In [26]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab2-sampling.ipynb")
```

In [27]:
```
import numpy as np
import pandas as pd
import altair as alt
```

Lab 2: Sampling design and statistical bias

In the following scenarios 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.

This lab has you explore sampling bias. The goal 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 design

The **sampling design** of a study refers to ***the way observational units are selected*** from the sampling frame (the collection of all observable 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 reasons to study adults (sampling frame), and decide to do a mail survey of 2000 randomly selected addresses in each state (sampling design). (This is not a random sample of all individuals 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 (observed) from its population counterpart (unobserved)***.

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 rarely (almost 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 under repeated sampling.

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 for the present topic ***because you can control the population properties***.

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: unbiased samples

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 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 [28]:
```
# 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[28]:

	diameter
seed	
0	0.831973
1	1.512187
2	0.977392

Question 1a

Calculate the mean diameter for the hypothetical population and set it to `mean_diameter` .

In [29]:
```
# solution
mean_pop_diameter = population.mean()

mean_pop_diameter
```

Out[29]:
```
diameter    1.018929
dtype: float64
```

In [30]:
```
grader.check("q1_a")
```

Out[30]: **q1_a** passed!

Question 1b

Calculate the standard deviation of diameters for the hypothetical population and set the result to `std_dev_pop_diameter` .

In [31]:
```python
# solution
std_dev_pop_diameter = population.std()

std_dev_pop_diameter
```

Out[31]:
```
diameter    0.72393
dtype: float64
```

In [32]:
```python
grader.check("q0_b")
```

Out[32]: **q0_b** 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.

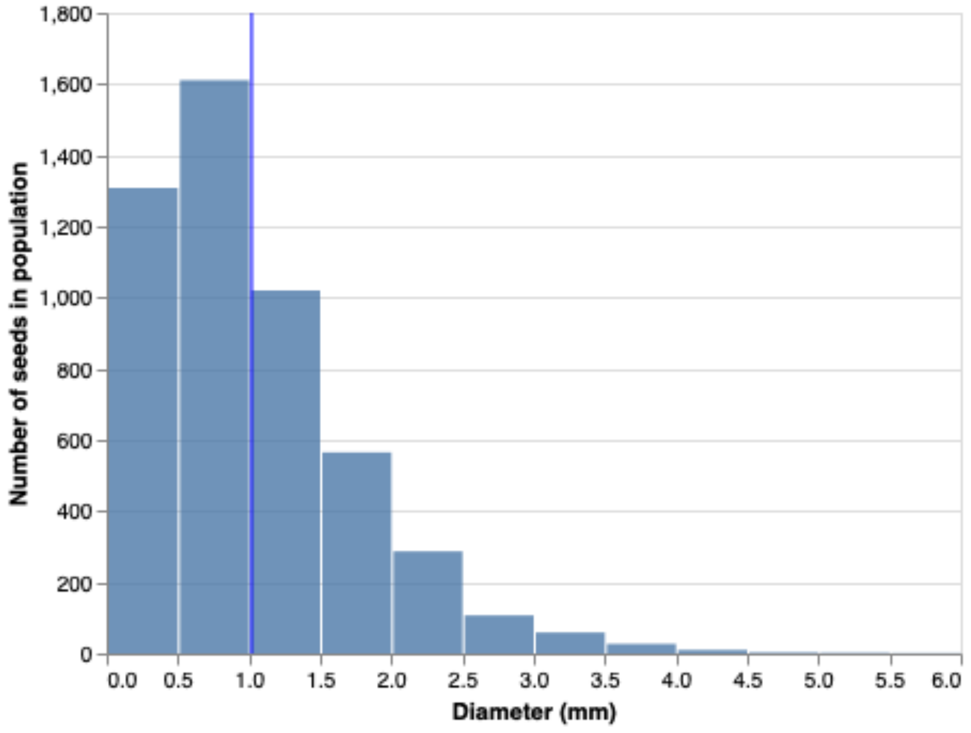
In [33]:
```python
# base layer
base_pop = alt.Chart(population).properties(width = 400, height = 300)

# 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
```

Out[33]:



Hypothetical sampling design

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 [34]:
```
# draw a random sample of seeds
np.random.seed(40221) # for reproducibility
sample = population.sample(n = 250, replace = False)
```

Question 1c.i

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

```
In [35]: # solution
         mean_sample_diameter = sample.mean()

         mean_sample_diameter
         #sample mean diameter is larger therefore it is an underestimation

Out[35]: diameter    0.977722
         dtype: float64
```

```
In [36]: grader.check("q1_c_i")

Out[36]: q1_c_i passed!
```

Question 1c.ii

Is it close to the population mean?

Type your answer here, replacing this text.

The cell below produces a histogram of the sample values, and displays it alongside the histogram of population values.

