lab11

May 17, 2022

Name: Allan Gongora

Section: 0131

1 Lab 11: Variance of Sample Means and Correlation

Welcome to Lab 11!

In this lab we will learn about the variance of sample means as well as ways to understand and quantify the association between two variables.

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests.

[1]:
```
pip install gofer-grader
```

```
Requirement already satisfied: gofer-grader in /opt/conda/lib/python3.7/site-
packages (1.1.0)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
packages (from jinja2->gofer-grader) (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# Run this cell, but please don't change it.

# These lines import the Numpy and Datascience modules.
import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
```

```
import warnings
warnings.simplefilter('ignore', FutureWarning)
warnings.simplefilter('ignore', UserWarning)

# These lines load the tests.
from gofer.ok import check
```

Recommended Reading: * Variability of the Sample Mean * Correlation

1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include:
   A) Sentence reponses to questions that ask for an explanation
   B) Numeric responses to multiple choice questions
   C) Programming code
2) Moreover, throughout this lab and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Otherwise, you will fail tests that you thought you were passing previously!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. How Faithful is Old Faithful?

(Note: clever title comes from here.)

Old Faithful is a geyser in Yellowstone National Park in Central United States. It's famous for erupting on a fairly regular schedule. You can see a video below.

[3]:
```
# For the curious: this is how to display a YouTube video in a
# Jupyter notebook.  The argument to YouTubeVideo is the part
# of the URL (called a "query parameter") that identifies the
# video.  For example, the full URL for this video is:
#   https://www.youtube.com/watch?v=wE8NDuzt8eg
from IPython.display import YouTubeVideo
YouTubeVideo("wE8NDuzt8eg")
```

[3]:

Some of Old Faithful's eruptions last longer than others. When it has a long eruption, there's generally a longer wait until the next eruption.

If you visit Yellowstone, you might want to predict when the next eruption will happen, so you can see the rest of the park and come to see the geyser when it erupts. To predict one variable from another, the first step is to understand the association between them.

The dataset has one row for each observed eruption. It includes the following columns: - **Duration**: Eruption duration, in minutes - **Wait**: Time between this eruption and the next, also in minutes

Run the next cell to load the dataset.

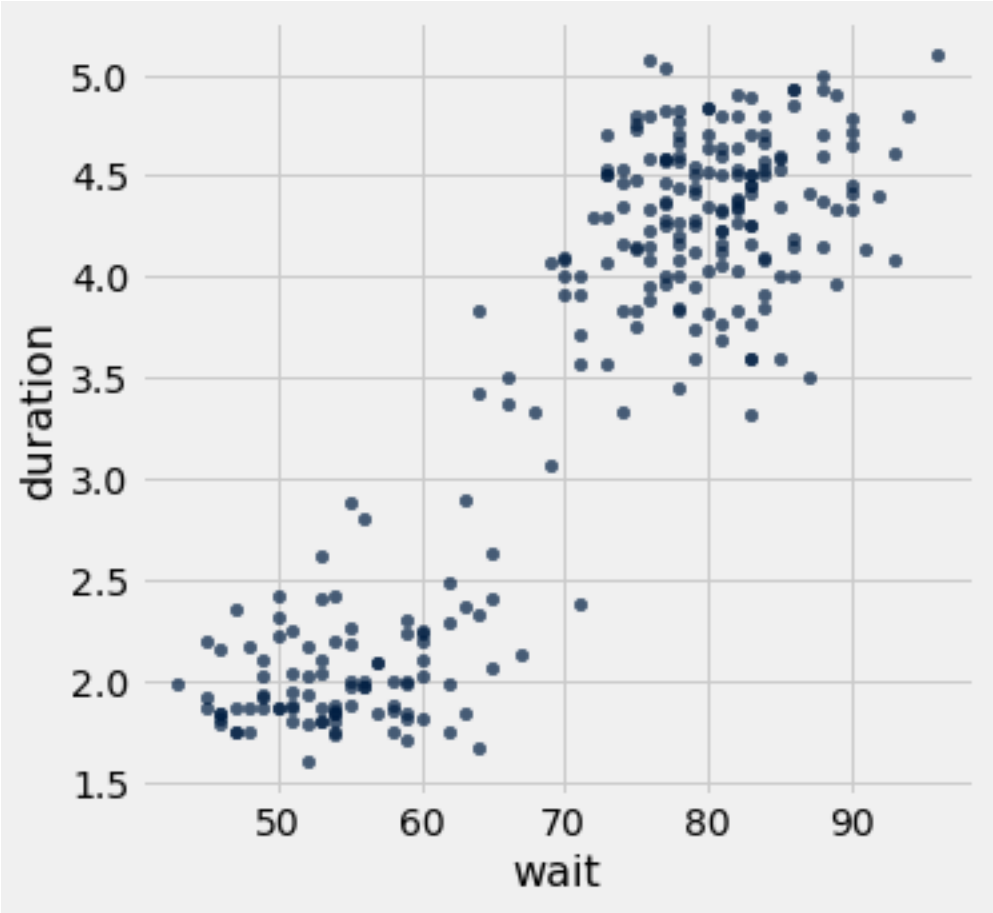
[4]:
```
faithful = Table.read_table("faithful.csv")
faithful
```

[4]:
```
duration | wait
3.6      | 79
1.8      | 54
3.333    | 74
2.283    | 62
4.533    | 85
2.883    | 55
4.7      | 88
```

```
3.6      | 85
1.95     | 51
4.35     | 85
… (262 rows omitted)
```

Question 1.1 Make a scatter plot of the data. It's conventional to put the column we will try to predict on the vertical axis and the other column on the horizontal axis.

[5]:
```
faithful.scatter("wait")
```



Look at the scatter plot. Does the association between wait times and eruption durations appear to be linear?

There's more going on than just a linear association. The eruption durations seem to cluster; there are a bunch of short eruptions and a bunch of longer ones. Within each of the clusters, these values appear to be roughly linearly correlated, but perhaps with a different correlation coefficient.

The overall relationship is positive, which means that longer eruptions have longer waiting times. Even when the association is more nuanced than a simple linear association, we can still compute the correlation.

First, we'll plot the data in standard units. Recall that, if `nums` is an array of numbers, then

`(nums - np.mean(nums)) / np.std(nums)`

is an array of those numbers in standard units.

Question 1.2 Compute the mean and standard deviation of the eruption durations and waiting times. **Then**, create a table called `faithful_standard` containing the eruption durations and waiting times in standard units. (The columns should be named `"duration (standard units)"` and `"wait (standard units)"`.)

2 Why doesn't this work

[6]:
```
duration_mean = np.mean(faithful["duration"]) # .mean()
duration_std = np.std(faithful["duration"]) # .std()
wait_mean = np.mean(faithful["wait"]) # why does .mean() not pass the test
wait_std = np.std(faithful["wait"]) # why does .std() not pass the test
# NOTHING PASSES THE TEST

faithful_standard = Table().with_columns(
    "duration (standard units)", (faithful["duration"] - duration_mean) /␣
 ↪duration_std,
    "wait (standard units)", (faithful["wait"] - wait_mean) / wait_std)
faithful_standard
```

[6]:
duration (standard units)	wait (standard units)
0.0984989	0.597123
-1.48146	-1.24518
-0.135861	0.228663
-1.0575	-0.655644
0.917443	1.03928
-0.530851	-1.17149
1.06403	1.26035
0.0984989	1.03928
-1.3498	-1.46626
0.756814	1.03928

