# Lab 04: Care For a Sample? – SOLUTIONS

**PSTAT 100, Summer Session A 2025 with Ethan P. Marzban**

MEMBER 1 (NetID 1) MEMBER 2 (NetID 2)
MEMBER 3 (NetID 3)

July 8, 2025

## Required Packages

```r
library(ottr)         # for checking test cases (i.e. autograding)
library(pander)       # for nicer-looking formatting of dataframe outputs
library(tidyverse)    # for graphs, data wrangling, etc.
```

## Logistical Details

**Logistical Details**

- This lab is due by **11:59pm on Wednesday, July 9, 2025**.
- Collaboration is allowed, and encouraged!
  - If you work in groups, list ALL of your group members' names and NetIDs (not Perm Numbers) in the appropriate spaces in the YAML header above.
  - Please delete any "MEMBER X" lines in the YAML header that are not needed.
  - No more than 3 people in a group, please.
- Ensure your Lab properly renders to a `.pdf`; non-`.pdf` submissions will not be graded and will receive a score of 0.
- Ensure all test cases pass (test cases that have passed will display a message stating `"All tests passed!"`)

## Lab Overview and Objectives

Welcome to another PSTAT 100 Lab! In this lab, we will cover the following:

- Sampling techniques

- Bias-correction through Inverse Probability Weighting
- Sampling distributions and basic inference

## Introduction to the Data

For today's lab, we will be considering a fictional dataset once again pertaining to cats. Located in the `data/` subfolder is a file called `cats_population.csv` - we are imgaining this to be our **target population** of cats. Recall that, in general, it is uncommon to have access to the entire population - the only reason we have access to the population in this case is because our data is simulated.

Each cat has three measurements on three variables:

- `Sex`: the sex of the cat (either `"Male"` or `"Female"`)
- `Breed`: the breed of the cat (either `"Domestic Short Hair"`, `"Domestic Medium Hair"`, or `"Domestic Long Hair"`)
- `Weight`: the weight of the cat (in pounds)

The following code reads in the `cats_population.csv` dataset, and assigns it to a variable called `pop` - if you are having trouble running this chunk, you may need to adjust your working directory.

```r
pop <- read.csv("data/cats_population.csv")
```

## Part I: Sampling Techniques

Let's begin with an exploration of some sampling techniques, as we discussed them in Lecture 8. Recall that there are three main types of sampling we discussed:

- **Simple Random Sampling:** each unit in the sampled population has an equal chance of being included in the sample
- **Stratified Random Sampling:** the sampled population is divided into **strata**, and a simple random sample is taken from each stratum
- **Cluster Random Sampling:** the sampled population is divided into **clusters**; a simple random sample of clusters is taken, and a simple random sample of units is taken from each of the selected clusters.

**Tip**

We will be making heavy use of the `slice_sample()` function; I encourage you to read the associated help file before beginning this lab.

**Important**

This lab will involve randomization. As such, to ensure the autograder runs properly, it is important that you set your seed within each question. Specifically, **set your seed to the value 100** at the start of each Question. Failure to do so may result in the autograder not running as expected.

**! Question 1**

Take a simple random sample of 600 cats from the population; assign your sample to a data frame called `sample1`.

