

A Skeptic's Guide to Using AI for Programming

PD12



About the Speaker

Frank Menius (he/him/his)

- Senior Trainer at Atorus Research
- Data Science and Data Standards SME
- 9 Years Statistical Programming and Clinical Research Experience
- Graduate of the University of North Carolina
- Post Graduate Applied Statistics and Data Management North Carolina State University
- Previous Career as Law Enforcement Officer for the City of Raleigh



Disclaimer

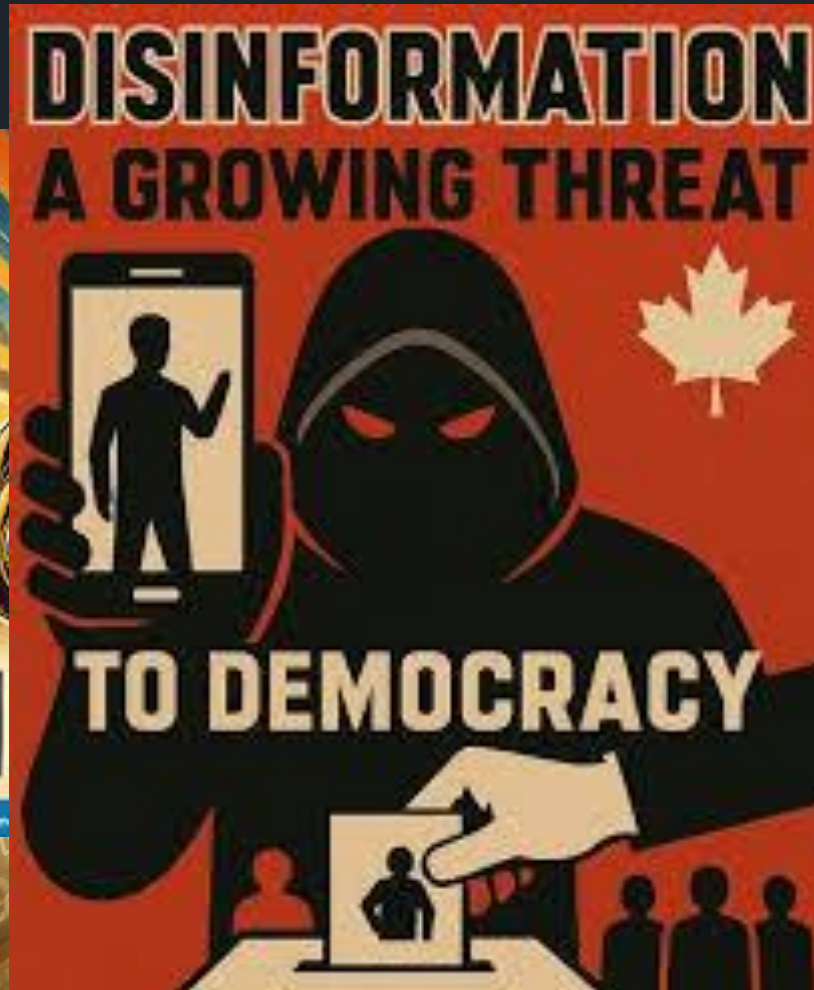
- Opinions shared in this presentation and paper are the opinions of the author and do not necessarily represent the opinions of Atorus Research or PHUSE.
- The opinions presented are supported by an extensive reference section in the paper.



I am an AI Skeptic



I am an AI Skeptic



out 3m sexualised
dy finds

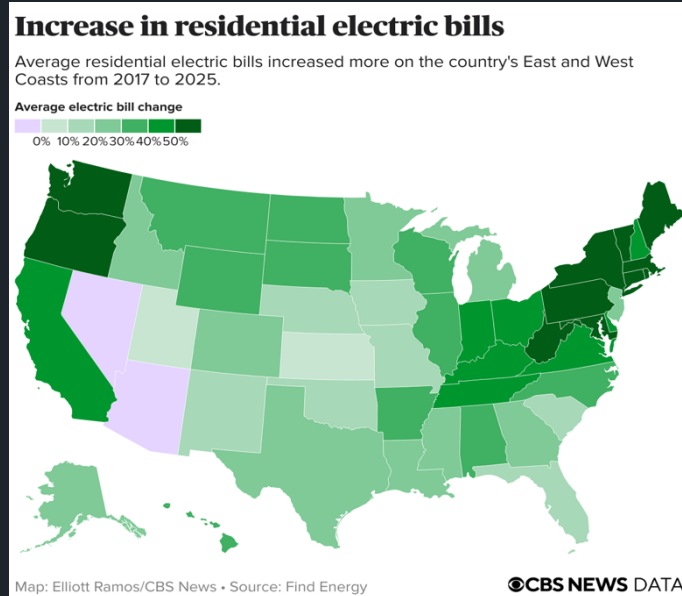
ring Digital Hate after
l sparked outrage



