```
In [1]:    # Initialize Otter
           import otter
           grader = otter.Notebook("lab2-sampling.ipynb")
```

```
In [2]:    import numpy as np
           import pandas as pd
           import altair as alt
```

Lab 2: Sampling designs and statistical bias

In this lab you'll explore through simulation how nonrandom sampling can produce datasets with statistical properties that are distored relative to the population that the sample was drawn from. This kind of distortion is known as **bias**.

In common usage, the word 'bias' means disproportion or unfairness. In statistics, the concept has the same connotation -- biased sampling favors certain observational units over others, and biased estimates are estimates that favor larger or smaller values than the truth. The goal of this lab is to refine your understanding about what statistical bias is and is not and develop your intuition about potential mechanisms by which bias is introduced and the effect that this can have on sample statistics.

Objectives:

- Simulate biased and unbiased sampling designs
- Examine the impact of sampling bias on the sample mean
- Apply a simple bias correction by inverse probability weighting

Background

Sampling designs

The **sampling design** of a study refers to ***the way observational units are selected*** from the collection of all observational units. Any design can be expressed by the probability that each unit is included in the sample. In a random sample, all units are equally likely to be included.

For example, you might want to learn about U.S. residents (population), but only be able for ethical or practical reasons to study adults (sampling frame), and decide to do a mail survey of 2000 randomly selected addresses in each state (sampling design). Each collection of 2000 addresses may constitute a random sample of households, but even with a 100% response rate the survey results will not be a random sample of adult U.S.

residents because individuals share addresses and the population sizes are different from state to state.

Bias

Formally, **bias** describes *the 'typical' deviation of a sample statistic the correspongind population value*.

For example, if a particular sampling design tends to produce an average measurement around 1.5 units, but the true average in the population is 2 units, then the estimate has a bias of -0.5 units. The language 'typical' and 'tends to' is important here. Estimates are never perfect, so just because an estimate is off by -0.5 units for one sample doesn't make it biased -- it is only biased if it is *consistently* off.

Although bias is technically a property of a sample statistic (like the sample average), it's common to talk about a biased *sample* -- this term refers to a dataset collected using a sampling design that produces biased statistics.

This is exactly what you'll explore in this lab -- the relationship between sampling design and bias.

Simulated data

You will be simulating data in this lab. **Simulation** is a great means of exploration *because you can control the population properties*, which are generally unknown in practice.

When working with real data, you just have one dataset, and you don't know any of the properties of the population or what might have happened if a different sample were collected. That makes it difficult to understand sampling variation and impossible to directly compare the sample properties to the population properties.

With simulated data, by contrast, you control how data are generated with exact precision -- so by extension, you know everything there is to know about the population. In addition, repeated simulation of data makes it possible to explore the typical behavior of a particular sampling design, so you can learn 'what usually happens' for a particular sampling design by direct observation.

Scenario 1: eucalyptus seed diameters

In this scenario you'll compare the sample mean and the distribution of sample values

for a single viariable with the population mean and distribution of population values for an unbiased sampling design.

Hypothetical population

To provide a little context to this scenario, imagine that you're measuring eucalyptus seeds to determine their typical diameter. The cell below simulates diameter measurements for a hypothetical population of 5000 seeds; imagine that this is the total number of seeds in a small grove at some point in time.

```
In [3]:  # simulate seed diameters
         np.random.seed(40221) # for reproducibility
         population = pd.DataFrame(
             data = {'diameter': np.random.gamma(shape = 2, scale = 1/2, size = 5000)
                     'seed': np.arange(5000)}
         ).set_index('seed')

         # check first few rows
         population.head(3)
```

Out[3]:

	diameter
seed	
0	0.831973
1	1.512187
2	0.977392

Question 1

Calculate the mean diameter for the hypothetical population and store it as `mean_diameter`.

```
In [4]:  mean_pop_diameter = population.mean()

         mean_pop_diameter
```

```
Out[4]:  diameter    1.018929
         dtype: float64
```

```
In [5]:  grader.check("q1")
```

Out[5]:

q1 passed! 💯

Question 2

Calculate the standard deviation of diameters for the hypothetical population and store it as `std_dev_pop_diameter`.

```
In [6]: std_dev_pop_diameter = population.std()
        std_dev_pop_diameter
```

```
Out[6]: diameter    0.72393
        dtype: float64
```

```
In [7]: grader.check("q2")
```

Out[7]:
q2 passed! 🍀

The cell below produces a histogram of the population values -- the distribution of diameter measurements among the hypothetical population -- with a vertical line indicating the population mean.

```python
In [8]: # base layer
        base_pop = alt.Chart(population).properties(width = 400, height = 300)

        # histogram of diameter measurements
        hist_pop = base_pop.mark_bar(opacity = 0.8).encode(
            x = alt.X('diameter',
                      bin = alt.Bin(maxbins = 20),
                      title = 'Diameter (mm)',
                      scale = alt.Scale(domain = (0, 6))),
            y = alt.Y('count()', title = 'Number of seeds in population')
        )

        # vertical line for population mean
        mean_pop = base_pop.mark_rule(color='blue').encode(
            x = 'mean(diameter)'
        )

        # display
        hist_pop + mean_pop
```
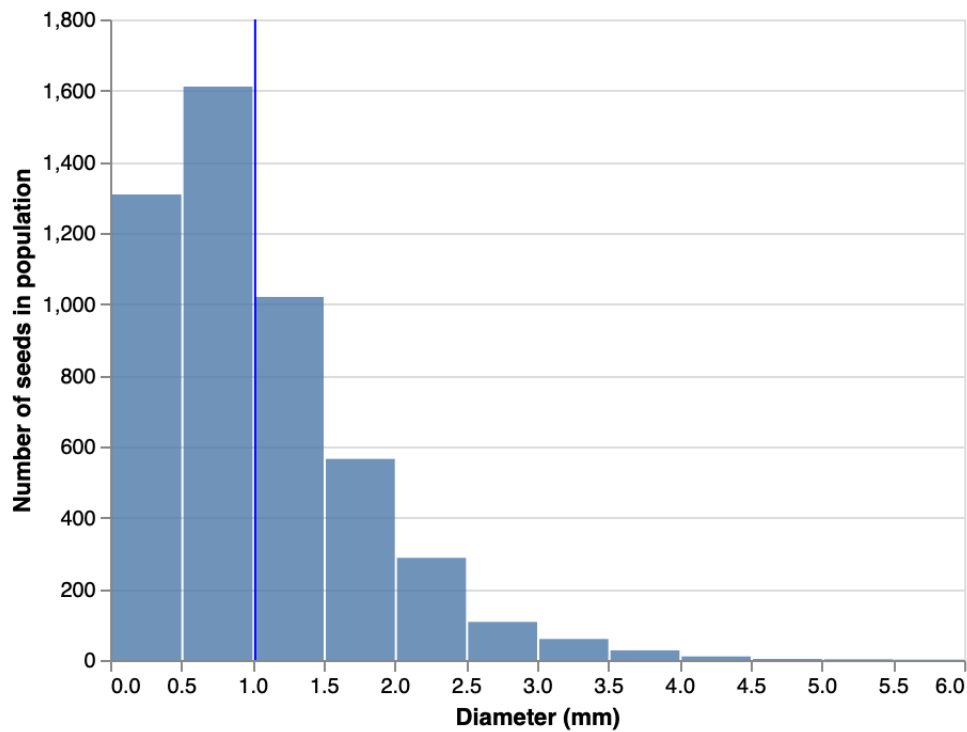
Random sampling

