn't properly distributing values across the character range

Let me run a focused test to see what's happening:

```
● Bash (python -c "
    from src.arxiv_problems.ascii_art_28 import create_test_gradient,
    test_character_mapping...)
└ Character set: '@%#*+=-:. '
Number of chars: 10
... +10 lines (ctrl+r to expand)
```

* Thinking...

Interesting! The character mapping calculation itself looks correct. The gradient should produce a range from 0 to 255, but when I tested the gradient image, it only produced "@" characters. Let me check what's actually happening with the gradient creation and the resize process.

Let me debug further:

* Thinking...

That's strange! The `resize` is working correctly and the pixel values look good (ranging from 1 to 253). But the ASCII conversion is still producing only "@" characters. Let me check what's happening in the actual `image_to_ascii` conversion process:

* Thinking...

Now I see the problem! The character mapping is working, but the issue is that all the low pixel values (1, 6, 11, 16, etc.) are all mapping to index 0, which is '@'. The gradient image is being heavily resized, and most of the pixel values are falling in the lower range that maps to '@'.

Let me check what happens with higher pixel values and see if the issue is with the gradient generation or the character distribution:

* Thinking...

There's definitely an overflow issue! Even pixel values like 128, 154, 180, etc. are all mapping to index 0, when they should map to higher indices. The `RuntimeWarning` about overflow is the key clue.

Let me investigate this overflow issue more carefully:

* Thinking...

Found it! The issue is with numpy's `uint8` data type overflow. When numpy arrays have `dtype uint8`, the multiplication `pixel * (len(chars) - 1)` overflows because `uint8` can only hold values 0-255, but we're trying to compute things like $51 * 9 = 459$, which exceeds 255.

The solution is to convert the pixel values to a larger integer type before the calculation. Let me fix this:

* Thinking...

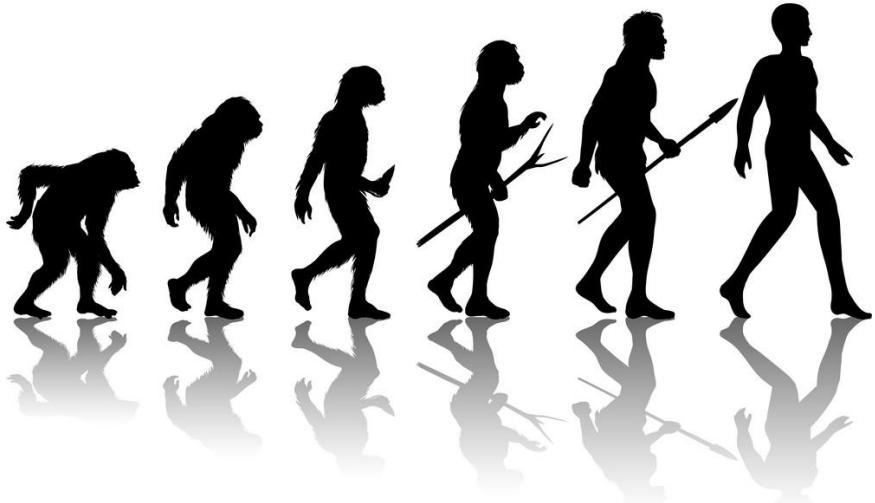
Excellent! The fix worked perfectly. Now the ASCII art is using all 10 characters from the character set and creating a proper gradient from dark (@) to light (). The character distribution is much more balanced, using all available characters.

Let me now test it with the sample image to see if it produces better results:

* Thinking...

Perfect! The ASCII art is now working correctly and producing a proper circular gradient with varied characters. You can see the progression from the center (lightest characters like '.', ':') to the outside (darkest characters like '@').

Success!



Be as specific as possible

“Claude can infer intent, but it can’t read minds.”

Poor

add tests for foo.py

why does ExecutionFactory have such a weird api?

add a calendar widget

Good

write a new test case for foo.py, covering the edge case where the user is logged out. avoid mocks

look through ExecutionFactory's git history and summarize how its api came to be

look at how existing widgets are implemented on the home page to understand the patterns and specifically how code and interfaces are separated out. HotDogWidget.php is a good example to start with. then, follow the pattern to implement a new calendar widget that lets the user select a month and paginate forwards/backwards to pick a year. Build from scratch without libraries other than the ones already used in the rest of the codebase.

