Lecture #17: MCMC & Missingness

aka STAT109A, AC209A, CSCIE-109A

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Gumb

Lecture Outline

- Catching our Breathe
- Rejection Sampling
- MCMC
- Metropolis-Hastings
- Dealing with Missingness
    - Types of Missingness
- Imputation Methods

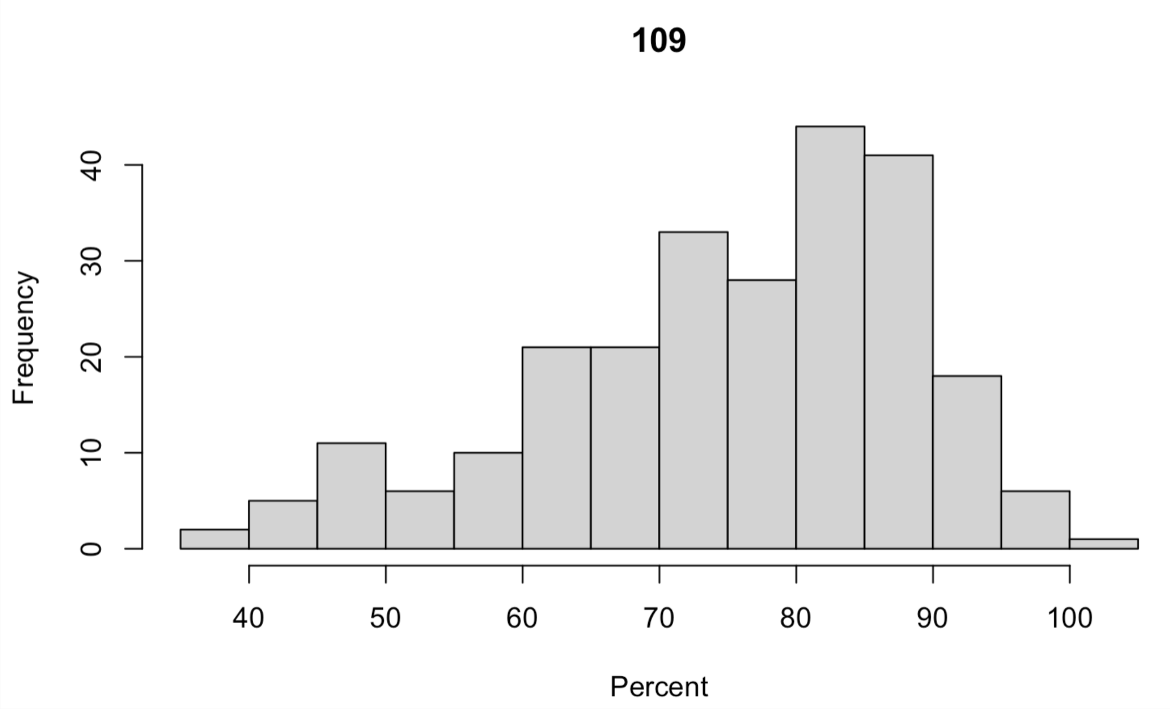
Stepping Back for a Moment

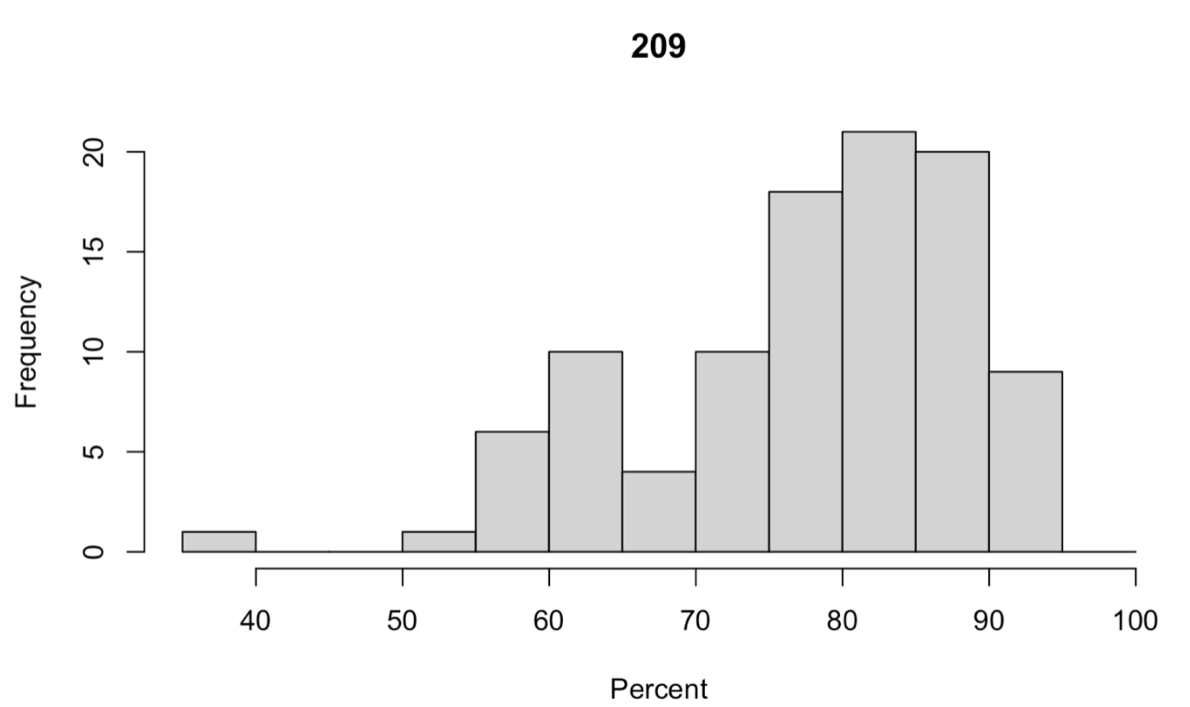
- When deriving the posterior distribution, we can drop scaling constants not including the unknown parameter.
    - These constants make the posterior area *sum* to 1.
- By combining posterior for θ with our data generating model, we can derive a posterior predictive distribution of future data.
    - A helpful tool for forecasting!
- For our choices of likelihood and prior, the posterior has been the same type of distribution as the prior so far.
    - Recall: this is called **conjugacy**.
- There are many examples of conjugate distributions (with their associated posterior predictive distributions).
    - See the Wikipedia page on conjugate priors for a thorough list.

Stepping Back for a Moment

- The conditional structure of Bayesian models will allow us to approximate lengthy integrals (aka, calculus) with (nested) simulations (Monte Carlo to come).
    - The complexity of what we have to simulate depends on the complexity of the distributions used in our models
- Monte Carlo methods can address difficult or intractable analysis
    - Not a one-size-fit-all solution.
    - Sometimes a little/lot of pen and paper is worth years of compute.

Midterm Results





$\bar{X}_{209} = 75.0$
$Median_{209} = 77.1$
$S_{209} = 13.6$

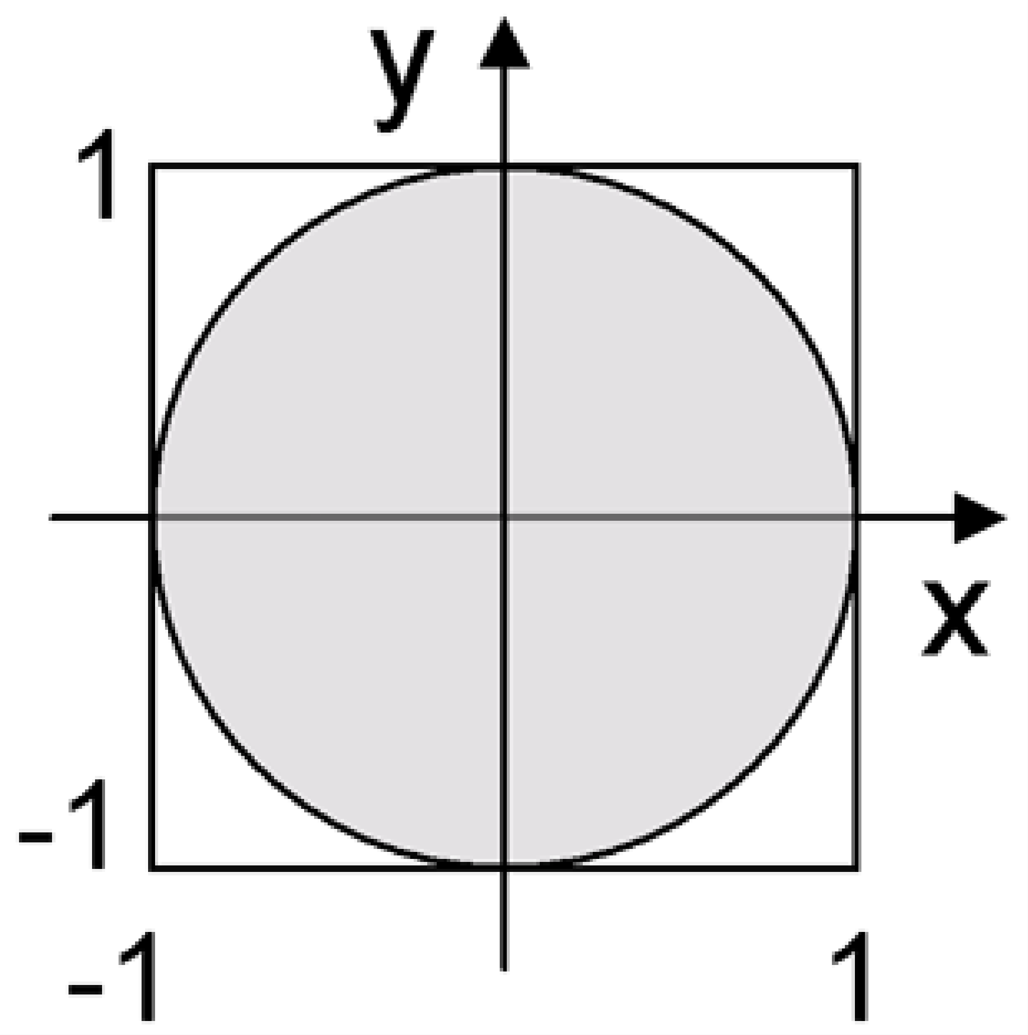
$\bar{X}_{209} = 77.6$
$Median_{209} = 80.1$
$S_{209} = 11.1$

Lecture Outline

- Catching our Breathe
- **Rejection Sampling**
- MCMC
- Metropolis-Hastings
- Dealing with Missingness
    - Types of Missingness
- Imputation Methods

Rejection Sampling

Let’s throw some darts!



