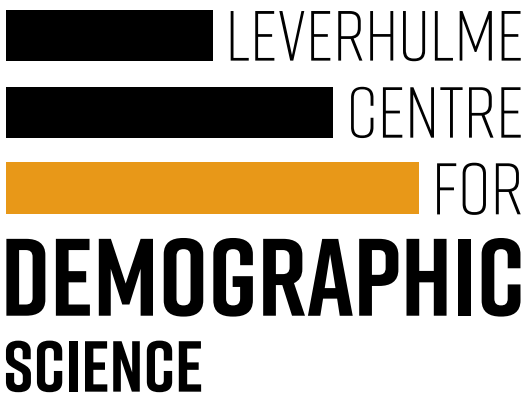
On the Responsible use of Pseudo-Random Number Generators

Charlie Rahal

LCDS and Nuffield College, University of Oxford

Max Planck Institute for Demographic Research, November 26th 2025



What is a Pseudo Random Number Generator?

"A pseudo-random number is a value produced by a deterministic algorithm (a PRNG) that is intended to mimic the statistical properties of true randomness. Given an internal state (often initialized from a seed), a PRNG computes a sequence of numbers that appear unpredictable for many applications but are fully reproducible if the algorithm and state are known. PRNGs are essential for simulations, sampling, any many other types of algorithms."

- GPT-5 mini

PRNGs are just one type of RNG!

	'RN'	'RN*' and 'Hardware'	'RN*' and 'Quantum'	'RN*' and 'Pseudo'	'RN*' and 'Quasi'
Health Sciences	0.406	0.000646	0.000407	0.00109	0.004201
Life Sciences	0.292	0.000516	0.001336	0.00179	0.001202
Physical Sciences	0.225	0.005430	0.007717	0.00726	0.003884
Social Sciences	0.092	0.000216	0.000242	0.00055	0.002420

A cursory scientometric analysis of Randon Numbers (RN*). Data comes from OpenAlex API. All numbers are % of the entire scientific record.

- More 'PRNG papers' in CS (4222) than others combined.
- %PRNG papers since 1970: 0.0008% $\longrightarrow$ 0.0048%.

PRNGs occur <u>everywhere</u>.

Computational Sciences:

- Numerical Integration
- Cryptography
- Genetic Algorithms
- Signal Processing
- Deep Learning
- Weather Forecasting
- Procedural Content
- ...

'Applied' (Health/Social) Sciences:

- RCTs/Survey Sampling
- Bootstrapping
- Epidemiological Modeling
- Econometric Estimation
- Data Imputation
- Topic Modeling
- ABMs
- ...

MT64: ubiquitous PRNG implementation (R, Stata, Python, ...)

prgn_stream_demo.py

```python
import random

floats = [random.random() for _ in range(5)]
print(floats)
```

```
[0.36381, 0.20691, 0.50947, 0.33315, 0.87402]
```

prgn_stream_demo.py

```python
import random

floats = [random.random() for _ in range(5)]
print(floats)
```

```
[0.4215, 0.01561, 0.9984, 0.7674, 0.274]
```

prgn_stream_demo.py

```python
import random

random.seed(123456789) # deterministic

floats = [random.random() for _ in range(5)]
print(floats)
```

```
[0.64140, 0.542189, 0.99317, 0.84325, 0.81173]
```

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?
  - Maybe you prefer `set.seed(42)`?

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?
  - Maybe you prefer `set.seed(42)`?
  - Or `numpy.random.seed(123)`?

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?
  - Maybe you prefer `set.seed(42)`?
  - Or `numpy.random.seed(123)`?
- Why did you do this?

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?
  - Maybe you prefer `set.seed(42)`?
  - Or `numpy.random.seed(123)`?
- Why did you do this?
  - Maybe to eliminate variation in algorithms with PRNGs?

So, why do we care about PRNGs?

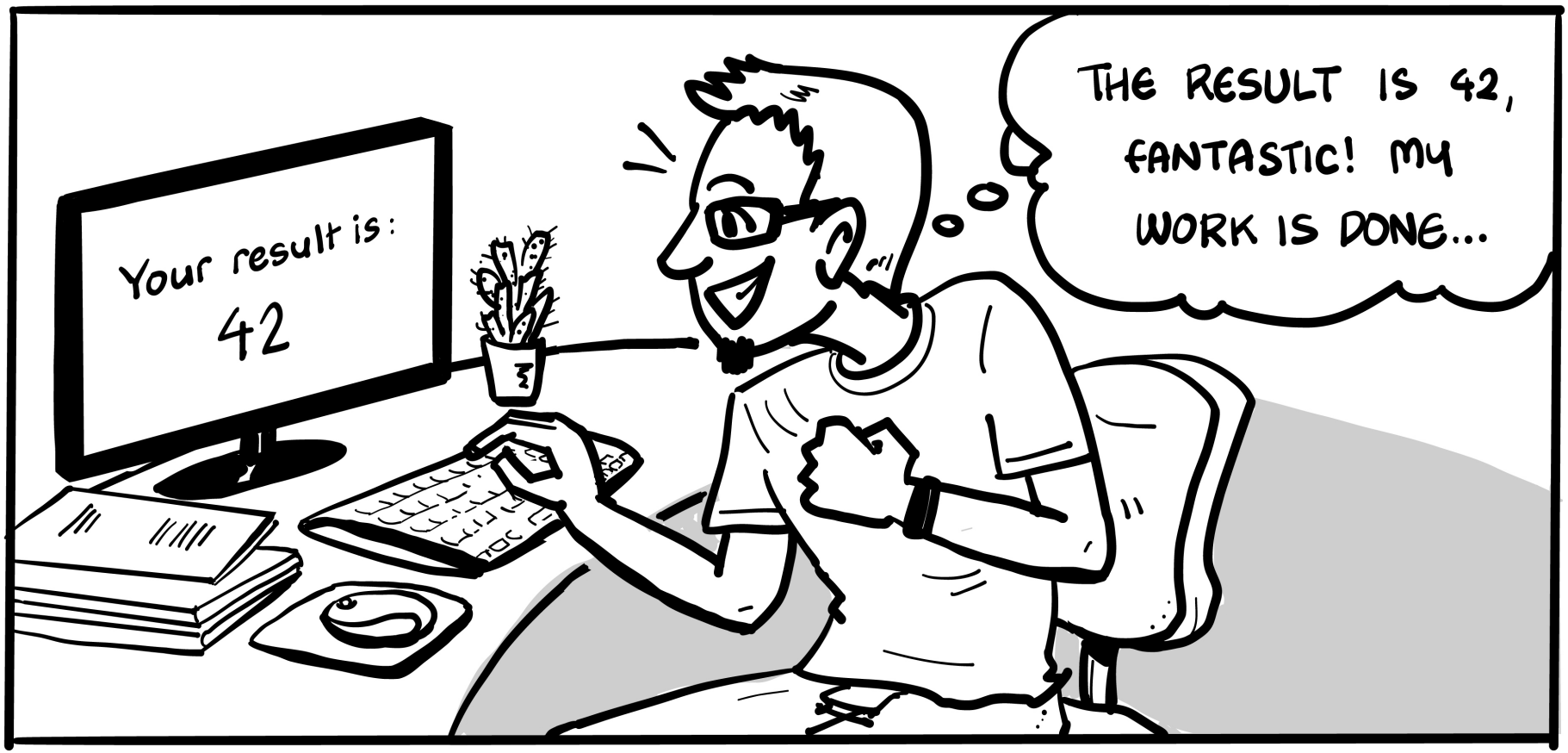
An invisible source of uncertainty in the scientific record: PRNGs!

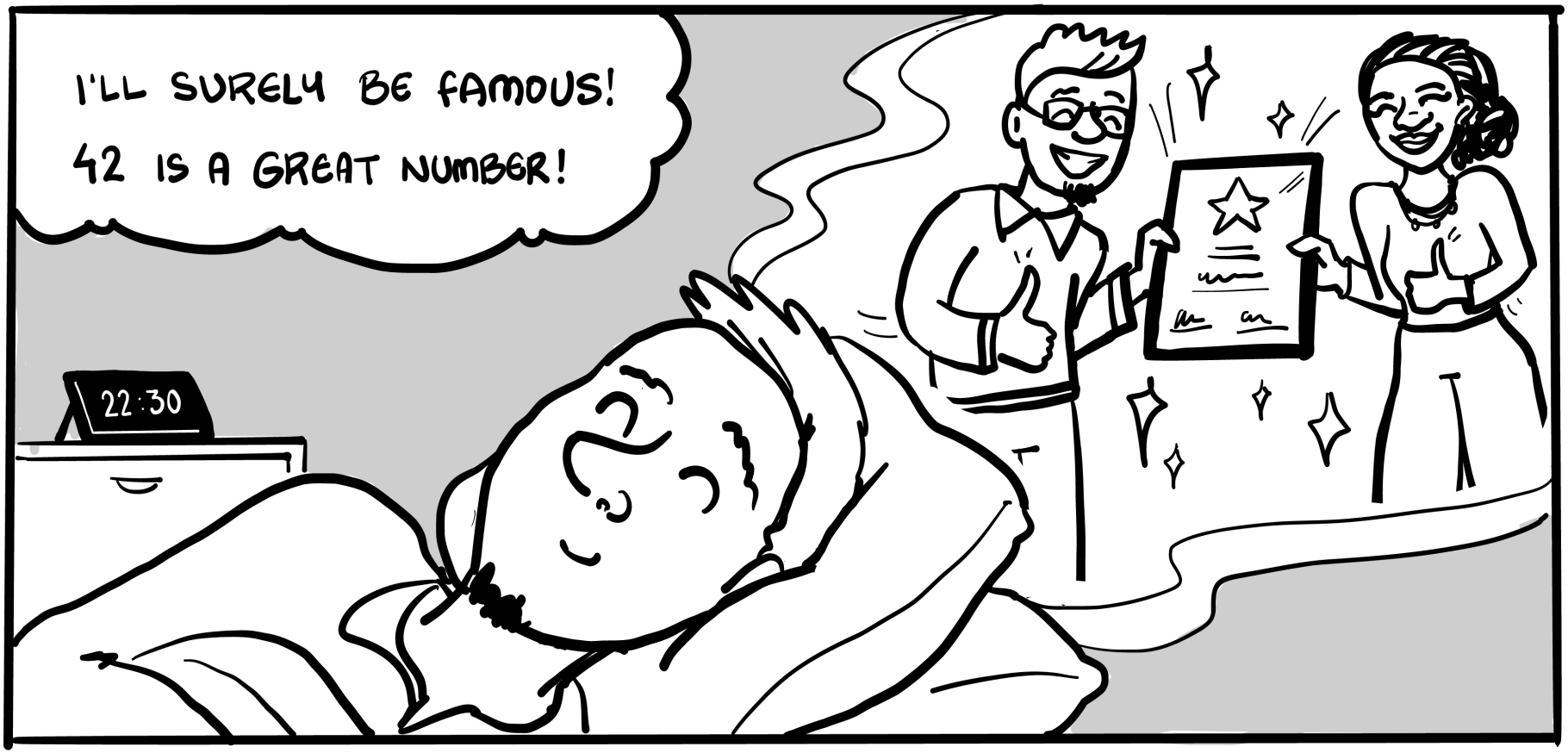
- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?
  - Maybe you prefer `set.seed(42)`?
  - Or `numpy.random.seed(123)`?
- Why did you do this?
  - Maybe to eliminate variation in algorithms with PRNGs?
  - This seems to be the current 'best practice' advice.