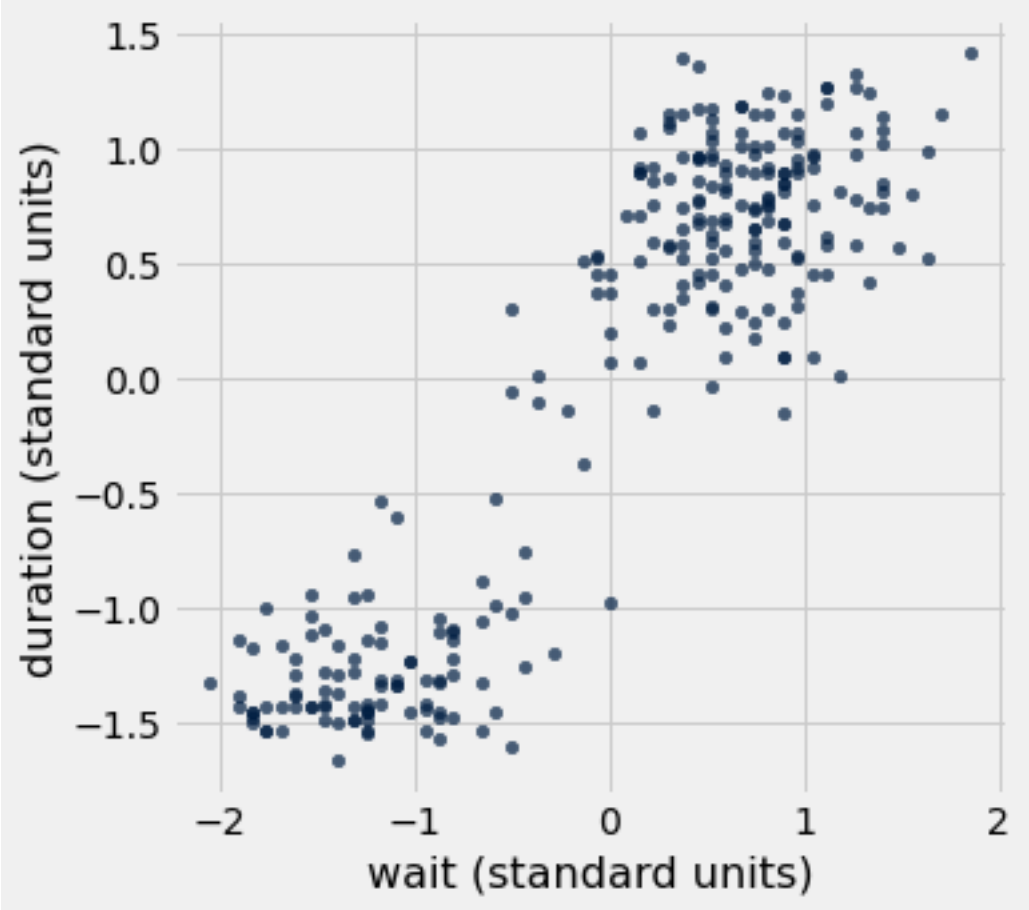
… (262 rows omitted)

[7]:
```
check('tests/q1_2.py')
```

[7]:
```
<gofer.ok.OKTestsResult at 0x7f8fedc91850>
```

Question 1.3 Plot the data again, but this time in standard units.

[8]:
```
faithful_standard.scatter("wait (standard units)")
```



You'll notice that this plot looks exactly the same as the last one! The data really are different, but the axes are scaled differently. (The method `scatter` scales the axes so the data fill up the available space.) So it's important to read the ticks on the axes.

Question 1.4 Among the following numbers, which would you guess is closest to the correlation between eruption duration and waiting time in this dataset?

- -1
- 0
- 1

Assign your answer to `closest_correlation`.

[9]:
```
closest_correlation = 1
```

[10]:
```
check('tests/q1_4.py')
```

[10]:
```
<gofer.ok.OKTestsResult at 0x7f8fedc91610>
```

Question 1.5 Compute the correlation `r`.

Hint: Use `faithful_standard`. Section 15.1 explains how to do this.

[11]:
```
r = np.mean(faithful_standard["duration (standard units)"] *␣
 ↪faithful_standard["wait (standard units)"])
r
```

[11]:
```
0.9008111683218132
```

[12]:
```
check('tests/q1_5.py')
```

[12]:
```
<gofer.ok.OKTestsResult at 0x7f8feb9a7e10>
```

2.1 2. Variability of the Sample Mean

By the Central Limit Theorem, the probability distribution of the mean of a large random sample is roughly normal. The bell curve is centered at the population mean. Some of the sample means are higher, and some lower, but the deviations from the population mean are roughly symmetric on either side, as we have seen repeatedly. Formally, probability theory shows that the sample mean is an unbiased estimate of the population mean.

In our simulations, we also noticed that the means of larger samples tend to be more tightly clustered around the population mean than means of smaller samples. In this section, we will quantify the variability of the sample mean and develop a relation between the variability and the sample size.

Let's take a look at the salaries of employees of the City of San Francisco in 2014. The mean salary reported by the city government was about $75,463.92.

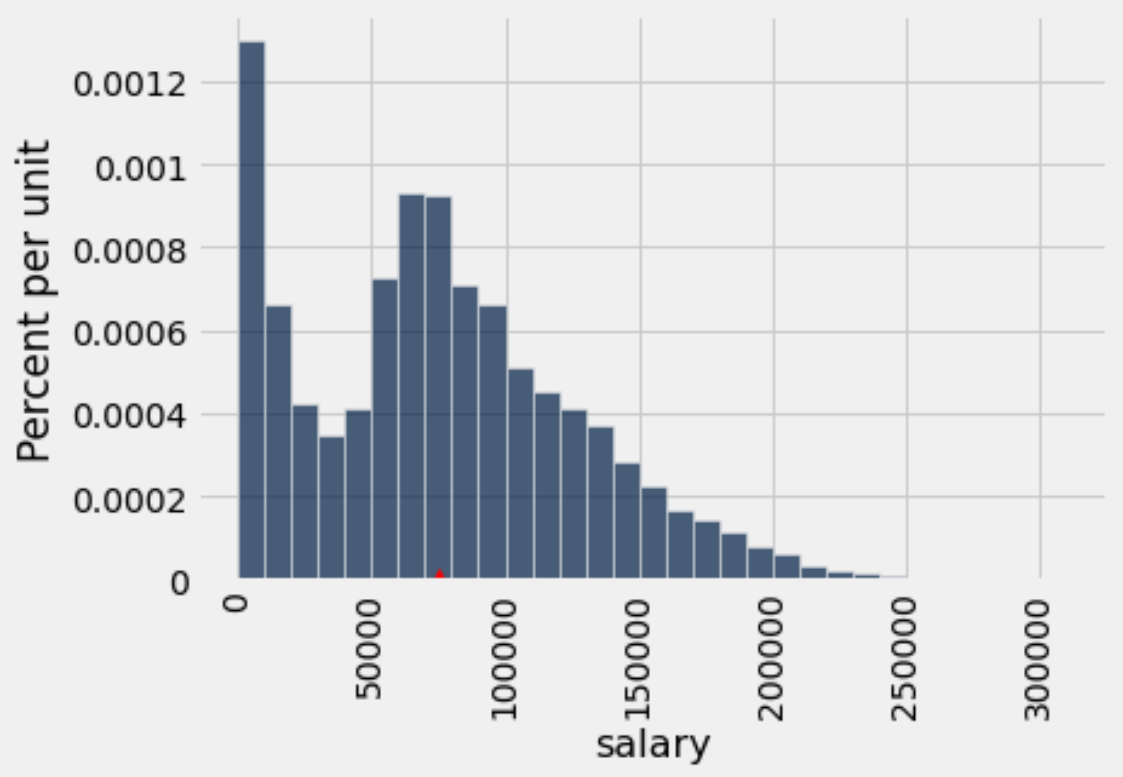
[13]:
```
salaries = Table.read_table('sf_salaries_2014.csv').select("salary")
salaries
```

[13]:
```
salary
471953
390112
339654
326717
326233
344187
311299
310161
335485
329391
… (38113 rows omitted)
```

[14]:
```
salary_mean = np.mean(salaries.column('salary'))
salary_mean
```

[14]:
```
75463.91814023031
```

[15]:
```
salaries.hist('salary', bins=np.arange(0, 300000+10000*2, 10000))
plots.scatter(salary_mean, 0, marker='^', color='red', s=100);
```



Question 2.1 Clearly, the population does not follow a normal distribution. Keep that in mind as we progress through these exercises.