Imagine that your sampling design involves collecting bunches of plant material from several locations in the grove and sifting out the seeds with a fine sieve until you obtaining 250 seeds. We'll suppose that using your collection method, any of the 5000 seeds is equally likely to be obtained, so that your 250 seeds comprise a *random sample* of the population.

We can simulate samples obtained using your hypothetical design by drawing values without replacement from the population.

```
In [9]:  # draw a random sample of seeds
         np.random.seed(40221) # for reproducibility
         sample = population.sample(n = 250, replace = False)
```

Question 3

Calculate the mean diameter of seeds in the simulated sample and set the result to `mean_sample_diameter`.

```
In [10]:  mean_sample_diameter = sample.mean()
          mean_sample_diameter
```

```
Out[10]:  diameter    0.977722
          dtype: float64
```

```
In [11]:  grader.check("q3")
```

Out[11]:

q3 passed! 🌈

You should see above that the sample mean is close to the population mean. In fact, *all* sample statistics are close to the population; this can be seen by comparing the distribution of sample values with the distribution of population values.

```
In [12]:  # base layer
          base_samp = alt.Chart(sample).properties(width = 400, height = 300)

          # histogram of diameter measurements
          hist_samp = base_samp.mark_bar(opacity = 0.8).encode(
              x = alt.X('diameter',
                        bin = alt.Bin(maxbins = 20),
                        scale = alt.Scale(domain = (0, 6)),
                        title = 'Diameter (mm)'),
              y = alt.Y('count()', title = 'Number of seeds in sample')
          )

          # vertical line for population mean
          mean_samp = base_samp.mark_rule(color='blue').encode(
              x = 'mean(diameter)'
          )

          # display
          hist_samp + mean_samp | hist_pop + mean_pop
```

Out[12]:
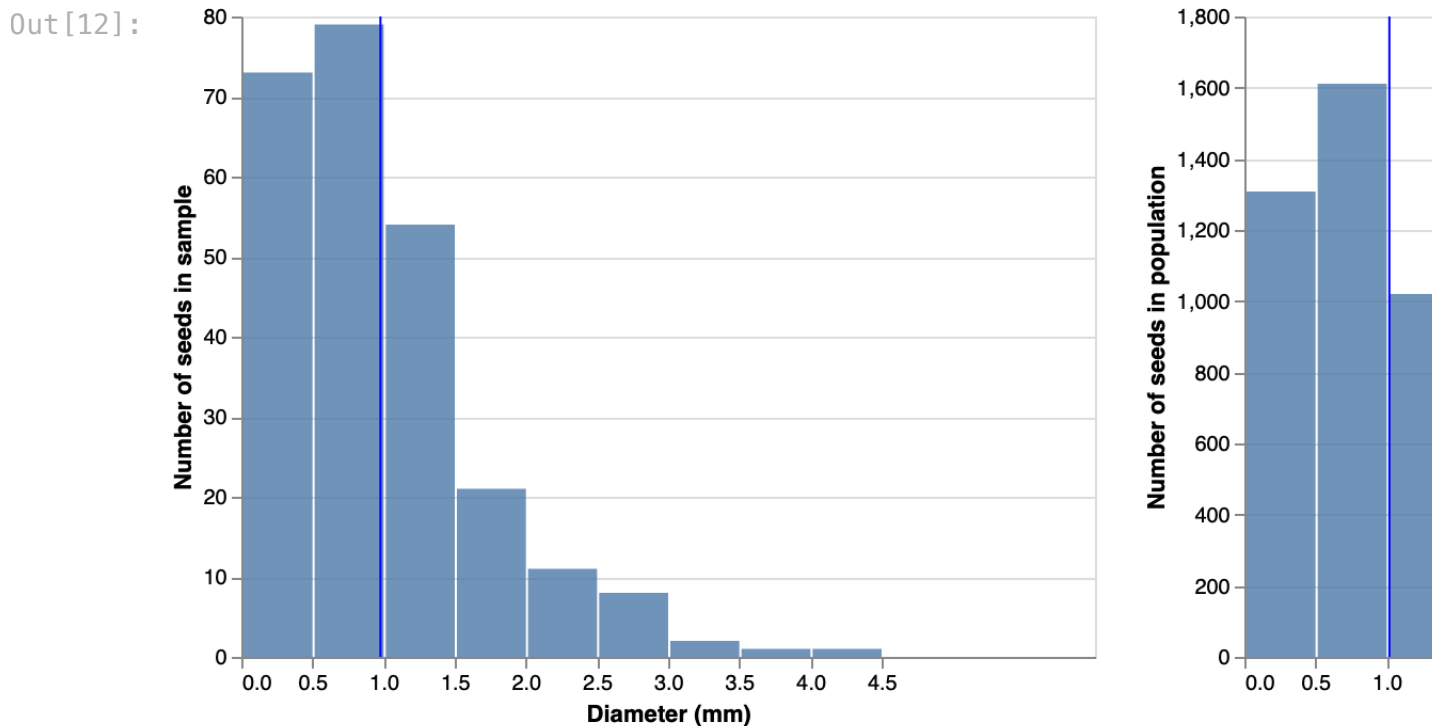
While there are some small differences, the overall shape is similar and the sample mean is almost exactly the same as the population mean. So with this sampling design, you obtained a dataset with few distortions of the population properties, and the sample mean is a good estimate of the population mean.