Rejection Sampling

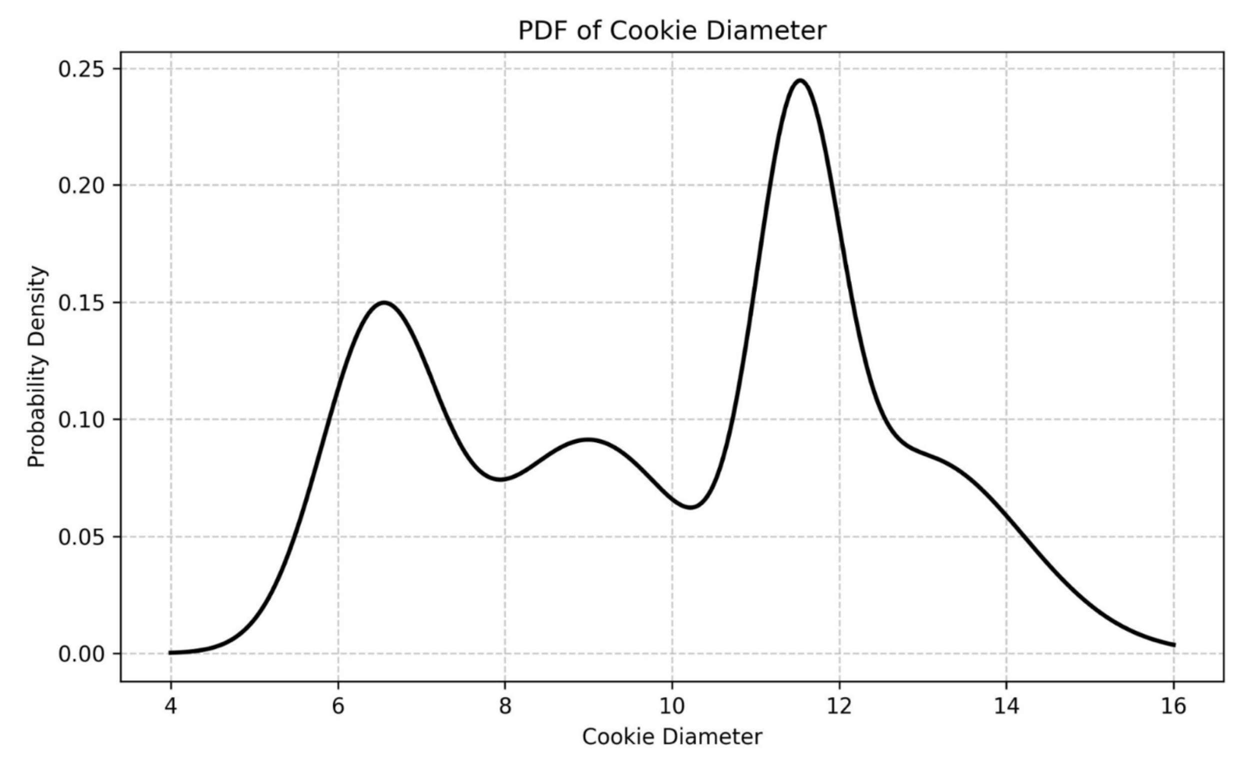
1. **Define a Target Distribution $f(x)$.** Specify the probability distribution you want to sample from.
2. **Choose a Proposal Distribution $g(x)$**: Select a simpler distribution from which you can easily sample, ensuring $f(x) \leq M \cdot g(x)$ for all x, where $M > 1$ is a scaling factor.
3. **Draw a Candidate Sample/Observation:** Generate a randomly sampled observation x from the proposal distribution $g(x)$**.**
4. **Generate a Uniform Random Number:** Draw $u \sim Uniform(0,1)$.
5. **Acceptance Condition:**
   - Compute the acceptance ratio $r = \frac{f(x)}{M \cdot g(x)}$.
   - Accept the candidate sample x if $u \leq r$; otherwise, reject it.
6. **Repeat Until Desired Sample Size:** Repeat steps 3–5 until you collect enough accepted observations.

Cookie Monster Needs Our Help!

Cookie Monster stumbled upon a famous bakery that **sells four varieties of cookies: chocolate chip, oatmeal raisin, peanut butter, and sugar cookies.**

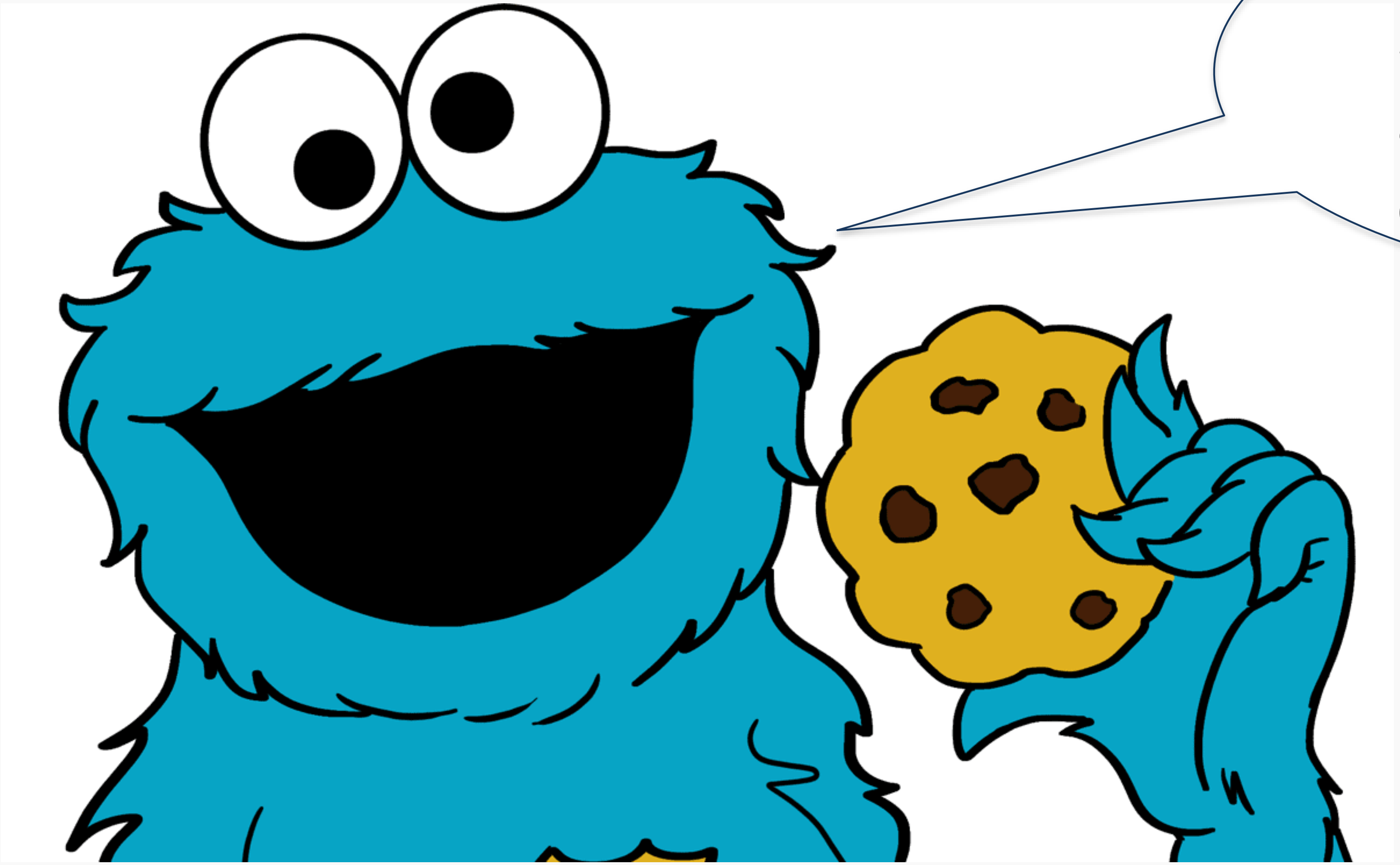
- Each variety has a diameter that varies slightly due to the baking process and the ingredients.
- The Cookie Monster is interested in sampling from the PDF of cookie diameters, but how?

Only Information Available: Cookie Diameter PDF



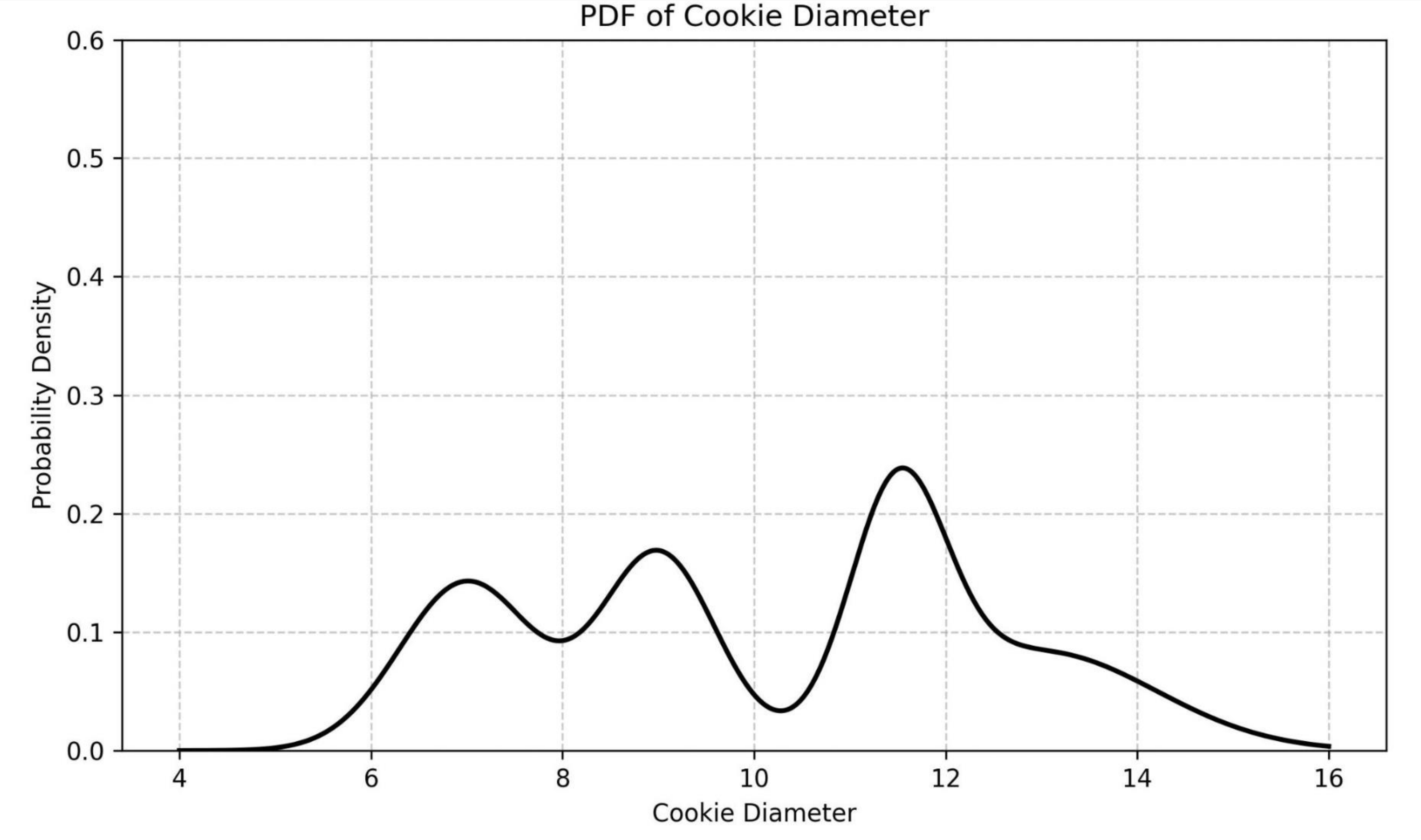
How would you describe this distribution? What is this suggestive of?

Cookie Monster is Wicked Smaht!