Let's take random samples and look at the probability distribution of the sample mean. As usual, we will use simulation to get an empirical approximation to this distribution.

We will define a function `simulate_sample_mean` to do this, because we are going to vary the sample size later. The arguments are the name of the table, the label of the column containing the variable, the sample size, and the number of simulations.

Complete the function `simulate_sample_mean`. It will not be graded, but if you haven't implemented it correctly, the rest of the lab won't work properly, so this step is crucial.

[16]:
```
"""Empirical distribution of random sample means"""

def simulate_sample_mean(table, label, sample_size, repetitions):

    means = []

    for i in np.arange(repetitions):
        new_sample = table.sample(sample_size)
```

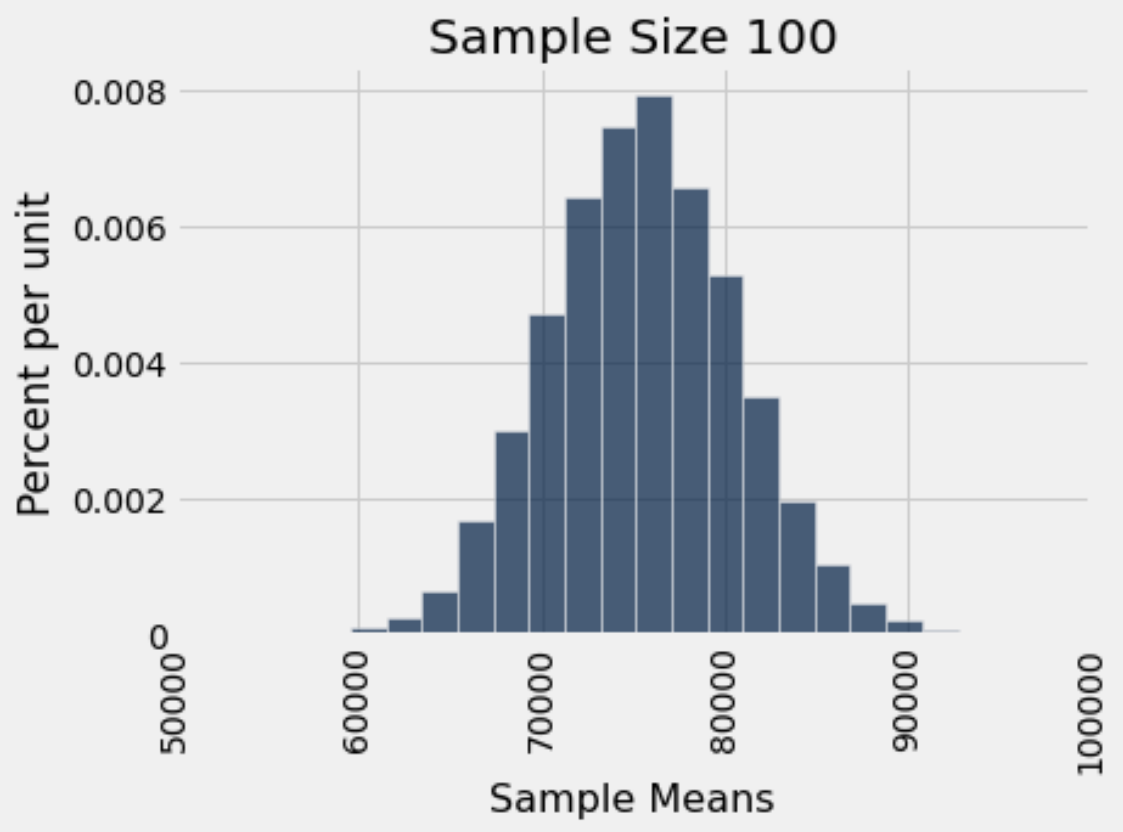
```
        # new_sample_mean =
        means.append(new_sample[label].mean())

    sample_means = Table().with_column('Sample Means', means)

    # Display empirical histogram and print all relevant quantities - don't␣
↪change this!
    sample_means.hist(bins=20)
    plots.xlabel('Sample Means')
    plots.title('Sample Size ' + str(sample_size))
    print("Sample size: ", sample_size)
    print("Population mean:", np.mean(table.column(label)))
    print("Average of sample means: ", np.mean(means))
    print("Population SD:", np.std(table.column(label)))
    print("SD of sample means:", np.std(means))
```

Question 2.2 In the following cell, we will create a sample of size 100 from the salaries table and graph it using our new `simulate_sample_mean` function.

[17]:
```
simulate_sample_mean(salaries, 'salary', 100, 10000)
plots.xlim(50000, 100000)
```

```
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75529.21310334999
Population SD: 51697.0349864653
SD of sample means: 5085.377695571287
```

[17]:
```
(50000.0, 100000.0)
```

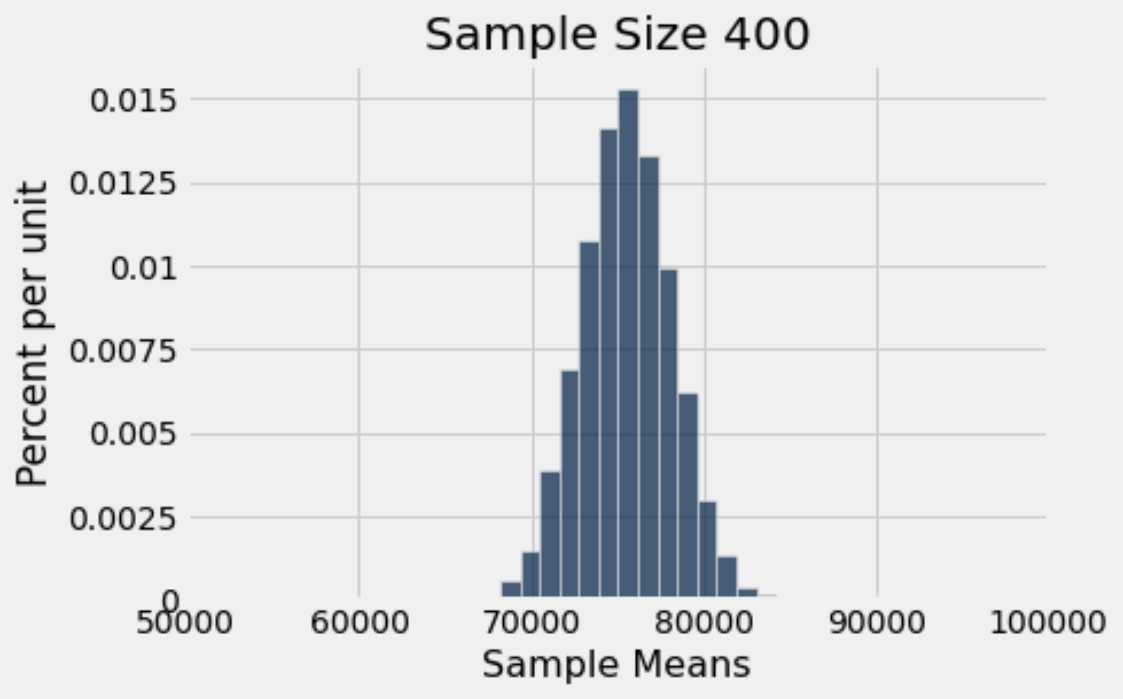


In the following two cells, simulate the mean of a random sample of 400 salaries and 625 salaries, respectively. In each case, perform 10,000 repetitions of each of these processes. Don't worry about the `plots.xlim` line – it just makes sure that all of the plots have the same x-axis.