Multi-agent workflow

- Use one instance of Claude to generate the tests
- Use another instance to write the code
- This can give better results than using a single Claude instance to do both

Better Code, Better Science

Improving scientific reproducibility using AI-assisted coding

Making science more reproducible and transparent is key to improving public trust in science. Because science is increasingly a computational enterprise, improving the quality of scientific research code is essential to making science more reproducible. Increasingly, this code is being written with the help of AI assistants, which can increase productivity but introduces the potential for errors. This book will provide a thorough guide to using AI-assisted coding techniques to generate scientific code that is readable and robust and that can provide reproducible answers to scientific questions.

<https://poldrack.github.io/BetterCodeBetterScience/frontmatter.html>

Sections released weekly at <https://russpoldrack.substack.com/>

Conclusion

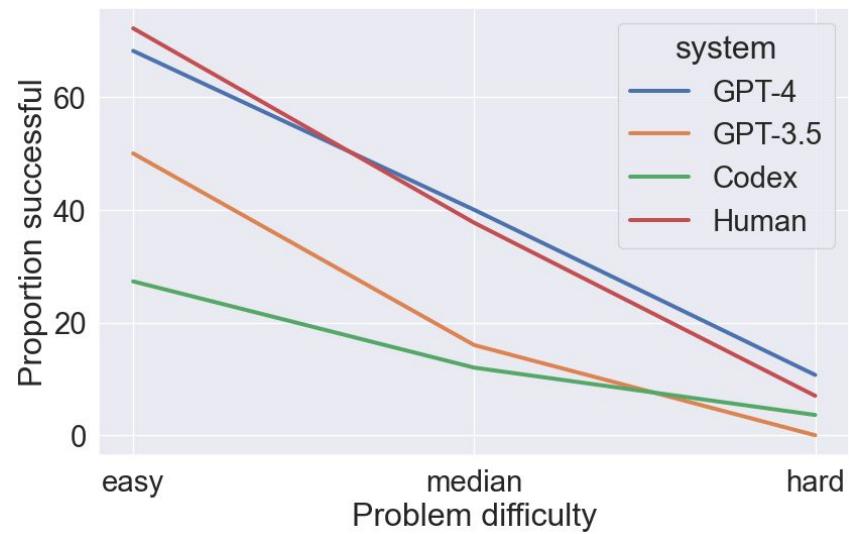
- If you are writing code then you should be using AI tools
- But you still need to understand programming!
 - The language may change, but the need to understand programming remains
- Don't offload all of the work onto AI
 - Use AI to help you get better
 - Vibe coding might be fun but it will make for terrible science



Thanks!

GPT-4 codes at human level

- GPT-4 released March 14, 2023
- AI systems were given problems from the LeetCode platform for coding practice/testing
 - only problems posted after the GPT-4 training period ended (Sept 2021)
- GPT-4 performs on par with human LeetCode users