Assessing bias through simulation

You may wonder: *does that happen all the time, or was this just a lucky draw?* This question can be answered by simulating a large number of samples and checking the average behavior to see whether the undistorted representation of the population is typical for this sampling design.

The cell below estimates the bias of the sample mean by:

- drawing 1000 samples of size 300;
- storing the sample mean from each sample;
- computing the average difference between the sample means and the population mean.

```
In [13]: np.random.seed(40221) # for reproducibility

         # number of samples to simulate
         nsim = 1000

         # storage for the sample means
         samp_means = np.zeros(nsim)

         # repeatedly sample and store the sample mean
         for i in range(0, nsim):
             samp_means[i] = population.sample(n = 250, replace = False).mean()
```

The bias of the sample mean is its average distance from the population mean. We can estimate this using our simulation results as follows:

```
In [14]: # bias
         samp_means.mean() - population.diameter.mean()
```

```
Out[14]: -0.0012458197406362004
```

So the average error observed in 1000 simulations was about 0.001 mm! This suggests that the sample mean is *unbiased*: on average, there is no error. Therefore, at least with respect to estimating the population mean, random samples appear to be *unbiased samples*.

However, **unbiasedness does not mean that you won't observe estimation error**. There is a natural amount of variability from sample to sample, because in each sample a different collection of seeds is measured. We can estimate this as well using the simulation results by checking the standard deviation of the sample means across all 1000 samples:

```
In [15]:   samp_means.std()
```

```
Out[15]:   0.04285044949708931
```

So on average, the sample mean varies by about 0.04 mm from sample to sample.

We could also check how much the sample mean deviates from the population mean on average by computing *root mean squared error*:

```
In [16]:   np.sqrt(np.sum((samp_means - population.diameter.mean())**2)/1000)
```

```
Out[16]:   0.0428685559463899
```

Note that this is very close to the variance of the sample mean across simulations, but not exactly the same; this latter calculation measures the spread around the population mean, and is a conventional measure of estimation accuracy.

The cell below plots a histogram representing the distribution of values of the sample mean across the 1000 samples you simulated (this is known as the *sampling distribution* of the sample mean). It shows a peak right at the population mean (blue vertical line) but some symmetric variation to either side -- most values are between about 0.93 and 1.12.

```
In [17]:   # plot the simulated sampling distribution
           sampling_dist = alt.Chart(pd.DataFrame({'sample mean': samp_means})).mark_ba
               x = alt.X('sample mean', bin = alt.Bin(maxbins = 30), title = 'Value of
               y = alt.Y('count()', title = 'Number of simulations')
           )

           sampling_dist + mean_pop
```

Out[17]:
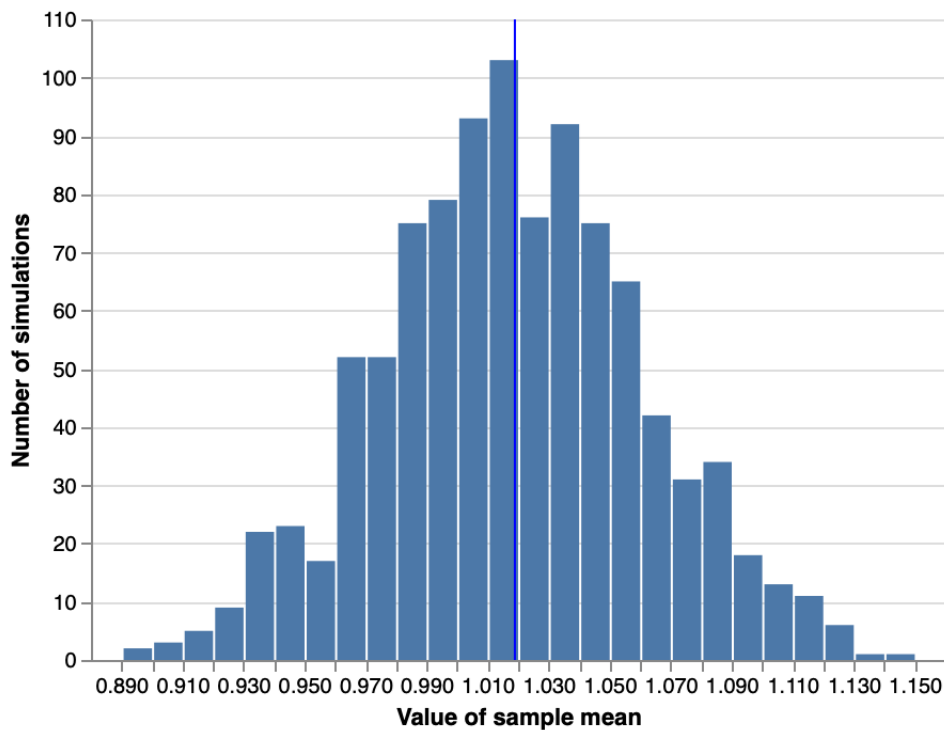
Biased sampling

In this scenario, you'll use the same hypothetical population of eucalyptus seed diameter measurements and explore the impact of a biased sampling design.

In the first design, you were asked to imagine that you collected and sifted plant material to obtain seeds. Suppose you didn't know that the typical seed is about 1mm in diameter and decided to use a sieve that is a little too coarse, tending only to sift out larger seeds and letting smaller seeds pass through. As a result, small seeds have a lower probability of being included in the sample and large seeds have a higher probability of being included in the sample.