So, why do we care about PRNGs?

An invisible source of uncertainty in the scientific record: PRNGs!

- **Q**: Has anyone ran a program twice, with different results?
- **Q**: Has anyone here ever set a 'seed'? Which seed?
  - Maybe you prefer `set.seed(42)`?
  - Or `numpy.random.seed(123)`?
- Why did you do this?
  - Maybe to eliminate variation in algorithms with PRNGs?
  - This seems to be the current 'best practice' advice.
- This is very well intentioned: it allows reproducibility. Yay!


Your result is:
42
THE RESULT IS 42, FANTASTIC! MY WORK IS DONE...

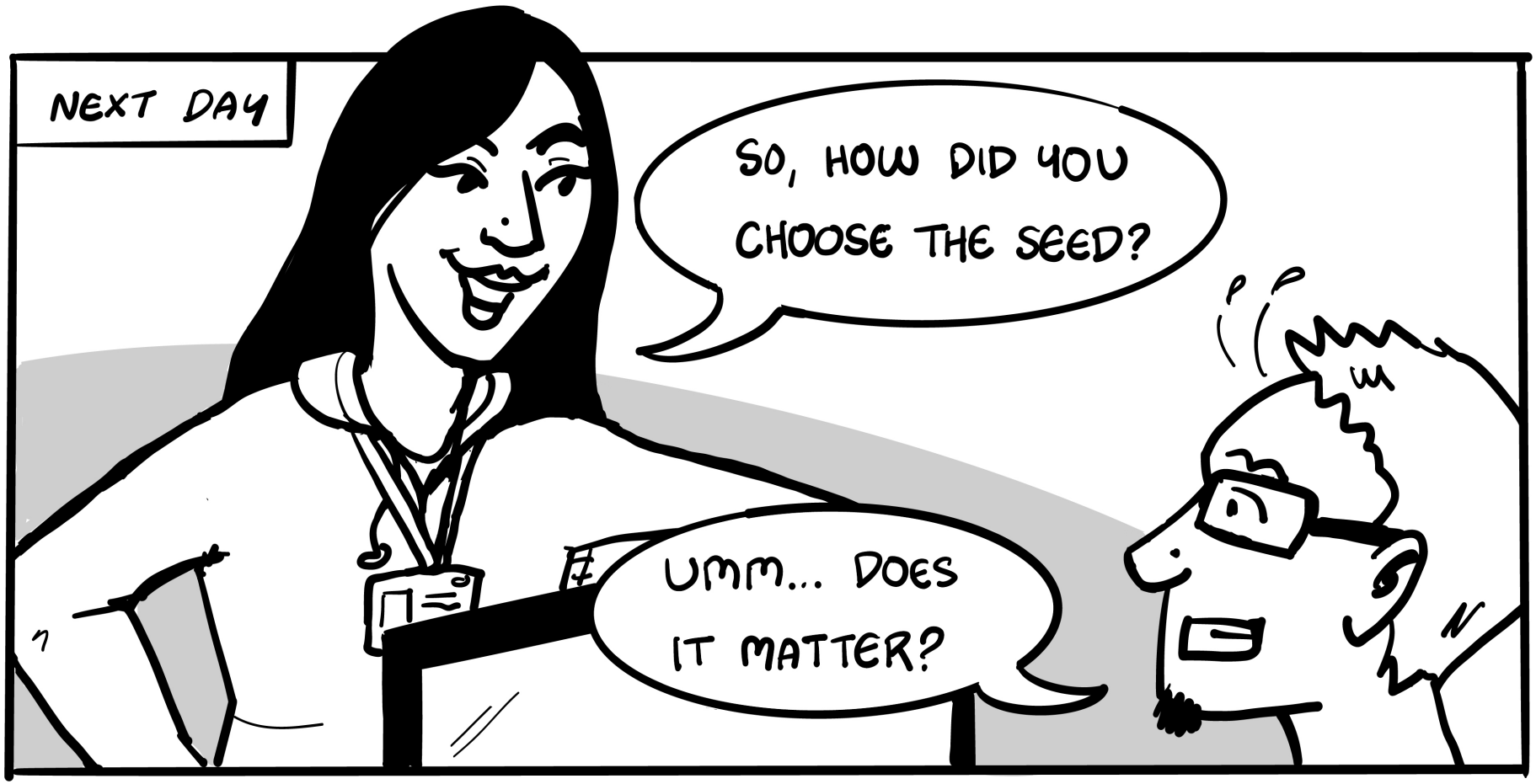

I'LL SURELY BE FAMOUS!
42 IS A GREAT NUMBER!
22:30


NEXT DAY
Your result is:
97
HUH?!


BEST PRACTICE: SET A SEED FOR REPRODUCIBILITY!


SHE DIDN'T SAY
WHAT SEED TO SET —
LET'S TRY '123'...
set-seed(123)
Your result is:
62

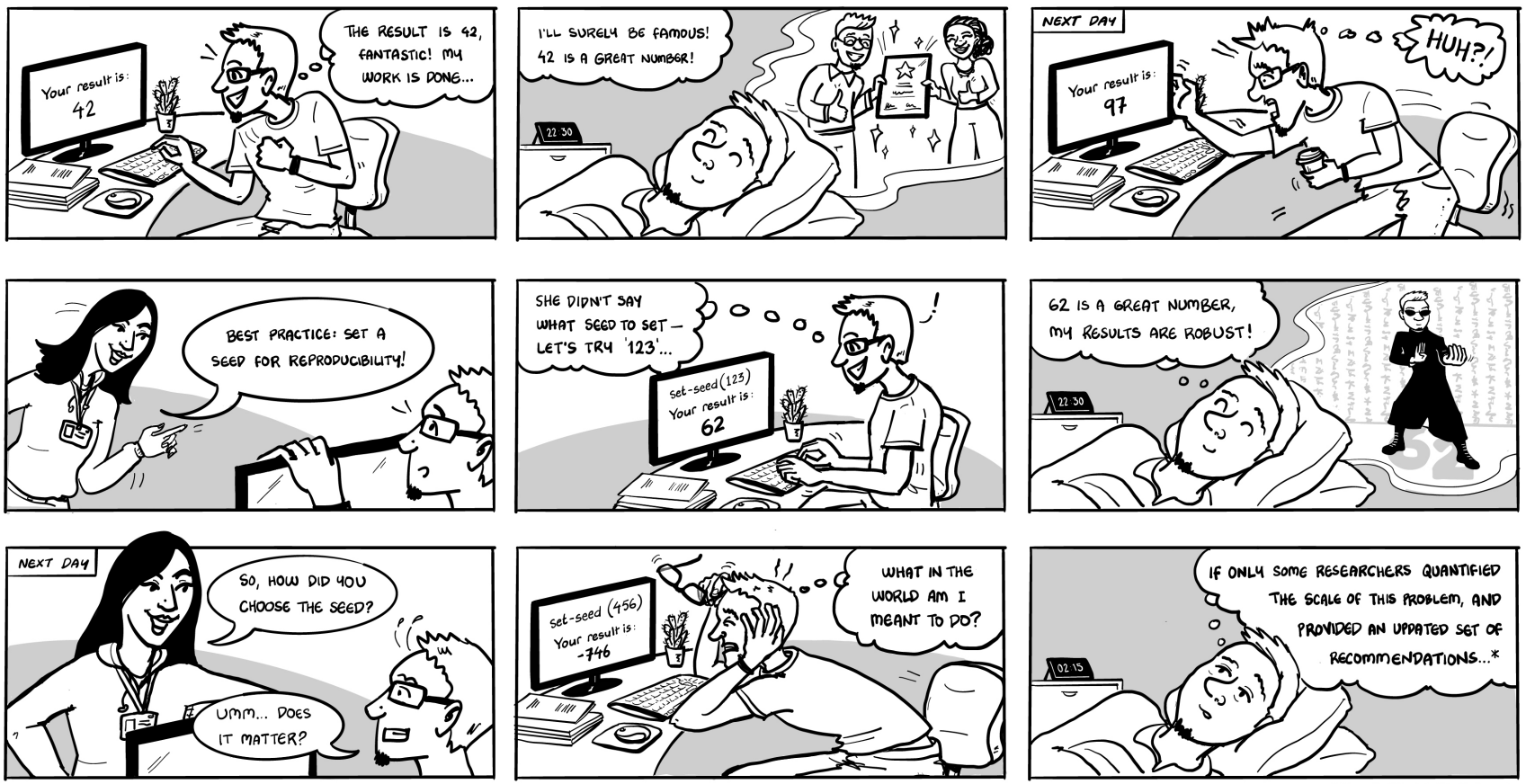

62 IS A GREAT NUMBER,
MY RESULTS ARE ROBUST!
22:30
62


NEXT DAY
SO, HOW DID YOU CHOOSE THE SEED?
UMM... DOES IT MATTER?


set-seed (456)
Your result is:
-746
WHAT IN THE WORLD AM I MEANT TO DO?


IF ONLY SOME RESEARCHERS QUANTIFIED THE SCALE OF THIS PROBLEM, AND PROVIDED AN UPDATED SET OF RECOMMENDATIONS...*
02:15

THE SEED OF DOUBT: A RANDOM TALE OF REPRODUCIBILITY



*See 'Rahal, C., Yan, J. And Verhagen, M. (2025), 'On the Responsible Use of Pseudo Random Number Generators', arXiv, pp. 1–26'

DRAWN BY LAURA SORVALA

"If any research design is susceptible to this kind of variation, there is a problem".