[18]:
```
simulate_sample_mean(salaries, "salary", 400, 10000)
plots.xlim(50000, 100000)
```

```
Sample size:  400
Population mean: 75463.91814023031
Average of sample means:  75475.50033659999
Population SD: 51697.0349864653
SD of sample means: 2589.490295253377
```

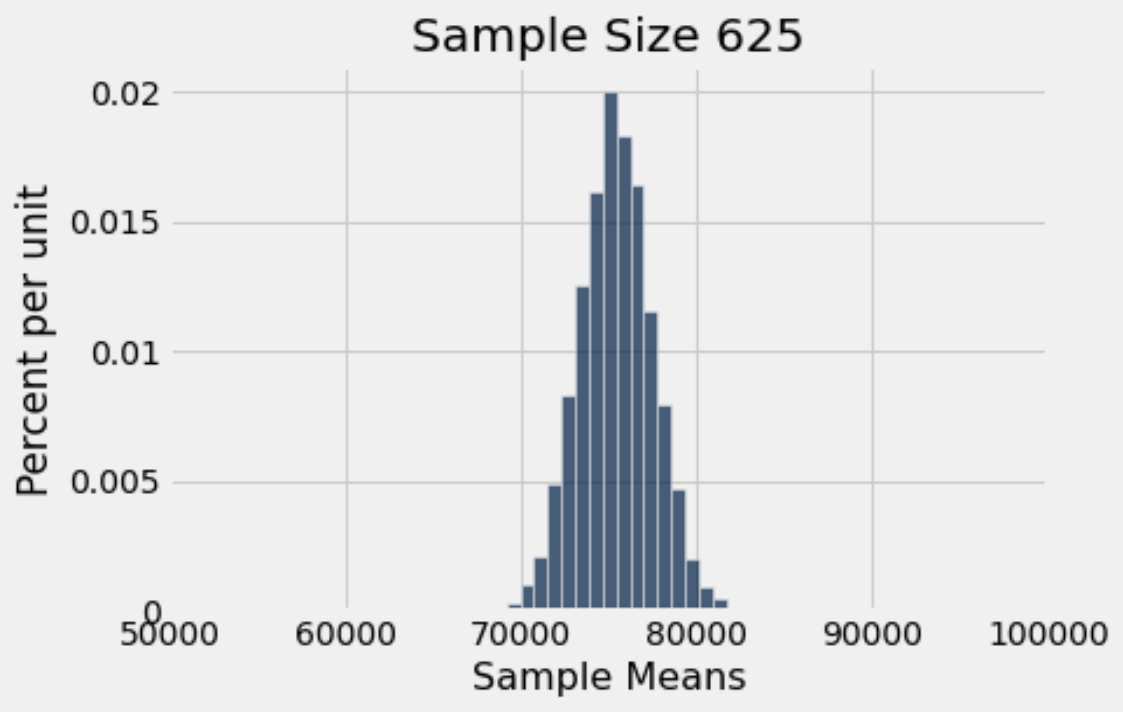
[18]: `(50000.0, 100000.0)`



[19]:
```
simulate_sample_mean(salaries, "salary", 625, 10000)
plots.xlim(50000, 100000)
```

```
Sample size:  625
Population mean: 75463.91814023031
Average of sample means:  75419.43419340801
Population SD: 51697.0349864653
SD of sample means: 2057.3609699522904
```

[19]: (50000.0, 100000.0)



We can see the Central Limit Theorem in action – the histograms of the sample means are roughly normal, even though the histogram of the salaries themselves is far from normal.

We can also see that each of the three histograms of the sample means is centered very close to the population mean. In each case, the "average of sample means" is very close to the population mean. Both values are provided in the printout above each histogram. As expected, the sample mean is an unbiased estimate of the population mean.

Question 2.3 Assign the variable `bootstrap_sampled_SD` to the integer corresponding to your answer to the following question:

When I increase the number of bootstrap samples that I take, for a fixed sample size, the SD of my sample mean will…

1. Increase
2. Decrease
3. Stay about the same
4. Vary widely

[20]:
```
bootstrap_sampled_SD = 3
```

[21]:
```
check('tests/q2_3.py')
```

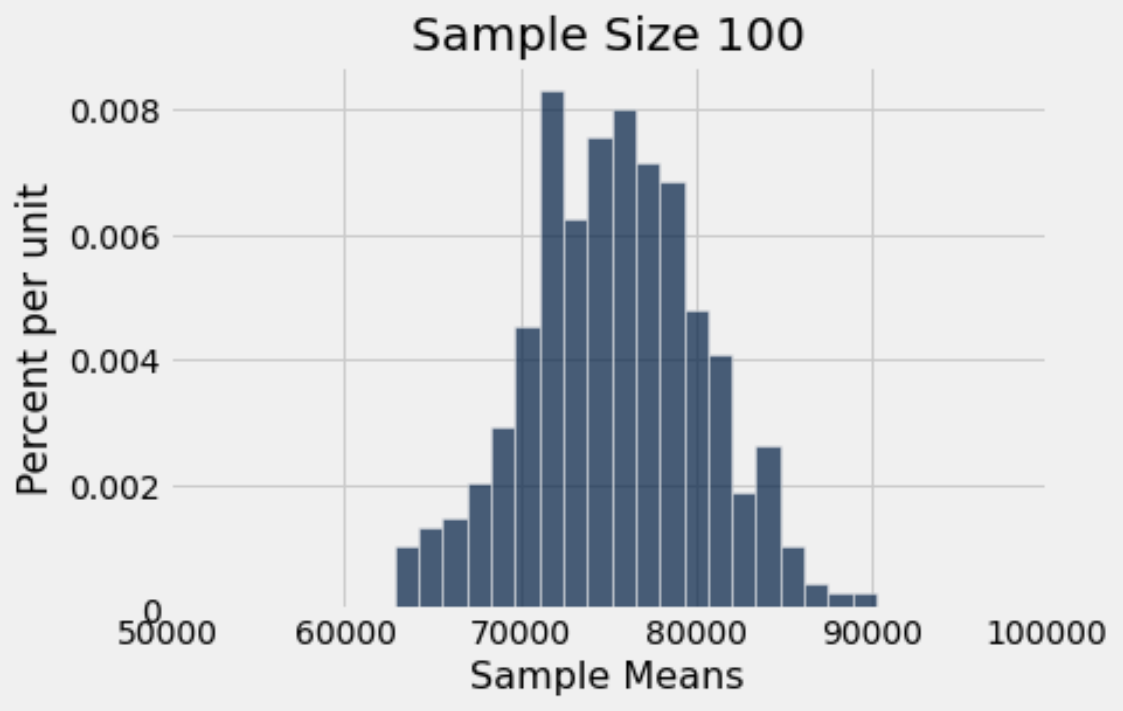
[21]:
```
<gofer.ok.OKTestsResult at 0x7f8feb7a5390>
```

Below, we'll look at what happens when we take a fixed sample, then bootstrap from it with different numbers of resamples. How does the distribution of the resampled means change?