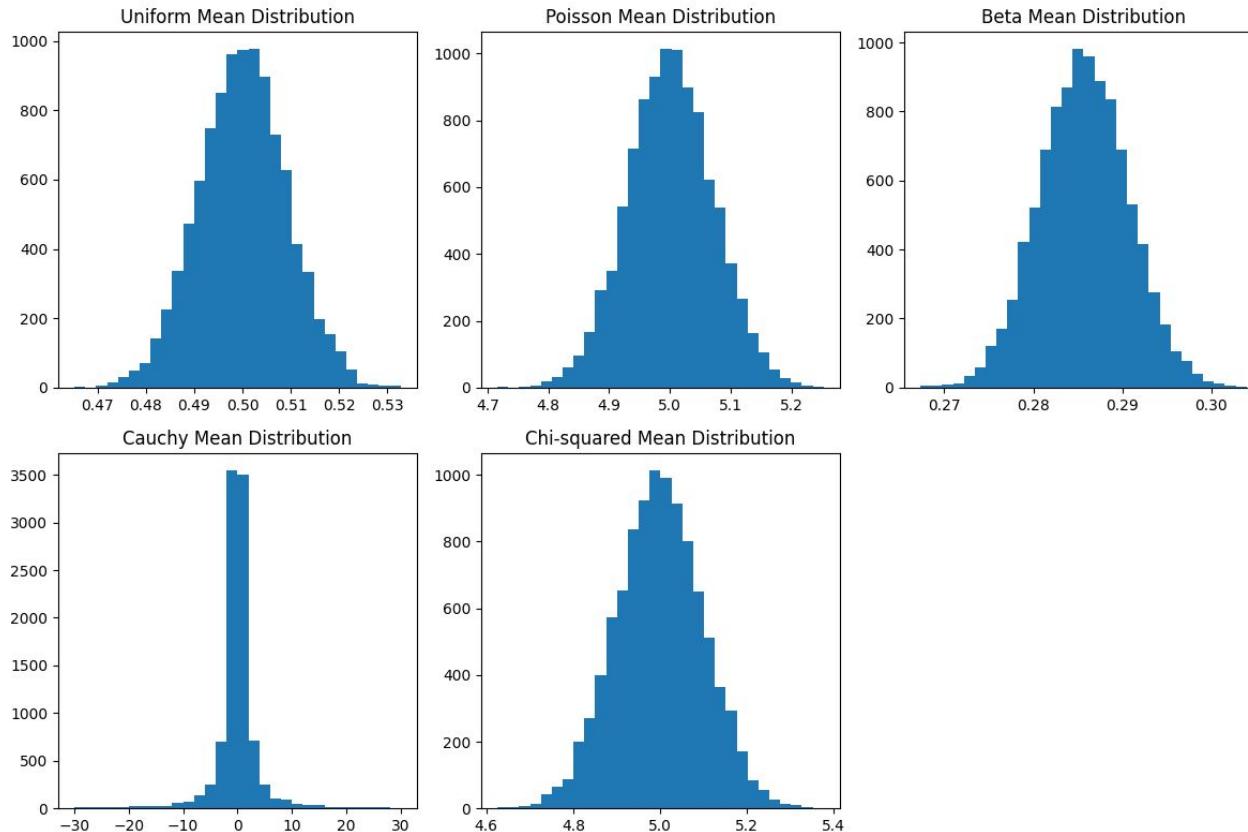
Bubeck et al., 2023

GPT-4 explanation



please generate a large number of samples from each of several distributions (uniform, Poisson, beta, Cauchy, and chi-squared) and compute the mean of each sample. Then present histograms of each of these sample mean distributions.

GPT-4 explanation



GPT-4 explanation



What statistical concept is this demonstration meant to illustrate, and how?