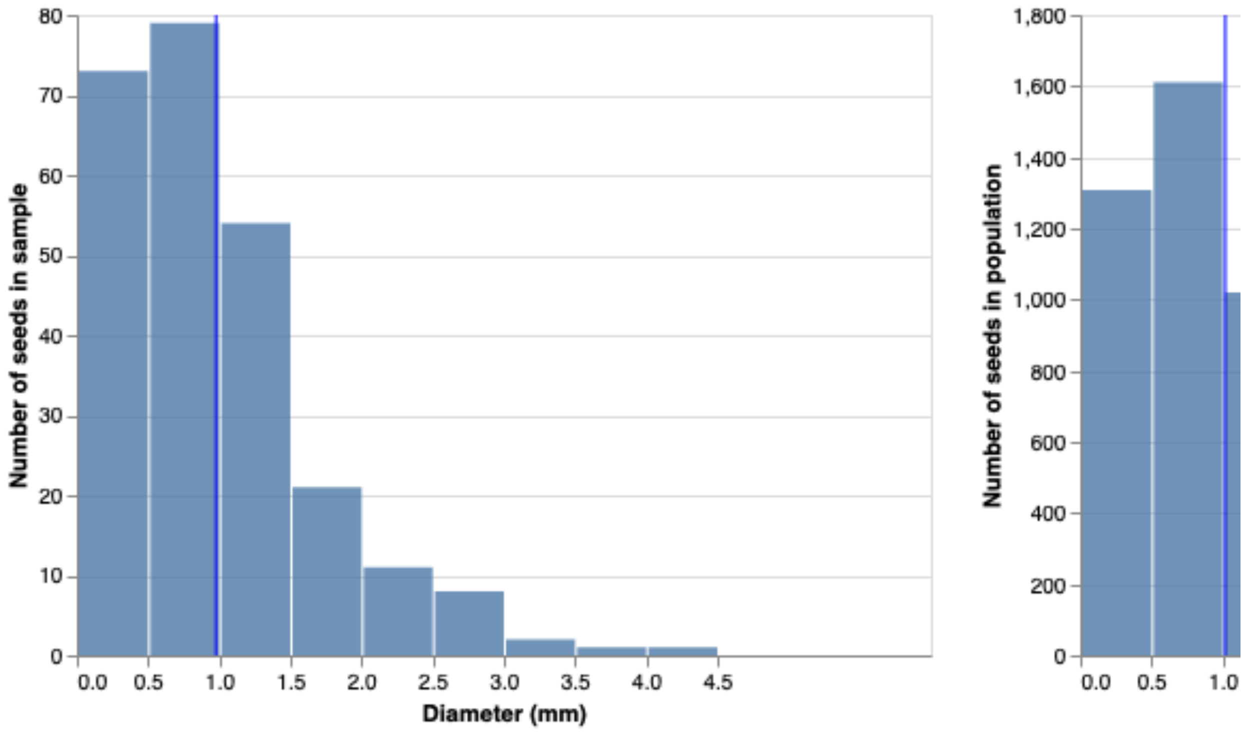
In [37]:
```python
# base layer
base_samp = alt.Chart(sample).properties(width = 400, height = 300)

# 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
#strictly sampling randomly so there is no bias
```

Out[37]:



Notice that 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

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 to see whether the undistorted representation of the population is typical for this sampling design. To simplify life a little, let's focus on whether the sample mean is usually accurate.

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 [38]:
```
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 [39]:
```
# bias
samp_means.mean() - population.diameter.mean()
```

Out[39]:
```
-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*.