**Solution:**

```r
## replace this line with your code
set.seed(100)
sample1 <- pop %>% slice_sample(n = 600)
```

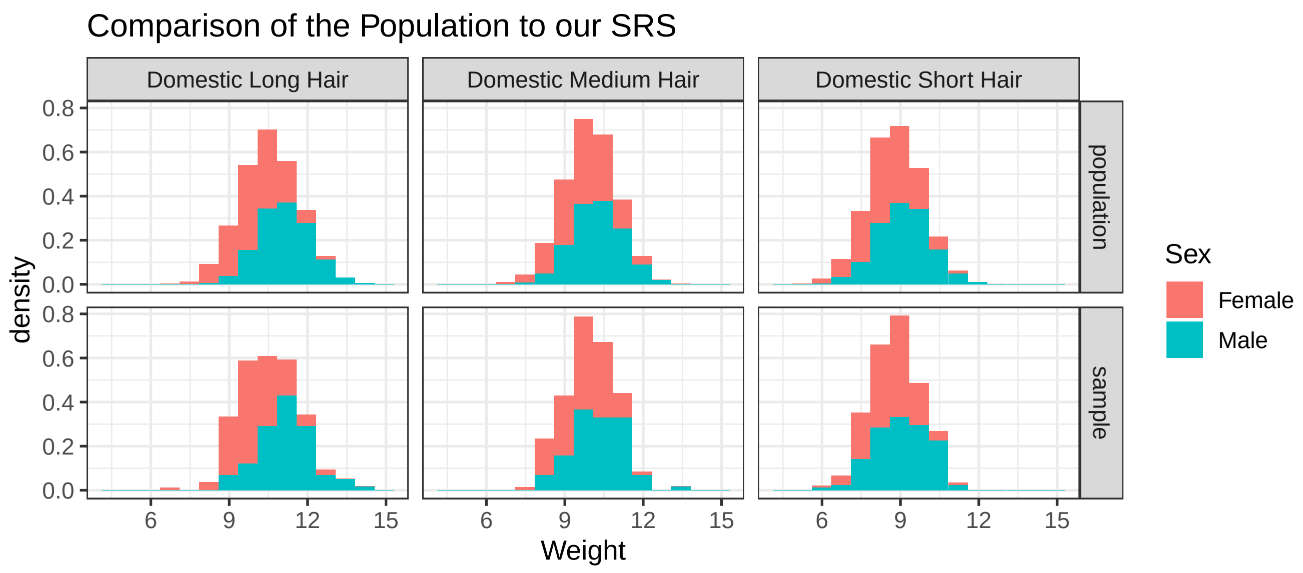
**Answer Check:**

```r
# DO NOT EDIT THIS LINE
invisible({check("tests/q1.R")})
```

```
All tests passed!
```

Recall that one of the hallmarks of simple random sampling is that our sample should preserve the structure of the population. Let's see if that is true! Assuming you completed Question 1 correctly, the following code should create a histogram that compares the distribution of weight values (colored by sex, and facetted by species) in both the population and our sample.

```r
rbind(pop, sample1) %>%
  mutate(indicator = c(rep('population', nrow(pop)), rep('sample', nrow(sample1)))
  ) %>%
  ggplot(aes(x = Weight)) +
  geom_histogram(bins = 15, aes(fill = Sex, y = after_stat(density))) +
  facet_grid(indicator ~ Breed) +
  theme_bw(base_size = 12) +
  ggtitle("Comparison of the Population to our SRS")
```



With any luck, you should see that the overall shapes of histograms for the sample and the population are quite similar! In other words, the distribution of weights in our sample appears to closely match the distribution of weights in the population.

As an alternative to simple random sampling, let's adopt stratified random sampling.

**Tip**

If the strata are defined by a variable present in the dataframe, stratified random sampling can be accomplished by first using `group_by()` to group by the stratifying variable and then using `slice_sample()` to take a simple random sample from each stratum. To be safe, you should pipe your resulting sample data frame into a call to `data.frame()` to ensure things aren't accidentally still grouped behind the scenes.

**Question 2**

Using `Breed` to define the strata, take a stratified random sample by taking a simple random sample of size 200 from each stratum. Assign your sample to a data frame called `sample2`.

**Solution:**

```r
## replace this line with your code
set.seed(100)
sample2 <- pop %>% group_by(Breed) %>% slice_sample(n = 200) %>% data.frame()
```