– Anonymous Colleague, Oxford (2019).

The General Premise

- **Problem Statement**: By setting **one** seed in applied algorithmic pipelines, we ignore the variation of our estimand as a function of how PRNGs were instantiated, This is computationally un-intensive, but scientifically dangerous.

The General Premise

- **Problem Statement**: By setting **one** seed in applied algorithmic pipelines, we ignore the variation of our estimand as a function of how PRNGs were instantiated, This is computationally un-intensive, but scientifically dangerous.

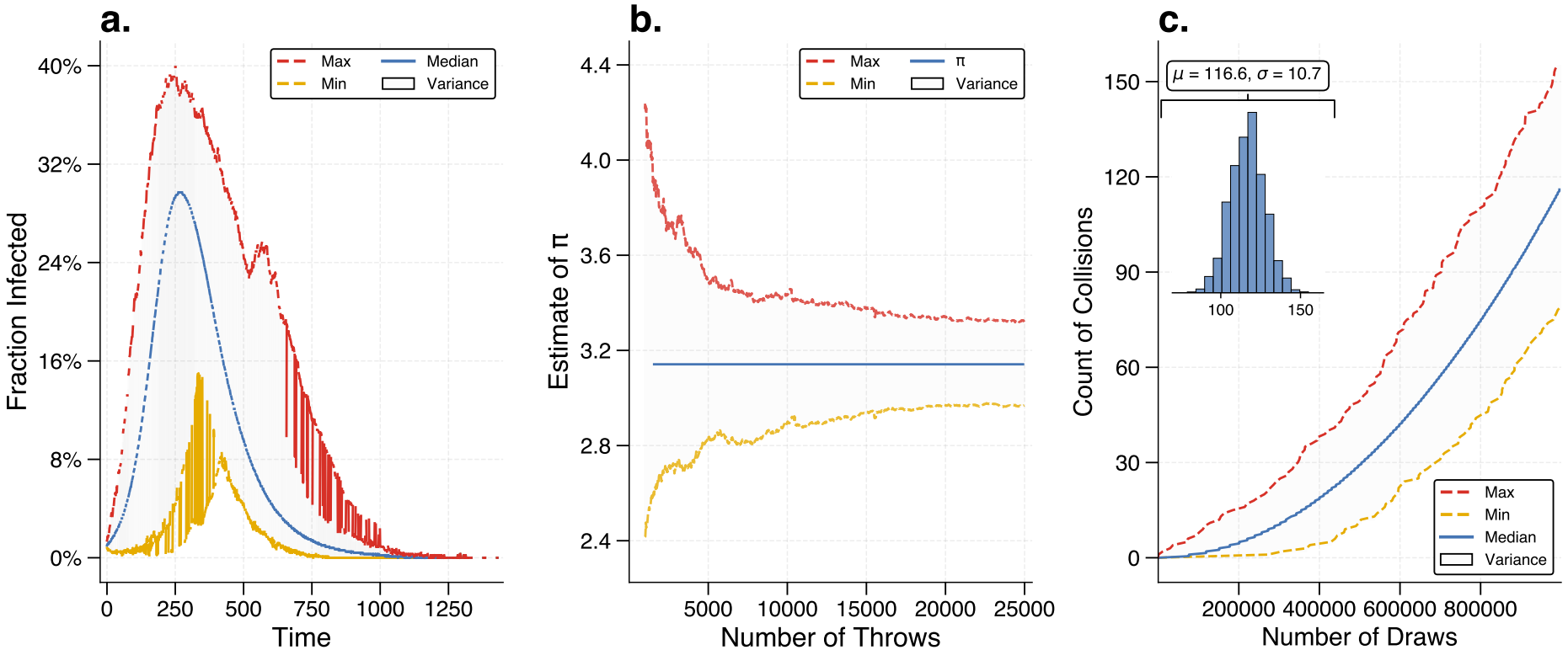
- **Solution**: Visualize the outcome space of a **large number** (10k? 100k?) of seeds simultaneously. This is computationally intensive, but scientifically responsible.

The General Premise

- **Problem Statement**: By setting **one** seed in applied algorithmic pipelines, we ignore the variation of our estimand as a function of how PRNGs were instantiated, This is computationally un-intensive, but scientifically dangerous.
- **Solution**: Visualize the outcome space of a **large number** (10k? 100k?) of seeds simultaneously. This is computationally intensive, but scientifically responsible.
- **Replication**: Similar to specification curves and p-hacking, we can retrospectively consider whether an original result is in the tail/IQR of the distribution of possible outcome space.

What do we do?

- Through **many** replications and simulations (20?), we:
  - Show some simple simulated examples.
  - Show seed variabiilty in data preprocessing
  - Show seed variability in modelling
  - Show seed variability in prediction
  - Show the potential compounding of these effects
  - Show implications for fairness
  - Show how this affects LLMs, too.
- Then provide guidance to hopefully improve scientific practice.
- All code and data is open source and available on GitHub.


a.
Max
Min
Median
Variance
40%
32%
24%
16%
8%
0%
Fraction Infected
0
250
500
750
1000
1250
Time
b.
Max
Min
π
Variance
4.4
4.0
3.6
3.2
2.8
2.4
Estimate of π
5000
10000
15000
20000
25000
Number of Throws
c.
μ = 116.6, σ = 10.7
100
150
150
120
90
60
30
0
Count of Collisions
200000
400000
600000
800000
Number of Draws
Max
Min
Median
Variance

What's going on here?

- These examples sample from `random` or `np.random()` directly.
  - No model fitting, no data processing, just raw sampling.
    1. Simple draws from a SIR model.
    2. A replication of Buffon's infamous needle problem.
    3. Replicating some recent work on random number collisions.

What's going on here?

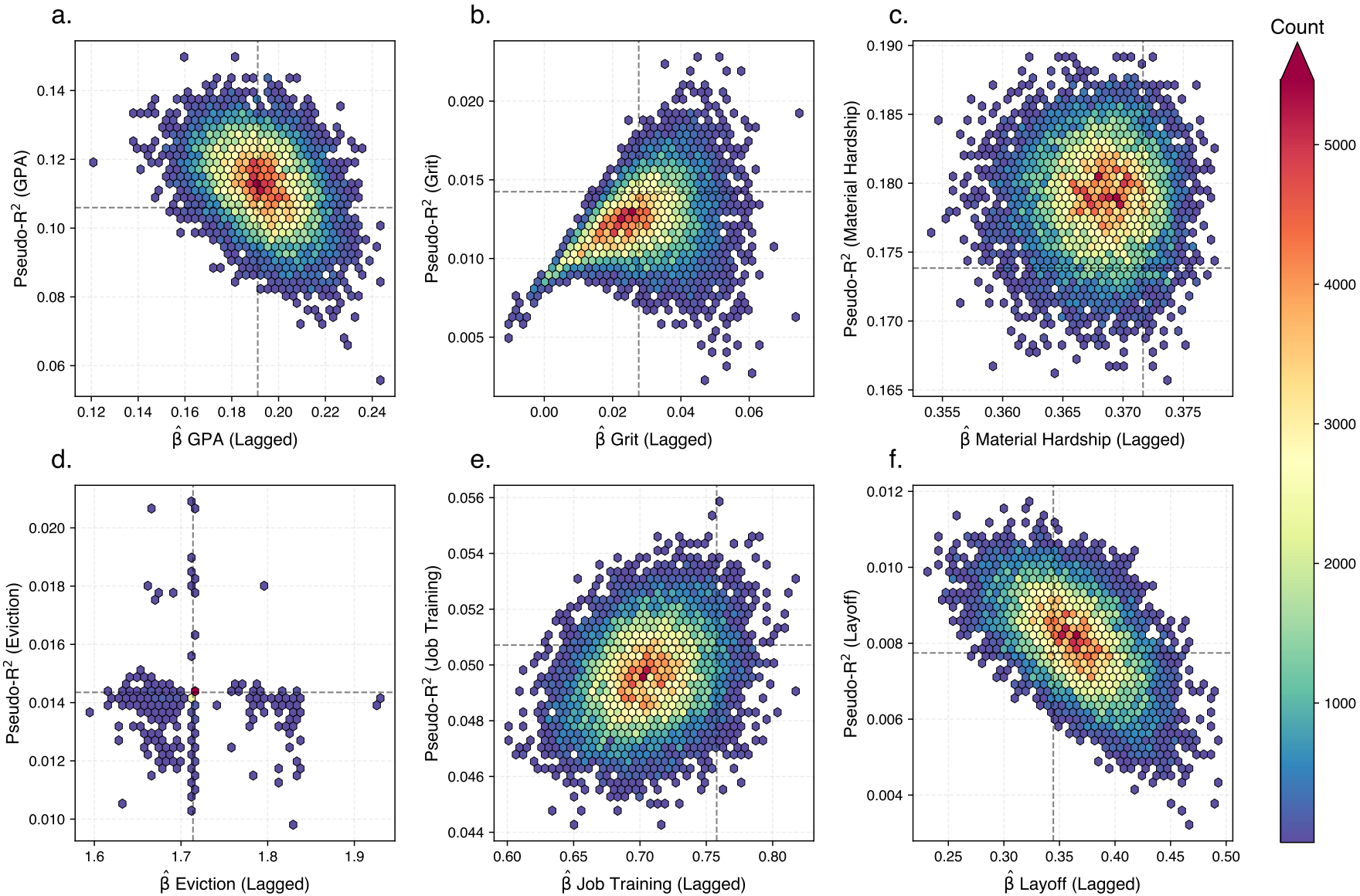
- These examples sample from `random` or `np.random()` directly.
  - No model fitting, no data processing, just raw sampling.
    1. Simple draws from a SIR model.
    2. A replication of Buffon's infamous needle problem.
    3. Replicating some recent work on random number collisions.
- Even in these simple cases, seed variability is **non-trivial**.

What's going on here?

- These examples sample from `random` or `np.random()` directly.
  - No model fitting, no data processing, just raw sampling.
    1. Simple draws from a SIR model.
    2. A replication of Buffon's infamous needle problem.
    3. Replicating some recent work on random number collisions.
- Even in these simple cases, seed variability is **non-trivial**.
- What are the implications for more complex pipelines?