This kind of sampling design can be described by assigning differential *sampling weights* $w_1, \ldots, w_N$ to each observation. The cell below defines some hypothetical weights such that larger diameters are more likely to be sampled.

In [18]:
```
population_mod1 = population.copy()
```

```
In [19]:  # inclusion weight as a function of seed diameter
          def weight_fn(x, r = 10, c = 1.5):
              out = 1/(1 + np.e**(-r*(x - c)))
              return out

          # create a grid of values to use in plotting the function
          grid = np.linspace(0, 6, 100)
          weight_df = pd.DataFrame(
              {'seed diameter': grid,
               'weight': weight_fn(grid)}
          )

          # plot of inclusion probability against diameter
          weight_plot = alt.Chart(weight_df).mark_area(opacity = 0.3, line = True).enc
              x = 'seed diameter',
              y = 'weight'
          ).properties(height = 100)

          # show plot
          weight_plot
```

Out[19]:
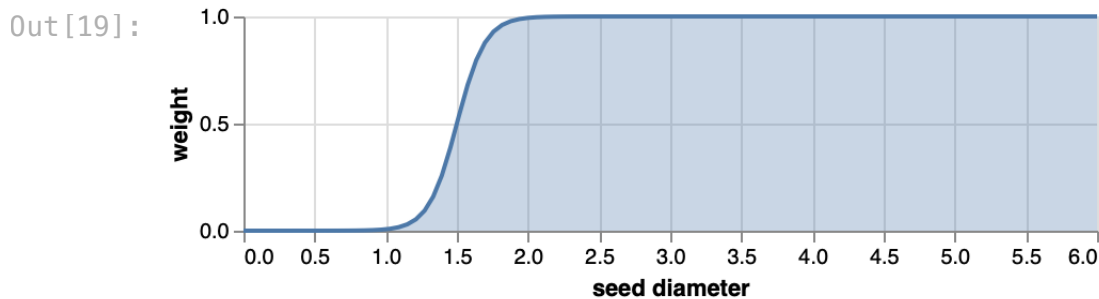
The actual probability that a seed is included in the sample -- its **inclusion probability** -- is proportional to the sampling weight. These inclusion probabilities π_i can be calculated by normalizing the weights w_i over all seeds in the population $i = 1, \ldots, 5000$:

$$\pi_i = \frac{w_i}{\sum_i w_i}$$

It may help you to picture how the weights will be used in sampling to line up this plot with the population distribution. In effect, we will sample more from the right tail of the population distribution, where the weight is nearest to 1.

```
In [20]:  hist_pop & weight_plot
```
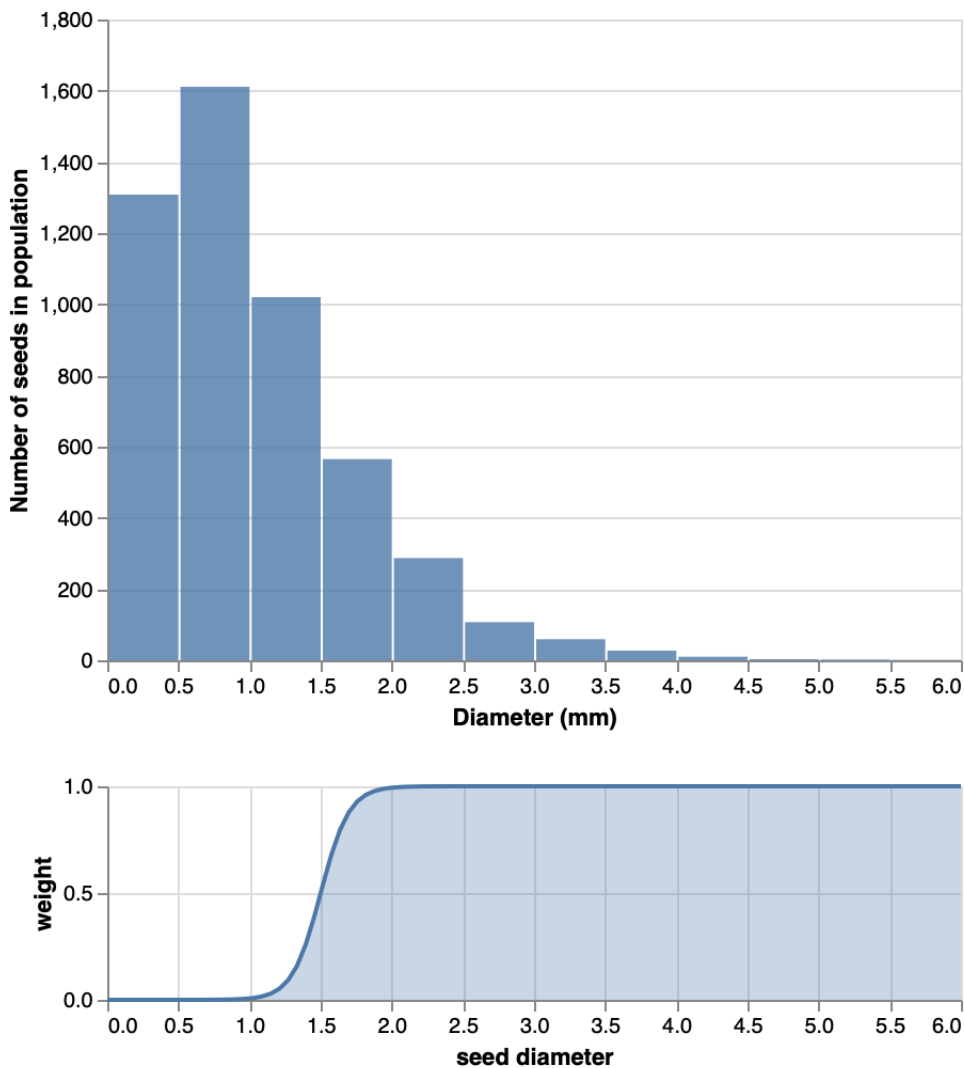
The following cell draws a sample with replacement from the hypothetical seed population *with seeds weighted according to the inclusion probability given by the function above.*

In [21]:
```python
# assign weight to each seed
population_mod1['weight'] = weight_fn(population_mod1.diameter)

# draw weighted sample
np.random.seed(40721)
sample2 = population_mod1.sample(n = 250, replace = False, weights = 'weight
```

Question 4

Calculate the mean diameter of seeds in the simulated sample and store as `mean_sample2_diameter` .