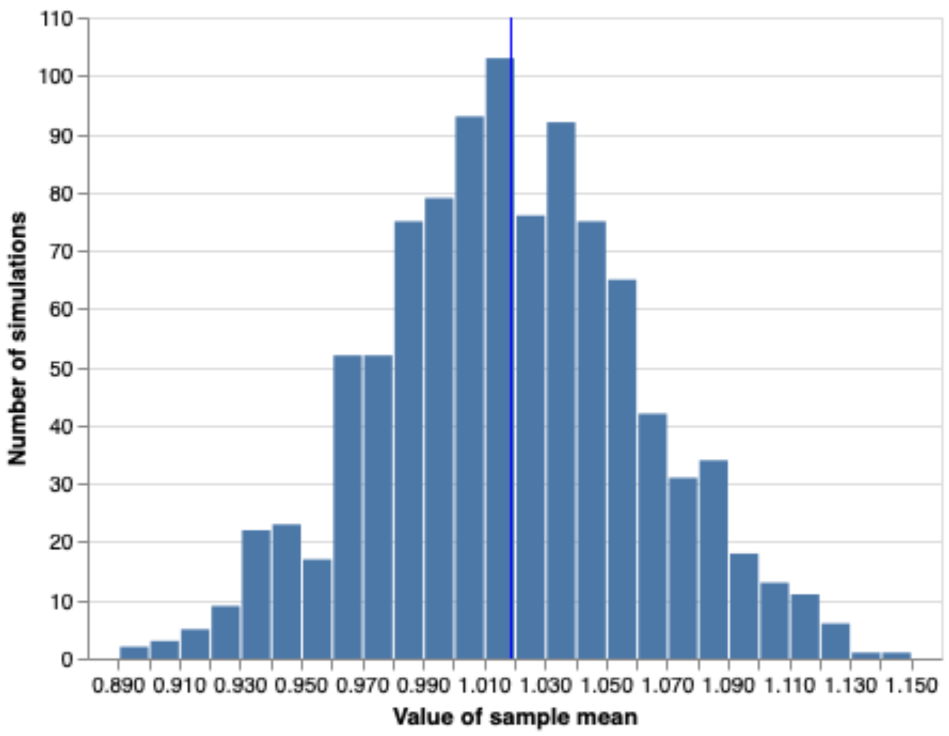
However, **unbiasedness does not mean that you won't observe estimation error**. There is a natural amount of variability from sample to sample, because in each sample a different collection of seeds is measured.

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 [40]:
```
# plot the simulated sampling distribution
sampling_dist = alt.Chart(pd.DataFrame({'sample mean': samp_means})).m
ark_bar().encode(
    x = alt.X('sample mean', bin = alt.Bin(maxbins = 30), title = 'Val
ue of sample mean'),
    y = alt.Y('count()', title = 'Number of simulations')
)

sampling_dist + mean_pop
```

Out[40]:



Scenario 2: 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.

Hypothetical 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* $w_1, \dots, w_N$ to each observation. The cell below defines a `weight_fn` that calculates a weight w_i between 0 and 1 according to diameter, so that larger diameters have larger weights and are more likely to be sampled.

In [41]:
```
population_mod1 = population.copy()
```

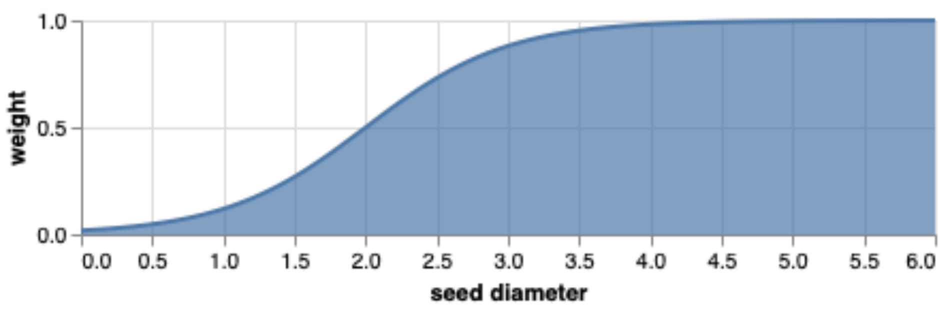
```
In [42]: # inclusion weight as a function of seed diameter
         def weight_fn(x, r = 2, c = 2):
             out = 1/(1 + np.e**(-r*(x - c)))
             return out

         # 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 = Tru
         e).encode(
             x = 'seed diameter',
             y = 'weight'
         ).properties(height = 100)

         # show plot
         weight_plot
```

Out[42]:



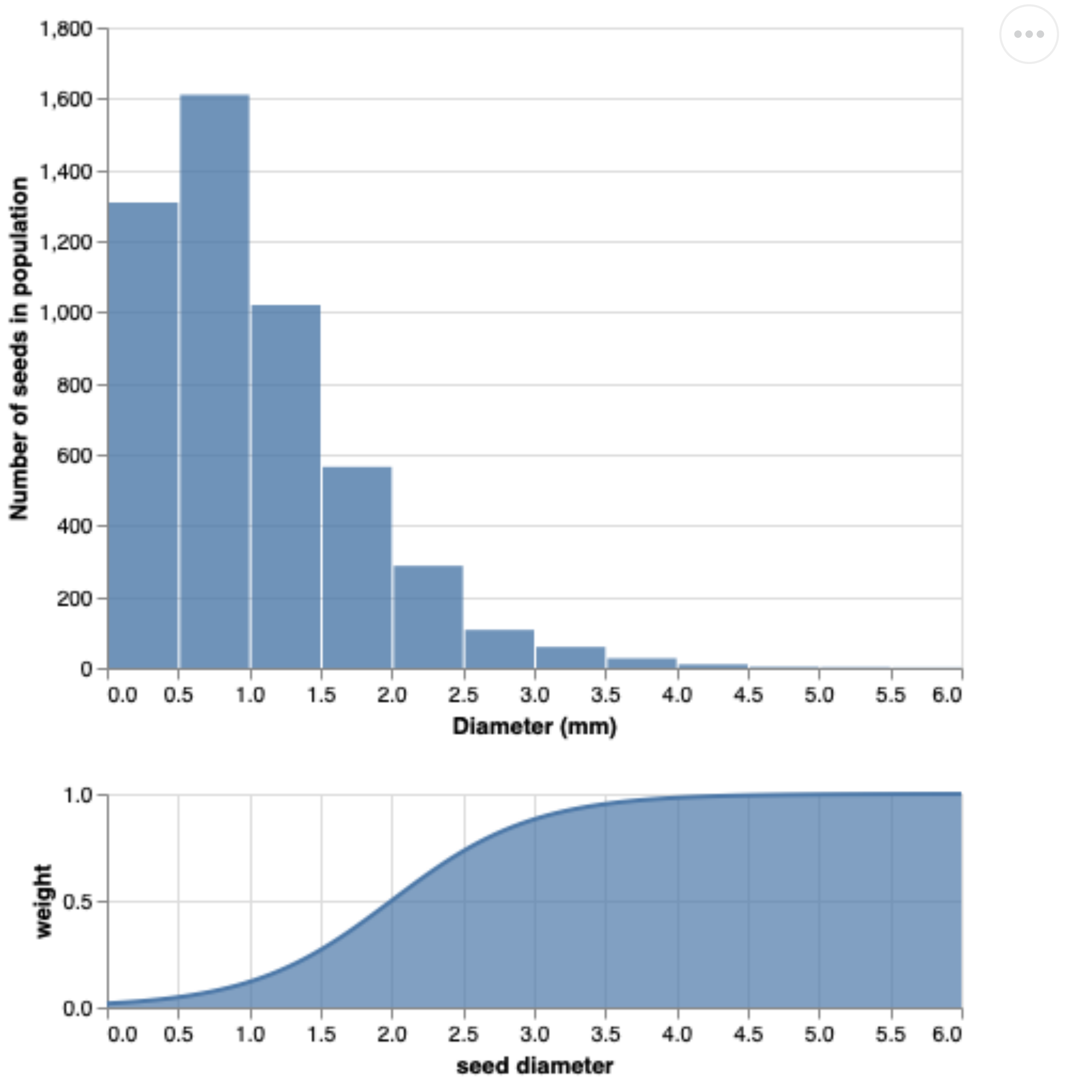
The actual probability that a seed is included in the sample -- its **inclusion probability** -- is proportional to the sampling weight. These inclusion probabilities π_i can be calculated by normalizing the weights w_i:

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

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 [43]: hist_pop & weight_plot
```

Out[43]:



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 [44]: # assign inclusion probability to each seed
population_mod1['inclusion_prob'] = weight_fn(population_mod1.diameter
)/(weight_fn(population_mod1.diameter)).sum()

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

Question 2a.i

Calculate the mean diameter of seeds in the simulated sample and set the result to `mean_sample2_diameter` .

```
In [45]: # solution
         mean_sample2_diameter = sample2.mean()