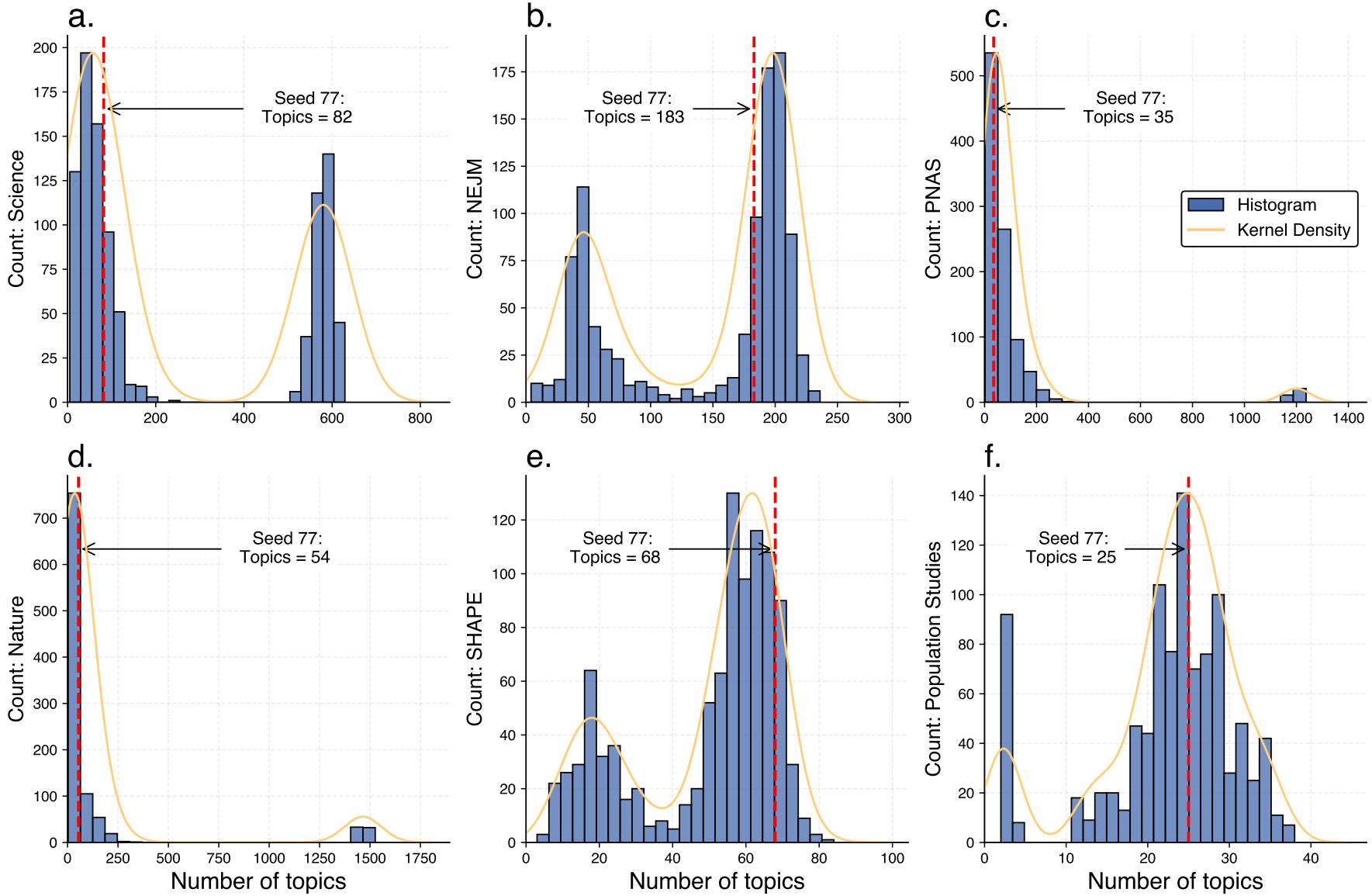
More Complex Replications

- Lets move into more complex and direct replications.
  - Major focus on the quantitative/computational social sciences.
    - Machine/Deep Learning
    - Sociology
    - Digital Humanities
    - Criminology
    - Medicine
    - Computer Vision
    - Econometrics
    - Time Series Analysis
    - Urban Segregation
    - LLMs
- Both recent and classical papers.
  - For each, we'll continue to give intuition as to what's going on.


a.
Pseudo-R² (GPA)
β̂ GPA (Lagged)
b.
Pseudo-R² (Grit)
β̂ Grit (Lagged)
c.
Pseudo-R² (Material Hardship)
β̂ Material Hardship (Lagged)
d.
Pseudo-R² (Eviction)
β̂ Eviction (Lagged)
e.
Pseudo-R² (Job Training)
β̂ Job Training (Lagged)
f.
Pseudo-R² (Layoff)
β̂ Layoff (Lagged)
Count

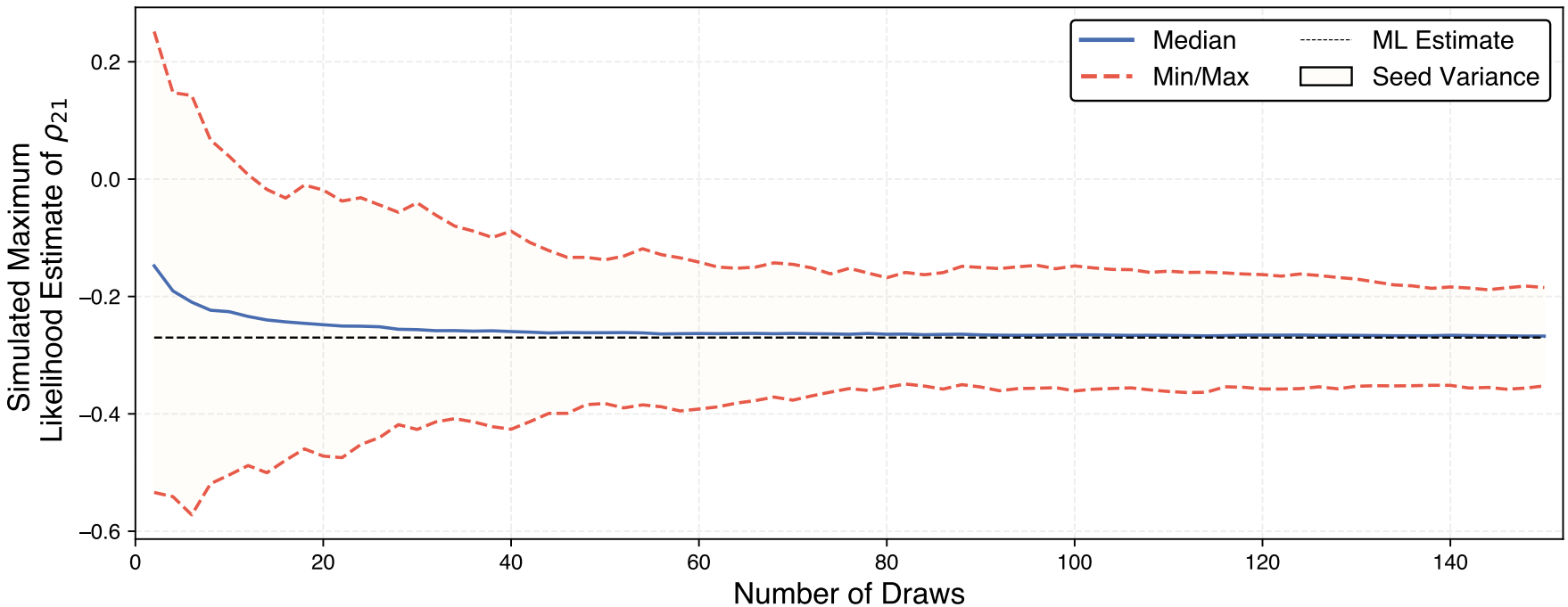
What's going on here?

- Amelia fills in missing data by drawing values from distributions.
- These distributions are based on observed data patterns.
- The imputed values reflect the uncertainty in the missing data.
- This is a form of Multiple Imputation.
- This results in entirely different models being built.
- This has implications for both inference and prediction.


a.
Seed 77:
Topics = 82
Count: Science
b.
Seed 77:
Topics = 183
Count: NEJM
c.
Seed 77:
Topics = 35
Count: PNAS
Histogram
Kernel Density
d.
Seed 77:
Topics = 54
Count: Nature
Number of topics
e.
Seed 77:
Topics = 68
Count: SHAPE
Number of topics
f.
Seed 77:
Topics = 25
Count: Population Studies
Number of topics

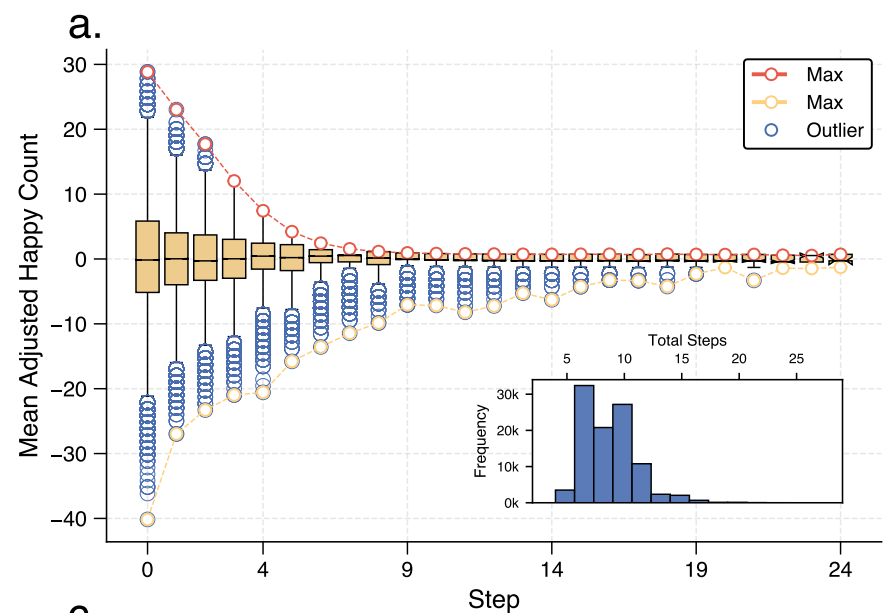
What's going on here?