In [22]:
```python
mean_sample2_diameter = sample2.mean()

mean_sample2_diameter
```

```
Out[22]: diameter    2.02672
         dtype: float64
```

```
In [23]: grader.check("q4")
```

Out[23]:

q4 passed! 🙌

Question 5

Show side-by-side plots of the distribution of sample values and the distribution of population values, with vertical lines indicating the corresponding mean on each plot.

Hint: copy the cell that produced this plot in scenario 1 and replace `sample` with `sample2`. Utilizing different methods is also welcome.

```
In [ ]:
```

```python
In [24]: # base layer
base_samp = alt.Chart(sample2).properties(width = 300, height = 300)

# histogram of diameter measurements
hist_samp = base_samp.mark_bar(opacity = 0.8).encode(
    x=alt.X('diameter',
            bin=alt.Bin(maxbins=20),
            scale = alt.Scale(domain = (0,6)),
            title = 'Diameter(mm)'),
    y=alt.Y('count()',
            title = 'Number of seeds in sample')
)

# vertical line for population mean
mean_samp = base_samp.mark_rule(color='red').encode(
    x='mean(diameter)'
)

# combine layers
hist_samp + mean_samp | hist_pop + mean_pop
```
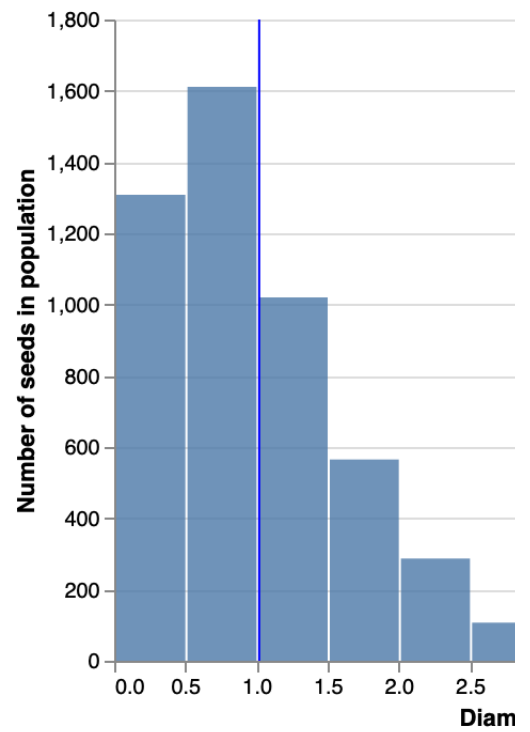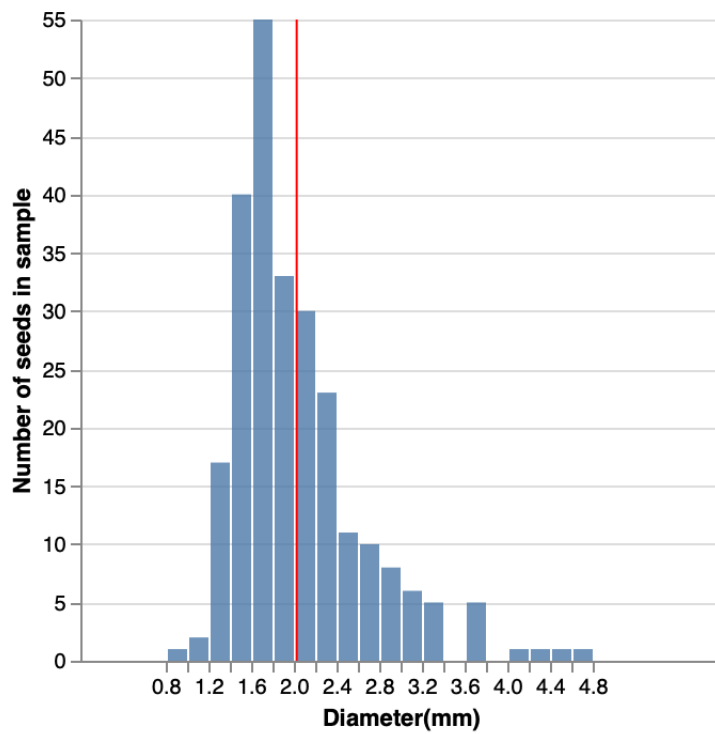
Assessing bias through simulation

Here you'll mimic the simulation done in scenario 1 to assess the bias of the sample mean under this new sampling design.

```
In [25]: population_mod1.head()
```

Out[25]:

seed	diameter	weight
0	0.831973	0.001254
1	1.512187	0.530430
2	0.977392	0.005346
3	2.874944	0.999999
4	0.506508	0.000048

Question 6

Investigate the bias of the sample mean by:

- drawing 1000 samples with observations weighted by inclusion probability;
- storing the collection of sample means from each sample as `samp_means`;
- computing the average difference between the sample means and the population mean (in that order!) and storing the result as `avg_diff`.

(*Hint*: copy the cell that performs this simulation in scenario 1, and be sure to replace `population` with `population_mod1` and adjust the sampling step to include `weights = ...` with the appropriate argument.)

```
In [ ]:
```

```
In [26]:  np.random.seed(40221) # for reproducibility

          # number of samples to simulate
          n_samp = 1000

          # storage for the sample means
          samp_means = np.zeros(n_samp)

          # repeatedly sample and store the sample mean in the samp_means array
          for i in range(n_samp):
              samp = population_mod1.sample(n = 250, replace = False, weights = popula
              samp_means[i] = samp['diameter'].mean()

          # bias
          avg_diff = samp_means.mean() - population_mod1['diameter'].mean()

          print(avg_diff)
```

```
1.0576986191465758
```

```
In [27]:  grader.check("q6")
```

```
Out[27]:
          q6 passed! 🎉
```

Question 7

Does this sampling design seem to introduce bias? If so, does the sample mean tend to over-estimate or under-estimate the population mean?