         mean_sample2_diameter
```

```
Out[45]: diameter    1.811721
         dtype: float64
```

```
In [46]: grader.check("q2_a_i")
```

Out[46]: **q2_a_i** passed!

Question 2a.ii

Is it close to the population mean?

Compared to the random sampled data, this calculated mean is further away from the population mean. It is also much greater than the population mean.

Question 2b.i

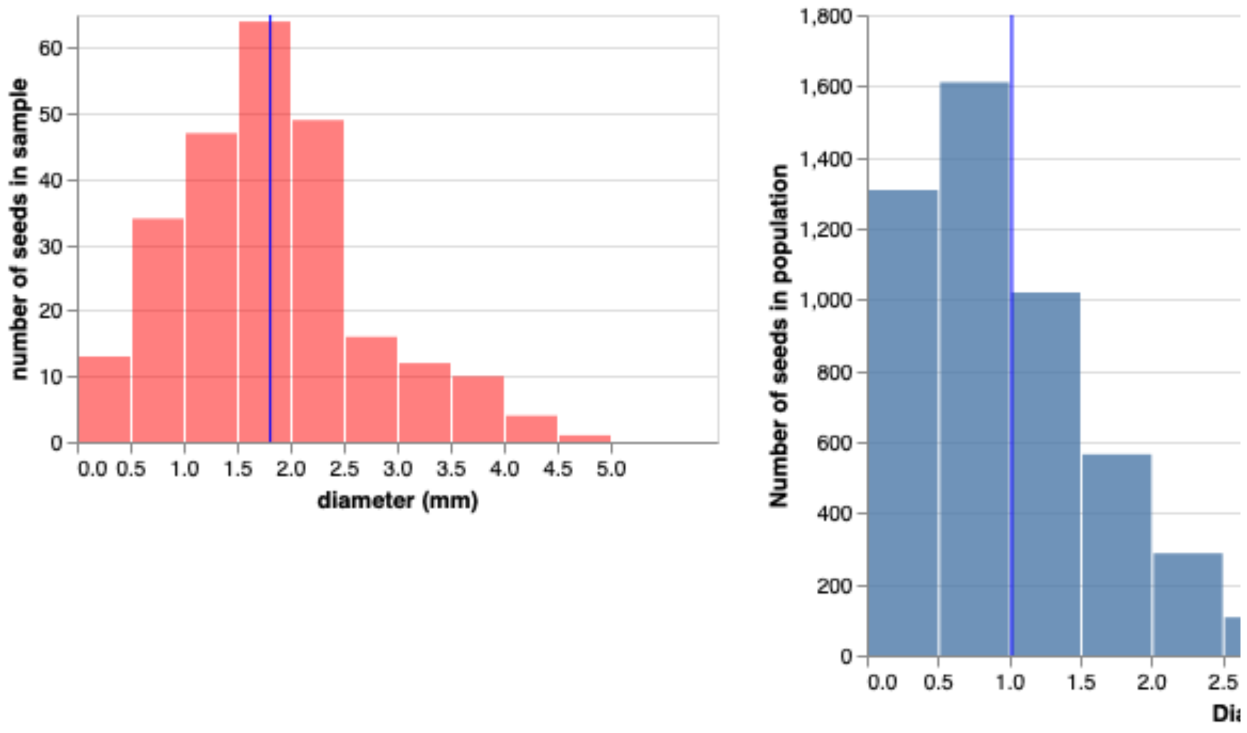
Show side-by-side plots of the distribution of sample values and the distribution of population values, with vertical lines indicating the corresponding mean on each plot. (*Hint*: copy the cell that produced this plot in scenario 1 and replace `sample` with `sample2` . Utilizing different methods is also welcome.)

In [47]:
```
# base layer
base_samp = alt.Chart(sample2).properties(width = 300, height = 200)

# histogram of diameter measurements
hist_samp = base_samp.mark_bar(opacity = 0.5, color = 'red').encode(
    x = alt.X('diameter',
              bin = alt.Bin(maxbins = 20),
              scale = alt.Scale(domain = (0, 6)),
              title = 'diameter (mm)'),
    y = alt.Y('count()',
              stack = None,
              title = 'number of seeds in sample')
)

# vertical line for population mean
mean_samp = base_samp.mark_rule(color='blue').encode(
    x = 'mean(diameter)'
)
# combine layers
hist_samp + mean_samp | hist_pop + mean_pop
```

Out[47]:



Question 2b.ii

Compare this plot with the original histogram that displays the distribution for the entire population. Does the distribution of diameters of seeds in the sample seem to accurately reflect the population?

The distribution of the diameter of seeds in the sample does not seem to match or reflect that of the population.

Assessing bias

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

In [48]:
```
population_mod1.head()
```

Out[48]:

seed	diameter	inclusion_prob
0	0.831973	0.000097
1	1.512187	0.000300
2	0.977392	0.000126
3	2.874944	0.000935
4	0.506508	0.000053

Question 2c

Investigate the bias of the sample mean by:

- drawing 1000 samples with observations weighted by inclusion probability;
- storing the sample mean from each sample in `samp_means` ;
- computing the average difference between the sample means and the population mean and storing the result in `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 [49]:
```
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 in the samp_means array
for i in range(0, nsim):
    samp_means[i] = population_mod1.sample(n = 250, replace = False, w
eights = 'inclusion_prob').drop(columns='incusion_prob').mean()

# bias
avg_diff = samp_means.means() - population_mod1.diameter.means()
```

```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent cal
l last)
<ipython-input-49-bac54f281b2a> in <module>
      8 # repeatedly sample and store the sample mean in the samp_me
ans array
      9 for i in range(0, nsim):
---> 10     samp_means[i] = population_mod1.sample(n = 250, replace
= False, weights = 'inclusion_prob').drop(columns='incusion_prob').m
ean()
     11 
     12 # bias

/opt/conda/lib/python3.7/site-packages/pandas/util/_decorators.py in
wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312 
    313         return wrapper

/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py in drop(
self, labels, axis, index, columns, level, inplace, errors)
   4911             level=level,
   4912             inplace=inplace,
-> 4913             errors=errors,
   4914         )
   4915 

/opt/conda/lib/python3.7/site-packages/pandas/core/generic.py in dro
p(self, labels, axis, index, columns, level, inplace, errors)
   4148         for axis, labels in axes.items():
   4149             if labels is not None:
-> 4150                 obj = obj._drop_axis(labels, axis, level=lev
el, errors=errors)
   4151 
   4152         if inplace:
```