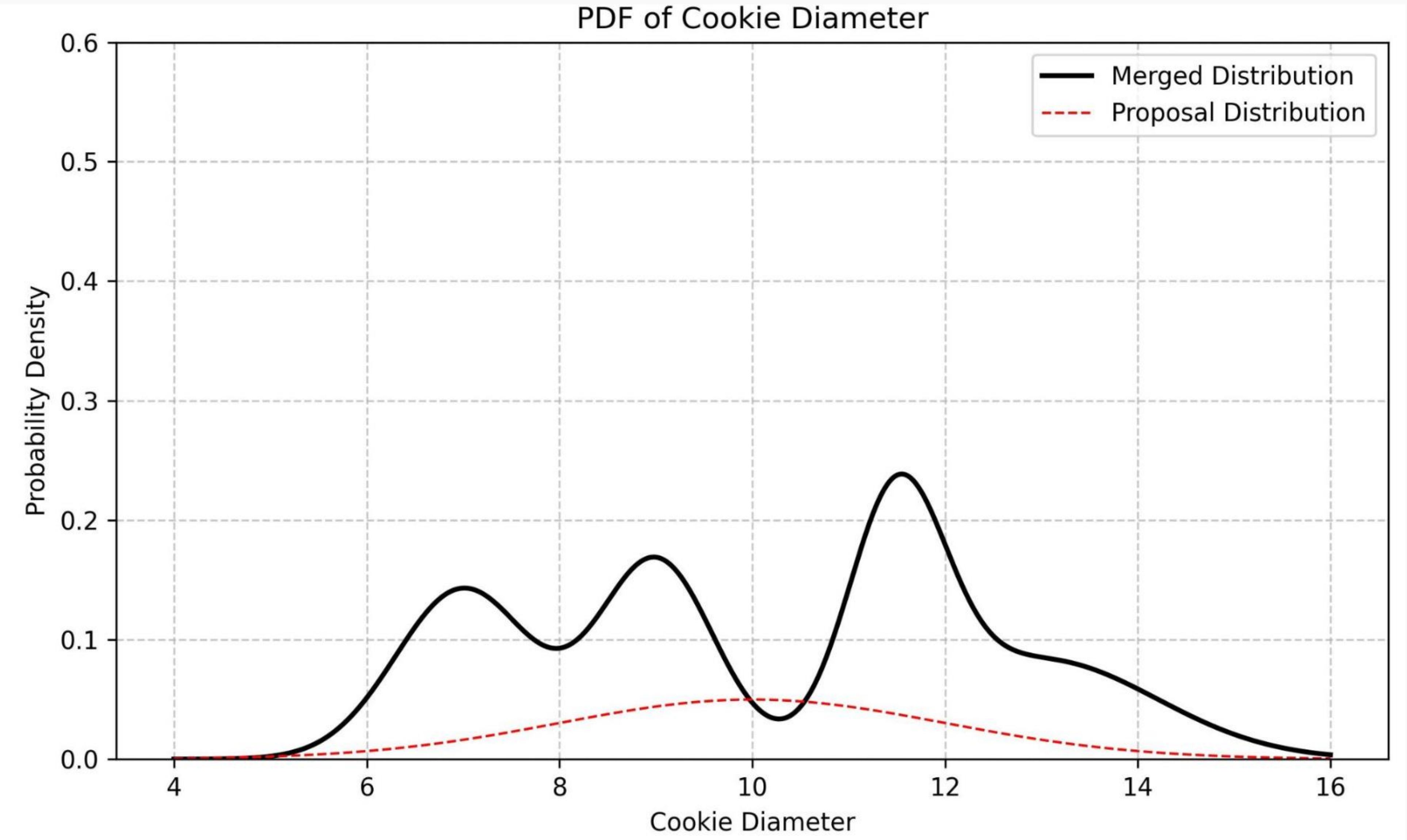


Can we use rejection sampling to generate a sample from this distribution?

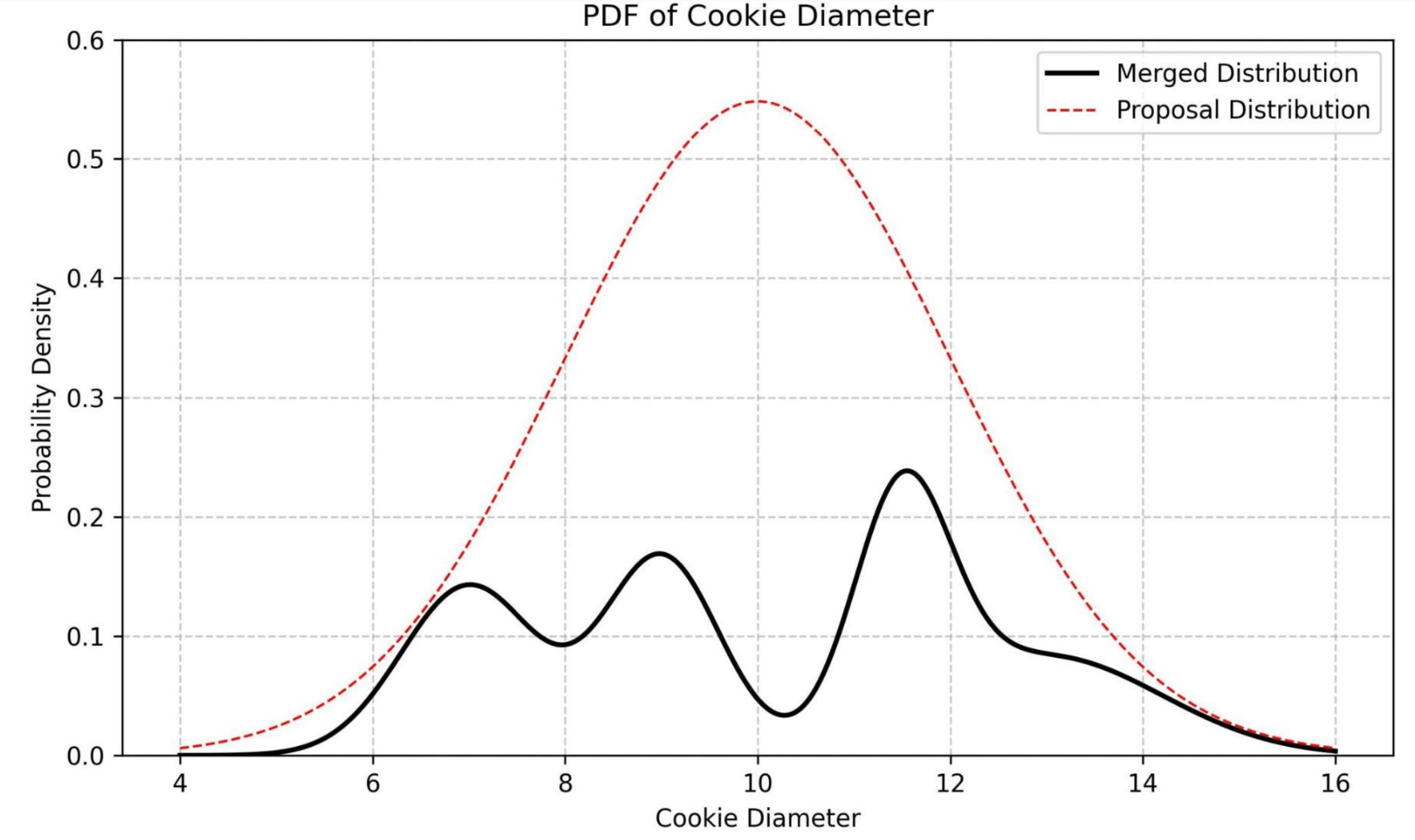
Step 1: Know your Target Distribution



Step 2A: Proposal Distribution



Step 2B: Set Proposal Distribution ($M = 11$)