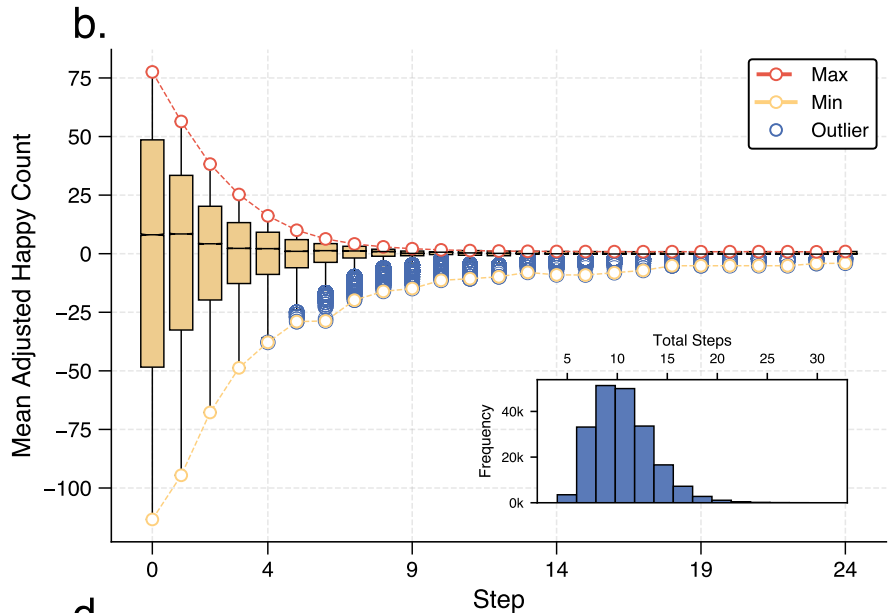
- BERTopic has many process: one particularly volatile to seeds.
- It commonly involves dimension reduction via UMAP.
  - **UMAP**: causes different low-dimensional embeddings.
- Note: CUML implementation of UMAP gives variable outputs for spectral initialisation *even when setting the seed*.
- If using HDBScan for clustering, no seed variability.
- In general here, the seed variance looks **enormous**.

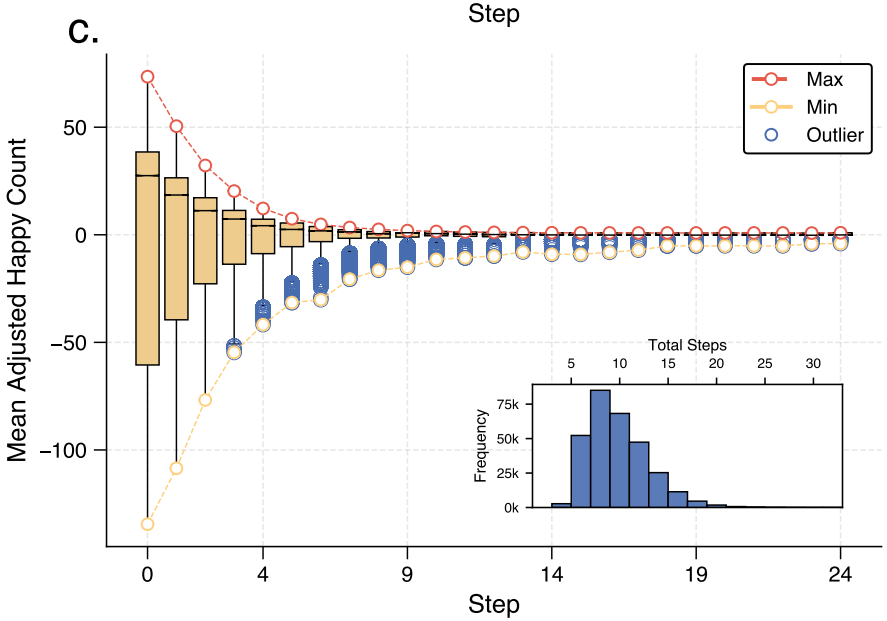

Median
ML Estimate
Min/Max
Seed Variance
Simulated Maximum Likelihood Estimate of ρ_{21}
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100
120
140
Number of Draws

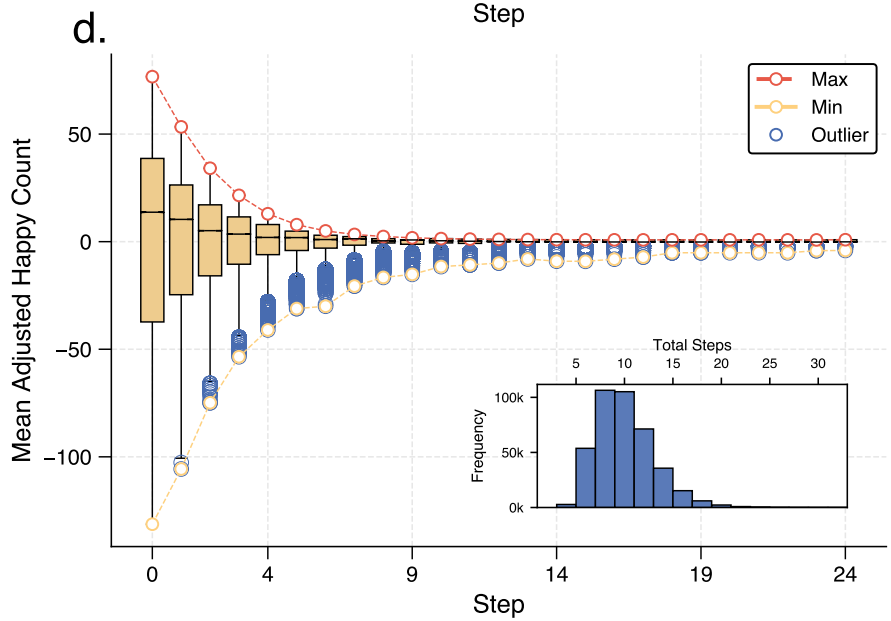
What's going on here?

- The `mvprobit` estimator requires evaluating M integrals jointly.
- This can be rewritten into a multivariate normal distributions.
  - Then, a sequence of normal distributions with certain properties.
- `mvprobit` samples from this to get at the underlying likelihood.
- Rewrite each individual normal to its CDF, which allows you to sample by just entering a uniform value between [0,1].
  - This random value is seed-dependent.
- Where CDF is steep on a small interval, seeds matter a lot.


a.
Mean Adjusted Happy Count
Step
Max
Max
Outlier
Total Steps
Frequency

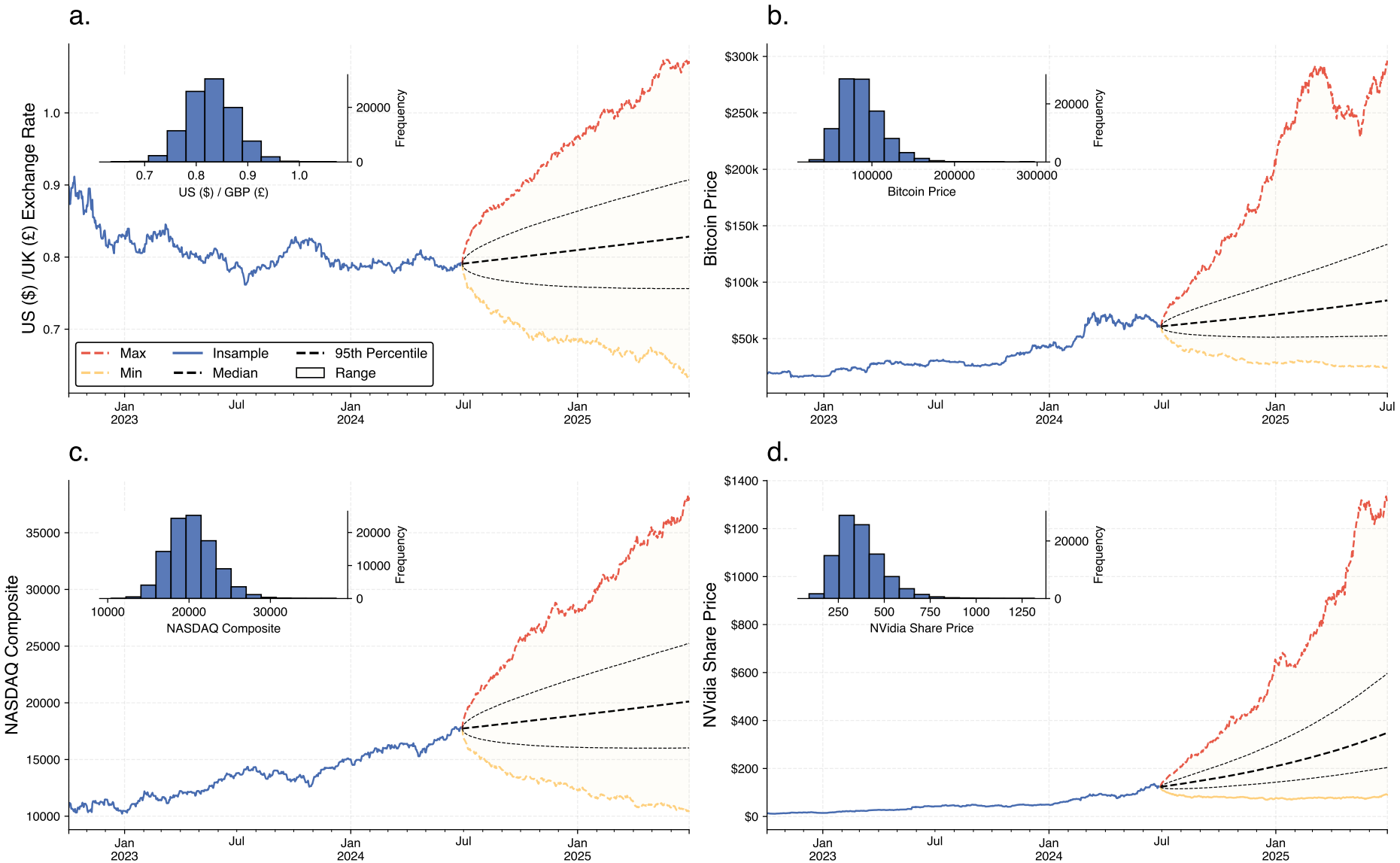

b.
Mean Adjusted Happy Count
Step
Max
Min
Outlier
Total Steps
Frequency


c.
Mean Adjusted Happy Count
Step
Max
Min
Outlier
Total Steps
Frequency


d.
Mean Adjusted Happy Count
Step
Max
Min
Outlier
Total Steps
Frequency

What's going on here?