```
/opt/conda/lib/python3.7/site-packages/pandas/core/generic.py in _dr
op_axis(self, labels, axis, level, errors)
   4183                 new_axis = axis.drop(labels, level=level,
errors=errors)
   4184             else:
-> 4185                 new_axis = axis.drop(labels, errors=errors)
   4186             result = self.reindex(**{axis_name: new_axis})
   4187

/opt/conda/lib/python3.7/site-packages/pandas/core/indexes/base.py
in drop(self, labels, errors)
   6015         if mask.any():
   6016             if errors != "ignore":
-> 6017                 raise KeyError(f"{labels[mask]} not found in
axis")
   6018             indexer = indexer[~mask]
   6019         return self.delete(indexer)

KeyError: "['incusion_prob'] not found in axis"
```

```
In [50]: grader.check("q2_c")
```

Out[50]: **q2_c results:**

q2_c - 1 result:

```
    Trying:
        avg_diff > 0
    Expecting:
        True
    *****************************************************************
******
    Line 1, in q2_c 0
    Failed example:
        avg_diff > 0
    Exception raised:
        Traceback (most recent call last):
          File "/opt/conda/lib/python3.7/doctest.py", line 1337, in
__run
            compileflags, 1), test.globs)
          File "", line 1, in
            avg_diff > 0
        NameError: name 'avg_diff' is not defined
```

q2_c - 2 result:

```
    Test case passed!
```

q2_c - 3 result:

```
    Trying:
        samp_means.mean() > 1
    Expecting:
        True
    *****************************************************************
******
    Line 1, in q2_c 2
    Failed example:
        samp_means.mean() > 1
    Expected:
        True
    Got:
        False
```

Question 2d

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

There seems to be quite a lot of bias in the sampling design. The sample mean tends to over estimate the population mean.

Scenario 3

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 [51]:
```
# for reproducibility
np.random.seed(40721)

# simulate hypothetical population
population_hawks = pd.DataFrame(
    data = {'length': np.random.normal(loc = 57.5, scale = 3, size = 3
000),
            'sex': np.repeat('female', 3000)}
).append(
    pd.DataFrame(
        data = {'length': np.random.normal(loc = 50.5, scale = 3, size
= 2000),
                'sex': np.repeat('male', 2000)}
    )
)

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

Out[51]:

	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).

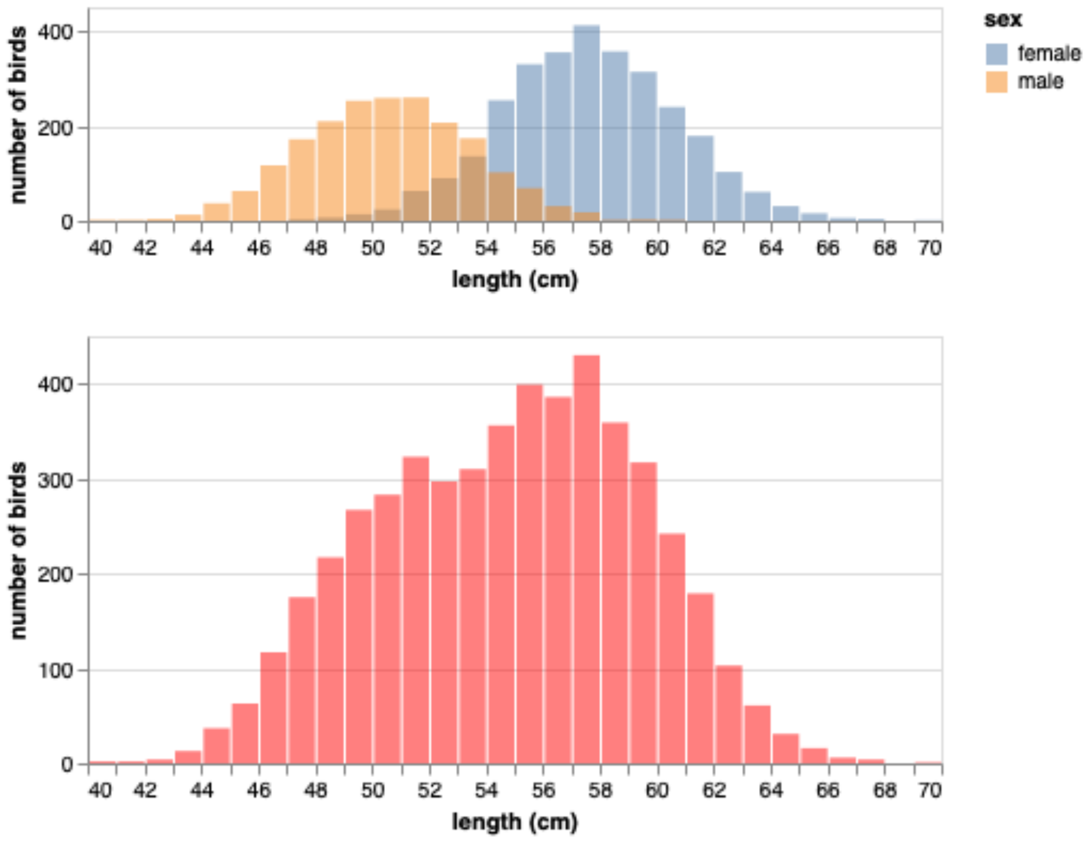
In [52]:
```python
base = alt.Chart(population_hawks).properties(height = 200)

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[52]:



The population mean -- average length of both female and male red-tailed hawks -- is shown below.

In [53]:
```
# population mean
population_hawks.mean()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:2: Futu
reWarning: Dropping of nuisance columns in DataFrame reductions (wit
h 'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the reduc
tion.
```

Out[53]:
```
length    54.737717
dtype: float64
```

First try drawing a random sample from the population:

In [54]:
```
# for reproducibility
np.random.seed(40821)