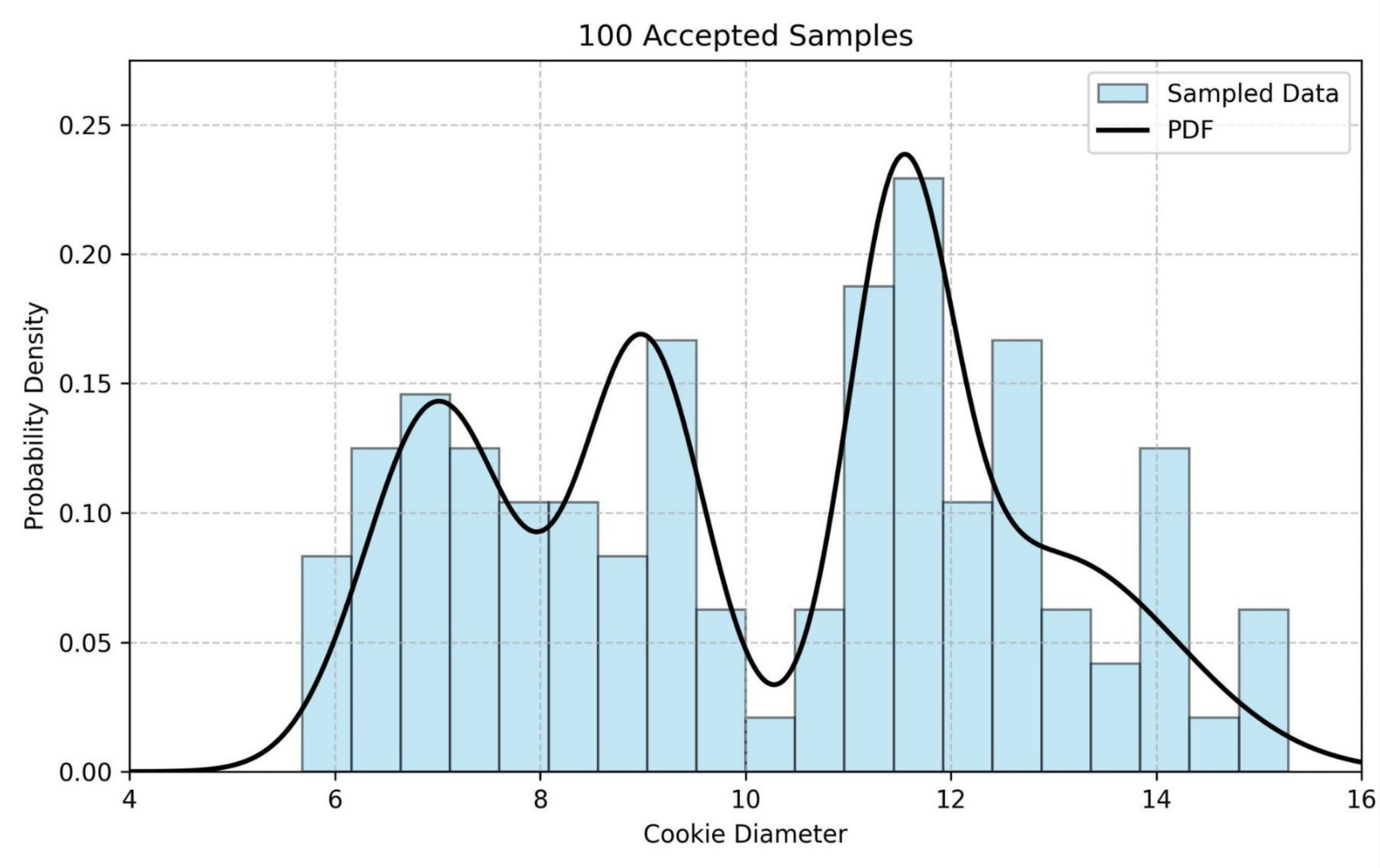
Steps 3-5: Draw x, u and compare

Rejection Sampling

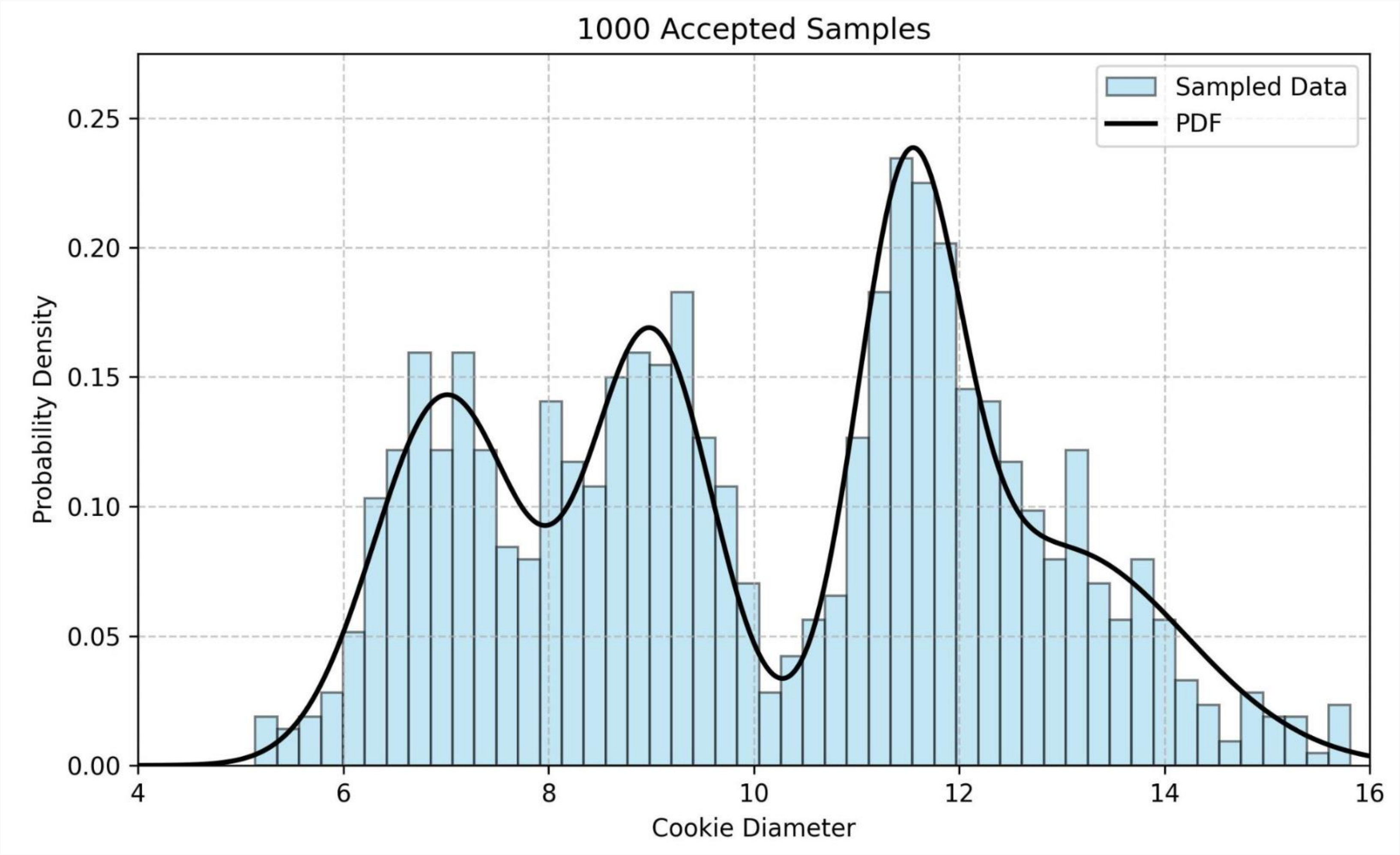
Now, we repeat the rest of the steps until we get a sample of some desired size:

3. **Draw a Candidate Sample /Observation:** Generate a randomly sampled observation x from the proposal distribution $g(x)$**.**
4. **Generate a Uniform Random Number:** Draw $u \sim Uniform(0,1)$.
5. **Acceptance Condition:**
    - Compute the acceptance ratio $r = \frac{f(x)}{M \cdot g(x)}$.
    - Accept the candidate sample x if $u \leq r$; otherwise, reject it.
6. **Repeat Until Desired Sample Size:** Repeat steps 3–5 until you collect enough accepted observations

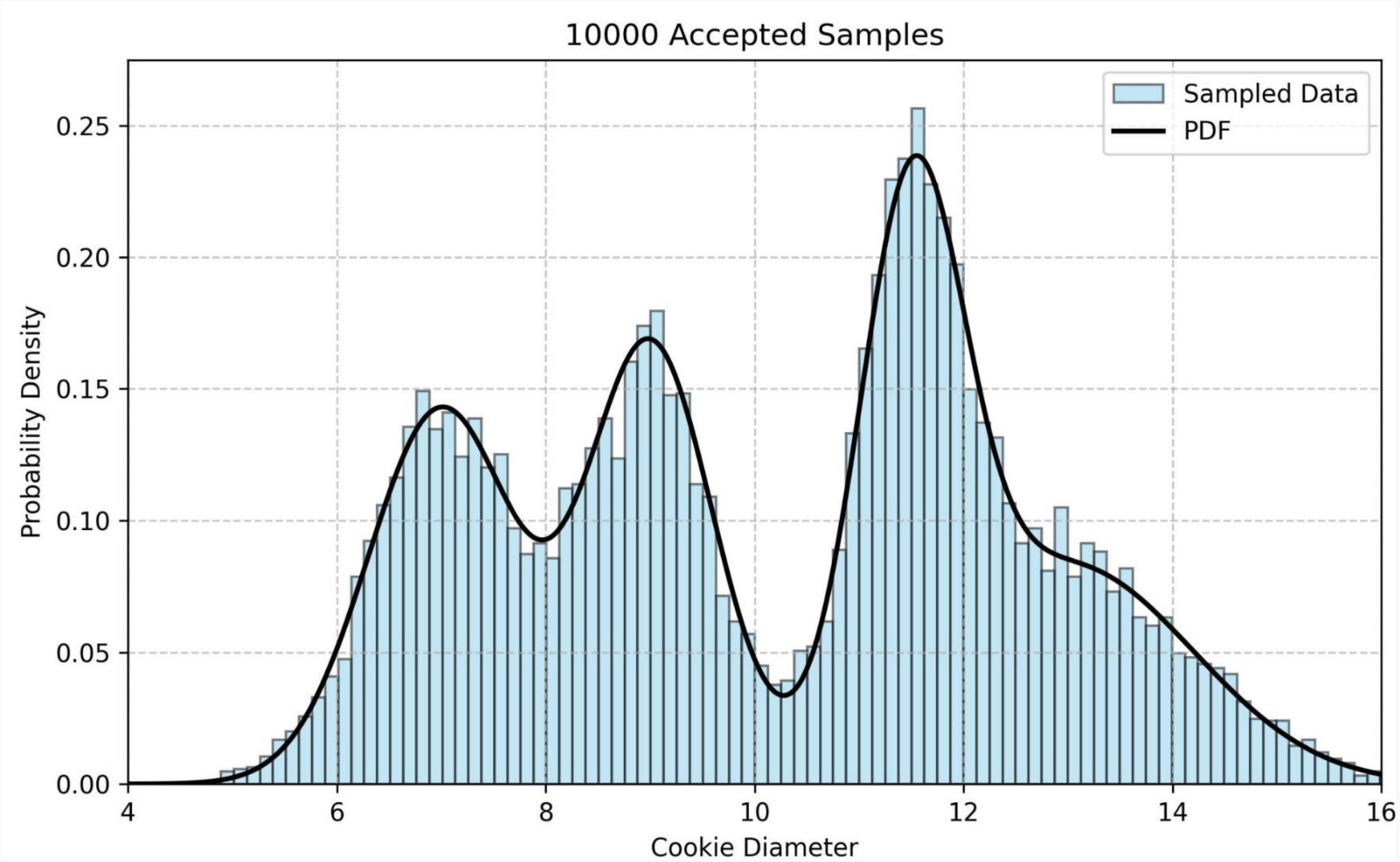
After 100 Accepted Samples…



After 1,000 Accepted Samples…



After 10,000 Accepted Samples…



Cookie Diameter: 1,000 Sample Results

True Expected Value:	$\mu = 10.125$
Estimated Expected Value:	$\hat{\mu} = 10.226$
True Variance:	$\sigma^2 = 5.932$
Estimated Variance:	$\hat{\sigma}^2 = 5.943$

Cookie Diameter: 10,000 Sample Results

True Expected Value:	$\mu = 10.125$
Estimated Expected Value:	$\hat{\mu} = 10.167$
True Variance:	$\sigma^2 = 5.932$
Estimated Variance:	$\hat{\sigma}^2 = 6.004$

Cookie Diameter: 100,000 Sample Results [~6 Seconds Runtime]

True Expected Value:	$\mu = 10.125$
Estimated Expected Value:	$\hat{\mu} = 10.130$
True Variance:	$\sigma^2 = 5.932$
Estimated Variance:	$\hat{\sigma}^2 = 5.944$

Importance Sampling: an aside

However, the previous algorithm is sometimes **inefficient**; instead of sampling from the distribution, we can use **importance sampling**:

1. Choose a proposal distribution $q(x)$ from which it is easy to sample.
2. Draw samples/observations$x_1, x_2, \dots, x_n$ from $q(x)$.
3. Assign a weight to each sample based on the ratio of the target distribution $p(x)$ to the proposal distribution $q(x)$:

$$w(x_i) = \frac{p(x_i)}{q(x_i)}$$

4. Use the weighted samples to compute the desired expectation of the original distribution (or integral). For example:

$$E_p[h(x)] = \int h(x)p(x)dx = \int h(x)w(x)q(x)dx \approx \sum_{i=1}^{n} h(x_i)w(x_i)$$

Lecture Outline

- Catching our Breathe
- Rejection Sampling
- **MCMC**
- Metropolis-Hastings
- Dealing with Missingness
    - Types of Missingness
- Imputation Methods

Limitation of Rejection (& Importance) Sampling

- The main problem with rejection sampling is **computational efficiency**:
  - It could take *ages* in rejection sampling before a sample is accepted.
  - It could take many, many samples in importance sampling before reweighted samples are sufficiently reliable to calculate posterior statistics.
- With both methods, the choice of proposal distribution makes a massive difference.

An Alternative Approach: MCMC!

- **Markov Chain Monte Carlo** is another method we can use.
  - **Markov Chain**: a memoryless random process.
  - **Monte Carlo**: estimates based on randomly generated samples.
    - We've been doing this already in rejection sampling and bootstrapping!

- **Main Idea**: Construct a random process which, as it evolves, has outputs that start resembling samples from the posterior distribution.

What is a Markov Chain?