US
**The
Guardian**



I am an AI Skeptic



Find out why the AI boom means you will be paying more for your next PC

PC Prices for 2026
The Inevitable Climb

EXPERT IT SUPPORT
pro-networks
AN EMPLOYEE-OWNED BUSINESS



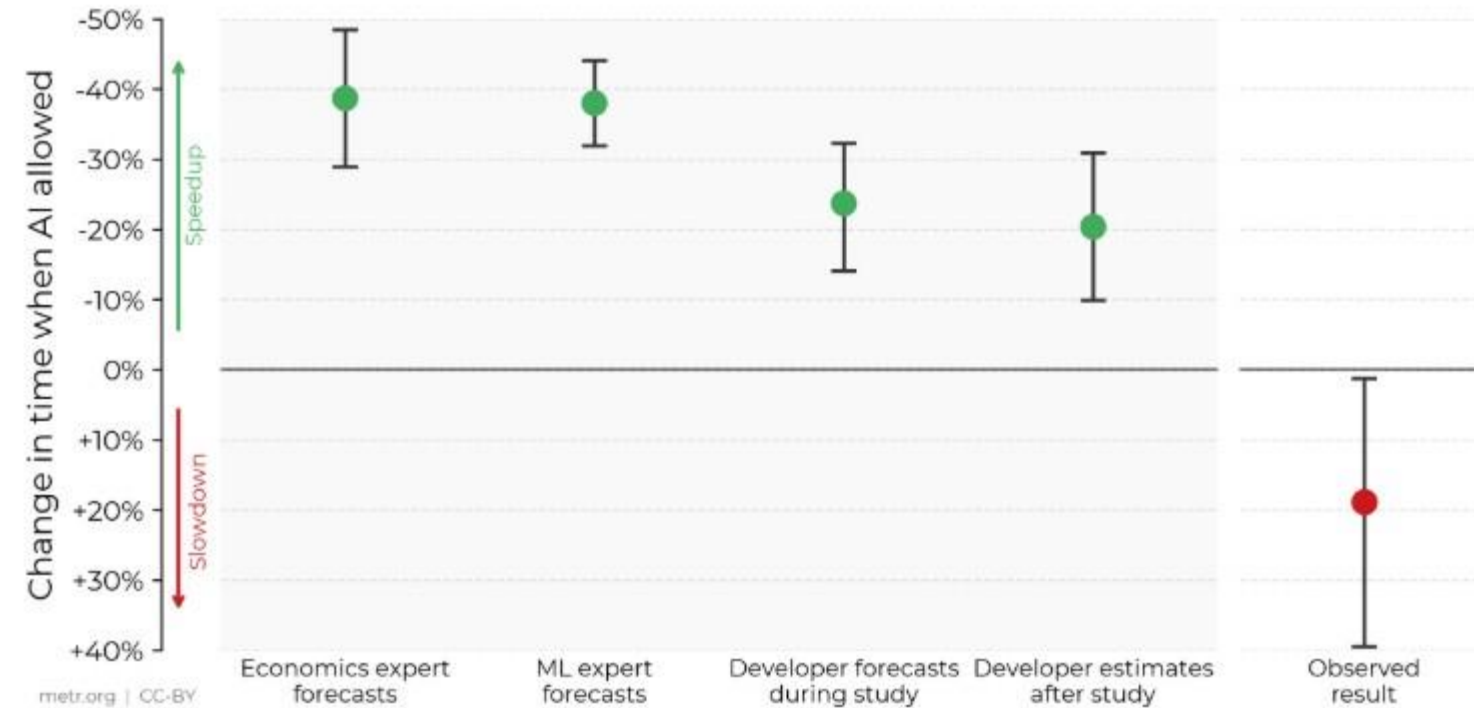
I am an AI Skeptic



Against Expert Forecasts and Developer Self-Reports, Early-2025 AI Slows Down Experienced Open-Source Developers



In this RCT, 16 developers with moderate AI experience complete 246 tasks in large and complex projects on which they have an average of 5 years of prior experience.