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

Question 3a

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 [55]:
```
# solution
proportion = sample_hawks.groupby('sex').count()/300
proportion_hawks_sample = proportion.rename(columns = {'length':'propo
rtion'})

proportion_hawks_sample
```

Out[55]:

	proportion
sex	
female	0.596667
male	0.403333

```
In [56]: grader.check("q3_a")
```

```
Out[56]: q3_a 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 [57]: sample_hawks.mean()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: Futu
reWarning: Dropping of nuisance columns in DataFrame reductions (wit
h 'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the reduc
tion.
  """Entry point for launching an IPython kernel.
```

```
Out[57]: 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.

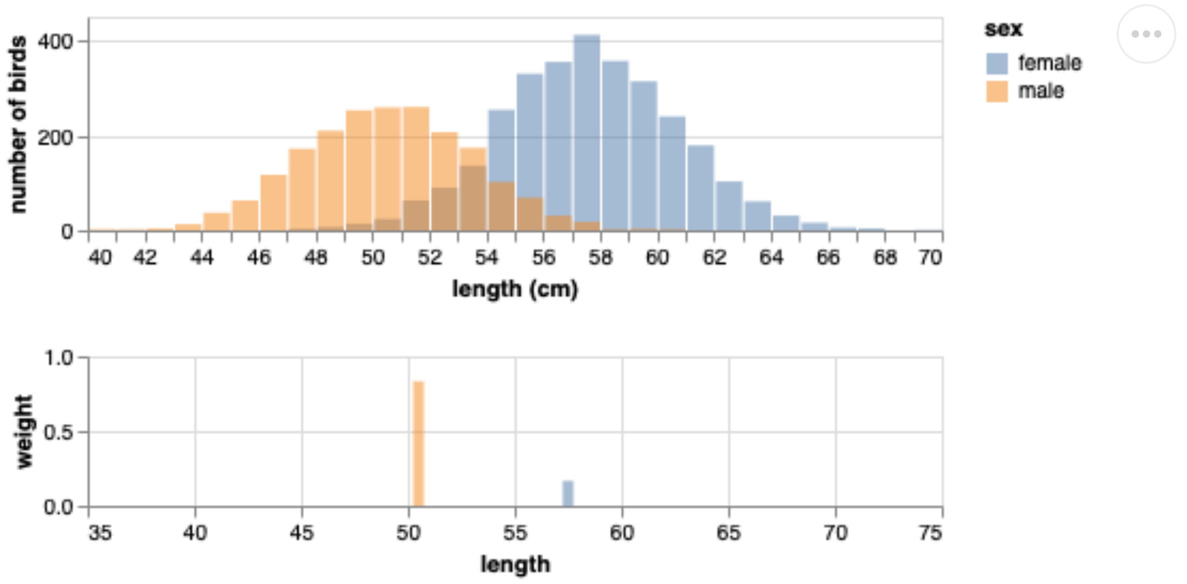
In [58]:
```
def weight_fn(sex, p = 5/6):
    if sex == 'male':
        out = p
    else:
        out = 1 - p
    return out

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
```

Out[58]:



Question 3b

Draw a weighted sample `sample_hawks_weighted` from the population `population_hawks` using the weights defined by `weight_fn`, and compute and store the sample mean in `sample_hawks_weighted_mean`.

In [59]:
```
# for reproducibility
np.random.seed(40821)

# assign weights
population_hawks['weight'] = population_hawks.sex.aggregate(func = wei
ght_fn)

# randomly sample
sample_hawks_weighted =  population_hawks.sample(n = 300, replace = Fa
lse, weights = 'weight').drop(columns = 'weight')

