Formula One race prediction

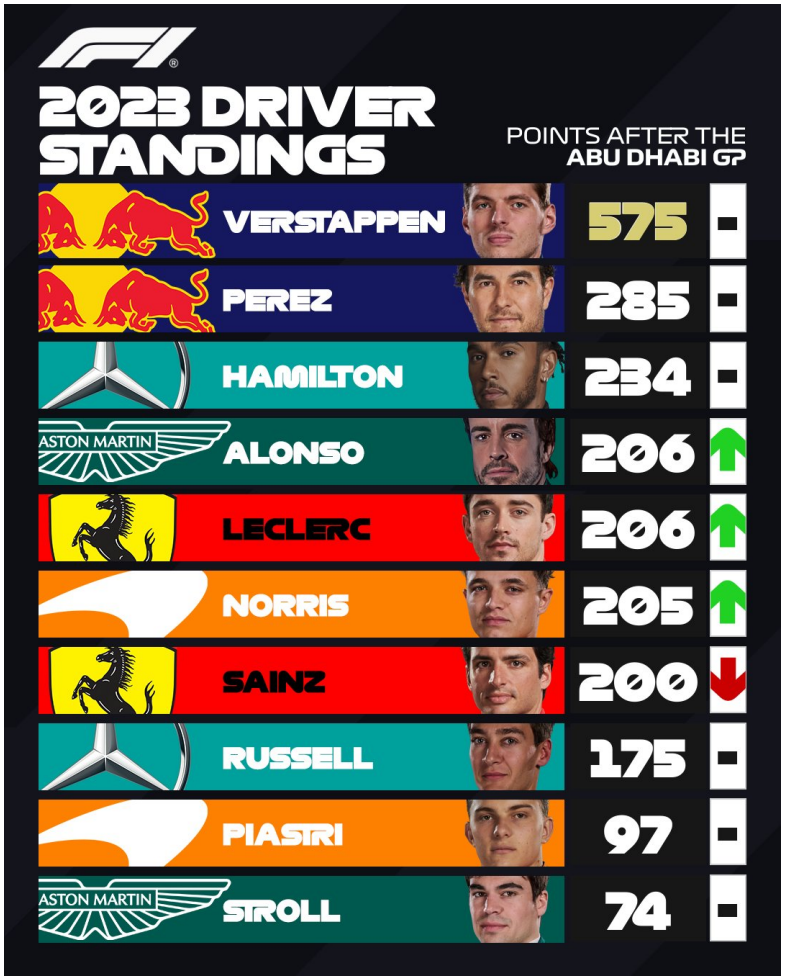
John Walker

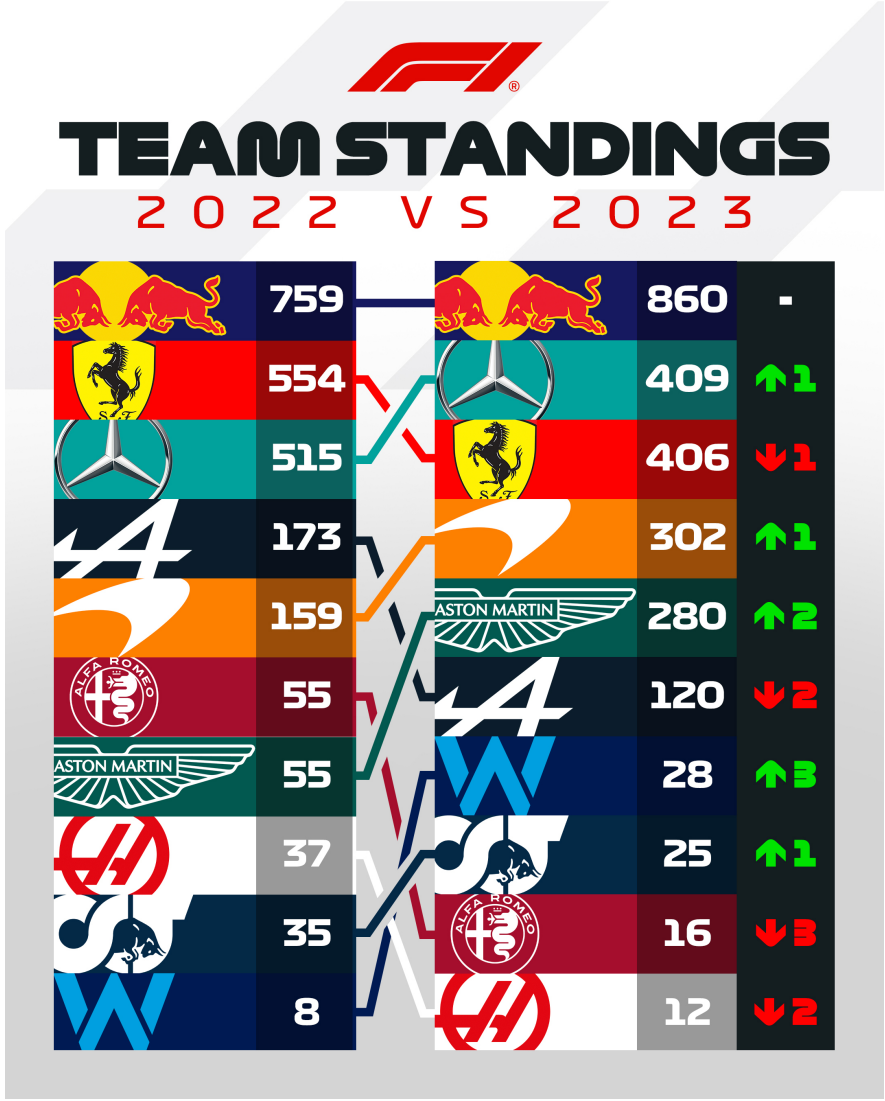
Formula One

Team Haas


MoneyGram
HAAS
Palm Angels
HAAS
MoneyGram
MoneyGram
HAAS F1 TEAM

(Top 10) Standings for this season





Problem

Original problem: predict race results using car telemetry data (lap times, tyre degradation, . . .)