- A Markov Chain is an algorithm that evolves in discrete times steps.
- $\theta^{(t)} \subset \Omega$ denotes state of algorithm at step t.
    - State space Ω is a set of all possible states for the algorithm.
- We denote the evolution of the algorithm using $\theta^{(0)}, \theta^{(1)}, \dots, \theta^{(t)}$
- To denote the probability density of $\theta^{(t)}$ given the history of the process, we use:

$$p\left(\theta^{(t)} | \theta^{(t-1)}, \dots, \theta^{(0)}\right)$$

The Markov-ian Property

- A central property of Markov Chains is that updates are random but memoryless:

$$p\left(\theta^{(t)}|\theta^{(t-1)},\ldots,\theta^{(0)}\right)=p\left(\theta^{(t)}|\theta^{(t-1)}\right)$$

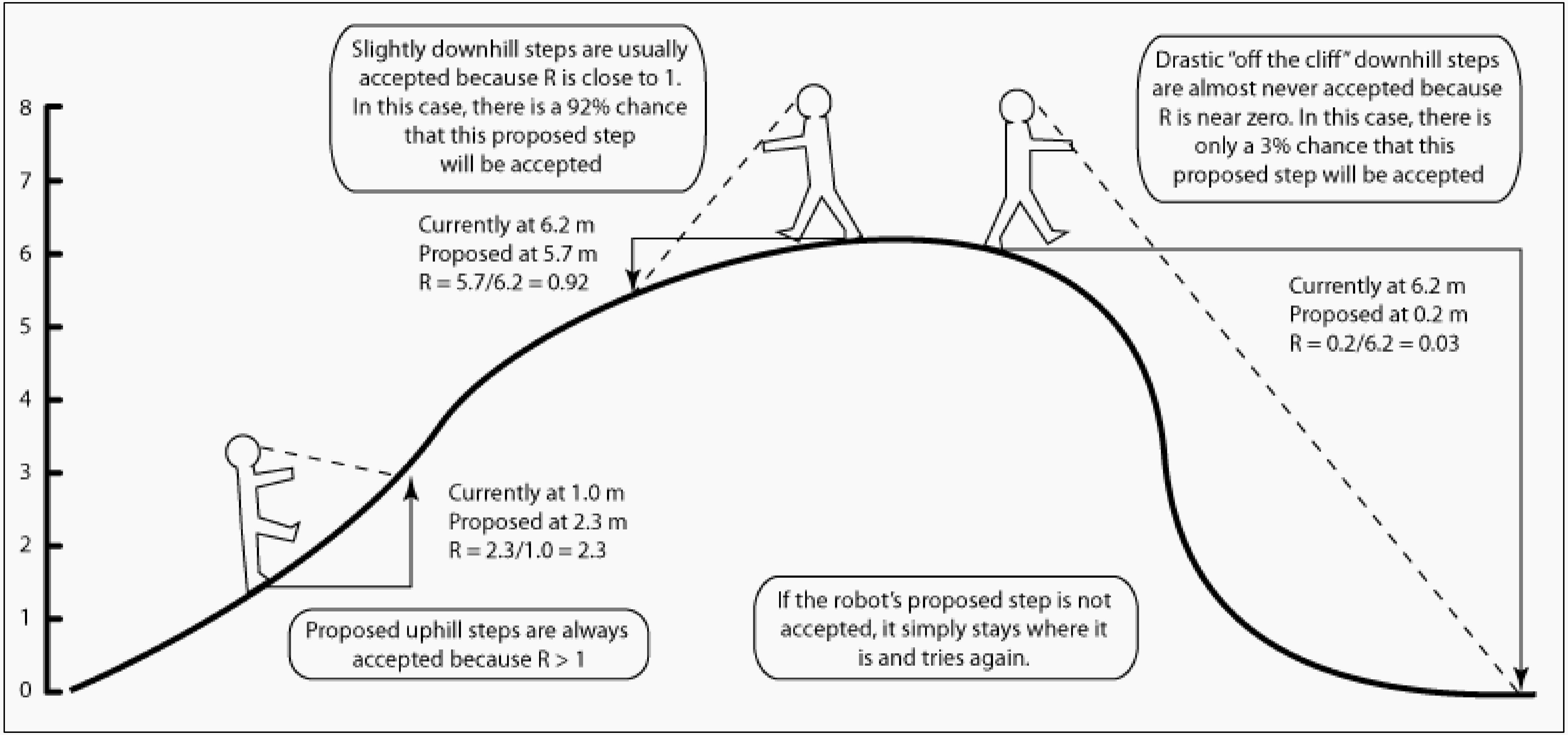
  - It doesn’t matter where I’ve been in the past.
  - It only matters where I am currently at to predict the next step.
- When does this property hold in real life in a ***random process***?

- **So how do we use this idea to sample from a posterior distribution?**

Lecture Outline

- Catching our Breathe
- Rejection Sampling
- MCMC
- **Metropolis-Hastings**
- Dealing with Missingness
  - Types of Missingness
- Imputation Methods

Metropolis-Hastings: a random walk through the distribution



8
7
6
5
4
3
2
1
0
Slightly downhill steps are usually accepted because R is close to 1. In this case, there is a 92% chance that this proposed step will be accepted
Drastic "off the cliff" downhill steps are almost never accepted because R is near zero. In this case, there is only a 3% chance that this proposed step will be accepted
Currently at 6.2 m
Proposed at 5.7 m
R = 5.7/6.2 = 0.92
Currently at 6.2 m
Proposed at 0.2 m
R = 0.2/6.2 = 0.03
Currently at 1.0 m
Proposed at 2.3 m
R = 2.3/1.0 = 2.3
Proposed uphill steps are always accepted because R > 1
If the robot's proposed step is not accepted, it simply stays where it is and tries again.

Metropolis-Hastings Algorithm

- Let $q(y|x)$ denote a chosen proposal distribution (we pick this).
  - $q(y|x)$ gives density for moving to y from x
- Let $p(\theta|data) = c \cdot f(\theta)$ be our target limiting distribution.

Algorithm (One Step): Currently at $\theta^{(t)}$

- Generate proposal: $\theta^* \sim q(y|\theta^{(t)})$
- Compute acceptance probability:

$$a = \min\left\{1, \frac{f(\theta^*)q(\theta^{(t)}|\theta^*)}{f(\theta^{(T)})q(\theta^*|\theta^{(t)})}\right\}$$

- With probability a set $\theta^{(t+1)} = \theta^*$. Otherwise, set $\theta^{(t+1)} = \theta^{(t)}$

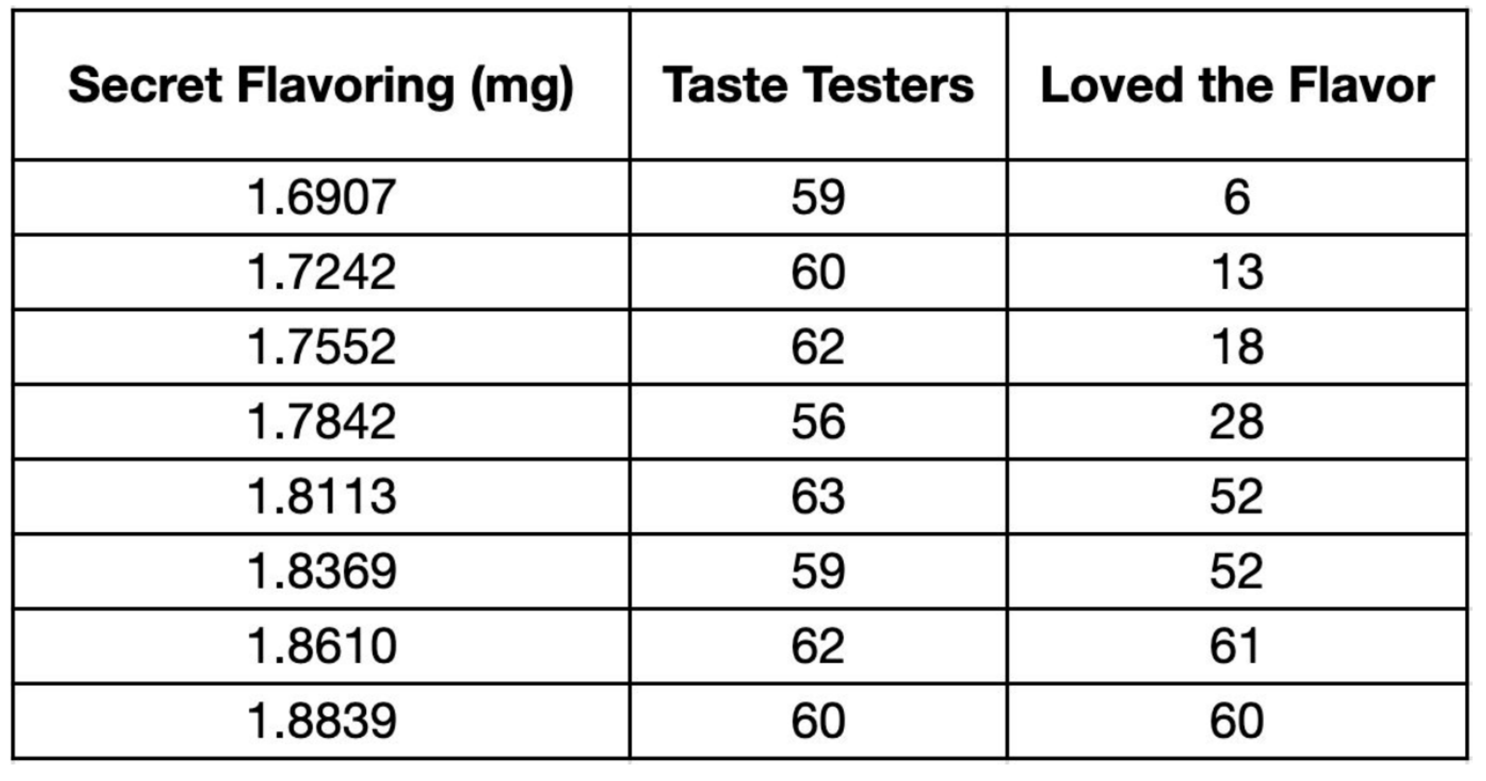
Interpreting Metropolis Hastings

$$a = \min\left\{1, \frac{f(\theta^*)q(\theta^{(t)}|\theta^*)}{f(\theta^{(T)})q(\theta^*|\theta^{(t)})}\right\}$$

- Uphill proposals (ones that take the Markov Chain to a local maximum) are always accepted.
- Downhill proposals (ones that move away from a local maximum) are accepted with probability equal to the relative heights of the posterior density at the proposed and current values.

Putting Things Into Practice: Skittles

The company behind Skittles is contemplating a new flavor: mango. They need to figure out how much of their secret flavoring to add to each skittle to maximize the number of people who enjoyed the new flavor. The company shared the following taste test data:



Secret Flavoring (mg)	Taste Testers	Loved the Flavor
1.6907	59	6
1.7242	60	13
1.7552	62	18
1.7842	56	28
1.8113	63	52
1.8369	59	52
1.8610	62	61
1.8839	60	60

Skittles Modeling

- What is the response variable? What is the predictor? What sort of model should we use?
- For each of the 8 observations: $Y_i \sim \text{Binom}(n_i, p_i)$
- What is the PMF (the likelihood)?

$$P(Y_i = y_i) = \binom{n_i}{y_i} p_i^{y_i}(1-p_i)^{n_i-y_i}$$

- We will use a logistic model: $\ln\left(\frac{p_i}{1-p_i}\right) = \alpha + \beta x_i$
  - Recall: this is equivalent to $p_i = \frac{e^{\alpha+\beta x_i}}{1+e^{\alpha+\beta x_i}}$
- The goal is to perform inference for the two parameters, α and β.
- Let’s take a Bayesian approach!

Priors become Posteriors

- We choose proper priors for both parameters of interest:

$$\alpha \sim N(0,100^2)$$
$$\beta \sim N(0,100^2)$$

- This essentially acts like a uniform distribution for each parameter because the variances are so large.