# compute mean
sample_hawks_weighted_mean = sample_hawks_weighted.mean()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:11: Fut
ureWarning: Dropping of nuisance columns in DataFrame reductions (wi
th 'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the reduc
tion.
  # This is added back by InteractiveShellApp.init_path()
```

In [60]:
```
grader.check("q3_b")
```

Out[60]: **q3_b** passed!

Question 3c

Investigate the bias of the sample mean by:

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

In [61]:
```
# solution
np.random.seed(40221) # for reproducibility

# number of samples to simulate
nsim = 1000
# storage for the sample means
samp_means_2 = np.zeros(nsim)

# repeatedly sample and store the sample mean in the samp_means array
for i in range(nsim):
    samp_means[i] = population_hawks.sample(n = 300, replace = False,
weights = 'weight').drop(columns = 'weight').mean()
# bias
biased_mean_diff = samp_means.mean() - population_hawks.length.mean()
biased_mean_diff
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:11: Fut
ureWarning: Dropping of nuisance columns in DataFrame reductions (wi
th 'numeric_only=None') is deprecated; in a future version this will
raise TypeError.  Select only valid columns before calling the reduc
tion.
  # This is added back by InteractiveShellApp.init_path()
```

Out[61]:
```
-2.5720649894415715
```

In [62]:
```
grader.check("q3_c")
```

Out[62]: **q3_c results:**

q3_c - 1 result:

```
    Test case passed!
```

q3_c - 2 result:

```
    Test case passed!
```

q3_c - 3 result:

```
    Trying:
        (samp_means_2.mean() > 40) and (samp_means_2.mean() < 60)
    Expecting:
        True
    *****************************************************************
******
    Line 1, in q3_c 2
    Failed example:
        (samp_means_2.mean() > 40) and (samp_means_2.mean() < 60)
    Expected:
        True
    Got:
        False
```

Question 3d

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.

I would expect the current sample mean to be an underestimate of the overall population mean because in our previous histograms and statistics calculated for the whole population of hawks, females tend to be larger than males.

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 scenario 2 -- 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:

In [63]:
```python
# 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_weight')
```

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

In [64]:
```python
# print sample and population means
pd.Series({'sample mean': sample3.diameter.mean(), 'population mean': population3.diameter.mean()})
```

Out[64]:
```
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 [65]:
```
# compute bias adjustment
sample3['bias_adjustment'] = (sample3.samp_weight**(-1))/(sample3.samp
_weight**(-1)).sum()

# weight diameter measurements
sample3['weighted_diameter'] = sample3.diameter*sample3.bias_adjustmen
t

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

Out[65]:
```
1.0139201948221341
```

Notice that the weighted average successfully corrected for the bias:

In [66]:
```
# print sample and population means
pd.Series({'sample mean': sample3.diameter.mean(),
           'weighted average': sample3.weighted_diameter.sum(),
           'population mean': population3.diameter.mean()})
```

Out[66]:
```
sample mean         1.811721
weighted average    1.013920
population mean     1.018929
dtype: float64
```

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 Checklist

1. Save file to confirm all changes are on disk
2. Run *Kernel* > *Restart & Run All* to execute all code from top to bottom
3. Save file again to write any new output to disk
4. Select *File* > *Download* (should save as .ipynb)
5. Submit to Gradescope

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

```
In [67]: grader.check_all()
```

```
Out[67]: q0_b results: All test cases passed!

         q1_a results: All test cases passed!
```

```
q1_c_i results: All test cases passed!

q2_a_i results: All test cases passed!

q2_c results:
    q2_c - 1 result:
        Trying:
            avg_diff > 0
        Expecting:
            True
        *******************************************************************
*********
        Line 1, in q2_c 0
        Failed example:
            avg_diff > 0
        Exception raised:
            Traceback (most recent call last):
              File "/opt/conda/lib/python3.7/doctest.py", line 1337,
in __run
                compileflags, 1), test.globs)
              File "<doctest q2_c 0[0]>", line 1, in <module>
                avg_diff > 0
            NameError: name 'avg_diff' is not defined

    q2_c - 2 result:
        Test case passed!

    q2_c - 3 result:
        Test case passed!

q3_a results: All test cases passed!

q3_b results: All test cases passed!

q3_c results:
    q3_c - 1 result:
        Test case passed!

    q3_c - 2 result:
        Test case passed!

    q3_c - 3 result:
        Trying:
            (samp_means_2.mean() > 40) and (samp_means_2.mean() < 60
)
        Expecting:
            True
        *******************************************************************
*********
        Line 1, in q3_c 2
        Failed example:
            (samp_means_2.mean() > 40) and (samp_means_2.mean() < 60
)
        Expected:
            True
        Got:
            False
```

```
In [ ]:
```