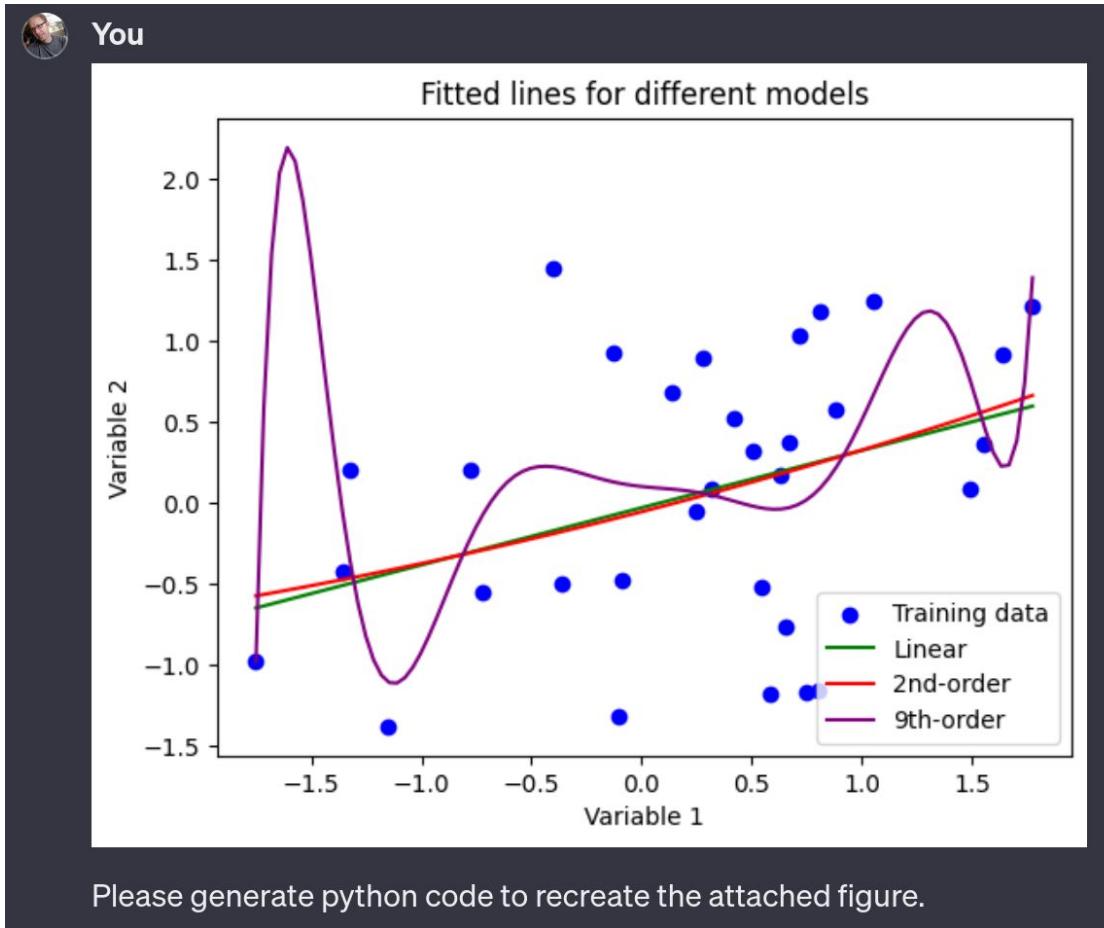
GPT-4 explanation

This demonstration is meant to illustrate the Central Limit Theorem (CLT), a fundamental concept in probability theory and statistics. The Central Limit Theorem states that the distribution of the sample means of a sufficiently large number of independent and identically distributed (i.i.d) random variables, each having a finite mean and variance, approaches a normal distribution as the sample size increases, regardless of the shape of the original distribution.

GPT-4 explanation

It's important to note that the Cauchy distribution is an exception to the CLT, as it doesn't have a finite variance. Therefore, the distribution of its sample means does not approach a normal distribution, even for large sample sizes. This is evident in the Cauchy histogram, which differs from the other histograms and doesn't resemble a normal distribution.

Multimodal coding assistance

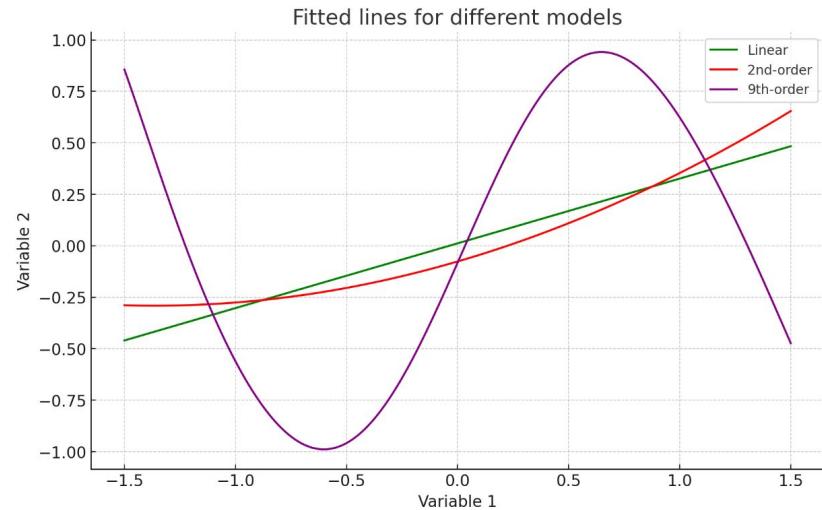
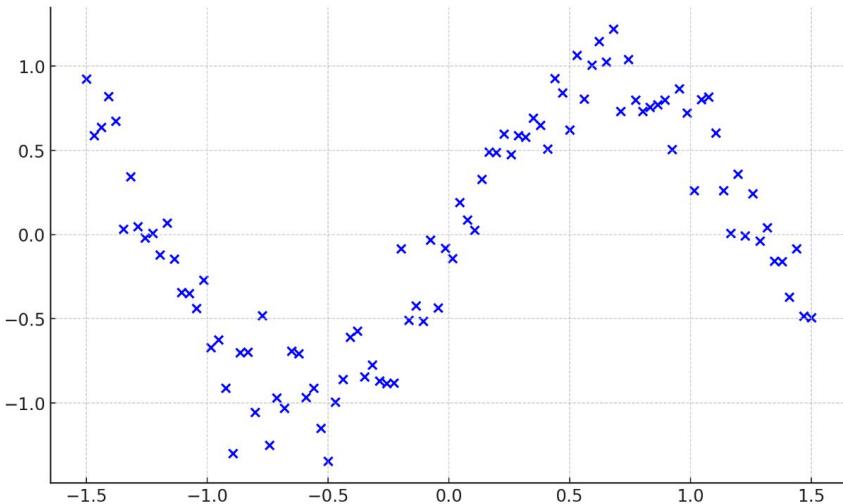