- too hard for me
- literature review: maybe simulation is better in this context

Actual problem: predict race results using race results

- in: driver positions in each completed race
- out: predicted driver positions in the next race

Method

[1] devises a model for "rating" players of multicompetitor games and [2] uses a slight variation of that method to try and find the "best" Formula 1 driver in recent times

- multicompetitor game: a competition where N players participate at once and are ranked $1, \ldots, N$
- [1]: assign each competitor i a skill number θ_i
- [2]: decompose $\theta_i = \theta_{\text{driver } i} + \theta_{\text{constructor } i}$

Idea: predict that the driver with the highest θ_i will win

Bayesian statistics

Bayes's theorem:

$$p(\theta|y) = p(y|\theta)p(\theta)/Z \qquad Z = \int p(y|\theta)p(\theta)d\theta$$

Allows information about the posterior $p(\theta|y)$ to be gained given data y, a prior $p(\theta)$, and sampling distribution $p(y|\theta)$

In Bayesian inference:

- a chosen model specifices $p(y|\theta)$
- you specify $p(\theta)$
- use data y to sample θ from $p(\theta|y)$ (not finding $p(\theta|y)$)
- make conclusions based on this (estimate θ, new data, ...)

Issue: how do you sample from $p(\theta|y)$?

Monte Carlo: illustration from physics

The probability of being in the state $|n\rangle$ with energy E_n is:

$$P(n) = \exp(-E_n/k_bT)/Z \qquad Z = \sum_i \exp(-E_i/k_bT)$$

Want to calculate expectations of operators A:

$$\langle A\rangle = \sum_n P(n)\langle n|A|n\rangle = \sum_n A_{nn}\exp(-E_n/k_bT)/Z$$

But the partition function Z is basically impossible to calculate.

Solution: Monte Carlo

Monte Carlo: illustration from physics

Method:

- begin with a state $|n\rangle$
- propose a new state $|m\rangle$
- accept with probability $\min(1, P(m)/P(n))$
- repeat
- Theorem: these samples are drawn from P
- estimate expectations by sample expectations

Main point: Z is no longer needed to generate samples because you divide it out

The Bayesian workflow

[3] summarizes the process of making a Bayesian model for a problem as follows:

1. Define a (joint) probability model over observables y and unobservables θ
2. Condition on observed data y to infer the posterior distribution of a quantity of interest
3. Evaluate the model

In practice, for each step you do the following:

1. Read papers
2. Code
3. Mix of coding and thinking if the results make sense

Model (slightly simplified) from [1]

N competitors in a multicompetitor game each have an ability θ_i. When they play the game, they each draw a latent performance Y_i:

$$Y_i \sim \text{Gumbel}(\theta_i) \qquad F_{\text{Gumbel}}(y|\theta) = \exp(-\exp(\theta - y))$$

The winner is the competitor with the largest Y_i down to the smallest.

It can be shown:

$$P(Y_1 > Y_2 > \cdots > Y_N|\theta) = \prod_{i=1}^{N-1} \exp(\theta_i) / \sum_{j=i}^{N} \exp(\theta_j)$$

For a given θ, this is maximized when $\theta_1 > \theta_2 > \cdots > \theta_N$: the most likely outcome of a game to be the ordering of the θ_i.

Model (slightly simplified) from [2]

Decompose each drivers θ_i into driver skill + constructor skill:

$$\theta_i = \theta_{\text{driver } i} + \theta_{\text{constructor } i}$$

Further decompose driver skill based on a belief that wet or dry racing environments affect skill, and similarly for constructor skill for street and circuit races:

$$\theta_{\text{driver } i} = \gamma_{0,i} + I(\text{wet})\gamma_{1,i} \qquad \theta_{\text{constructor } i} = \beta_{i,i} + I(\text{circuit})\beta_{1,i}$$

All combinations lead to 4 different submodels

Do this for each season of interest

My model

- I care about results now, so I used only one season
- Mostly interested in this most recent season, which was pretty dry and sometimes the rain came later in the race
- I believe circuit type is mostly reflected in driver skill instead of constructor

I looked at 3 submodels:

$$\theta_i = \theta_i$$

$$\theta_i = \theta_{\text{driver } i} + \theta_{\text{constructor } i}$$

$$\theta_{\text{driver } i} = \gamma_{0,i} I(\text{street}) + \gamma_{1,i} I(\text{circuit})$$

Next step in the Bayesian workflow is to code this

Coding

Data preparation was done using `Python` with the `fastf1` library. For model fitting, `Stan` was used

Structure of a `Stan` program:

```
functions {}
data {}
transformed data {}
parameters {}
transformed parameters {}
model {}
generated quantities {}
```

Submodel comparison

Before doing a model evaluation, one best submodel is selected

I used the `R` package `loo` for this

- "efficient leave one out cross validation"
- efficient because it just needs the likelihood of each sample
- no refitting required

Output:

- `elpd_loo`: measure of cross validation goodness; greater is better
- `p_loo`: effective number of parameters
- `pareto k diagnostic values`: measure of how reliable the `elpd_loo` calculation is

Submodel comparison

`loo` results:

Model	`elpd_loo`	`pareto k`
Single skill	-863.3 ± 14.4	good
Con + dri	-863.7 ± 15	good
Con + dri + cir	-865.4 ± 16	bad