- There are multiple (!) sources of PRNG in the Schelling Model:
  1. The initialisation of the agents on the grid.
  2. The shuffling of unhappy agents and empty cells.
- Agents might start more/less segregated from the beginning.
  - This affects the speed of total segregation over time.
- Even if the model reaches similar segregation, configuration of agents and trajectories will differ.
- As shown in the previous slide, parameterisation matters (a lot).


a.
US ($) /UK (£) Exchange Rate
US ($) / GBP (£)
Frequency
Max
Min
Insample
Median
95th Percentile
Range
Jan 2023
Jul
Jan 2024
Jul
Jan 2025
b.
Bitcoin Price
Bitcoin Price
c.
NASDAQ Composite
NASDAQ Composite
d.
NVidia Share Price
NVidia Share Price

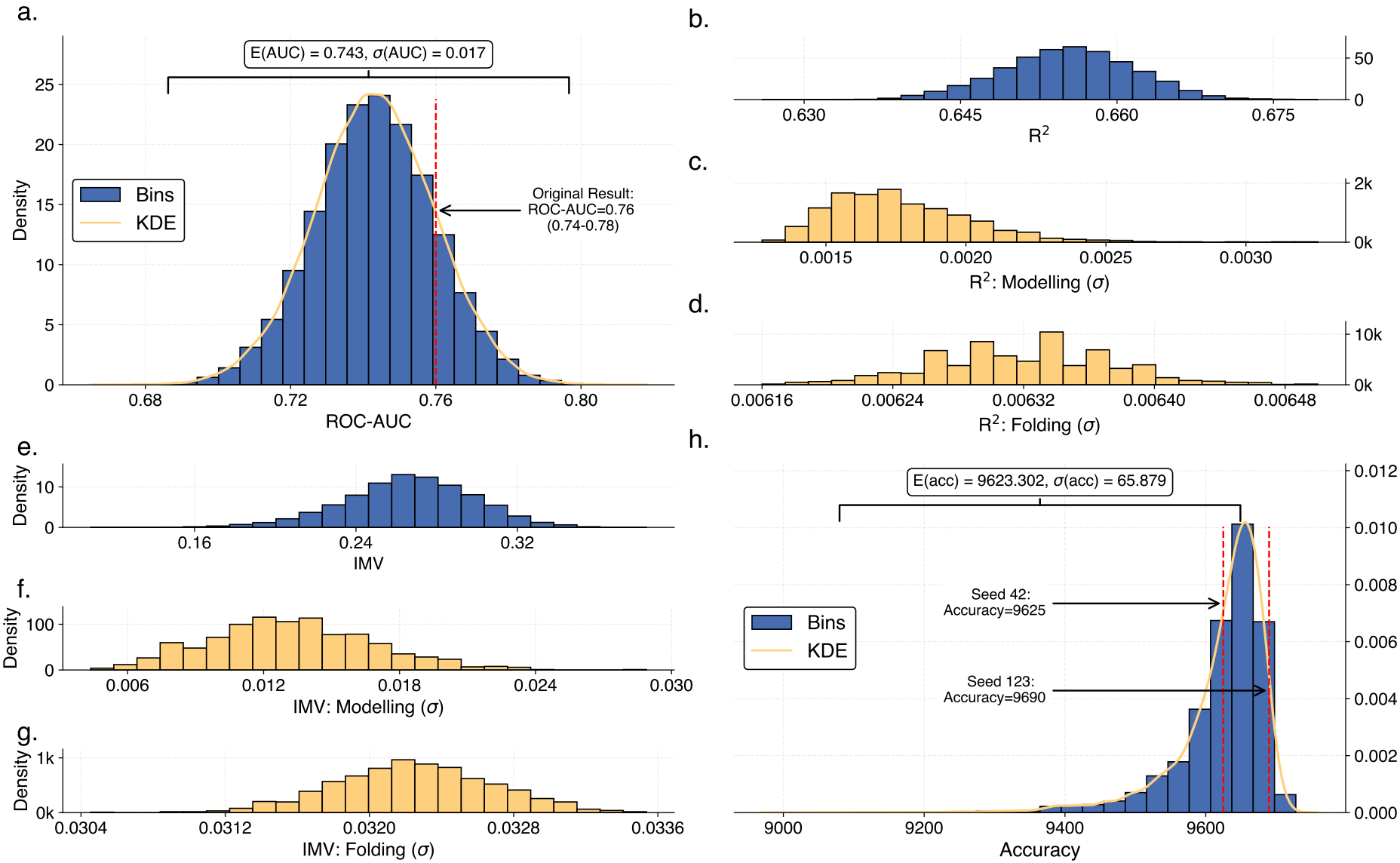
What's going on here?

- Seeds modify sequences of PRNG draws, affecting y_t.
- Moments of previous changes $(\mu, \sigma/2)$ control magnitude.
- Draws of $\mathcal{N}(\mu, \sigma/2)$ impact asset price trajectory.
- The seed affects long-term trends by influencing both direction and size of early random steps.

This can be formally written as:

$$y_t = y_{t-1} + \epsilon_t \quad \text{where} \quad \epsilon_t \sim \mathcal{N}(\mu, \sigma/2) \tag{1}$$

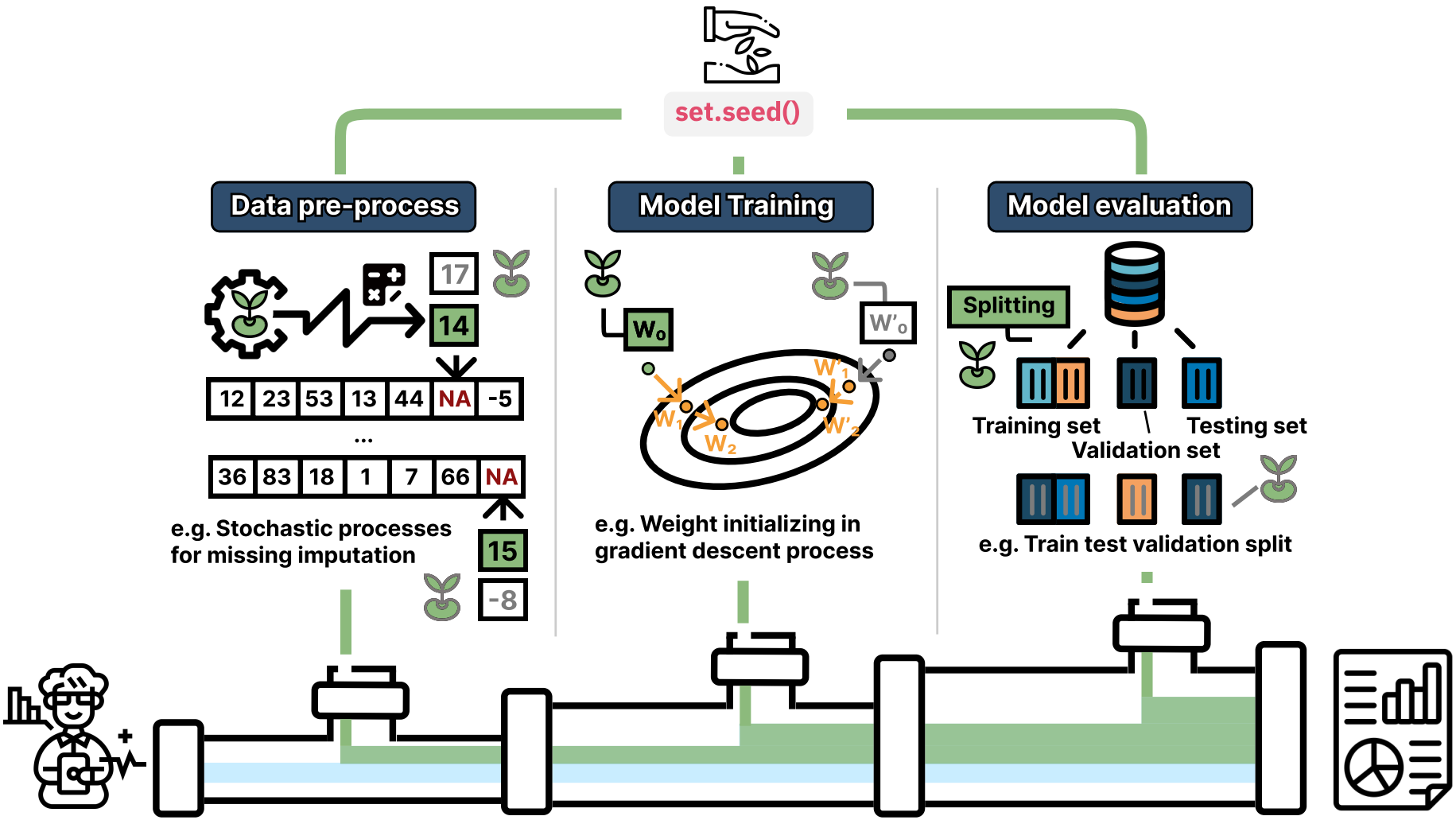
- Previously thought that nothing outperforms RW for ForEx.
- RWs commonly used as a benchmark for structural models.