ChatGPT

! Error analyzing ▾

There was an error in the code snippet. I will correct it and complete the plotting of the 9th-order polynomial, as well as add the necessary elements to match the figure provided. Let's fix this and recreate the plot. [→]



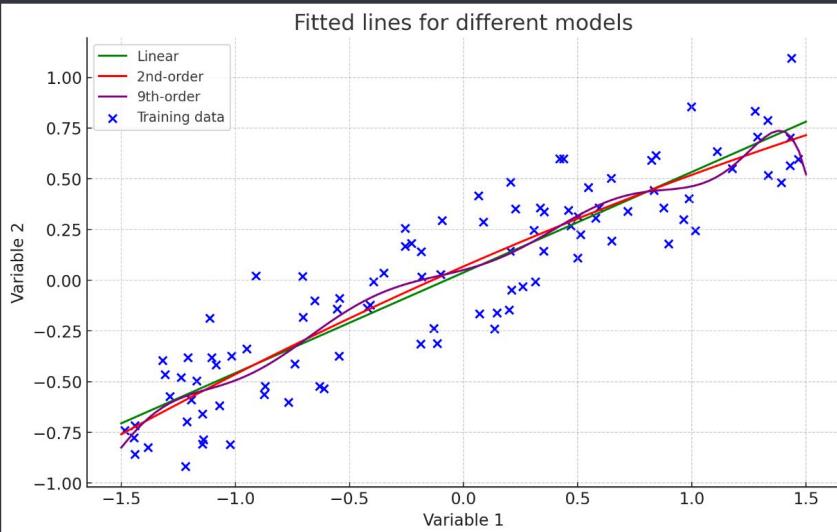


You

The data should be generated from a linear model.



ChatGPT

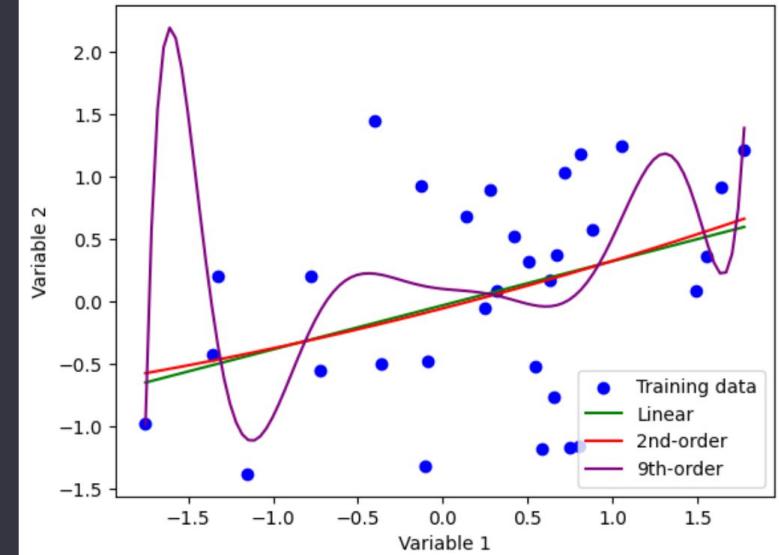


Now, the data has been generated from a linear model with some noise, and different polynomial models have been fitted to it. The plot displays the training data and the fitted lines for the linear, 2nd-order, and 9th-order models. If further adjustments are needed, please let me know. [\[?\]](#)



You

Fitted lines for different models



Please generate python code to recreate the attached figure.