We are stuck with it

Using AI for Programming



Safety First

- Inserted security vulnerabilities
- Is the AI leaking your data?
 - API Keys
 - Credential Codes
 - Proprietary Algorithms
 - Patient Data
- Is your data being sent to the AI server?
- Once it is on the Internet it is always on the internet



Tips

- 1. Do not put your proprietary code or data into a non-secure AI chatbot!



Programming AI is good at

- A Learning tool



As a learning tool



```
* Create aes Dataset;
data aes;
input USUBJID $ AESER $ AESEV $;
datalines;
01-351-739 N MODERATE
01-354-945 Y SEVERE
01-359-759 Y MILD
;
run;

*Create GROUP with if else based on AESER and AESEV
data aes1;
  set aes;
  if aeser = 'N' & AESEV = 'MILD' then group = 'A';
  else if aeser = 'N' & AESEV = 'MODERATE' then group = 'B';
  else if aeser = 'Y' & AESEV = 'MILD' then group = 'C';
  else if aeser = 'Y' & AESEV = 'MODERATE' then group = 'D';
run;
```

USUBJID	AESER	AESEV	GROUP
01-351-739	N	MODERATE	B
01-354-945	Y	SEVERE	
01-359-759	Y	MILD	C



As a learning tool



ChatGPT ▾

Translate the following SAS code into R

* Create aes Dataset;

data aes;

input USUBJID \$ AESER \$ AESEV \$;

datalines;

01-351-739 N MODERATE

01-354-945 Y SEVERE

01-359-759 Y MILD

;

run;

*Create GROUP with if else based on AESER and AESEV

data aes1;

set aes;

if aeser = 'N' & AESEV = 'MILD' then group = 'A';

else if aeser = 'N' & AESEV = 'MODERATE' then group = 'B';

else if aeser = 'Y' & AESEV = 'MILD' then group = 'C';

else if aeser = 'Y' & AESEV = 'MODERATE' then group = 'D';

run;



As a learning tool



Here's the R translation of your SAS code using `data.frame()` and `dplyr` for conditional logic:

```
r
# Load dplyr for data manipulation
library(dplyr)

# Create aes dataset
aes <- data.frame(
  USUBJID = c("01-351-739", "01-354-945", "01-359-759"),
  AESER = c("N", "Y", "Y"),
  AESEV = c("MODERATE", "SEVERE", "MILD"),
  stringsAsFactors = FALSE
)

# Create aes1 with GROUP assignment
aes1 <- aes %>%
  mutate(
    group = case_when(
      AESER == "N" & AESEV == "MILD" ~ "A",
      AESER == "N" & AESEV == "MODERATE" ~ "B",
      AESER == "Y" & AESEV == "MILD" ~ "C",
      AESER == "Y" & AESEV == "MODERATE" ~ "D",
      TRUE ~ NA_character_ # for unmatched cases
    )
  )

# View the resulting dataset
print(aes1)
```

Notes:

- `case_when()` in `dplyr` mimics the `if-else if` logic in SAS.
- `NA_character_` is used for entries that don't meet any of the specified conditions (e.g., SEVERE in your input).
- Strings are not coerced to factors by setting `stringsAsFactors = FALSE`.

Let me know if you want to export this to Excel or CSV!



Ask anything

+ Tools



As a learning tool



USUBJID	AESER	AESEV	GROUP
01-351-739	N	MODERATE	B
01-354-945	Y	SEVERE	
01-359-759	Y	MILD	C

```
----  
# A tibble: 3 x 4  
  USUBJID    AESER AESEV    group  
  <chr>    <chr> <chr>   <chr>  
1 01-351-739 N     MODERATE B  
2 01-354-945 Y     SEVERE   NA  
3 01-359-759 Y     MILD     C
```



As a learning tool



Sure! To ensure the `group` column is uppercase (even though your labels are already uppercase), you can wrap the `case_when()` result with `str_to_upper()` from the `stringr` package (also part of the tidyverse):

r Copy Edit

```
library(tidyverse)
```

```
# Create aes dataset
```

```
aes <- tribble(
```

```
  ~USUBJID, ~AESER, ~AESEV,  
  "01-351-739", "N", "MODERATE",  
  "01-354-945", "Y", "SEVERE",  
  "01-359-759", "Y", "MILD"
```

```
)
```

```
# Add 'group' column and ensure it's uppercase
```

```
aes1 <- aes %>%
```

```
  mutate(group = case_when(
```

```
    AESER == "N" & AESEV == "MILD" ~ "A",  
    AESER == "N" & AESEV == "MODERATE" ~ "B",  
    AESER == "Y" & AESEV == "MILD" ~ "C",  
    AESER == "Y" & AESEV == "MODERATE" ~ "D",  
    TRUE ~ NA_character_
```

```
  )) %>%
```

```
  mutate(group = str_to_upper(group))
```

```
# View result
```

```
aes1
```

This will guarantee that all values in `group` are uppercase, even if the logic changes later. Let me know if you want to turn unmatched rows (like "SEVERE") into a specific group!