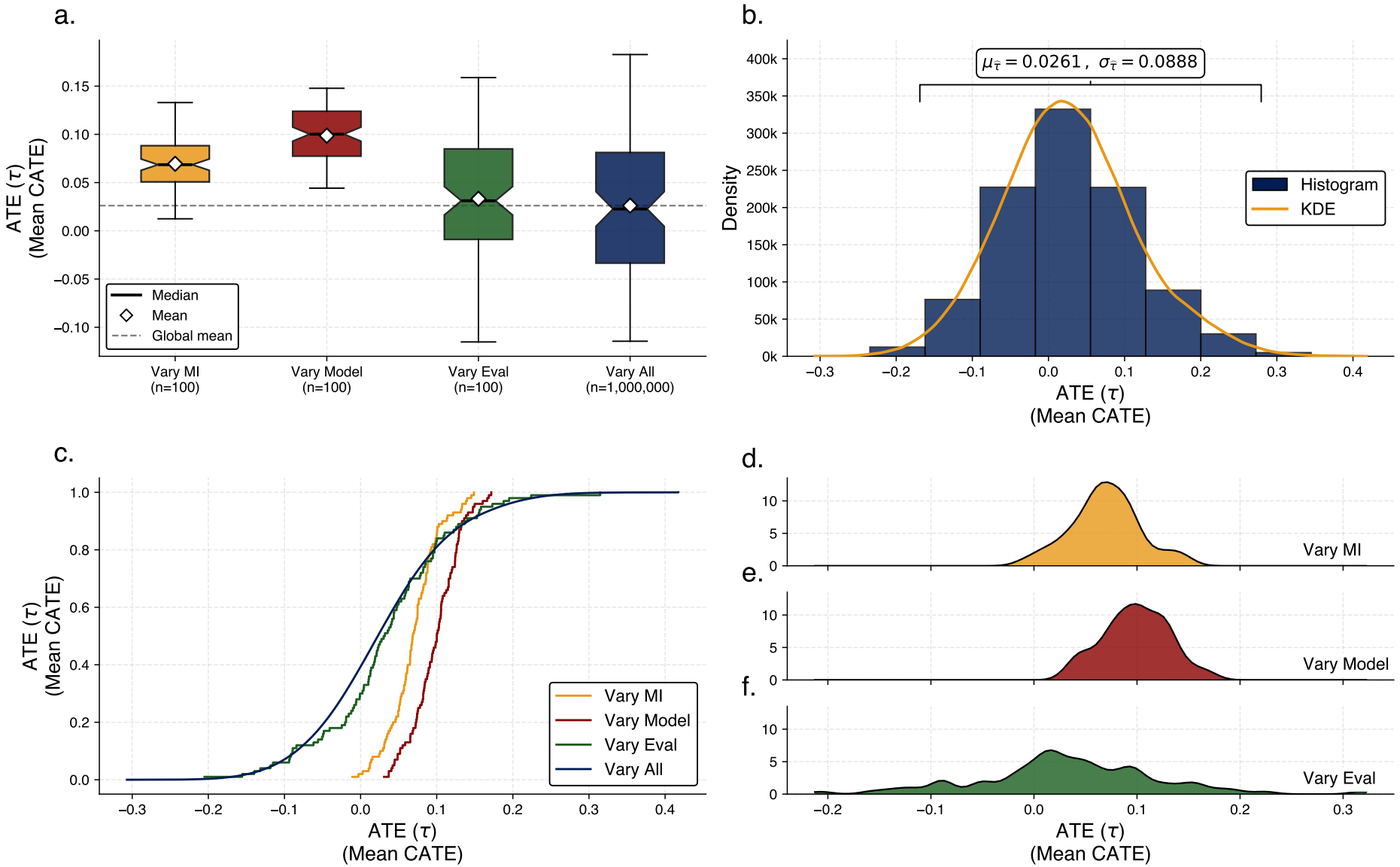

a.
E(AUC) = 0.743, σ(AUC) = 0.017
Bins
KDE
Original Result:
ROC-AUC=0.76
(0.74-0.78)
Density
ROC-AUC
b.
R²
c.
R²: Modelling (σ)
d.
R²: Folding (σ)
e.
IMV
f.
IMV: Modelling (σ)
g.
IMV: Folding (σ)
h.
E(acc) = 9623.302, σ(acc) = 65.879
Seed 42:
Accuracy=9625
Seed 123:
Accuracy=9690
Accuracy

What's going on here?

In addition to modelling, ML/DL pipelines involve stochastic folding.

- This happens with every form of CV.
- The only exception is LOOCV.
- The fewer the splits, the more the seed matters.
- We are able to decompose types of seed effect.
  - This is done with Titanic and housing examples.
- The final example shows no folding CV, only model variability.


set.seed()
Data pre-process
Model Training
Model evaluation
17
14
12 23 53 13 44 NA -5
...
36 83 18 1 7 66 NA
15
-8
e.g. Stochastic processes for missing imputation
W0
W'0
W1
W2
W'1
W'2
e.g. Weight initializing in gradient descent process
Splitting
Training set
Validation set
Testing set
e.g. Train test validation split

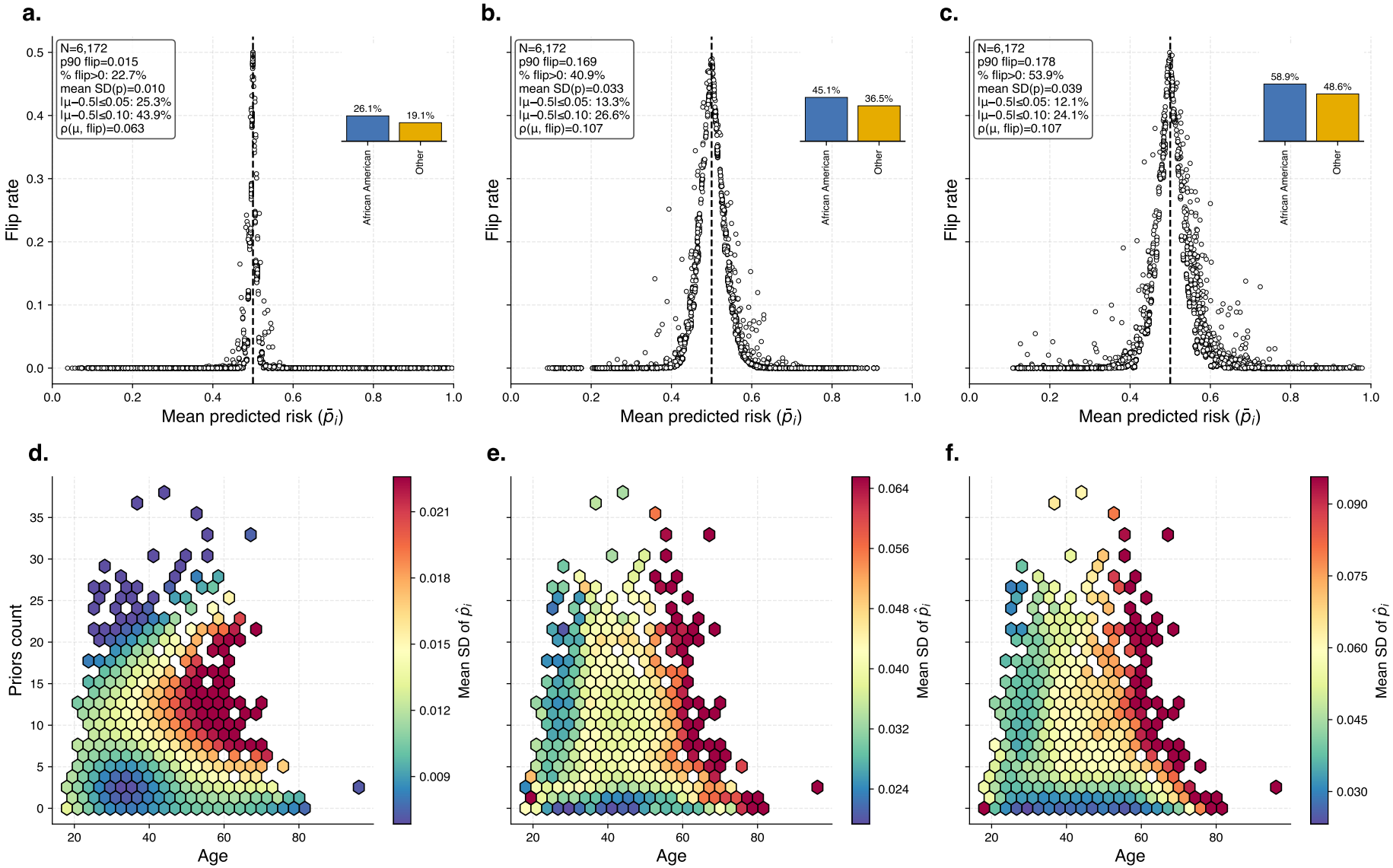

a.
ATE (τ)
(Mean CATE)
Median
Mean
Global mean
Vary MI
(n=100)
Vary Model
(n=100)
Vary Eval
(n=100)
Vary All
(n=1,000,000)
b.
$\mu_{\hat{\tau}} = 0.0261$, $\sigma_{\hat{\tau}} = 0.0888$
Density
Histogram
KDE
ATE (τ)
(Mean CATE)
c.
ATE (τ)
(Mean CATE)
Vary MI
Vary Model
Vary Eval
Vary All
ATE (τ)
(Mean CATE)
d.
Vary MI
e.
Vary Model
f.
Vary Eval
ATE (τ)
(Mean CATE)

Compound Simulation: Core Idea

- Assess how randomness from imputation, model fitting, and evaluation affects downstream ATE estimates.
- Importantly, it shows how seed variability compounds.
- Pipeline components:
  - Simulate one dataset with MAR data.
  - IterativeImputer (MICE-style stochastic draws),
  - CausalForestDML (cross-fitted).
- Compare five seed-varying schemes: `baseline_fixed`, `vary_imp`, `vary_model`, `vary_eval`, `vary_all`.

Simulation Parameters

- $N = 600$
- $d = 16$
- $\tau = 0.10$
- miss_rate $= 0.50$
- $R_{\text{imp}} = 100$
- $R_{\text{model}} = 100$
- $R_{\text{eval}} = 100$
- $M_{\text{imputations}} = 1$
- n_estimators $= 200$
- subforest_size $= 4$
- cv_folds $= 3$
- n_jobs $= -1$


a.
N=6,172
p90 flip=0.015
% flip>0: 22.7%
mean SD(p)=0.010
|μ−0.5|≤0.05: 25.3%
|μ−0.5|≤0.10: 43.9%
ρ(μ, flip)=0.063
26.1%
19.1%
African American
Other
Flip rate
Mean predicted risk ($\bar{p}_i$)
b.
N=6,172
p90 flip=0.169
% flip>0: 40.9%
mean SD(p)=0.033
|μ−0.5|≤0.05: 13.3%
|μ−0.5|≤0.10: 26.6%
ρ(μ, flip)=0.107
45.1%
36.5%
c.
N=6,172
p90 flip=0.178
% flip>0: 53.9%
mean SD(p)=0.039
|μ−0.5|≤0.05: 12.1%
|μ−0.5|≤0.10: 24.1%
ρ(μ, flip)=0.107
58.9%
48.6%
d.
e.
f.
Priors count
Age
Mean SD of $\hat{p}_i$

COMPAS: seed-driven OOF instability

Emphasizes the implications for distributional fairness.

- Seed variability: because of model instantiation and folding.
- Uses the infamous Pro-Publica dataset on recidivism.
- Repeated 2-fold OOF across many seeds for RF, LR, and MLP.
- Summarise per-UID instability (variance / flip-rate).