[22]:
```
simulate_sample_mean(salaries, 'salary', 100, 500)
plots.xlim(50000, 100000)
```

```
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75384.17485280002
Population SD: 51697.0349864653
SD of sample means: 4972.062832981087
```

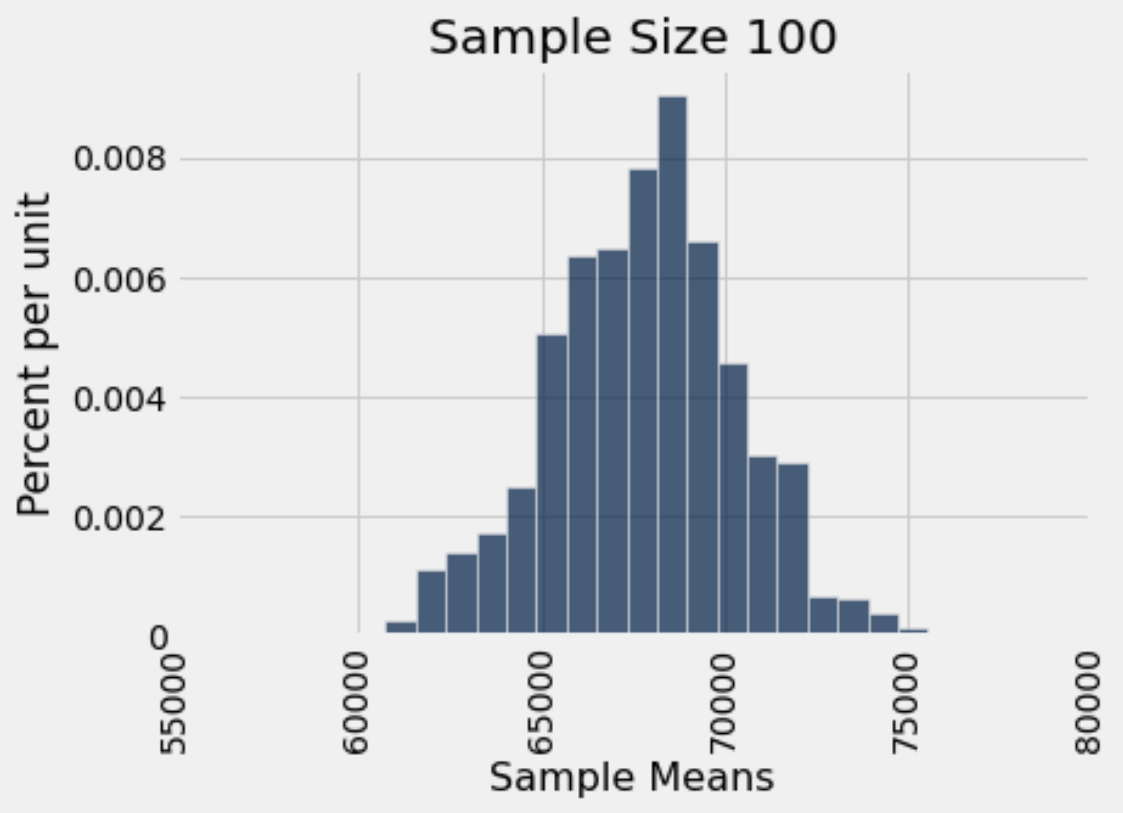
[22]: (50000.0, 100000.0)



[23]:
```
simulate_sample_mean(salaries, 'salary', 100, 1000)
plots.xlim(50000, 100000)
```

```
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75627.3035523
Population SD: 51697.0349864653
SD of sample means: 5052.407916245221
```

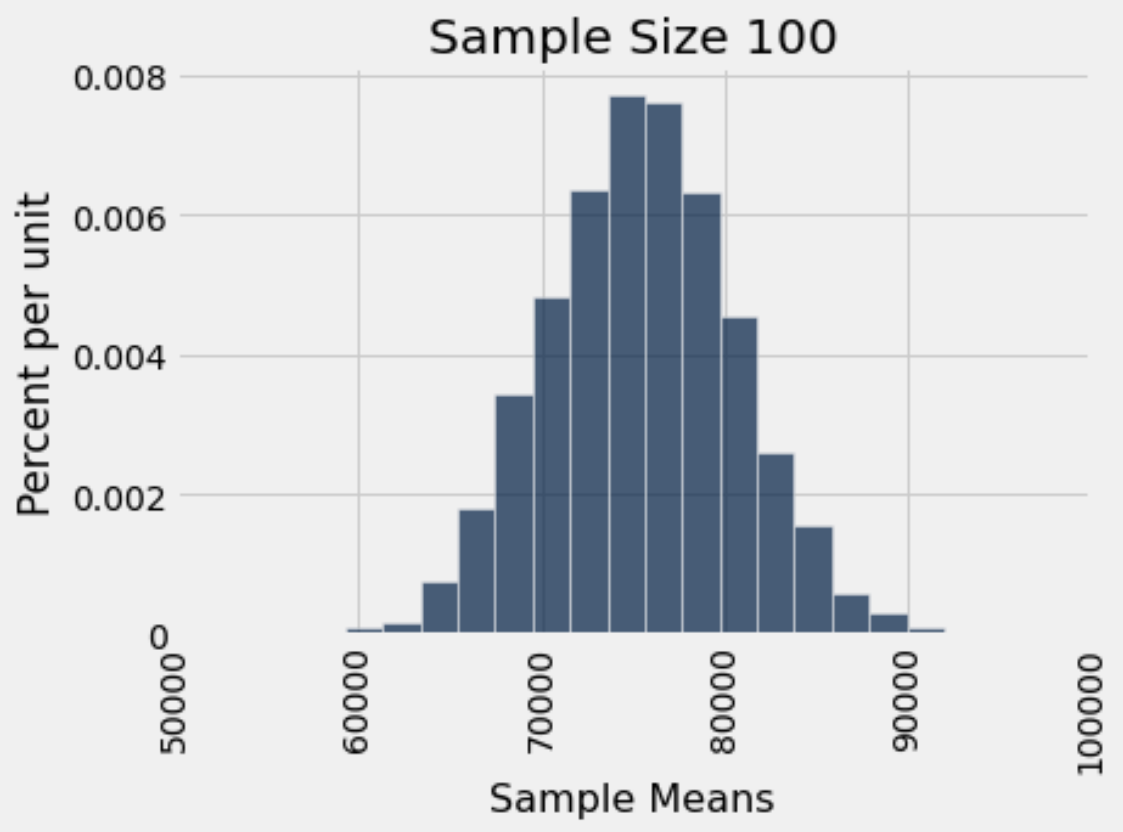
[23]: (50000.0, 100000.0)



[24]:
```
simulate_sample_mean(salaries, 'salary', 100, 5000)
plots.xlim(50000, 100000)
```

```
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75420.03141708
Population SD: 51697.0349864653
SD of sample means: 5135.143889821718
```

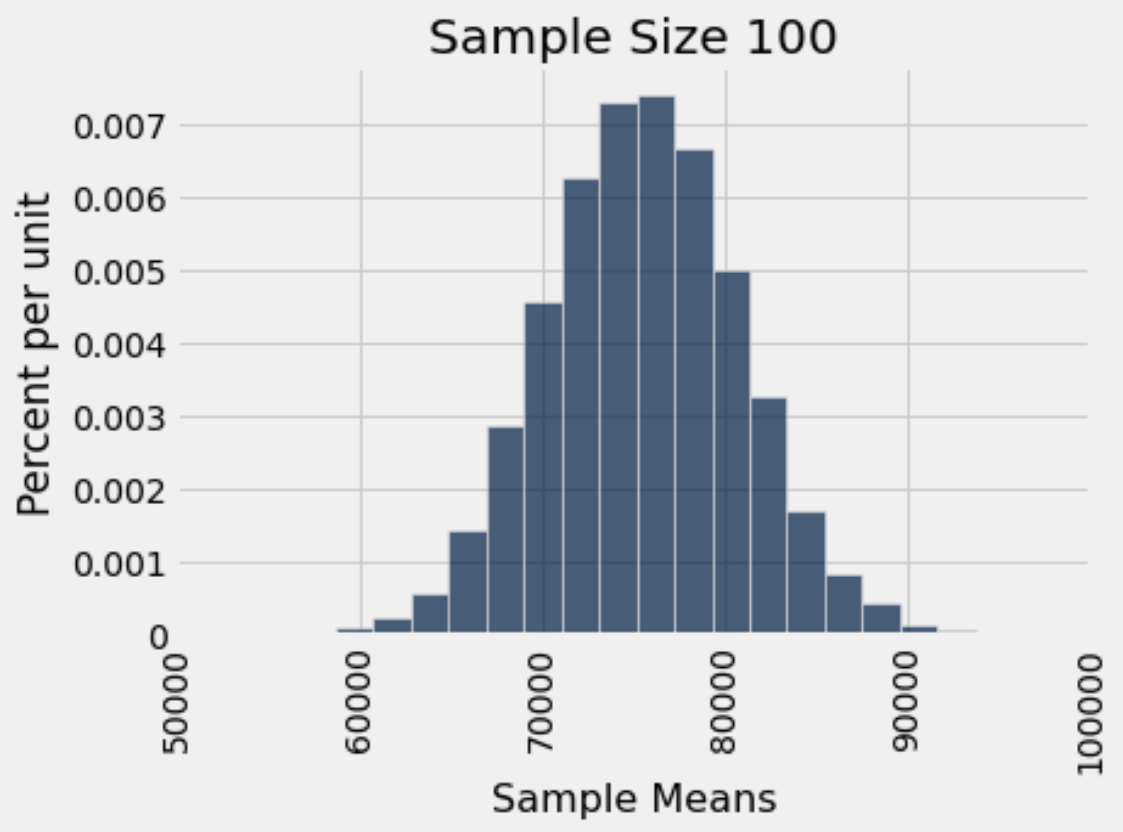
[24]: (50000.0, 100000.0)