While the single skill model evaluates the best, I chose the con + dri model as best since it evaluates about the same and is more interpretable

Results

Recall the con + dri model:

$$\theta_i = \theta_{\text{driver } i} + \theta_{\text{constructor } i}$$

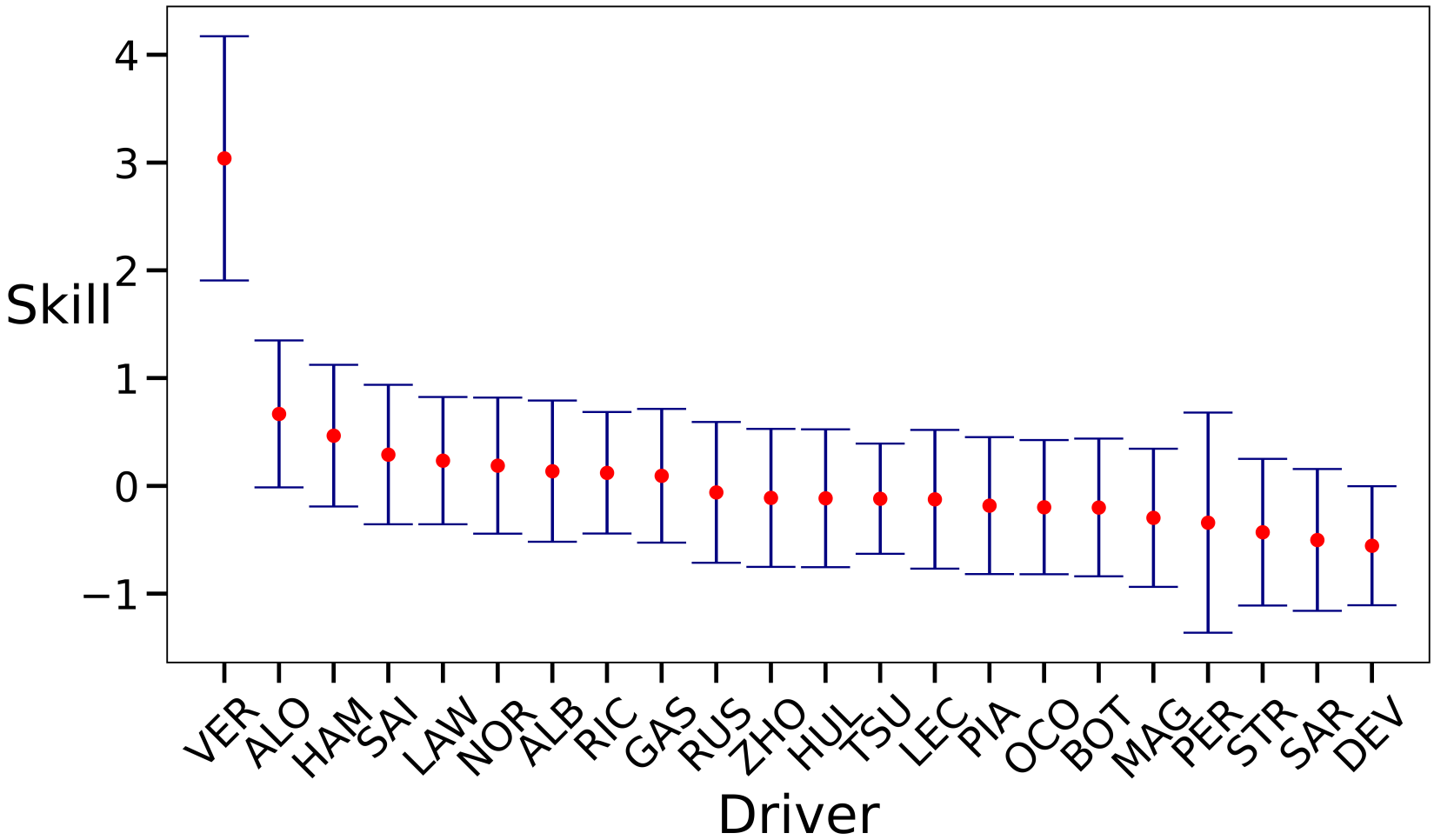
What running `Stan` did was sample numerous $\theta_{\text{driver } i}$ and $\theta_{\text{constructor } i}$

Estimates:

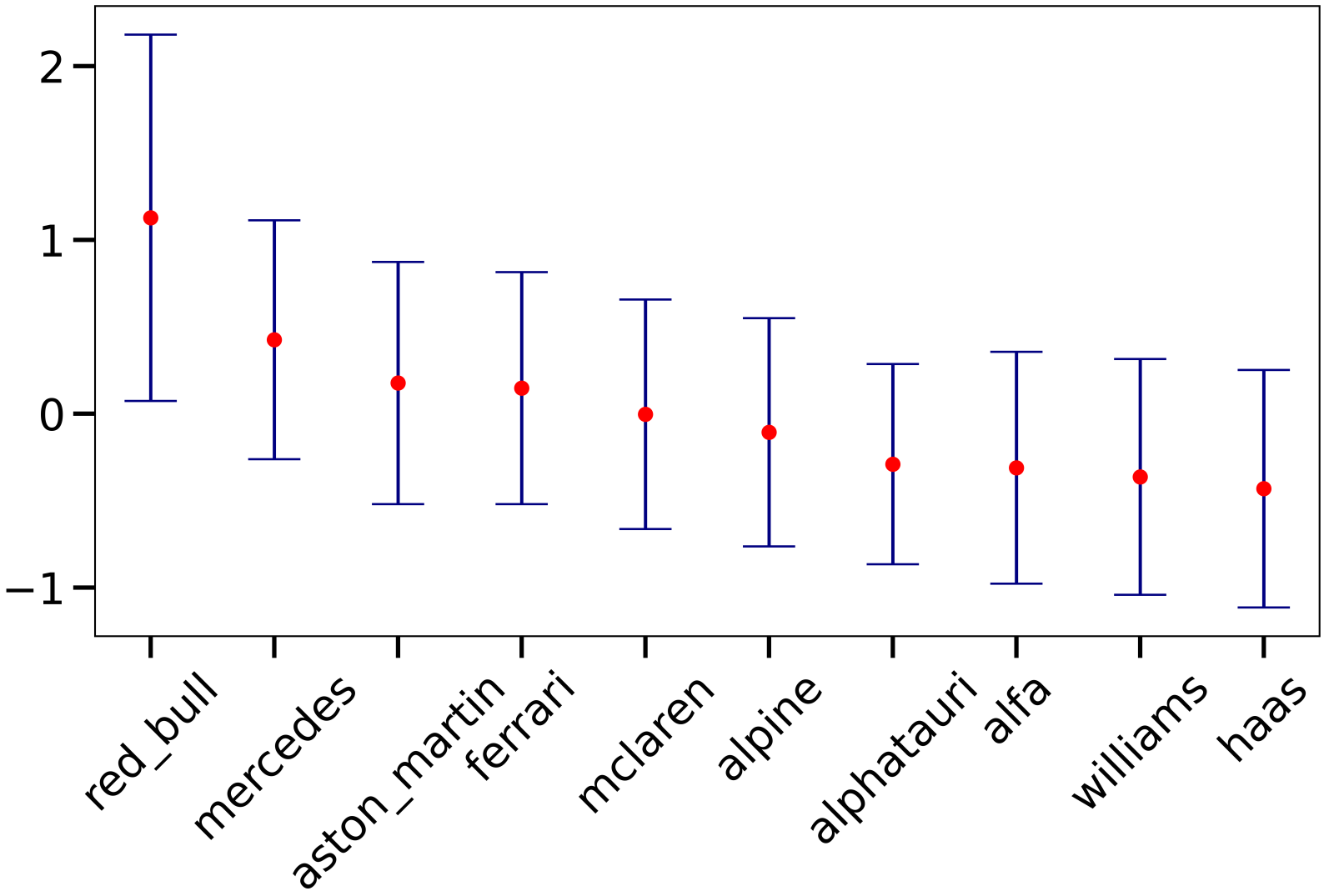
- $\hat{\theta}_{\text{driver } i} = \text{mean}(\theta_{\text{driver } i})$
- $\hat{\theta}_{\text{constructor } i} = \text{mean}(\theta_{\text{constructor } i})$

Set $\hat{\theta}_i = \hat{\theta}_{\text{driver } i} + \hat{\theta}_{\text{constructor } i}$ and my prediction for the next race is the ordering of these $\hat{\theta}_i$'s