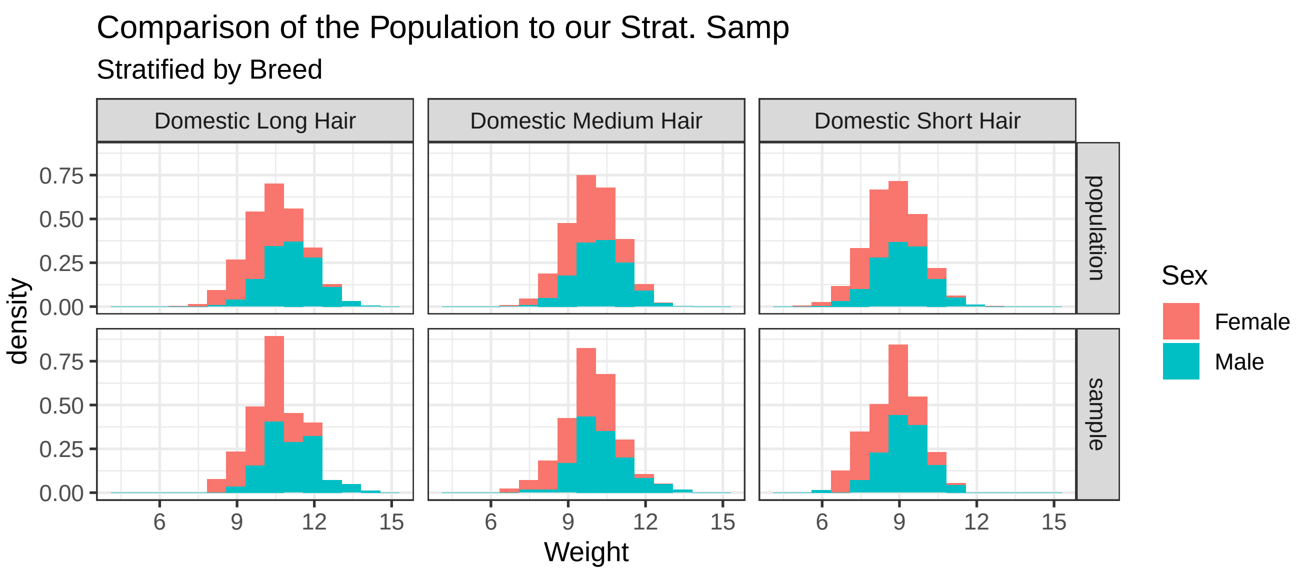
**Answer Check:**

```r
# DO NOT EDIT THIS LINE
invisible({check("tests/q2.R")})
```

```
All tests passed!
```

Just for fun, let's replicate our histogram from above, now using our stratified random sample:

```r
rbind(pop, sample2) %>%
  mutate(indicator = c(rep('population', nrow(pop)), rep('sample', nrow(sample1)))
  ) %>%
  ggplot(aes(x = Weight)) +
  geom_histogram(bins = 15, aes(fill = Sex, y = after_stat(density))) +
  facet_grid(indicator ~ Breed) +
  theme_bw(base_size = 12) +
  ggtitle("Comparison of the Population to our Strat. Samp",
          subtitle = "Stratified by Breed")
```



Notice that, once again, our samples preserve the structure of the population relatively well. As additional practice, let's see what happens if we stratify based on `Sex`:

**! Question 3**

Using `Sex` to define the strata, take a stratified random sample by taking a simple random sample of size 300 from each stratum. Assign your sample to a data frame called `sample3`.

**Solution:**

```
## replace this line with your code
set.seed(100)
sample3 <- pop %>% group_by(Sex) %>% slice_sample(n = 300) %>% data.frame()
```