Our sample is biased with a bias of 1. The positive number shows that the sample is overestimating the bias

Scenario 2: hawks

In this scenario, you'll explore sampling from a population with group structure -- frequently bias can arise from inadvertent uneven sampling of groups within a population.

Hypothetical population

Suppose you're interested in determining the average beak-to-tail length of red-tailed hawks to help differentiate them from other hawks by sight at a distance. Females and males differ slightly in length -- females are generally larger than males. The cell below generates length measurements for a hypothetical population of 3000 females and 2000 males.

In [28]:
```python
# for reproducibility
np.random.seed(40721)

# simulate hypothetical population
female_hawks = pd.DataFrame(
    data = {'length': np.random.normal(loc = 57.5, scale = 3, size = 3000),
            'sex': np.repeat('female', 3000)}
)

male_hawks = pd.DataFrame(
    data = {'length': np.random.normal(loc = 50.5, scale = 3, size = 2000),
            'sex': np.repeat('male', 2000)}
)

population_hawks = pd.concat([female_hawks, male_hawks], axis = 0)

# preview
population_hawks.groupby('sex').head(2)
```

Out[28]:

|	length	sex
0	53.975230	female
1	60.516768	female
0	53.076663	male
1	49.933166	male

The cell below produces a histogram of the lengths in the population overall (bottom panel) and when distinguished by sex (top panel).

```
In [29]:  base = alt.Chart(population_hawks).properties(height = 200)

          hist = base.mark_bar(opacity = 0.5, color = 'red').encode(
              x = alt.X('length',
                        bin = alt.Bin(maxbins = 40),
                        scale = alt.Scale(domain = (40, 70)),
                        title = 'length (cm)'),
              y = alt.Y('count()',
                        stack = None,
                        title = 'number of birds')
          )

          hist_bysex = hist.encode(color = 'sex').properties(height = 100)

          hist_bysex & hist
```

Out[29]:
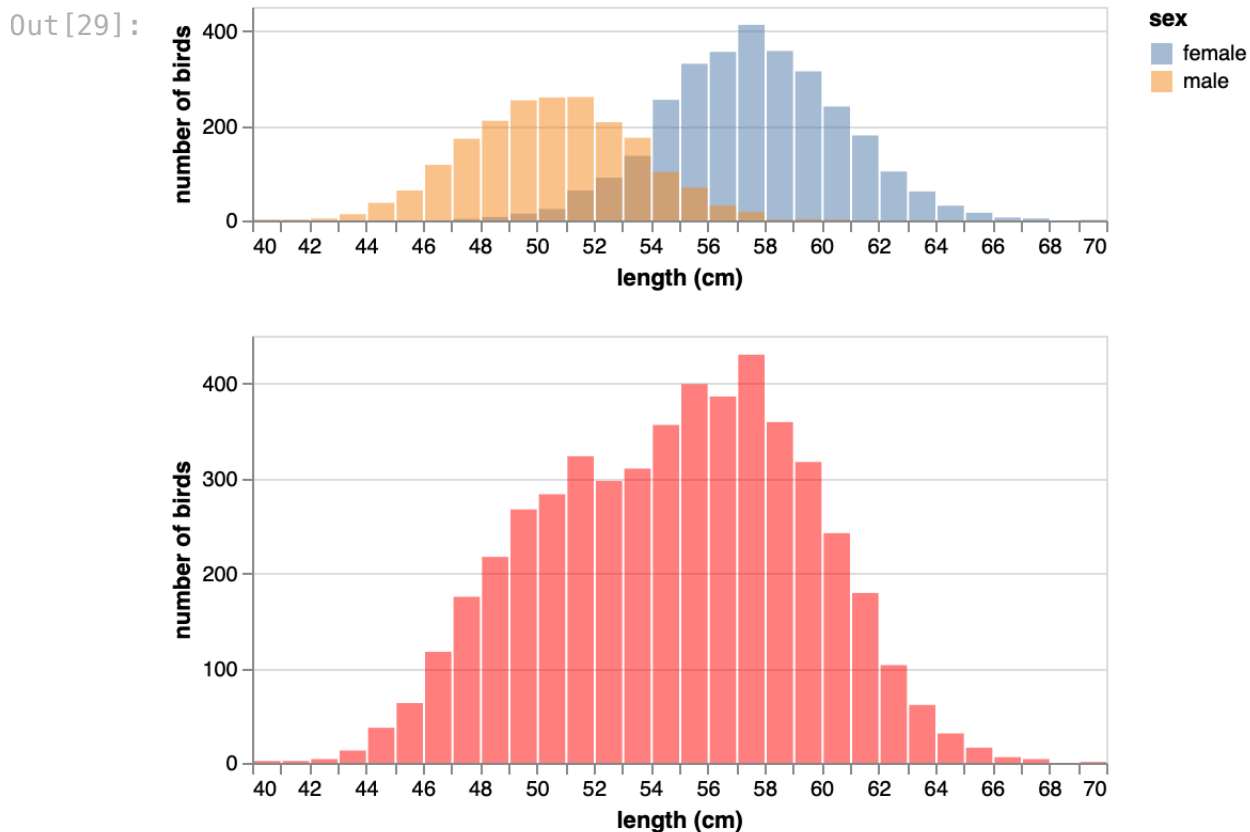
The population mean -- average length of both female and male red-tailed hawks -- is shown below.

```
In [30]:  # population mean
          population_hawks.mean(numeric_only = True)
```

Out[30]: length   54.737717
          dtype: float64

First try drawing a random sample from the population:

```
In [31]:  # for reproducibility
          np.random.seed(40821)

          # randomly sample
          sample_hawks = population_hawks.sample(n = 300, replace = False)
```

Question 8

Do you expect that the sample will contain equal numbers of male and female hawks?
Think about this for a moment (you don't have to provide a written answer), and then
compute the proportions of individuals in the sample of each sex and store the result as
`proportion_hawks_sample`.