[25]:
```
simulate_sample_mean(salaries, 'salary', 100, 10000)
plots.xlim(50000, 100000)
```

```
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75512.41859069
Population SD: 51697.0349864653
SD of sample means: 5206.475748485946
```

[25]: (50000.0, 100000.0)



What did you notice about the sample means of the four bootstrapped samples above?

Question 2.4 Next, let's think about how the relationships between population SD, sample SD, and SD of sample means change with varying sample size. Which of the following is true? Again, assign the variable `pop_vs_sample` to the integer corresponding to your answer. To gain some intuition, you can run the simulation cells below.

1. Sample SD gets smaller with increasing sample size, SD of sample means gets smaller with increasing sample size
2. Sample SD gets larger with increasing sample size, SD of sample means stays the same with increasing sample size
3. Sample SD becomes more consistent with population SD with increasing sample size, SD of sample means gets smaller with increasing sample size
4. Sample SD becomes more consistent with populatoin SD with increasing sample size, SD of smaple means stays the same with increasing sample size

Hint: This lab has you guess what happens next. So don't fret when it asks you to figure out what will happen by choosing one of the answers. If you don't guess right first time around, run through the work below it and come back up to the question. You should understand it better at that point. The checks for this lab works so you will know when you get the correct answer.

[26]:
```
pop_vs_sample = 3
```

[27]:
```
check('tests/q2_4.py')
```

[27]:
```
<gofer.ok.OKTestsResult at 0x7f8feb546cd0>
```

Let's see what happens: First, we calculate the population SD so that we can compare the SD of each sample to the SD of the population.

[28]:
```
pop_sd = np.std(salaries.column("salary"))
pop_sd
```

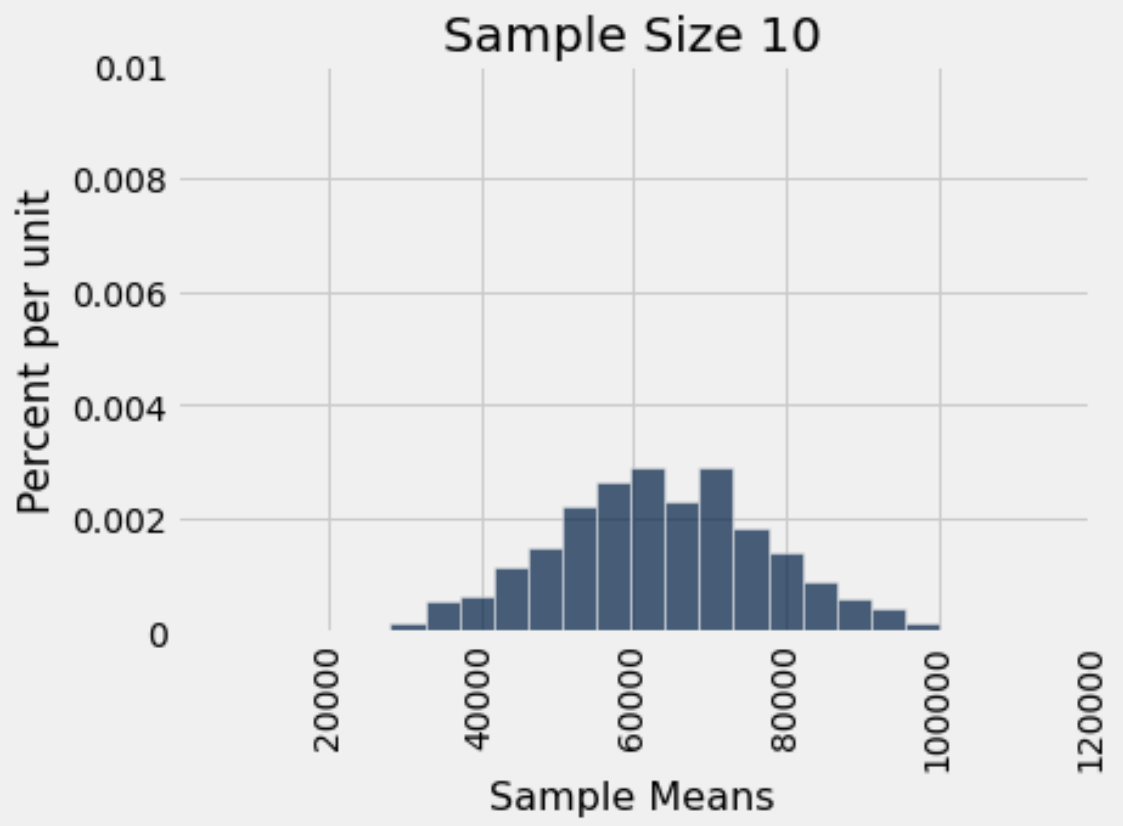
[28]:
```
51697.0349864653
```

Let's then see how a small sample behaves. Run the following cells multiple times to see how the SD of the sample changes from sample to sample. Adjust the bins as necessary.

[29]:
```
sample_10 = salaries.sample(10)
sample_10.hist("salary")
print("Sample SD: ", np.std(sample_10.column("salary")))
simulate_sample_mean(sample_10, 'salary', 10, 1000)
plots.xlim(5,120000)
plots.ylim(0, .0001);
```

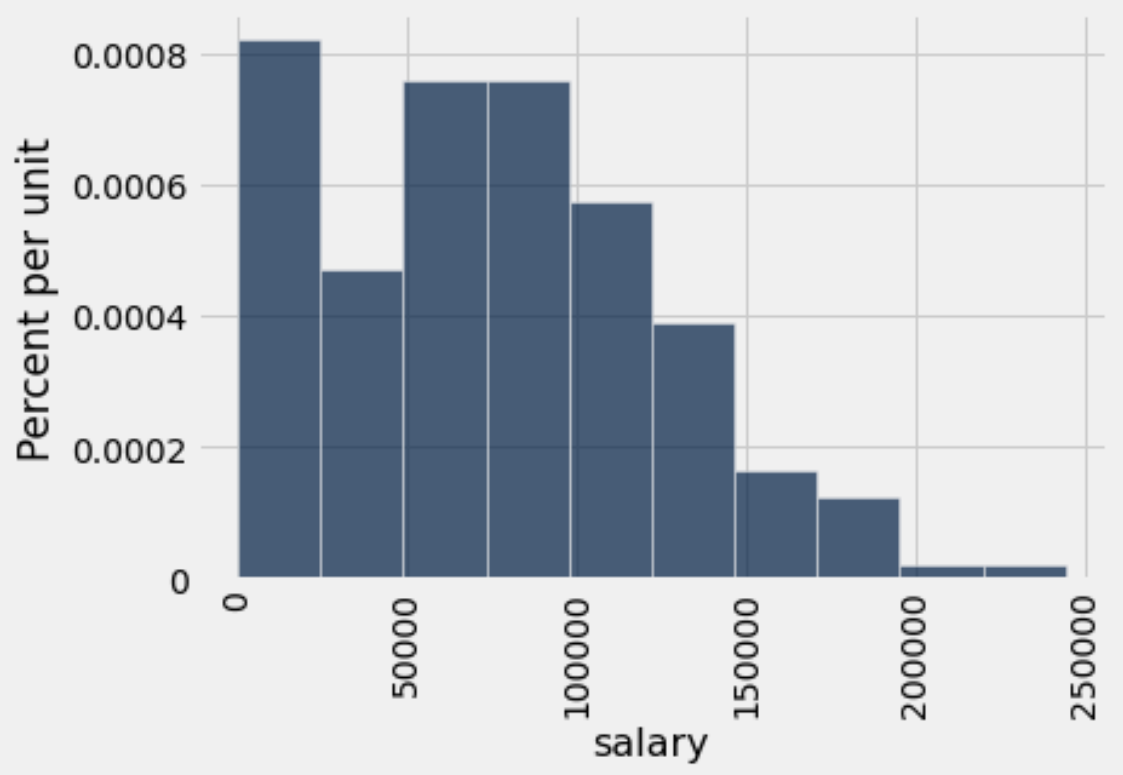
```
Sample SD:  44144.500964422296
Sample size:  10
Population mean: 63755.203000000016
Average of sample means:  63399.671589000005
Population SD: 44144.500964422296
SD of sample means: 14112.831542042884
```