**Answer Check:**

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q3.R")})
```

```
All tests passed!
```

**! Question 4**

Do you think cluster sampling is a good idea, given the population we have? Explain briefly.

**Solution:**
*Replace this line with your answer.*

**It is definitely not a good idea to cluster by sex; there are clear differences between the weights within each sex, so omitting one sex entirely would lead to skewed samples. One could possibly argue that there aren't too many differences across breeds, however even then there are only three breeds present so taking a sample of breeds first wouldn't necessarily provide much benefit. So, overall, cluster sampling is probably not a good idea.**

**Answer Check:**
There is no autograder for this question; your TA will manually check that your answers are correct.

## Part II: Biased Sampling

Generally, simple random sampling, stratified random sampling, and cluster random sampling are considered to be "unbiased" sampling techniques (we won't explicitly define what this means until a bit later). In this part of the lab, we'll explore the effects of utilizing a *biased* sampling scheme to generate a sample. Though biased sampling may seem hopeless to untangle, it *is* possible in some cases to apply **bias corrections** to obtain more robust estimates, provided we have good information on *how* the bias entered.

We are going to imagine that heavier cats are more likely to have been included in our sample than lighter cats. There is a real-world reason to believe this mght be the case! Suppose we collected our data by examining records from a veterinarian's office. It is the case that pet owners are, perhaps, more likely to go to the vet if they believe their cat is overweight - hence, it is possible that heavier cats are more likely to be included in our sample.

This type of sampling can be modeled by assigning a **weight** $w_i \in [0, 1]$ to each unit. The inclusion probability of unit $i$ is then computed by

$$\pi_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

where $n$ denotes the number of elements in the sample. For the simulation in this lab, we will utilize the **logistic function** to compute the weights:

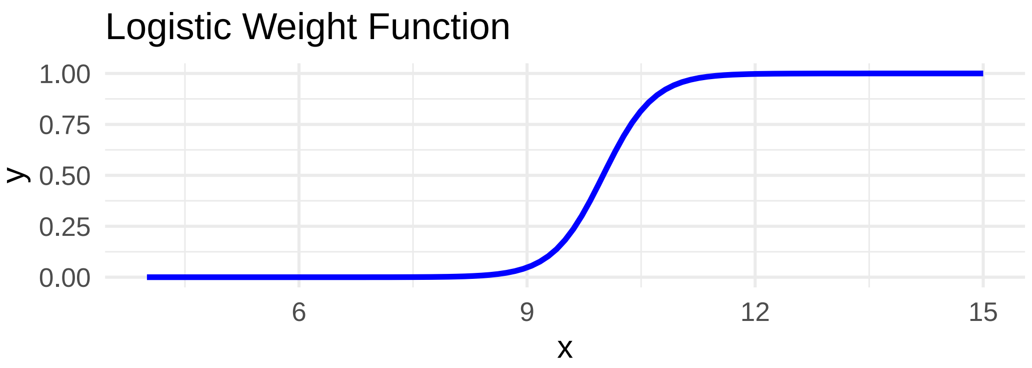
$$w_i(x) = \frac{1}{1 + e^{-k(x - x_0)}}$$

That is, for a given weight $x$, we compute the weight $w_i$ using the function above. Here is a code cell that creates this function in `R`, and assigns it to a function named `weight_fnt()`:

```r
weight_fnt <- function(x, k = 3, x0 = 10){
  return(1 / (1 + exp(-k * (x - x0))))
}
```

Below is a plot of what our weight function looks like. Recall that the idea is that we take a height (x-value) and use the `weight_fnt()` to obtain a weight (y-value) that can be used to compute the corresponding inclusion probability ($\pi_i$).

```r
data.frame(x = 4:15) %>% ggplot(aes(x = x)) +
  stat_function(fun = weight_fnt, linewidth = 1, col = "blue") +
  theme_minimal(base_size = 12) + ggtitle("Logistic Weight Function")
```



Just to make sure we don't mess up our original population dataframe, let's make a copy of it (called `pop_copy`) and use this copy of the population going forward:

```r
pop_copy <- pop
```

**! Question 5**

Take a sample of size 600, using the weights prescribed by the `weight_fnt()` function. Specifically:

- Mutate the `pop_copy` dataframe to include a column with the weights obtained as a function of the corresponding `Weight`
- Use `slice_sample()`, and specify the `weight_by` argument

Store your result in a dataframe called `samp_biased`. Note that your `samp_biased` data frame should still have a column of (probability) weight values.

**Solution:**

```r
## replace this line with your code
set.seed(100)
samp_biased <- pop_copy %>% mutate(weights = weight_fnt(Weight)) %>%
  slice_sample(n = 600, weight_by = weights)
```

**Answer Check:**

```
# DO NOT EDIT THIS LINE
invisible({check("tests/q5.R")})
```

```
All tests passed!
```

So, why are we calling this a "biased" sample? Well, when we introduce unequal inclusion probabilities, we can obtain sample values that differ from their corresponding population counterparts. For example, here is the mean weight of cats in the population:

```
(mean_pop_weight <- pop$Weight %>% mean())
```

```
[1] 9.683299
```

and here is the mean of weights in our sample:

```
(mean_samp_weight <- samp_biased$Weight %>% mean())
```

```
[1] 10.74878
```

As expected, our sample mean is larger than our population mean. (Recall that this was expected, since we explicitly modeled heavier cats as being more likely to be included in our sample!) However, because we have a good sense of what the inclusion probabilities were, we can perform a **bias correction** on our mean-weight calculation. One particular type of bias correction is called **inverse probability weighting** (IPW), in which we compute a bias-corrected average as:

$$\text{IPW average} = \sum_{i=1}^{n} \left[ \left( \frac{w_i^{-1}}{\sum_{i=1}^{n} w_i^{-1}} \right) \cdot \text{value}_i \right]$$

**! Question 6**

Calculate the IPW average of cat weights in the `biased_sample` dataframe **using the true weights $w_i$ that were computed from the logistic function**, and assign this to a variable called `ipw_avg`. Compare this to the weight of cats in the population.

**Solution:**