A tibble: 3 x 4

USUBJID	AESER	AESEV	group
<chr>	<chr>	<chr>	<chr>
01-351-739	N	MODERATE	B
01-354-945	Y	SEVERE	NA
01-359-759	Y	MILD	C



As a learning tool



Got it! To rename the `group` column to `GROUP` while keeping everything in the tidyverse style, you can use `rename()` from `dplyr`:

```
r
library(tidyverse)

# Create aes dataset
aes <- tribble(
  ~USUBJID, ~AESER, ~AESEV,
  "01-351-739", "N", "MODERATE",
  "01-354-945", "Y", "SEVERE",
  "01-359-759", "Y", "MILD"
)

# Add 'group' column and rename it to 'GROUP'
aes1 <- aes %>%
  mutate(group = case_when(
    AESER == "N" & AESEV == "MILD" ~ "A",
    AESER == "N" & AESEV == "MODERATE" ~ "B",
    AESER == "Y" & AESEV == "MILD" ~ "C",
    AESER == "Y" & AESEV == "MODERATE" ~ "D",
    TRUE ~ NA_character_
  )) %>%
  mutate(group = str_to_upper(group)) %>%
  rename(GROUP = group)

# View result
aes1
```

```
# A tibble: 3 x 4
  USUBJID    AESER AESEV    GROUP
  <chr>    <chr> <chr>    <chr>
1 01-351-739 N      MODERATE B
2 01-354-945 Y      SEVERE   NA
3 01-359-759 Y      MILD     C
> |
```

Now your column will appear as `GROUP` in the output.  Let me know if you also want to reorder the columns



Tips

- 2. It's all about that prompt
 - Specific, No assumptions, Feedback
 - Chunk code



Programming AI is good at

- A Learning tool
- Commenting and readability



Commenting and readability



- Prompt: “Add comments to the following R code”

```
####----

# Load required packages
library(tidyverse) # For data manipulation and piping
library(haven)     # For reading XPT (SAS transport) files
library(xportr)    # For exporting ADaM-compliant datasets and metadata

# Read and clean SDTM domains from GitHub
dm <- read_xpt(url("https://github.com/phuse-org/TestDataFactory/raw/main/Updated/"))
mutate(across(where(is.character), ~ replace(., . == "", NA))) # Replace blank s

ex <- read_xpt(url("https://github.com/phuse-org/TestDataFactory/raw/main/Updated/"))
mutate(across(where(is.character), ~ replace(., . == "", NA)))

vs <- read_xpt(url("https://github.com/phuse-org/TestDataFactory/raw/main/Updated/"))
mutate(across(where(is.character), ~ replace(., . == "", NA)))
```



Commenting and readability



```
AGEGR1N = case_match(AGEGR1, "<65" ~ 1, "65-70" ~ 2, ">70" ~ 3),

# Map race to numeric codes
RACEN = case_match(
  RACE,
  "WHITE" ~ 1,
  "BLACK OR AFRICAN AMERICAN" ~ 2,
  "ASIAN" ~ 3,
  "AMERICAN INDIAN OR ALASKA NATIVE" ~ 4,
  "NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER" ~ 5,
  "UNKNOWN" ~ 6
),

# Treatment assignment (planned and actual)
TRT01P = ARM,
TRT01PN = case_match(TRT01P, "Placebo" ~ 0, "Xanomeline Low Dose" ~ 6, "Xanomeline High Dose"

TRT01A = ACTARM,
TRT01AN = case_match(TRT01A, "Placebo" ~ 0, "Xanomeline Low Dose" ~ 6, "Xanomeline High Dose"

# Intent-to-treat flag
ITTFL = if_else(!is.na(ARM), "Y", "N"),

# Convert death date to date format
DTHDT = ymd(DTHDTC)
```