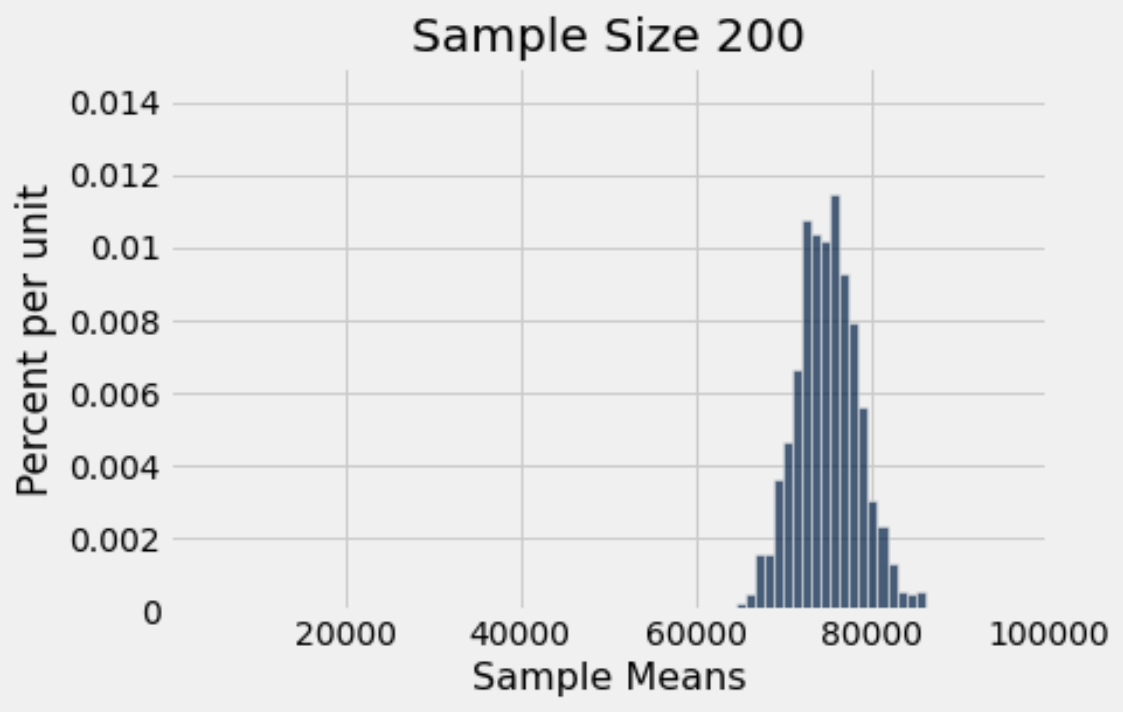

0.0015
0.00125
0.001
0.00075
0.0005
0.00025
0
Percent per unit
20000
40000
60000
80000
100000
120000
140000
salary


Sample Size 10
0.01
0.008
0.006
0.004
0.002
0
Percent per unit
20000
40000
60000
80000
100000
120000
Sample Means

[30]:
```
sample_200 = salaries.sample(200)
sample_200.hist("salary")
print("Sample SD: ", np.std(sample_200.column("salary")))
simulate_sample_mean(sample_200, 'salary', 200, 1000)
plots.xlim(5,100000)
plots.ylim(0, .00015);
```

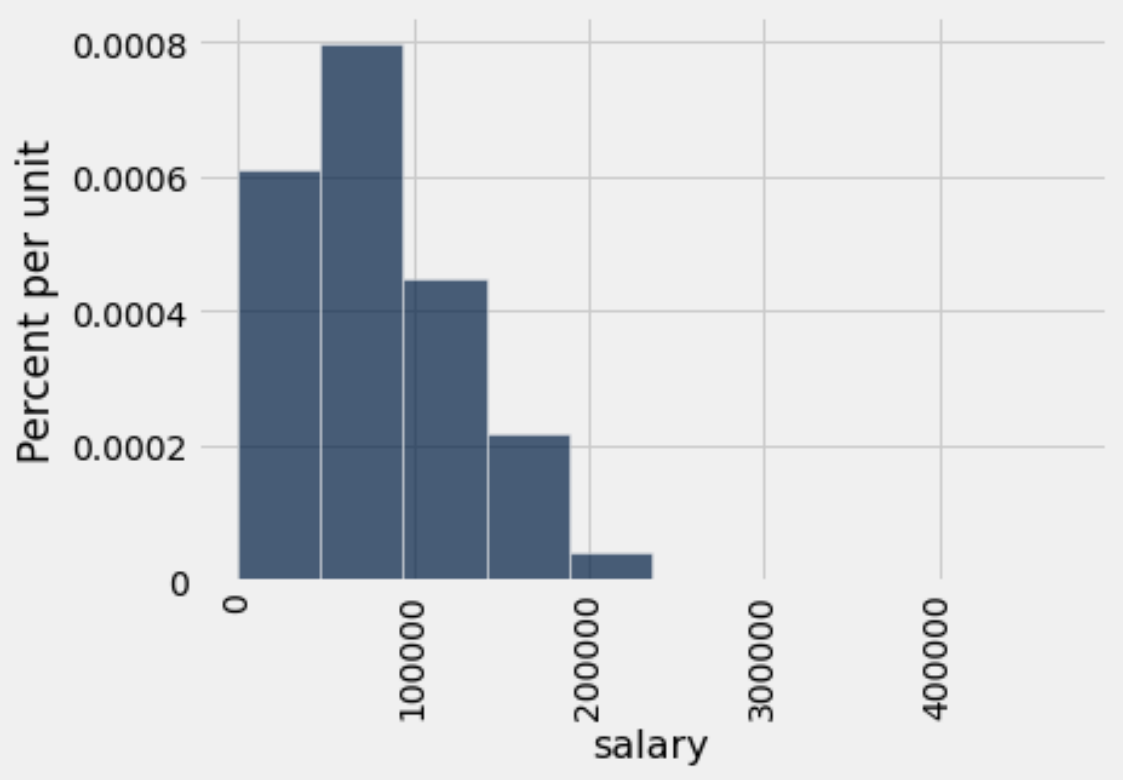
```
Sample SD:  49953.0610240402
Sample size:  200
Population mean: 75017.6121
Average of sample means:  74852.5656929
Population SD: 49953.0610240402
SD of sample means: 3558.2235203894415
```