Hint: group by sex, use `.count()`, and divide by the sample size. Be sure to rename
the output column appropriately, as the default behavior produces a column called
`length`.

```
In [32]:  proportion_hawks_sample = sample_hawks.groupby('sex').count().rename(columns

          proportion_hawks_sample
```

Out[32]:

sex	proportion
female	0.596667
male	0.403333

```
In [33]:  grader.check("q8")
```

Out[33]:
q8 passed! 🌟

The sample mean is shown below, and is fairly close to the population mean. This should
be expected, since you already saw in scenario 1 that random sampling is an unbiased
sampling design with respect to the mean.

```
In [34]:  sample_hawks.mean(numeric_only = True)
```

```
Out[34]:  length    54.952103
          dtype: float64
```

Biased sampling

Let's now consider a biased sampling design. Usually, length measurements are collected from dead specimens collected opportunistically. Imagine that male mortality is higher, so there are better chances of finding dead males than dead females. Suppose in particular that specimens are five times as likely to be male; to represent this situation, we'll assign sampling weights of 5/6 to all male hawks and weights of 1/6 to all female hawks.

```python
def weight_fn(sex, p = 5/6):
    if sex == 'male':
        out = p
    else:
        out = 1 - p
    return out

weight_df = pd.DataFrame(
    {'length': [50.5, 57.5],
     'weight': [5/6, 1/6],
     'sex': ['male', 'female']})

wt = alt.Chart(weight_df).mark_bar(opacity = 0.5).encode(
    x = alt.X('length', scale = alt.Scale(domain = (40, 70))),
    y = alt.Y('weight', scale = alt.Scale(domain = (0, 1))),
    color = 'sex'
).properties(height = 70)

hist_bysex & wt
```

In [35]:

Out[35]:
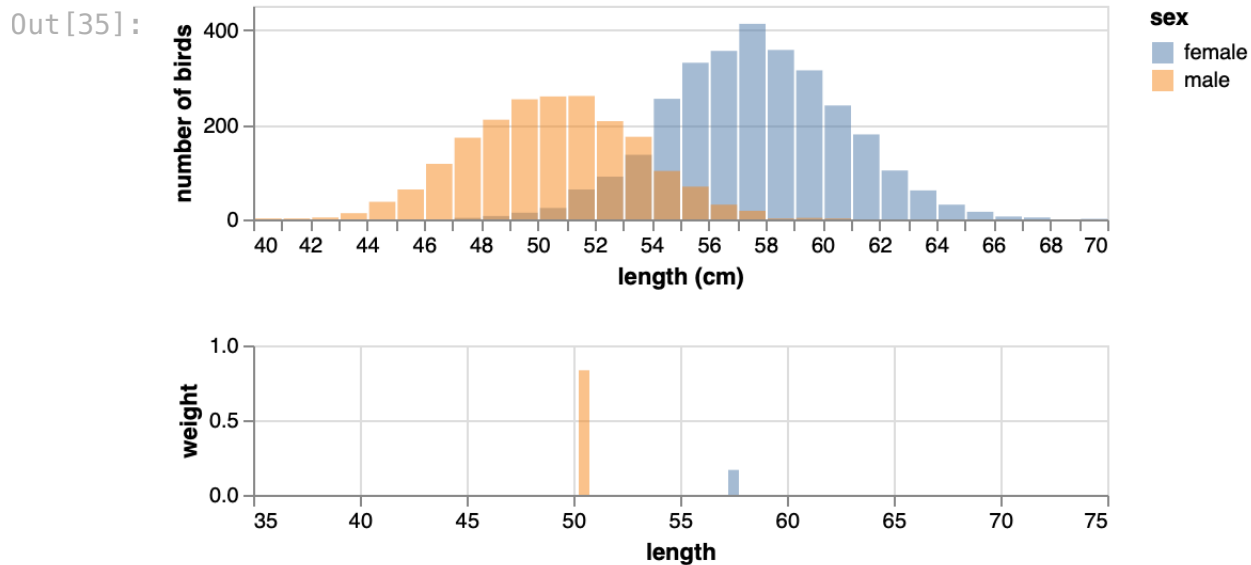
Question 9

Draw a weighted sample `sample_hawks_biased` from the population `population_hawks` using the weights defined by `weight_fn`, and compute and store the sample mean as `sample_hawks_biased_mean`.

```
In [ ]:
```

```
In [36]: # for reproducibility
         np.random.seed(40821)

         # assign weights
         population_hawks['weight'] = population_hawks.sex.aggregate(func = weight_fr

         # randomly sample
         sample_hawks_biased = population_hawks.sample(n = 300, replace = False,  wei

         # compute mean
         sample_hawks_biased_mean=sample_hawks_biased.mean(numeric_only=True)
```

```
In [37]: grader.check("q9")
```

```
Out[37]:
```
q9 passed! 🙌

Question 10

Investigate the bias of the sample mean by:

- drawing 1000 samples with observations weighted by `weight_fn`;
- storing the sample mean from each sample as `samp_means_hawks`;
- computing the average difference between the sample means and the population mean and storing the result as `avg_diff_hawks`.

```
In [38]:  np.random.seed(40221) # for reproducibility

          # number of samples to simulate
          nsim = 1000

          # storage for the sample means
          samp_means_hawks = np.zeros(nsim)

          # repeatedly sample and store the sample mean in the samp_means array
          for i in range(nsim):
              samp_means_hawks[i] = population_hawks.sample(n=300, replace=False, weig

          # bias
          avg_diff_hawks = samp_means_hawks.mean() - population_hawks['length'].mean()

          print(avg_diff_hawks)

          -2.5720649894415715
```

```
In [39]:  grader.check("q10")
```

Out[39]:
q10 passed! ✨

Question 11

Reflect a moment on your simulation result in question 3c. If instead *female* mortality is higher and specimens for measurement are collected opportunistically, as described in the previous sampling design, do you expect that the average length in the sample will be an underestimate or an overestimate of the population mean? Explain why in 1-2 sentences, and carry out a simulation to check your intuition.

Overestimate, since the average diff is negative for men it would be the opposite for females. When the female has teh higher length they are much easier to be found so the estimate should be much higher.

```
In [ ]:
```

Bias correction

What can be done if a sampling design is biased? Is there any remedy?