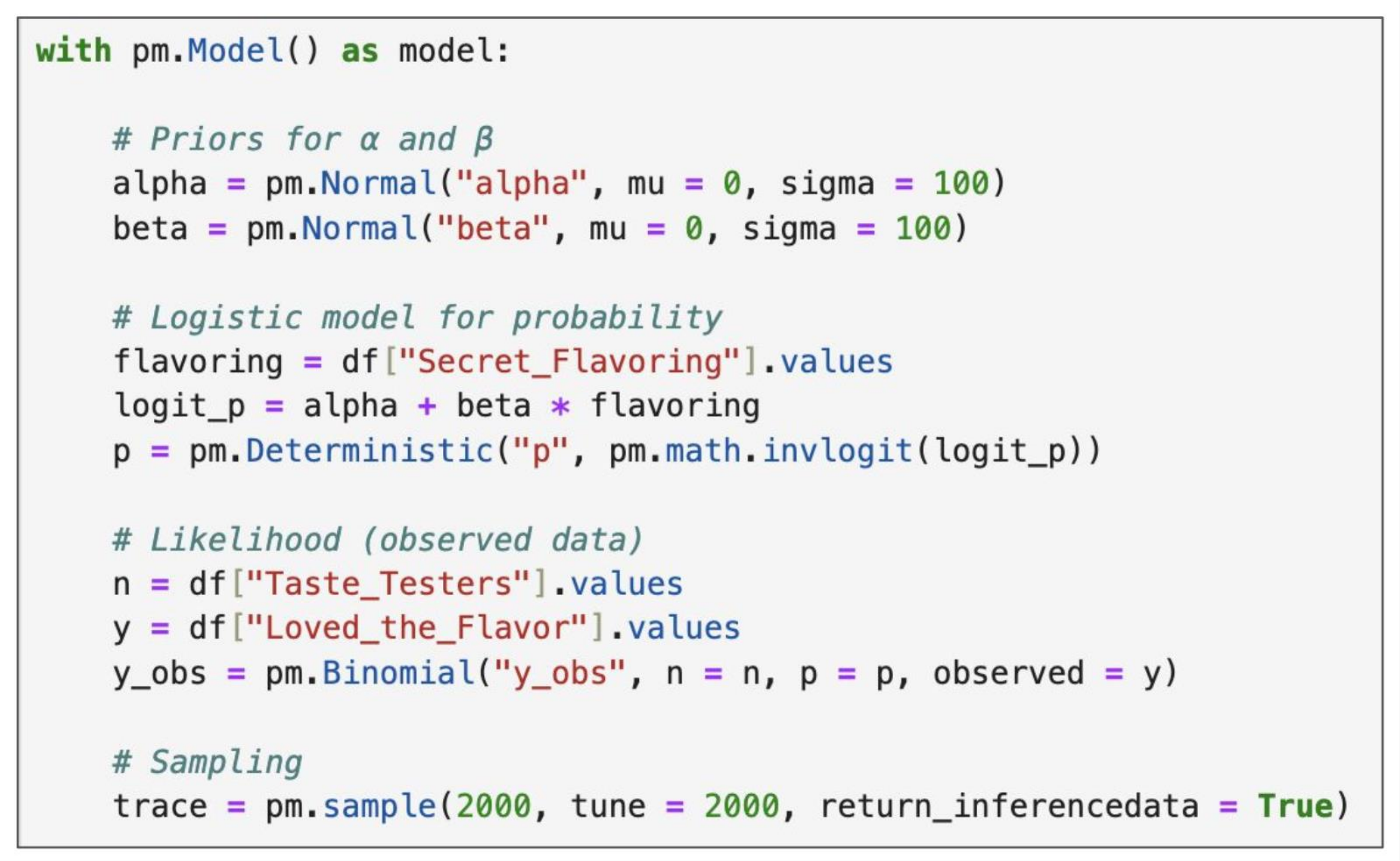
- By Bayes rule, the posterior distribution can be written as:

$$p(\alpha,\beta|y) \propto e^{-\left(\frac{\alpha^2}{2\cdot 100^2}\right)} \cdot e^{-\left(\frac{\beta^2}{2\cdot 100^2}\right)} \cdot \prod_{i=1}^{8} p_i^{y_i}(1-p_i)^{n_i-y_i}$$

Using pymc for Modeling

- The pymc library allows us to abstract away from the details of implementing MCMC and the Metropolis-Hastings algorithm by hand:
- It runs several parallel MCMC samplers with different starting values.
- It simulates values from the Markov chains for a “burn-in” period (before the Markov chains have converged to the stationary distribution), and discard the burn-in simulations.
- It saves simulated values after the burn-in period. These will be the simulated values on which we can perform inferential summaries.

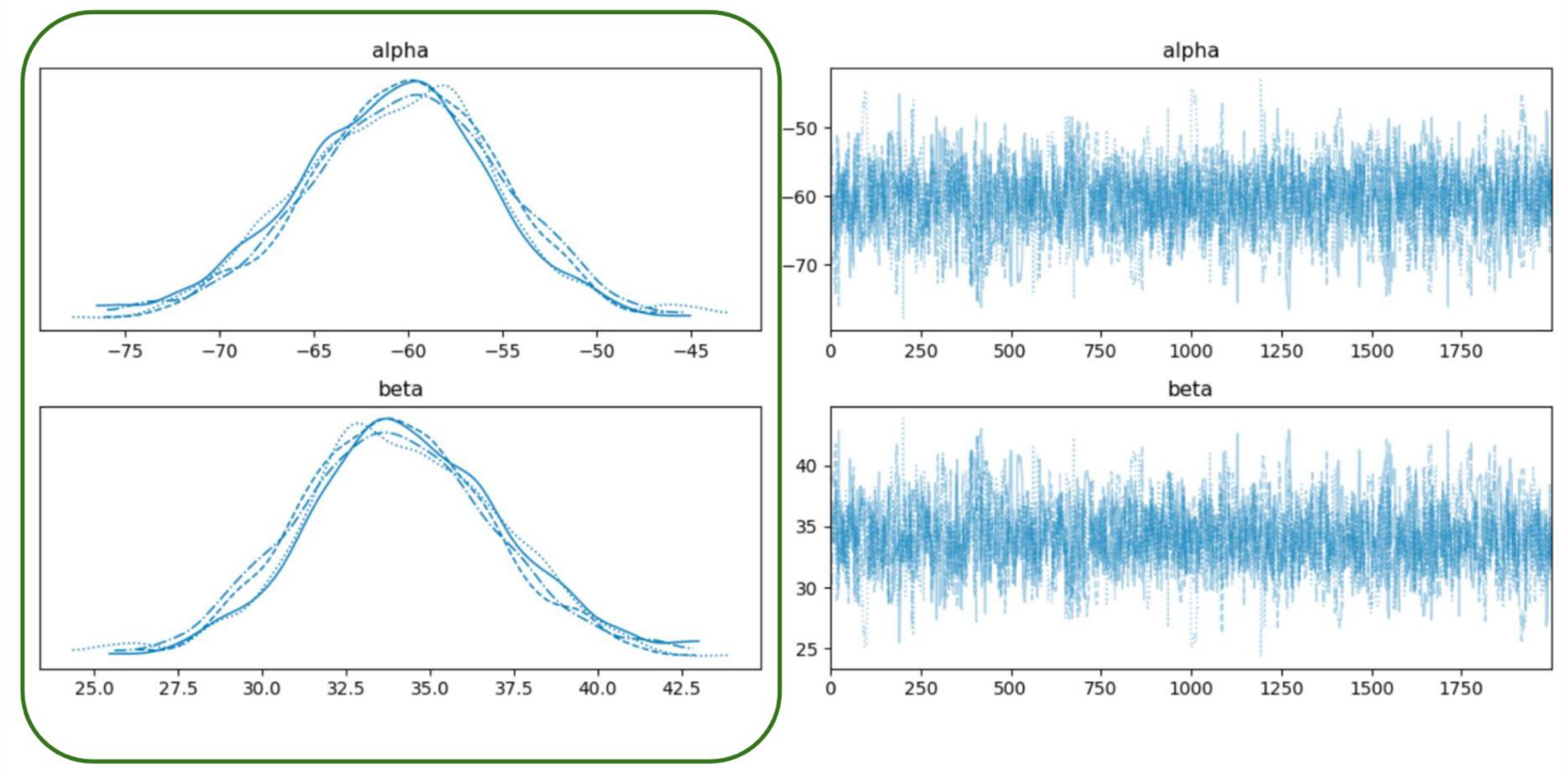
Using pymc for Modeling (cont.)



```python
with pm.Model() as model:

    # Priors for α and β
    alpha = pm.Normal("alpha", mu = 0, sigma = 100)
    beta = pm.Normal("beta", mu = 0, sigma = 100)

    # Logistic model for probability
    flavoring = df["Secret_Flavoring"].values
    logit_p = alpha + beta * flavoring
    p = pm.Deterministic("p", pm.math.invlogit(logit_p))

    # Likelihood (observed data)
    n = df["Taste_Testers"].values
    y = df["Loved_the_Flavor"].values
    y_obs = pm.Binomial("y_obs", n = n, p = p, observed = y)

    # Sampling
    trace = pm.sample(2000, tune = 2000, return_inferencedata = True)
```

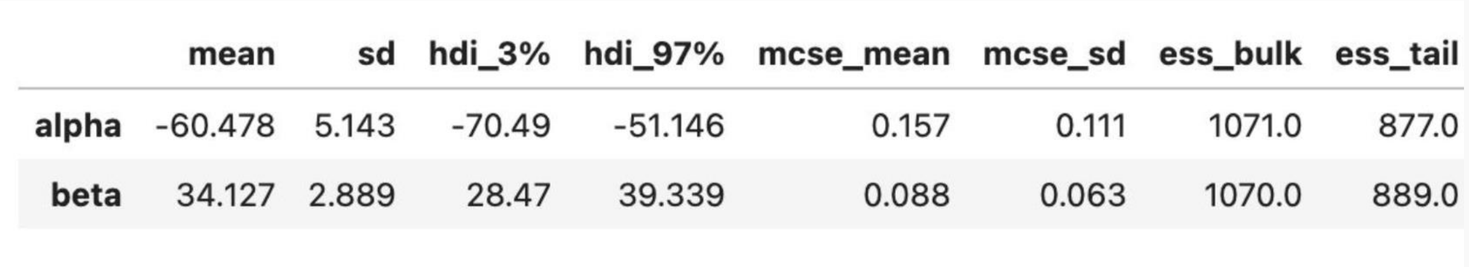
Interpreting pymc Results

The pymc library gives us sampled posterior distributions for the parameters (this is from pymc.sample):



Interpreting pymc Results

- We are also provided with some summary information on the stability of the sampling (pymc.summary):

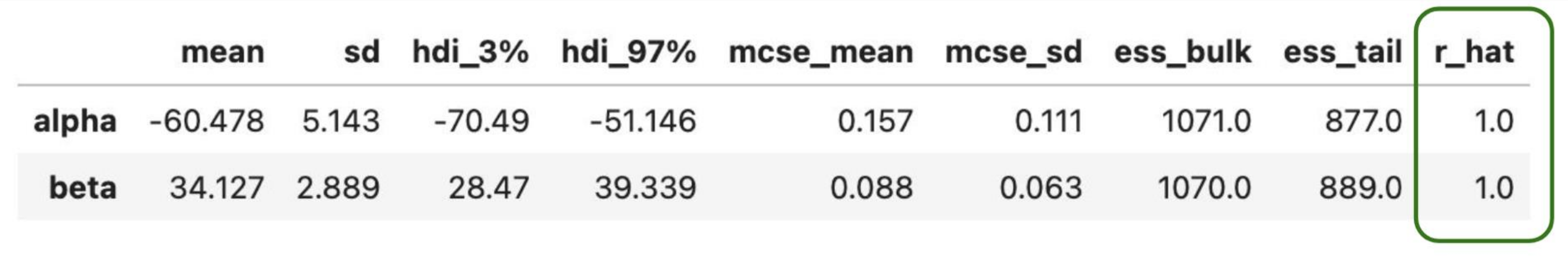


	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail
alpha	-60.478	5.143	-70.49	-51.146	0.157	0.111	1071.0	877.0
beta	34.127	2.889	28.47	39.339	0.088	0.063	1070.0	889.0

- Provides the Following:
  - Posterior mean and posterior standard deviation
  - 94% Credible Interval
  - Effective sample size (of the middle 90% of the distribution and of the 10% tails of the distribution).
  - sd/(effective sample size) & sd/(#draws)

Interpreting pymc Results

- We are also provided with some summary information on the stability of the sampling:



	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	-60.478	5.143	-70.49	-51.146	0.157	0.111	1071.0	877.0	1.0
beta	34.127	2.889	28.47	39.339	0.088	0.063	1070.0	889.0	1.0

- The r_hat statistic is a numerical measure that indicates whether the Markov chain was run long enough to reach convergence.
  - Values near 1.0 indicate convergence.
  - Large values (say 1.3 or greater) indicate either that the procedure has not been run long enough, or that the parameter itself may be difficult to obtain reasonable samples given strong autocorrelation in the sampler.
  - The statistic essentially computes a ratio of between-chain variance and within-chain variance.