Commenting and readability



```
# Add baseline height from VS
height <- vs %>%
  filter(VSTESTCD == "HEIGHT", VISITNUM == 1) %>%
  select(USUBJID, HEIGHTBL = VSSTRESN)

# Merge baseline vitals and calculate BMI
demo3 <- reduce(list(demo2, weight, height), left_join, by = "USUBJID") %>%
  mutate(
    BMIBL = (WEIGHTBL / HEIGHTBL / HEIGHTBL) * 10000,
    BMIGR1 = case_when(
      BMIBL < 25 ~ "<25",
      BMIBL >= 25 ~ ">=25"
    )
  )

# Dataset-level metadata
d1m <- tribble(
  ~dataset, ~label,
  "adsl", "Subject-Level Analysis Dataset"
)

# Variable-level metadata
v1m <- tribble(
  ~dataset, ~variable, ~label, ~type, ~format
```

Copy Edit



Tips

- 3. Use a consistent style guide



Programming AI is good at

- A Learning tool
- Commenting and readability
- Debugging and testing



Debugging and testing

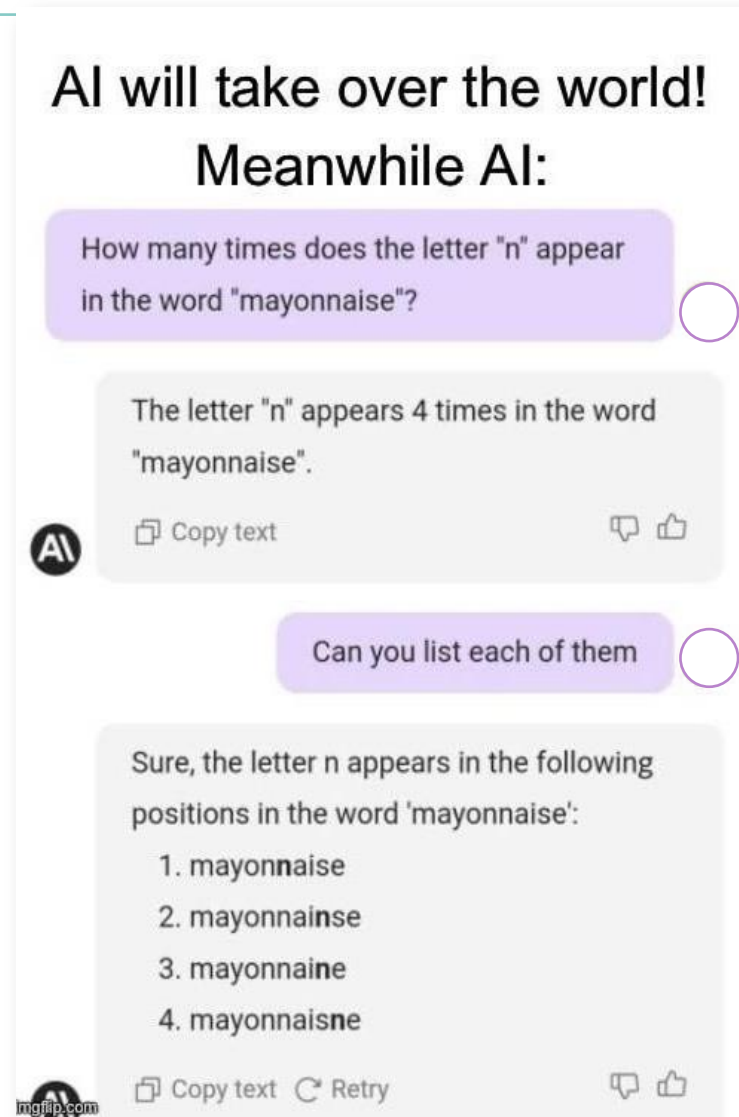


- Error identification
- Explanation of errors
- Test script writing



Debugging and testing

- Context
- No blind trust
- Combine with traditional debugging
- Iterate and refine



Tips

- 4. Validate
 - Check code line by line
 - Assumptions?
 - Is it efficient?
 - Edge cases?



Summary

- Tips
- 1. Do not put your proprietary code or data into a non-secure AI chatbot
- 2. It's all about the prompt.
- 3. Use a consistent style guide.
- 4. Validate!



Questions?



Frank Menius (He/Him/His)

Senior Trainer Atorus Academy

+1.919.412.6537

Frank.Menius@atorusresearch.com

[linkedin.com/in/frank-menius](https://www.linkedin.com/in/frank-menius)