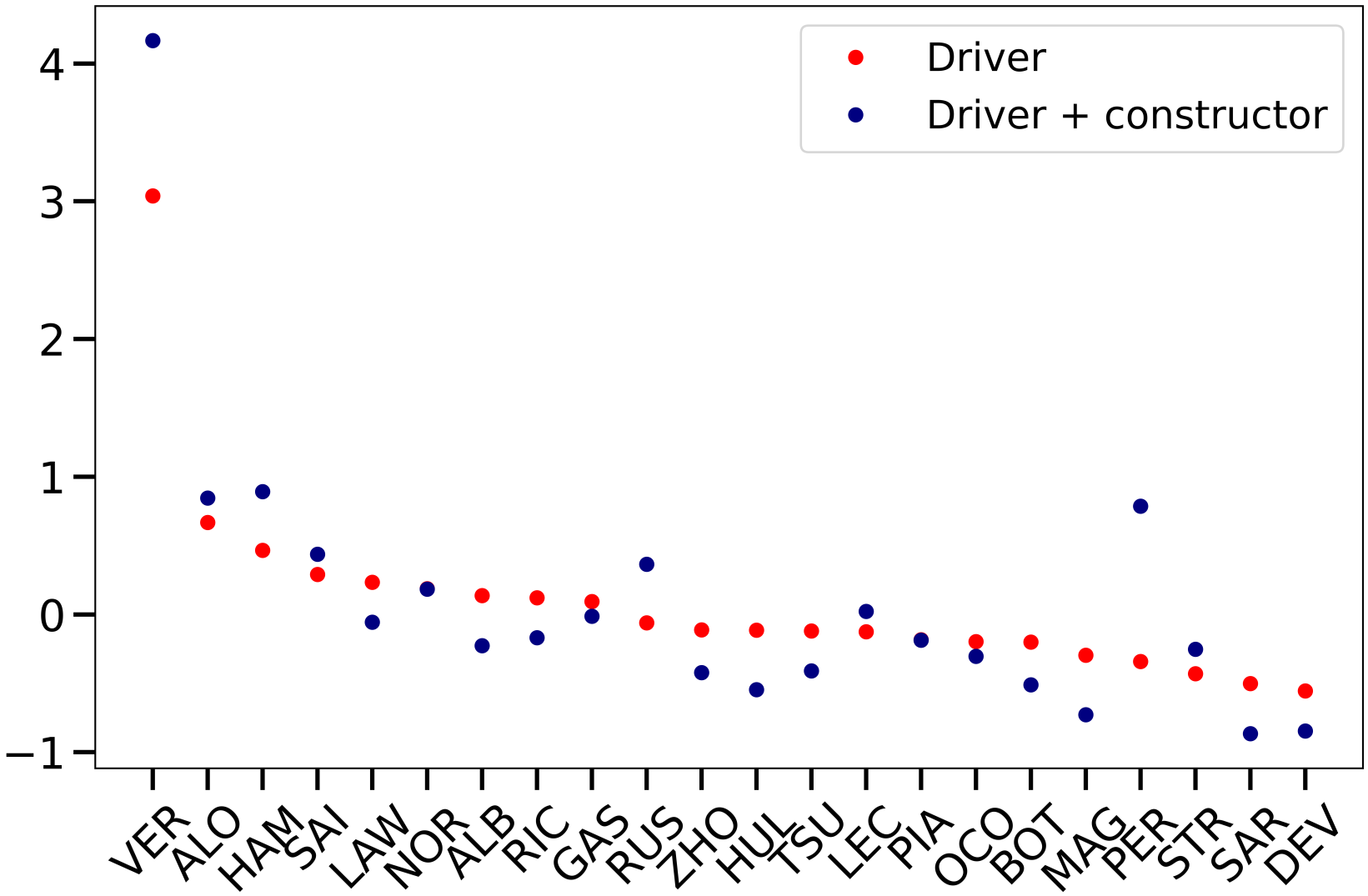
Results: driver skill 2023 season



Results: constructor skill 2023 season



Results: total skill 2023 season (errorbars omitted)



Comparison with standings (for 20 finishing drivers)

Driver	Actual	Predicted	Driver	Actual	Predicted
VER	1	1	GAS	11	9
PER	2	4	OCO	12	14
HAM	3	2	ALB	13	12
ALO	4	3	TSU	14	15
LEC	5	8	BOT	15	17
NOR	6	7	HUL	16	18
SAI	7	5	RIC	17	10
RUS	8	6	ZHO	18	16
PIA	9	12	MAG	19	19
STR	10	14	SAR	20	20

Ranking of total skills is not far off from the actual overall rankings

Posterior predictive check

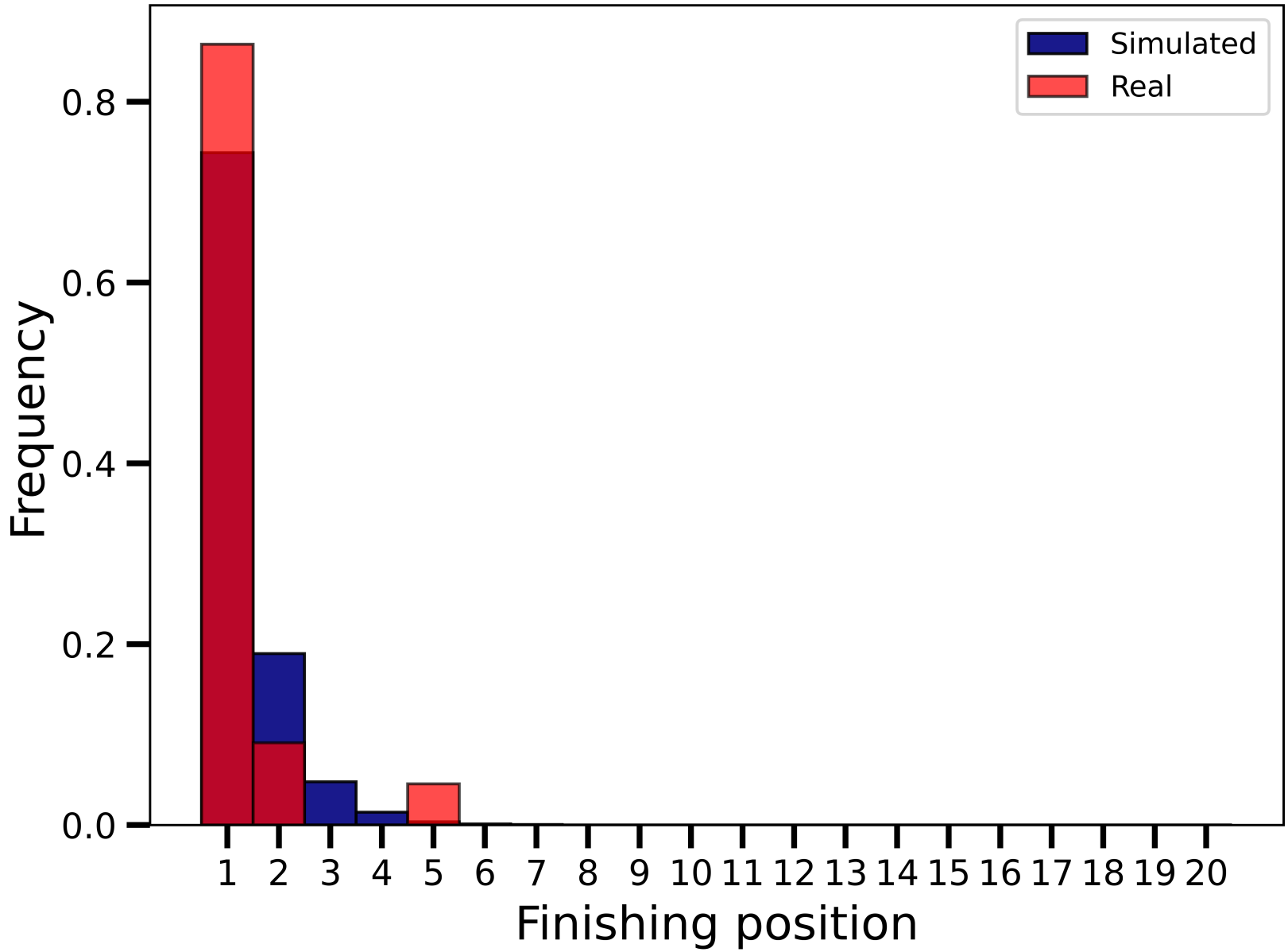
Posterior predictive inference: predict new data $\tilde{y}$ given existing data y [3]