You've seen some examples above of how bias can arise from a sampling mechanism in which units have unequal chances of being selected in the sample. Ideally, we'd work with random samples all the time, but that's not very realistic in practice. Fortunately, biased sampling is not a hopeless case -- **it is possible to apply bias corrections if you have good information about which individuals were more likely to be sampled**.

To illustrate how this would work, let's revisit the biased sampling in scenario 1 -- sampling larger eucalyptus seeds more often than smaller ones. Imagine you realize the mistake and conduct a quick experiment with your sieve to determine the proportion of seeds of each size that pass through, and use this to estimate the inclusion probabilities. (To simplify this excercise, we'll just use sampling weights we defined to calculate the actual inclusion probabilities.)

The cell below generates the population and sample from scenario 2 again:

```python
In [40]:    # simulate seed diameters
            np.random.seed(40221) # for reproducibility
            population3 = pd.DataFrame(
                data = {'diameter': np.random.gamma(shape = 2, scale = 1/2, size = 5000)
                        'seed': np.arange(5000)}
            ).set_index('seed')

            # probability of inclusion as a function of seed diameter
            def weight_fn(x, r = 2, c = 2):
                out = 1/(1 + np.e**(-r*(x - c)))
                return out

            # assign inclusion probability to each seed
            population3['samp_weight'] = weight_fn(population3.diameter)

            # draw weighted sample
            np.random.seed(40721)
            sample3 = population3.sample(n = 250, replace = False, weights = 'samp_weigh
```

The sample mean and population mean you calculated earlier are shown below:

```python
In [41]:    # print sample and population means
            pd.Series({'sample mean': sample3.diameter.mean(), 'population mean': popula
```

```
Out[41]:    sample mean         1.811721
            population mean      1.018929
            dtype: float64
```

We can obtain an unbiased estimate of the population mean by computing a *weighted average* of the diameter measurements instead of the sample average after weighting the measurements in inverse proportion to the sampling weights:

$$\text{weighted average} = \sum_{i=1}^{250} \underbrace{\left(\frac{w_i^{-1}}{\sum_{j=1}^{250} w_j^{-1}} \right)}_{\text{bias adjustment}} \times \text{diameter}_i$$

This might look a little complicated, but the idea is simple -- the weighted average corrects for bias by simply up-weighting observations with a lower sampling weight and down-weighting observations with a higher sampling weight.

The cell below performs this calcuation.

```
In [42]:  # compute bias adjustment
          sample3['bias_adjustment'] = (sample3.samp_weight**(-1))/(sample3.samp_weigh

          # weight diameter measurements
          sample3['weighted_diameter'] = sample3.diameter*sample3.bias_adjustment

          # sum to compute weighted average
          sample3.weighted_diameter.sum()
```

Out[42]: 1.0139201948221341

Notice that the weighted average successfully corrected for the bias:

```
In [43]:  # print sample and population means
          pd.Series({'sample mean': sample3.diameter.mean(),
                     'weighted average': sample3.weighted_diameter.sum(),
                     'population mean': population3.diameter.mean()})
```

Out[43]: sample mean   1.811721
          weighted average   1.013920
          population mean   1.018929
          dtype: float64

Question 12

Perform a bias correction for the hawk scenario, using the actual inclusion probabilities, in the following steps:

- redraw `sample_hawks_biased` exactly (use the same RNG seed) and retain the `weight` column
- compute the 'raw' sample mean and store the result as `sample_mean`
- compute a bias adjustment factor as illustrated above by normalizing the inverse weights, and append the adjustment factor as a new column in `sample_hawks_biased` named `bias_adjustment`
- compute a weighted average length and store the result as `sample_mean_corrected`

Print the raw and corrected sample means.

```
In [44]:  np.random.seed(40821)

          #random sample
          sample_hawks_biased = population_hawks.sample(n=300, replace = False, weight

          #size correction
          sample_hawks_biased['bias_adjustment'] = (sample_hawks_biased.weight ** (-1)

          #weiht lengths and compute sum
          sample_hawks_biased['length_weighted']=sample_hawks_biased.length * sample_h
          sample_mean_corrected=sample_hawks_biased.length_weighted.sum()

          #compare with unweighted
          sample_mean=sample_hawks_biased.length.mean()

          print('weighted mean', sample_mean_corrected)
          print('sample mean', sample_mean)
```

```
weighted mean 54.46787565637176
sample mean 51.88710627046723
```

```
In [45]:  grader.check("q12")
```

Out[45]:
q12 passed! 🚀

Takeaways

These simulations illustrate through a few simple examples that random sampling -- a sampling design where each unit is equally likely to be selected -- produces unbiased sample means. That means that 'typical samples' will yield sample averages that are close to the population value. By contrast, deviations from random sampling tend to yield biased sample averages -- in other words, nonrandom sampling tends to distort the statistical properties of the population in ways that can produce misleading conclusions (if uncorrected).

Here are a few key points to reflect on:

- bias is not a property of an individual sample, but of a *sampling design*

  - unbiased sampling designs tend to produce faithful representations of populations
  - but there are no guarantees for individual samples
- if you hadn't known the population distributions, there would have been no computational method to detect bias

  - in practice, it's necessary to *reason* about whether the sampling design is sound
- the sample statistic (sample mean) was only reliable when the sampling design was sound

  - the quality of data collection is arguably more important for reaching reliable conclusions than the choice of statistic or method of analysis

Submission

1. Save the notebook.
2. Restart the kernel and run all cells. (**CAUTION**: if your notebook is not saved, you will lose your work.)
3. Carefully look through your notebook and verify that all computations execute correctly. You should see **no errors**; if there are any errors, make sure to correct them before you submit the notebook.
4. Download the notebook as an `.ipynb` file. This is your backup copy.
5. Export the notebook as PDF and upload to Gradescope.

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [46]: grader.check_all()
```

Out[46]: q1 results: All test cases passed!

q10 results: All test cases passed!

q12 results: All test cases passed!

q2 results: All test cases passed!

q3 results: All test cases passed!

q4 results: All test cases passed!

q6 results: All test cases passed!

q8 results: All test cases passed!

q9 results: All test cases passed!