Taste the Rainbow!

Lecture Outline

- Catching our Breathe
- Rejection Sampling
- MCMC
- Metropolis-Hastings
- **Dealing with Missingness**
  - Types of Missingness
- Imputation Methods

What is missing data?

Often times when data is collected, there are some missing values apparent in the dataset. This leads to a few questions to consider:

- How does this show up in pandas?
- How does sklearn handle these NaNs?
- How does this effect our modeling?

What are the simplest ways to handle missing data?

Naively handling missingness

- Drop the observations that have any missing values.
  - Use pd.DataFrame.dropna(axis=0)
- Impute the mean/median (if quantitative) or most common class (if categorical) for all missing values.
  - Use pd.DataFrame.fillna(value=x.mean())

How do statsmodels and sklearn handle these NaNs?

What are some consequences in handling missingness in these fashions?

Missingness Indicator Variable

One simple way to handle missingness in a variable, X_j, is to impute a value (like 0 or $\bar{X}_j$), then create a new variable, $X_{j,miss}$, that indicates this observation had a missing value. If X_j is categorical then just impute 0.

Then include both $X_{j,miss}$ and X_j as predictors in any model.

Illustration is to the right.

X_1	X_2	X_1^*	X_2^*	$X_{1,miss}$	$X_{2,miss}$
10	.	10	0	0	1
5	1	5	1	0	0
21	0	21	0	0	0
15	0	15	0	0	0
16	.	16	0	0	1
.	.	0	0	1	1
21	1	21	1	0	0
12	0	12	0	0	0
.	1	0	1	1	0

Why use a Missingness Indicator Variable?

How does this missingness indicator variable improve the model?

Because the group of individuals with a missing entry may be systematically different than those with that variable measured. Treating them equivalently could lead to bias in quantifying relationships (the β’s) and underperform in prediction.

For example: imagine a survey that asks whether or not someone has ever recreationally used opioids, and some people chose not to respond. Does the fact that they did not respond provide extra information? Should we treat them equivalently as never-users?

This approach essentially creates a third group for this predictor: the “did not respond” group.

Lecture Outline

- Catching our Breathe
- Rejection Sampling
- MCMC
- Metropolis-Hastings
- Dealing with Missingness
    - **Types of Missingness**
- Imputation Methods

Sources of Missingness

Missing data can arise from various places in data:

- A survey was conducted and values were just randomly missed when being entered in the computer.
- A respondent chooses not to respond to a question like `Have you ever recreationally used opioids?'.
- You decide to start collecting a new variable (due to new actions: like a pandemic) partway through the data collection of a study.
- You want to measure the speed of meteors, and some observations are just 'too quick' to be measured properly.

The source of missing values in data can lead to the major types of missingness:

Types of Missingness

There are 3 major types of missingness to be concerned about:

1. **Missing Completely at Random (MCAR)** - the probability of missingness in a variable is the same for all units. Like randomly poking holes in a data set.
2. **Missing at Random (MAR)** - the probability of missingness in a variable depends only on available information (in other predictors).
3. **Missing Not at Random (MNAR)** - the probability of missingness depends on information that has not been recorded and this information also predicts the missing values.

What are examples of each these 3 types?

Missing completely at random (MCAR)

Missing Completely at Random is the best case scenario, and the easiest to handle:

- Examples: a coin is flipped to determine whether an entry is removed. Or when values were just randomly missed when being entered in the computer.
- Effect if you ignore: there is no effect on inferences (estimates of β).
- How to handle: lots of options, but best to impute (more on next slide).

Missing at random (MAR)

Missing at Random is still a case that can be handled.

- Example(s): men and women respond to the question "have you ever felt harassed at work?" at different rates (and may be harassed at different rates).
- Effect if you ignore: inferences are biased (estimates of β's) and predictions are usually worsened.
- How to handle: use the information in the other predictors to build a model and **impute** a value for the missing entry.

Key: we can fix any biases by modeling and imputing the missing values based on what is observed!

Missing Not at Random (MNAR)

Missing Not at Random is the worst case scenario, and impossible to handle properly:

- Example(s): patients drop out of a study because they experience some really bad side effect that was not measured. Or cheaters are less likely to respond when asked if you've ever cheated.
- Effect if you ignore: there are major effects on inferences (estimates of beta) or predictions.
- How to handle: you can 'improve' things by dealing with it like it is MAR, but you [likely] may never completely fix the bias. And incorporating a **missingness indicator variable** may actually be the best approach (if it is in a predictor).

What type of missingness is present?

Can you ever tell based on your data what type of missingness is actually present?

Since we asked the question, the answer must be no.

It generally cannot be determined whether data really are missing at random, or whether the missingness depends on unobserved predictors or the missing data themselves. The problem is that these potential “lurking variables” are unobserved (by definition) and so can never be completely ruled out.

In practice, a model with as many predictors as possible is used so that the `missing at random' assumption is reasonable.

Lecture Outline

- Catching our Breathe
- Rejection Sampling
- MCMC
- Metropolis-Hastings
- Dealing with Missingness
  - Types of Missingness
- **Imputation Methods**

Handling Missing Data

When encountering missing data, the approach to handling it depends on…

1. …whether the missing values are in the response or in the predictors. Generally speaking, it is much easier to handle missingness in predictors.
2. …whether the variable is quantitative or categorical.
3. …how much missingness is present in the variable. If there is too much missingness, you may be doing more harm than good.

Generally speaking, it is a good idea to attempt to **impute** (or 'fill in') entries for missing values in a variable (assuming your method of imputation is a good one).

Imputation Methods

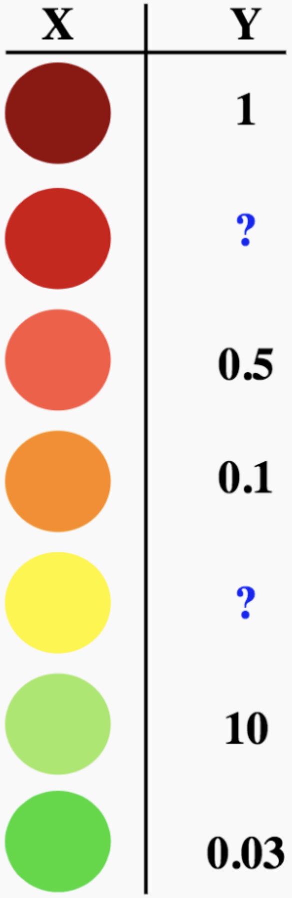
There are several different approaches to imputing missing values:

1. **Impute the mean or median** (quantitative) or most common class (categorical) for all missing values in a variable.
2. Create a new variable that is an **indicator of missingness**, and include it in any model to predict the response (also plug in zero or the mean in the actual variable).
3. **Hot deck imputation**: for each missing entry, randomly select an observed entry in the variable and plug it in.
4. **Model the imputation**: plug in predicted values ($\hat{y}$) from a model based on the other observed predictors.
5. **Model the imputation with uncertainty**: plug in predicted values plus randomness ($\hat{y} + \epsilon$) from a model based on the other observed predictors.

What are the advantages and disadvantages of each approach?

Schematic: imputation through modeling

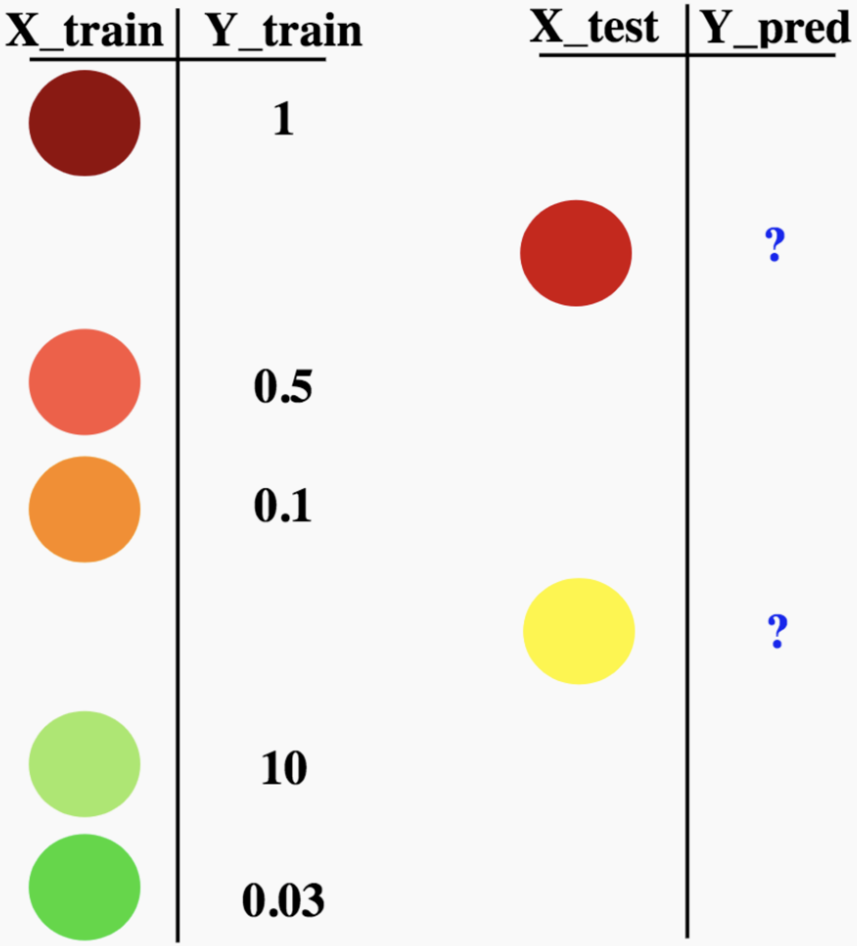
How do we use models to fill in missing data?