We used race result data y to get skill samples $\theta^{(m)}$. We can use these samples to simulate races $\tilde{y}$

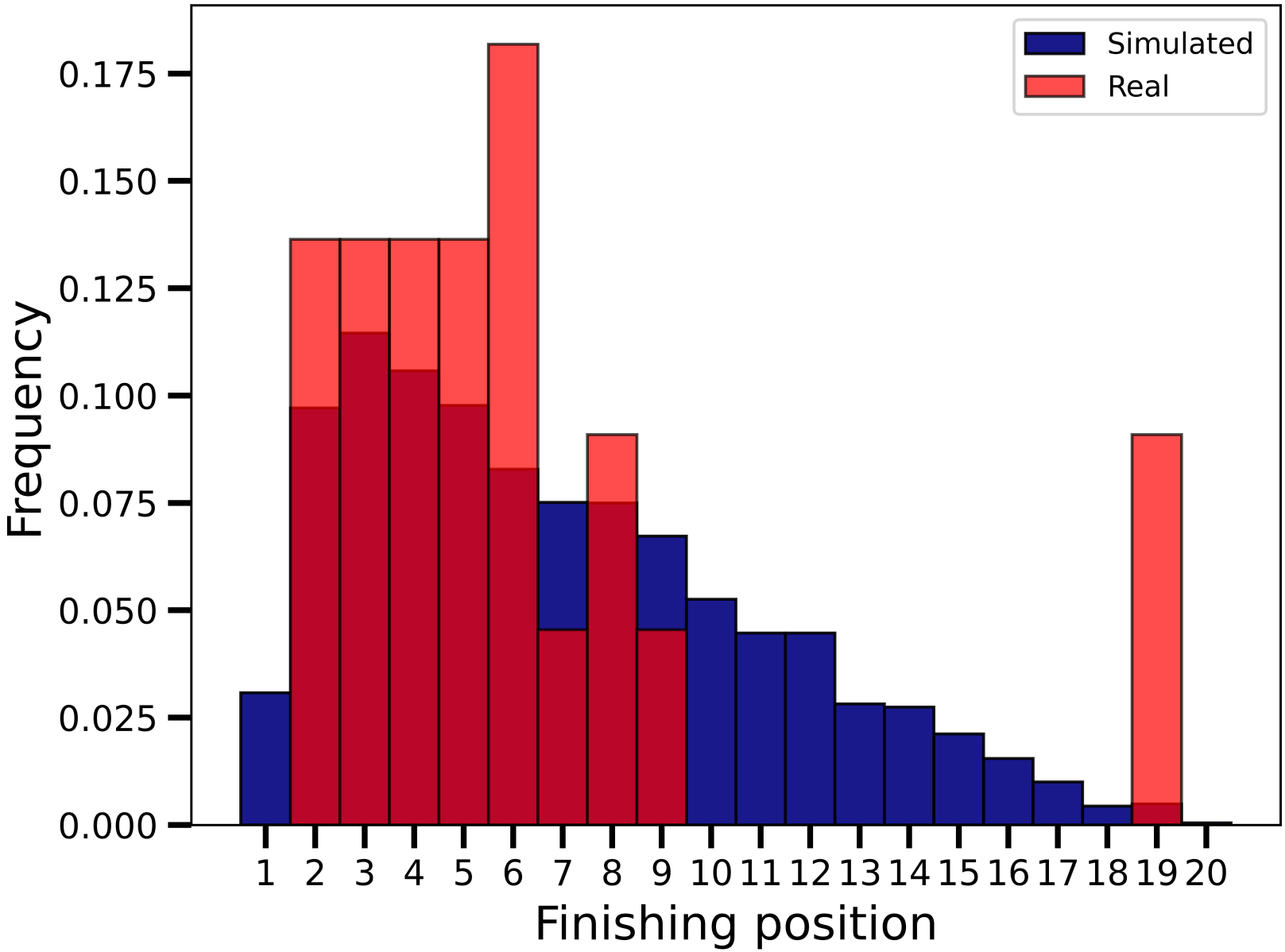
Race simulation:

- for a race, find the drivers that participated in that race and take a sample $\theta^{(m)}$ with all of their skills
- sample a performance $Y \sim \text{Gumbel}(\theta^{(m)})$
- order Y to determine the outcome of the race
- repeat a bunch of times for each race, obtaining a frequency of positions