COMPAS: seed-driven OOF instability

Emphasizes the implications for distributional fairness.

- Seed variability: because of model instantiation and folding.
- Uses the infamous Pro-Publica dataset on recidivism.
- Repeated 2-fold OOF across many seeds for RF, LR, and MLP.
- Summarise per-UID instability (variance / flip-rate).

Other obvious extensions:

COMPAS: seed-driven OOF instability

Emphasizes the implications for distributional fairness.

- Seed variability: because of model instantiation and folding.
- Uses the infamous Pro-Publica dataset on recidivism.
- Repeated 2-fold OOF across many seeds for RF, LR, and MLP.
- Summarise per-UID instability (variance / flip-rate).

Other obvious extensions:

1. Different domains (mortgage finance, health)

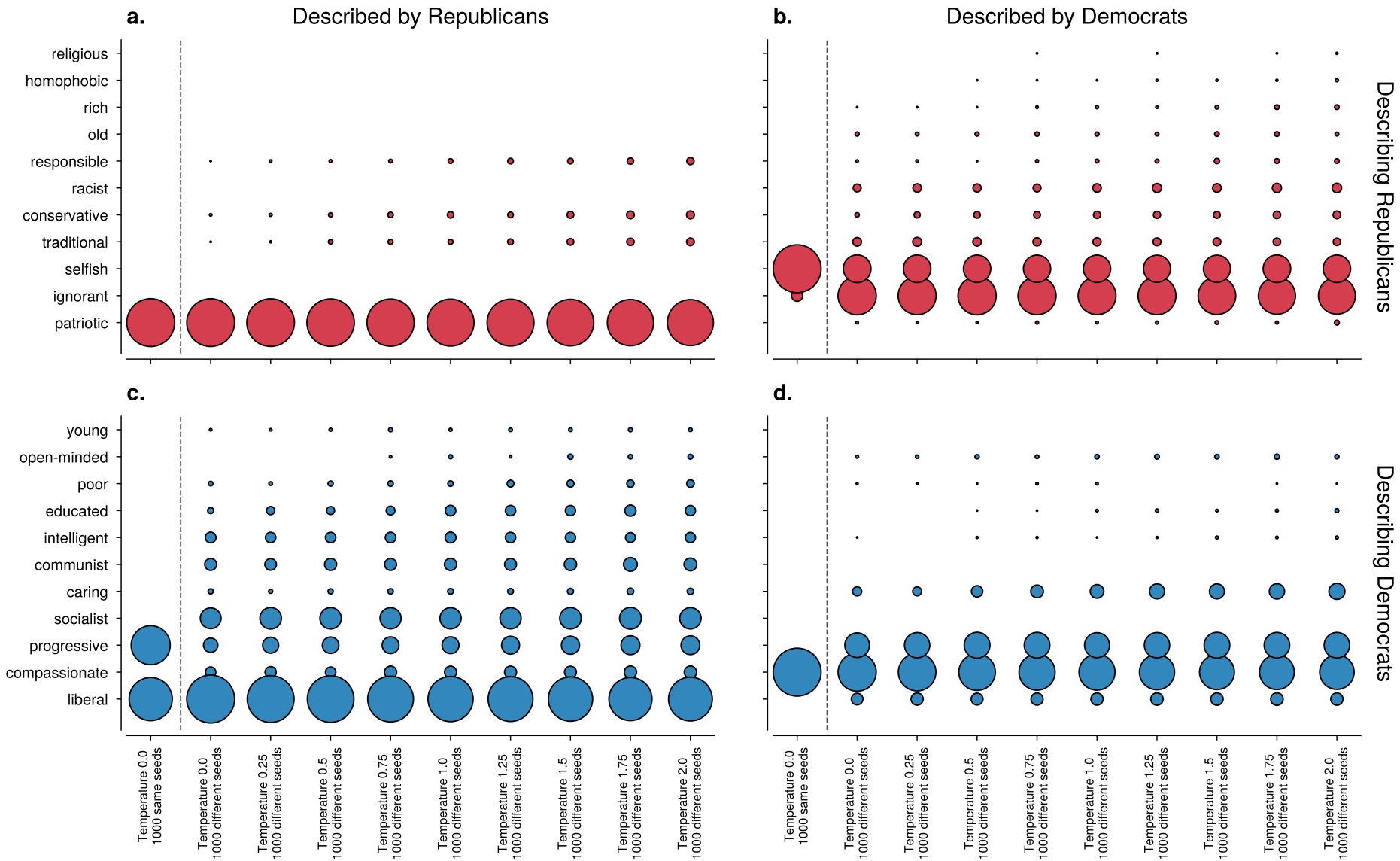
COMPAS: seed-driven OOF instability

Emphasizes the implications for distributional fairness.

- Seed variability: because of model instantiation and folding.
- Uses the infamous Pro-Publica dataset on recidivism.
- Repeated 2-fold OOF across many seeds for RF, LR, and MLP.
- Summarise per-UID instability (variance / flip-rate).

Other obvious extensions:

1. Different domains (mortgage finance, health)
2. Alternate metrics, e.g., statistical parity.


a.
Described by Republicans
b.
Described by Democrats
c.
d.
Describing Republicans
Describing Democrats
religious
homophobic
rich
old
responsible
racist
conservative
traditional
selfish
ignorant
patriotic
young
open-minded
poor
educated
intelligent
communist
caring
socialist
progressive
compassionate
liberal
Temperature 0.0 1000 same seeds
Temperature 0.0 1000 different seeds
Temperature 0.25 1000 different seeds
Temperature 0.5 1000 different seeds
Temperature 0.75 1000 different seeds
Temperature 1.0 1000 different seeds
Temperature 1.25 1000 different seeds
Temperature 1.5 1000 different seeds
Temperature 1.75 1000 different seeds
Temperature 2.0 1000 different seeds

Seed variability in the experiment

- Outer loop varies temperature across nine fixed values.
- Inner loop iterates over up to 1000 seeds.
- Outputs depend jointly on temperature and seed.
- Fixed-seed experiment repeats 1000 calls at temperature zero.
- Temperature has little effect on variability compared to seed.
- The seed regulates the sampling step – the choice among logits at each token.
  - But the logits themselves come from a huge distributed system whose compute is not bit-deterministic.
- Even with *fixed seed* (and `temp=0`): non-determinism abounds!

Concluding Thoughts (Part One)

- Matt Salganik commented: 'That sounds great, but expensive!'
  - Expense no-longer prohibitive: everything here ran locally.
  - We live in an age of vast computational resources.

Concluding Thoughts (Part One)

- Matt Salganik commented: 'That sounds great, but expensive!'
  - Expense no-longer prohibitive: everything here ran locally.
  - We live in an age of vast computational resources.
- What are our suggestions?

Concluding Thoughts (Part One)

- Matt Salganik commented: 'That sounds great, but expensive!'
  - Expense no-longer prohibitive: everything here ran locally.
  - We live in an age of vast computational resources.
- What are our suggestions?
  - **Researchers:** Visualize your seed variability where affordable!
    - There are times when you can eliminate/reduce seed variability.
    - Pedagogically, lets teach students to parralelize code effectively.

Concluding Thoughts (Part One)

- Matt Salganik commented: 'That sounds great, but expensive!'
  - Expense no-longer prohibitive: everything here ran locally.
  - We live in an age of vast computational resources.
- What are our suggestions?
  - **Researchers:** Visualize your seed variability where affordable!
    - There are times when you can eliminate/reduce seed variability.
    - Pedagogically, lets teach students to parralelize code effectively.
  - **Reviewers:** Question papers you read – is there stochasticity?

Concluding Thoughts (Part One)

- Matt Salganik commented: 'That sounds great, but expensive!'
  - Expense no-longer prohibitive: everything here ran locally.
  - We live in an age of vast computational resources.
- What are our suggestions?
  - **Researchers:** Visualize your seed variability where affordable!
    - There are times when you can eliminate/reduce seed variability.
    - Pedagogically, lets teach students to parralelize code effectively.
  - **Reviewers:** Question papers you read – is there stochasticity?
    - Even well established results are susceptible.

Concluding Thoughts (Part One)

- Matt Salganik commented: 'That sounds great, but expensive!'
  - Expense no-longer prohibitive: everything here ran locally.
  - We live in an age of vast computational resources.
- What are our suggestions?
  - **Researchers:** Visualize your seed variability where affordable!
    - There are times when you can eliminate/reduce seed variability.
    - Pedagogically, lets teach students to parralelize code effectively.
  - **Reviewers:** Question papers you read – is there stochasticity?
    - Even well established results are susceptible.
  - **Editors:** Ubiquitous mandate for responsible use of PRNGs!

simple_ranges.py

```python
import numpy as np # randomisation
import matplotlib.pyplot as plt # plotting

def analytical_function(func_input, seed):
        '''Simple analytical function: can be anything'''
        np.random.seed(seed)
        return func_input*np.random.normal()

results = [] # store results
inputs = 42 # dataset, figure path, etc.

for seed in range(0, 100000): # simple loop; can be distributed
        results.append(analytical_function(inputs, seed))

plt.hist(results) # plot results
```

better_seed_algorithm.py

```python
import numpy as np # randomisation
import matplotlib.pyplot as plt # plotting

def analytical_function(func_input, seed):
      '''Simple analytical function: can be anything'''
      np.random.seed(seed)
      return func_input*np.random.normal()

results = [] # store results
inputs = 42 # dataset, figure path, etc.

# Use predefined 'secret' seeds we provide
with open(os.path.join('..', 'assets', 'seed_list.txt')) as f:
      seed_list = [int(line.rstrip('\n')) for line in f]

for seed in seed_list: # simple loop; can be distributed
      results.append(analytical_function(inputs, seed))

plt.hist(results) # plot results
```

Concluding Thoughts (Part Two)

- Ideally, we want to move towards distributional reproducibility.
- Moving foward, PRNGs shouldn't exist (QRNGs the norm).
- Does anyone have ideas for other types of seed variability?
- Can this be corrective? Index historical seed variability?
- Prospectively: we make available a list of seeds (replication materials), encouraging their use as a pre-specified set.
- **TLDR**: when variation can't be eliminated, should be visualised.

Thanks to my Coauthors!

Mark Verhagen

Jiani Yan

Thanks to my Coauthors!

Mark Verhagen

Jiani Yan

And thanks to you for your attention and attendance!

Replication Materials

Naturally, containerised code available on GitHub at the QR code.