```
## replace this line with your code
(ipw_avg <- samp_biased %>% mutate(ipw = (1/weights) / sum(1/weights),
                                   ipw_weight = Weight * ipw) %>%
  pull(ipw_weight) %>%
  sum())
```

```
[1] 9.968533
```

**Answer Check:**

```r
# DO NOT EDIT THIS LINE
invisible({check("tests/q6.R")})
```

```
All tests passed!
```

Now, you may note that we used the "true" weights (i.e. those computed from the logistic function) to make things easier. In practice, these weights won't be known - we won't discuss how to estimate them, but just know that there exist methods that can be used to make our procedure in Question 6 above more self-contained.

## Part III: Sampling Distributions

Finally, let's pivot away from sampling techniques and explore some of the topics from *inferential statistics* we learned in lecture today.

First, recall that we define a **population parameter** to be a parameter that governs the population. In general, the true value of a population parameter is unknown; as such, we seek to **estimate** it. Specifically, we construct **estimators** by taking a random sample and constructing a random variable from this random sample that we believe, in some way, mirrors the value of the corresponding population parameter.

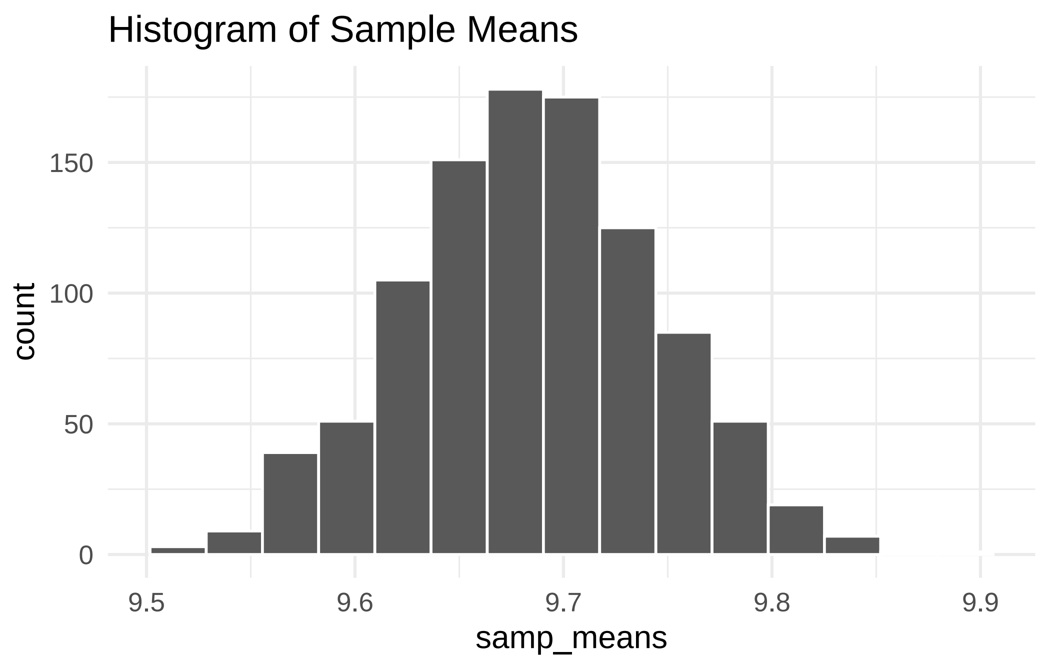
Crucially, estimators are *random variables* - this means they have distributions, and we call these distributions **sampling distributions**. In lecture, we explored the sampling distributions of the sample mean and sample variance. As a warm-up, let's re-do some of these simulations using our cats population from above.

**! Question 7**

Repeat the following 1000 times: Generate an SRS *with replacement* of size 500 from the population, and record the average (mean) weight. Generate a histogram of these 1000 sample averages - this is an approximation to the sampling distribution of the sample mean $\overline{Y}_n$. What distribution does this look like? Does this agree with what we discussed in lecture?

**Solution:**