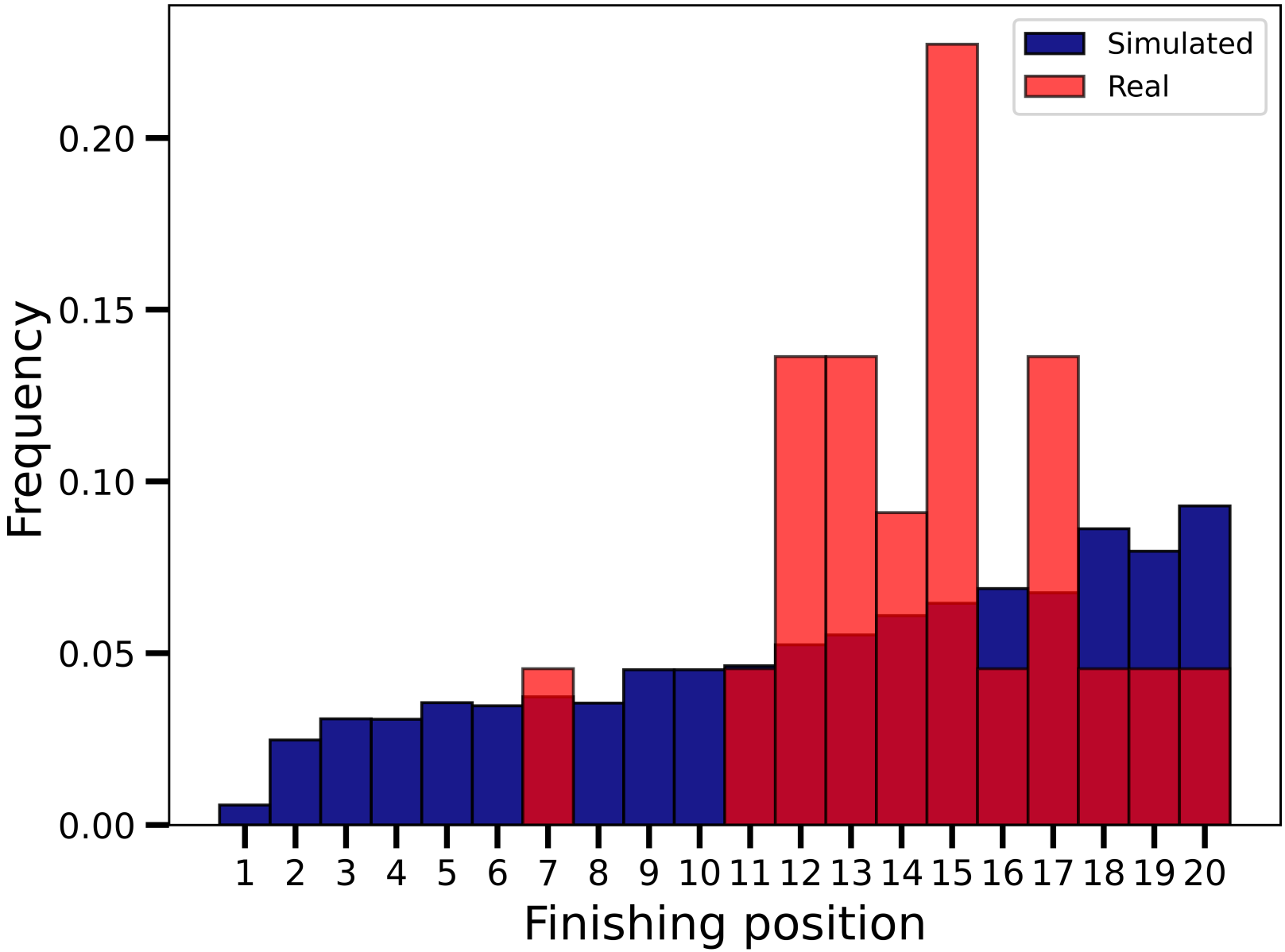
Simulation: Verstappen 2023



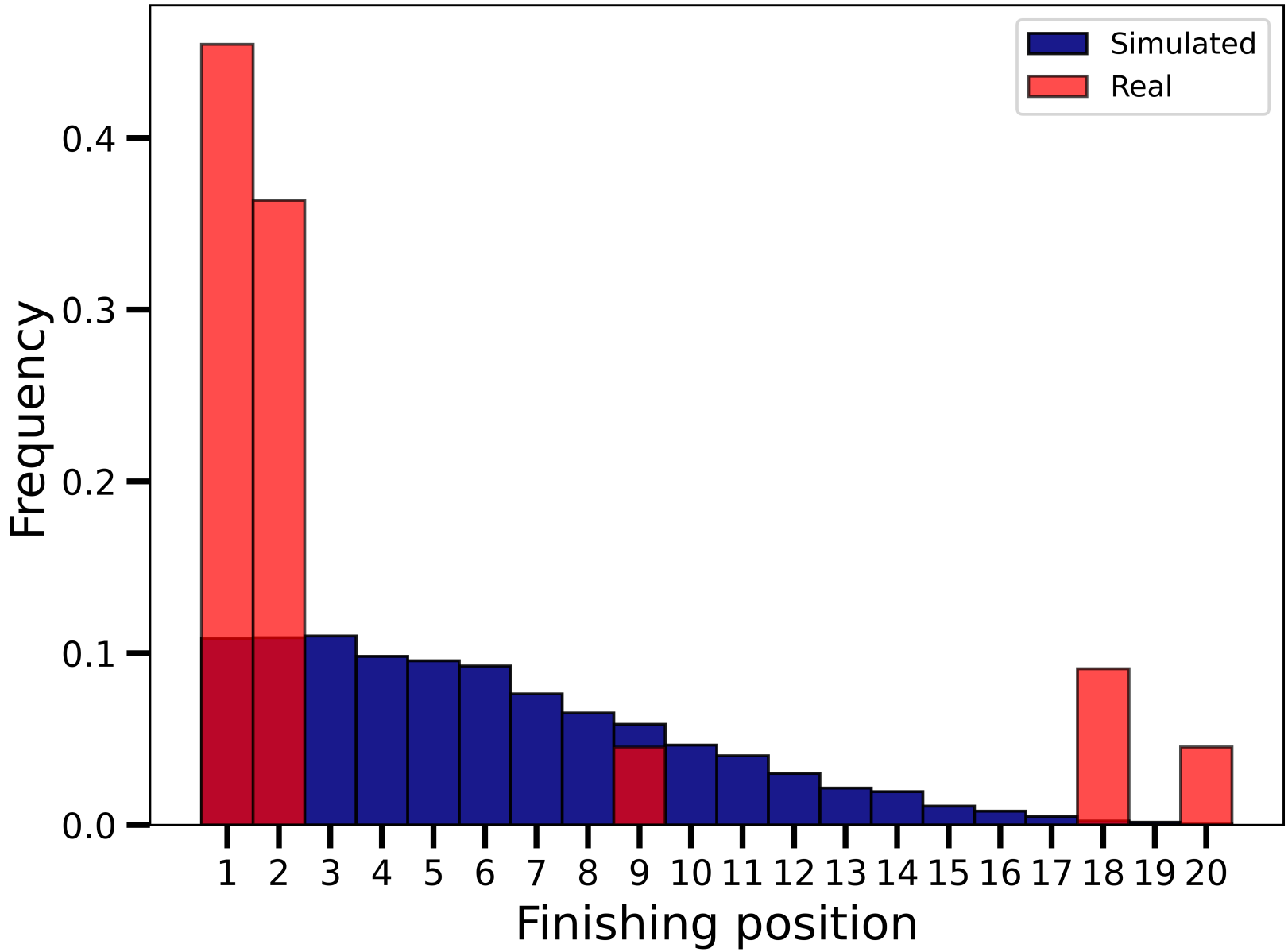
Simulation: Hamilton 2023



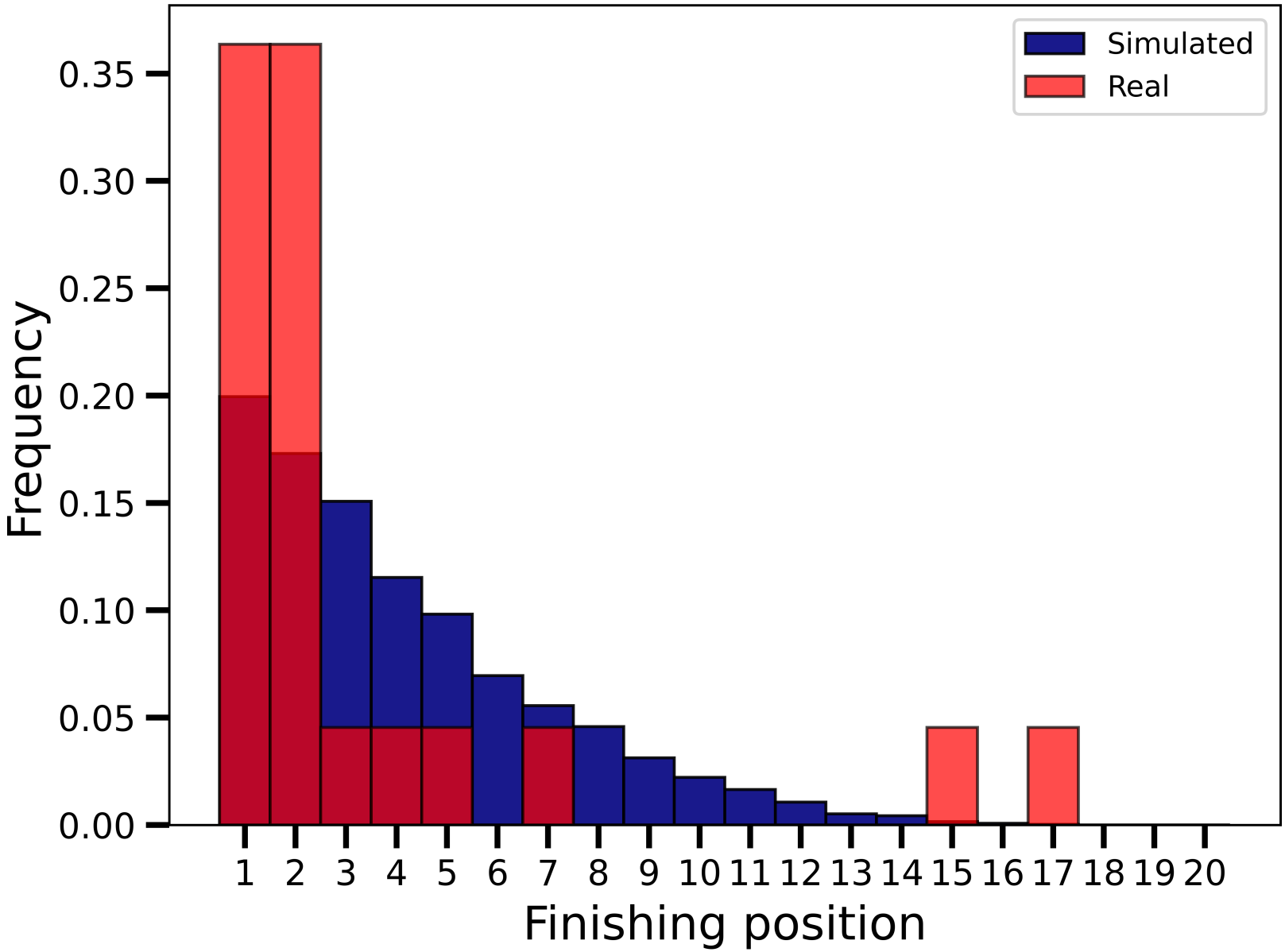
Simulation: Hulkenberg 2023



Simulation: Verstappen 2021



Simulation: Hamilton 2021



Conclusion

The model is okay

- I think partitioning the season into parts based on car upgrades might improve it (pre and post summer break might be good)
- I think actual simulation would predict a lot better
- I think a sharper distribution that performances are drawn from would be better

References

[1] Glickman and Hennessy A stochastic rank ordered logit model for rating multi-competitor games and sports `http://www.glicko.net/research/multicompetitor.pdf`

[2] Kesteren and Bergkamp Bayesian Analysis of Formula One Race Results: Disentangling Driver Skill and Constructor Advantage `https://arxiv.org/pdf/2203.08489.pdf`

[3] Carpenter Getting Started with Bayesian Statistics `https://bob-carpenter.github.io/stan-getting-started/`

Code (rough version for now): `https://github.com/Meromorphics/f1project`