X	Y
🔴	1
🔴	?
🔴	0.5
🟠	0.1
🟡	?
🟢	10
🟢	0.03

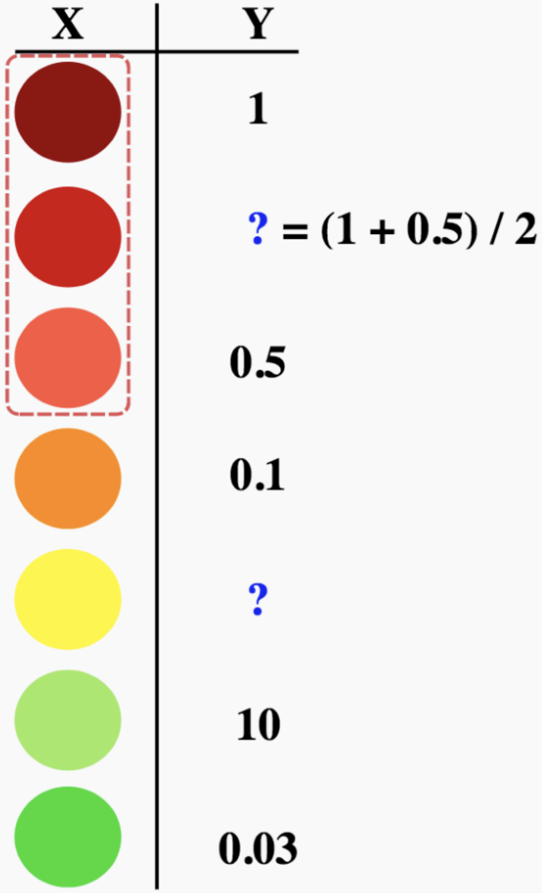
Schematic: imputation through modeling

How do we use models to fill in missing data?



Schematic: imputation through modeling

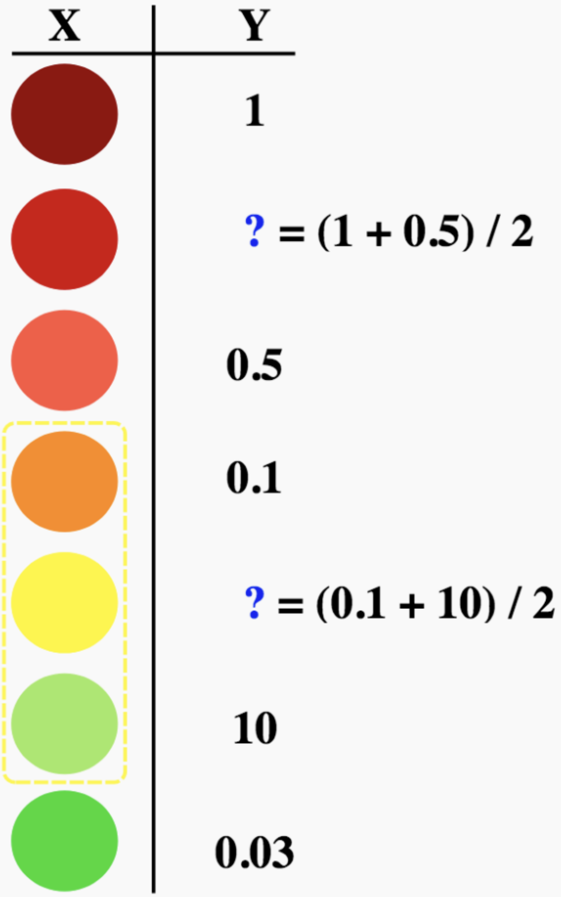
How do we use models to fill in missing data? Using *k*-NN for *k* = 2?



X	Y
●	1
●	? = (1 + 0.5) / 2
●	0.5
●	0.1
●	?
●	10
●	0.03

Schematic: imputation through modeling

How do we use models to fill in missing data? Using *k*-NN for *k* = 2?



X

Y

1

? = (1 + 0.5) / 2

0.5

0.1

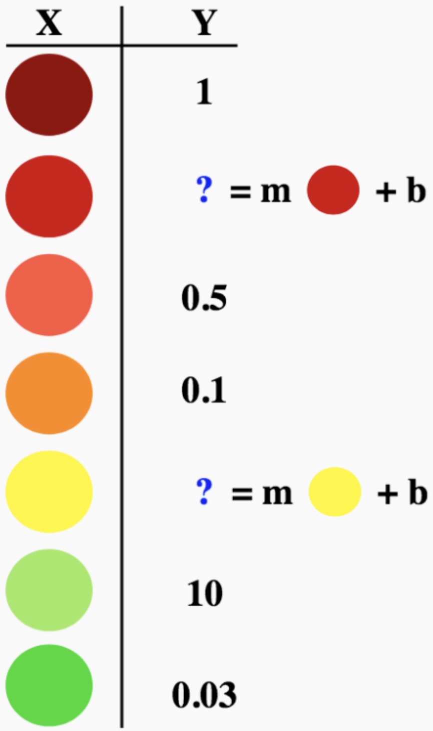
? = (0.1 + 10) / 2

10

0.03

Schematic: imputation through modeling

How do we use models to fill in missing data? Using linear regression?



X	Y
●	1
●	? = m ● + b
●	0.5
●	0.1
●	? = m ● + b
●	10
●	0.03

Where m and b are computed from the observations (rows) that do not have missingness (we should call them $b = \beta_0$ and $m = \beta_1$).

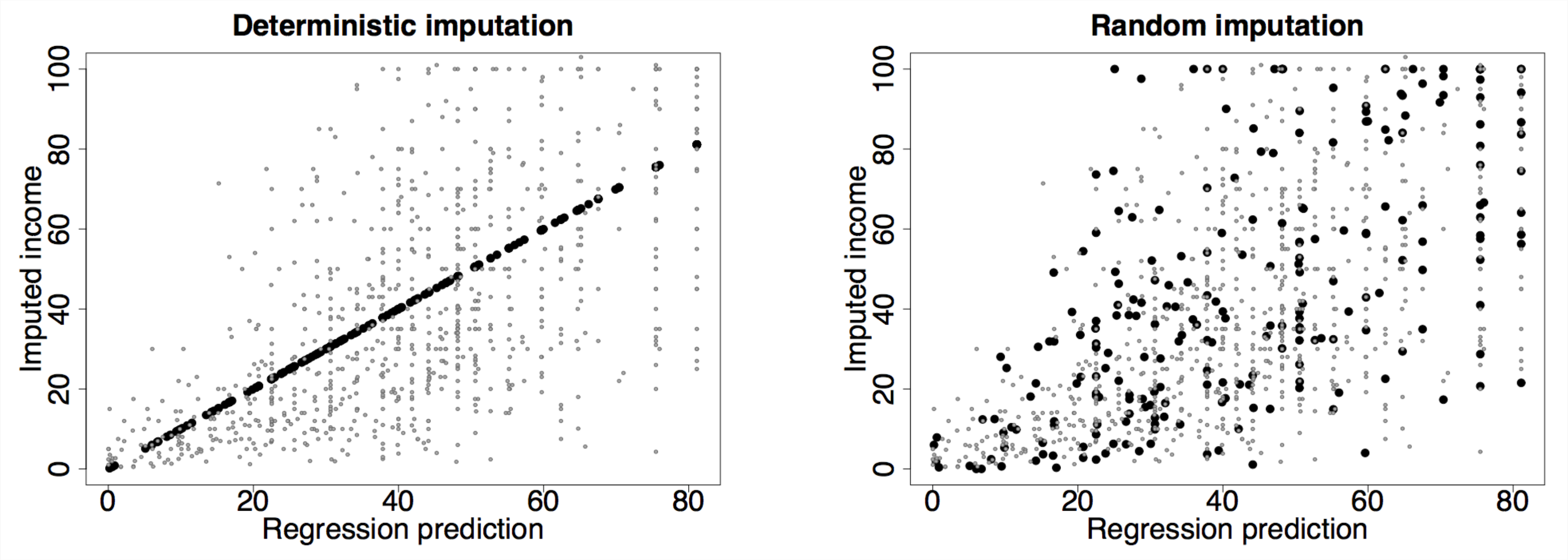
Imputation through modeling with uncertainty

The schematic in the last few slides ignores the fact of imputing with uncertainty. What happens if you ignore this fact and just use the ‘best’ model to impute values solely on $\hat{y}$?

The distribution of the imputed values will be too narrow and not represent real data (see next slide for illustration). The goal is to impute values that include the uncertainty of the model.

How can this be done in practice in k-NN? In linear regression? In logistic regression?

Imputation: modeling with uncertainty (an illustration)



Imputation: modeling with uncertainty (an algorithm)

Recall the probabilistic model in linear regression:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_P + \epsilon \quad \text{where } \epsilon \sim N(0, \sigma^2)$$

How can we take advantage of this model to impute with uncertainty?

It's a 3 step process:

1. Fit a model to predict the predictor variable with missingness from all the other predictors.
2. Predict the missing values from the model in the previous part.
3. Add in a measure of uncertainty to this prediction by randomly sampling from a $N(0, \sigma^2)$ distribution*, where σ^2 is the mean square error (MSE) from the model.

   *Instead of sampling from a $N(0, \sigma^2)$ distribution, you can bootstrap sample from the observed residuals!

Imputation: modeling with uncertainty (*k*-NN regression)

How can we use *k*-NN regression to impute values that mimic the error in our observations?

Two ways:

- Use $k = 1$.
- Use any other k, but randomly select from the nearest neighbors in $\mathcal{N}_0$. This can be done with equal probability or with some weighting (inverse to the distance measure used).

Imputation through modeling with uncertainty: classifiers

For classifiers, this imputation with uncertainty/randomness is a little easier process. How can it be implemented?

If a classification model (logistic, *k*-NN, etc…) is used to predict the variable with missingness on the observed predictors, then all you need to do is flip a ‘biased coin’ (or multi-sided die) with the probabilities of coming up for each class equal to the predicted probabilities from the model.

Warning: do not just classify blindly using the predict command in sklearn!

Imputation across multiple variables

If only one variable has missing entries, life is easy. But what if all the predictor variables have a little bit of missingness (with some observations having multiple entries missing)? How can we handle that?

It's an iterative process. Impute X_1 based on $X_2, ..., X_p$. Then impute X_2 based on X_1 and $X_3, ..., X_p$. And continue down the line.

Any issues? Yes, not all of the missing values may be imputed with just one ’run’ through the data set. So you will have to repeat these ’runs’ until you have a completely filled in data set.

Multiple imputation: beyond this class

What is an issue with treating your now ‘complete’ data set (a mixture of actually observed values and imputed values) as simply all observed values?

Any inferences or predictions carried out will be tuned and potentially overfit to the random entries imputed for the missing entries. How can we prevent this phenomenon?

By performing **multiple imputation**: rerun the imputation algorithm many times, refit the model on the response many times (one time each), and then ’average’ the predictions or estimates of β coefficients to perform inferences (also incorporating the uncertainty involved).

Note: this is beyond what we would expect in this class. But it generally a good thing to be aware of.

sklearn’s impute module

Want to do imputation in sklearn? Of course there are functions for that!

The 4 main functions in this module are:

1. **`sklearn.impute.SimpleImputer`**: perform mean, median, mode, or any specific value to impute
2. **`sklearn.impute.IterativeImputer`**: perform multivariate imputation that estimates each predictor from all the others (user-supplied model/estimator).
3. **`sklearn.impute.KNNImputer`**: perform imputation automatically from a *k*-NN model from all the other predictors.
4. **`sklearn.impute.MissingIndicator`**: to create binary indicator(s) for where the missing values occur.

What we are trying to avoid…