```r
## replace this line with your code
samp_means <- c()
for(b in 1:1000){
  temp_samp <- sample(pop$Weight, size = 500, replace = T)
  samp_means <- c(samp_means, mean(temp_samp))
}
data.frame(samp_means) %>% ggplot(aes(x = samp_means)) +
  geom_histogram(bins = 15, col = "white") +
  theme_minimal(base_size = 12) +
  ggtitle("Histogram of Sample Means")
```



**Answer Check:**
There is no autograder for this question; your TA will manually check that your answers are correct.

As mentioned in lecture, we don't have to restrict ourselves to means! Specifically, let's see if we perform inference on the *largest observable cat weight*; i.e. a **population maximum**. In other words, define $\theta$ to be the true largest weight (in pounds) a cat could theoretically have. A natural estimator for $\theta$ is the **sample maximum**; using simulations, we can easily construct an approximation to the sampling distribution for the sample maximum.

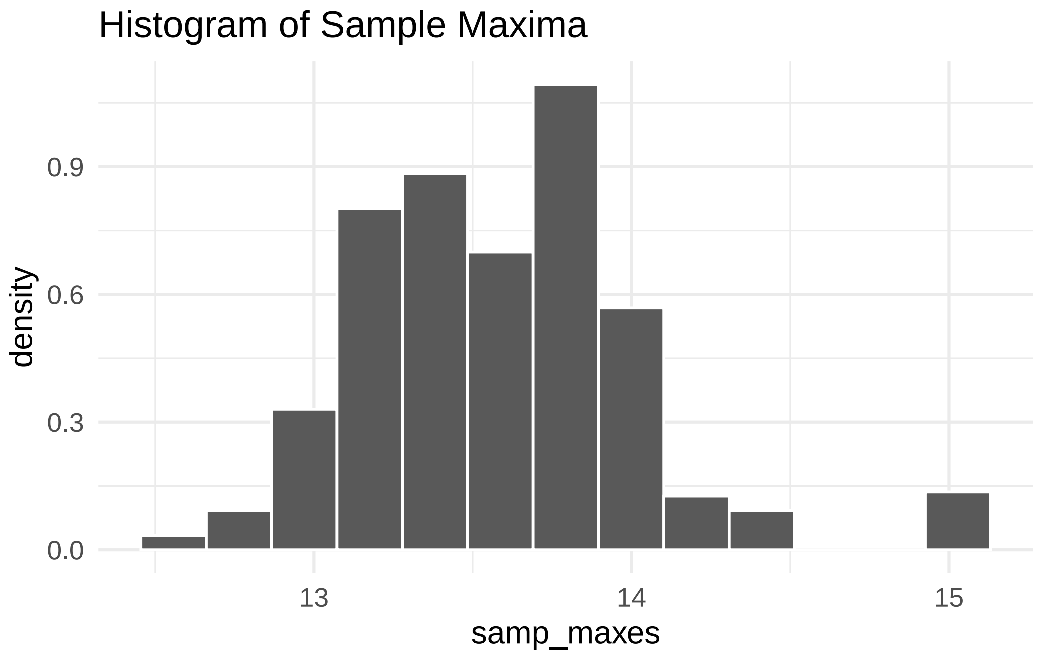
**! Question 8**

Repeat the following 1000 times: Generate an SRS *with replacement* of size 500 from the population, and record the maximum weight. Generate a histogram of these 1000 sample maxima.

**Solution:**

```r
## replace this line with your code
samp_maxes <- c()
for(b in 1:1000){
  temp_samp <- sample(pop$Weight, size = 500, replace = T)
  samp_maxes <- c(samp_maxes, max(temp_samp))
}

data.frame(samp_maxes) %>% ggplot(aes(x = samp_maxes)) +
  geom_histogram(bins = 13, col = "white",
                 aes(y = after_stat(density))) +
  theme_minimal(base_size = 12) +
  ggtitle("Histogram of Sample Maxima")
```



**Answer Check:**
There is no autograder for this question; your TA will manually check that your answers are correct.

The distribution of the sample maximum is *not* normally distributed. One can show (and the proof is a bit outside the scope of PSTAT 100) that the density of the sample maximum of a series of $n$ i.i.d. $\mathcal{N}(\mu, \sigma^2)$ random variables is given by

$$f_X(x) = \frac{n}{\sigma}\Phi\left(\frac{x-\mu}{\sigma}\right)^{n-1}\phi\left(\frac{x-\mu}{\sigma}\right)$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ denote the standard normal CDF and PDF, respectively. The code chunk below creates a function called `true_max()` with density equal to the one provided above:

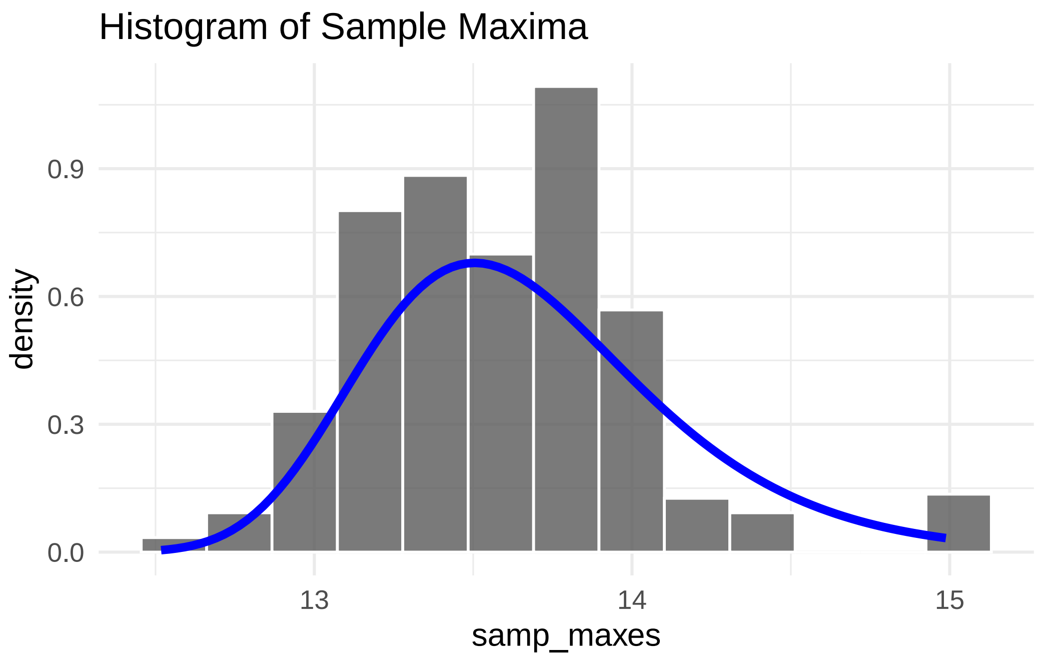
```r
true_max <- Vectorize(function(x, n, mu, sigma){
  (n / sigma) * pnorm(x, mu, sigma)^(n - 1) * dnorm(x, mu, sigma)
})
```

**! Question 9**

Copy-paste your histogram from Question 7 above, and overlay the true sampling density atop. Use $\mu$ and $\sigma$ values computed from the population; that is, since we are assuming we have access to the full population, compute the true values of $\mu$ and $\sigma$ (which, again, we wouldn't be able to do in real-world settings). **Hint:** use `stat_function()`.

**Solution:**

```r
## replace this line with your code
data.frame(samp_maxes) %>% ggplot(aes(x = samp_maxes)) +
  geom_histogram(bins = 13, col = "white",
                 aes(y = after_stat(density)),
                 alpha = 0.8) +
  theme_minimal(base_size = 12) +
  stat_function(fun = true_max,
                args = list(n = 500,
                            mu = pop$Weight %>% mean(),
                            sigma = pop$Weight %>% sd()),
                col = "blue", linewidth = 1.5) +
  ggtitle("Histogram of Sample Maxima")
```



**Answer Check:**
There is no autograder for this question; your TA will manually check that your answers are correct.

## Submission Details

Congrats on finishing this PSTAT 100 lab! Please carry out the following steps:

**Submission Details**

1) Check that all of your tables, plots, and code outputs are rendering correctly in your final `.pdf`.
2) Check that you passed all of the test cases (on questions that have autograders). You'll know that you passed all tests for a particular problem when you get the message "All tests passed!".
3) Submit **ONLY** your `.pdf` to Gradescope. Make sure to **match ALL pages to the ONE question on Gradescope**; failure to do so will incur a penalty of 0.1 points.