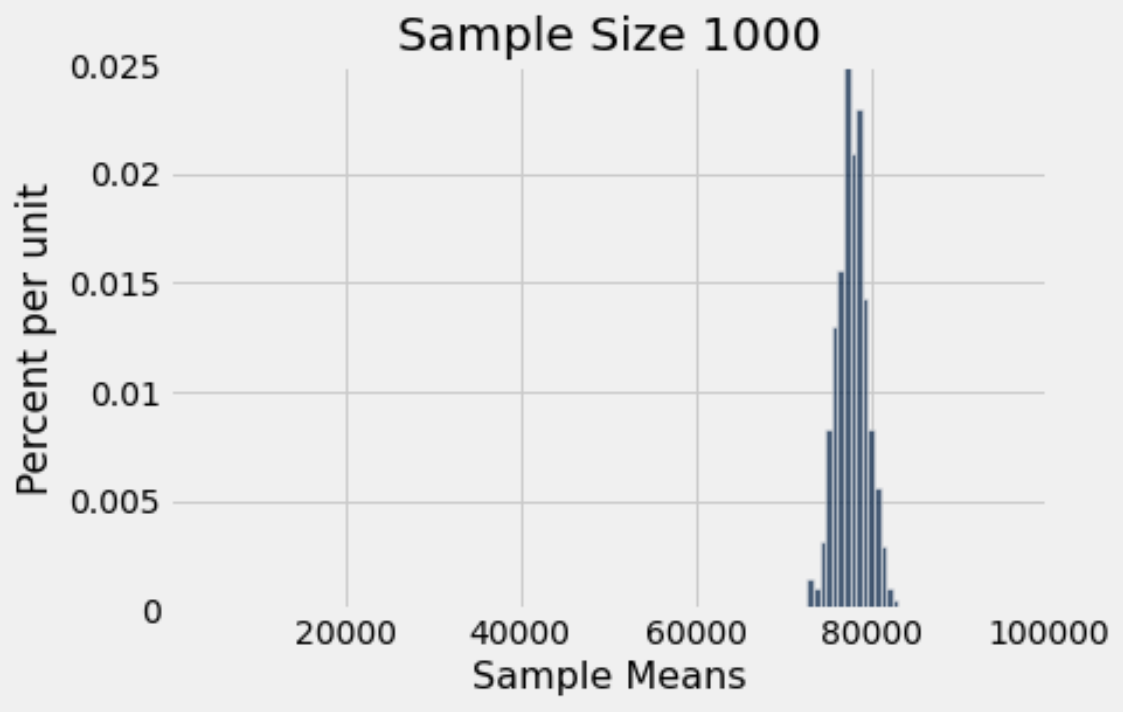




[31]:
```
sample_1000 = salaries.sample(1000)
sample_1000.hist("salary")
print("Sample SD: ", np.std(sample_1000.column("salary")))
simulate_sample_mean(sample_1000, 'salary', 1000, 1000)
plots.xlim(5,100000)
plots.ylim(0, .00025);
```

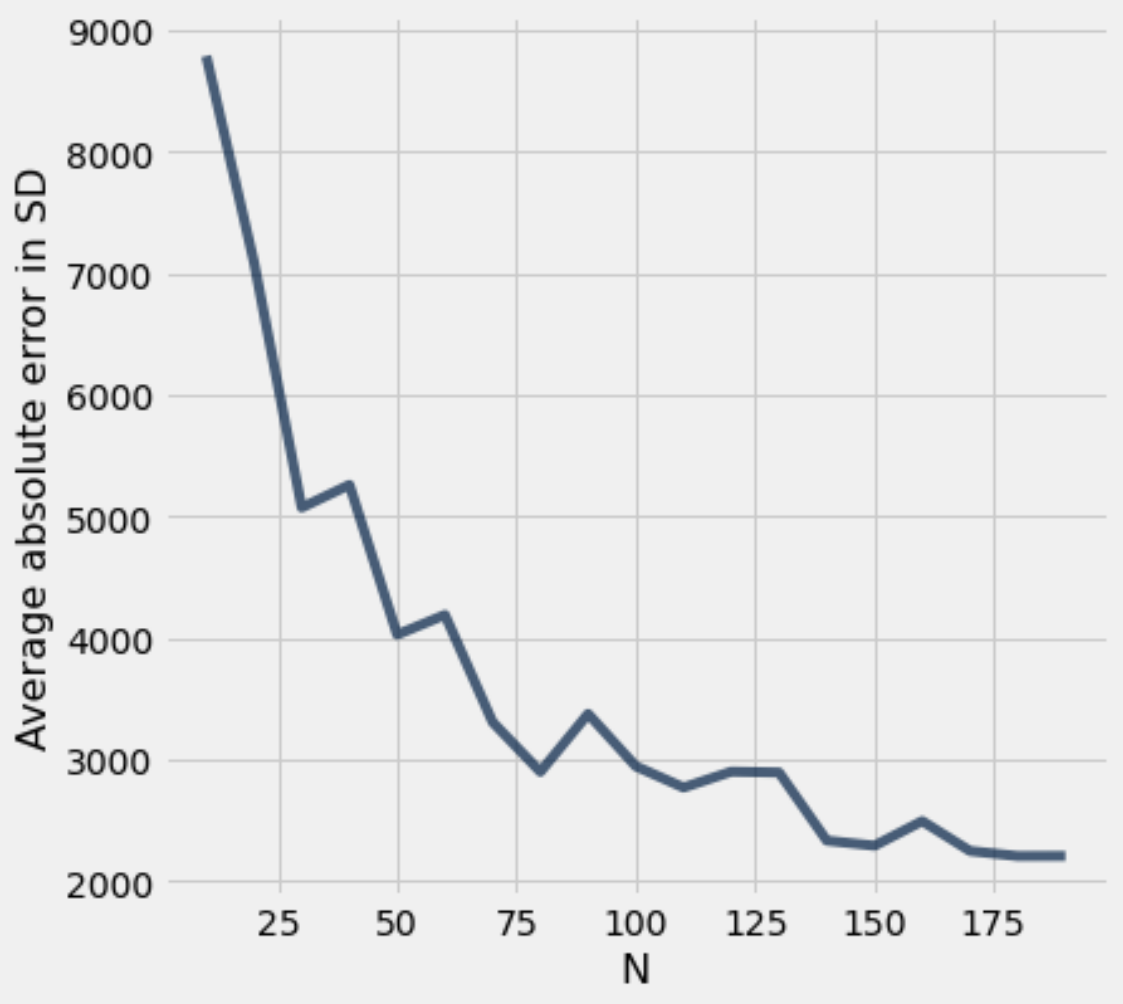
```
Sample SD:  52885.431427561714
Sample size:  1000
Population mean: 77569.23503999999
Average of sample means:  77652.21922905999
Population SD: 52885.431427561714
SD of sample means: 1728.7620989588504
```


0.0008
0.0006
0.0004
0.0002
0
Percent per unit
0
100000
200000
300000
400000
salary


Sample Size 1000
0.025
0.02
0.015
0.01
0.005
0
Percent per unit
20000
40000
60000
80000
100000
Sample Means

Let's illustrate this trend. Below, you will see how the average absolute error of SD from the population changes with sample size (N).

[32]:
```
# Don't change this cell, just run it!
sample_n_errors = make_array()
for i in np.arange(10, 200, 10):
    sample_n_errors = np.append(sample_n_errors, np.average([abs(np.
 ↪std(salaries.sample(i).column("salary"))-pop_sd)
                                                             for d in np.
 ↪arange(100)]))
Table().with_columns("Average absolute error in SD", sample_n_errors, "N", np.
 ↪arange(10, 200, 10)).plot("N", "Average absolute error in SD")
```



You should notice that the distribution of means gets spiker, and that the distribution of the sample increasingly looks like the distribution of the population as we get to larger sample sizes.

Is there a relationship between the sample size and absolute error in standard deviation? Identify

this relationship – if you're having trouble, take a look at this section in our textbook about the variability of sample means.

2.2 3. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

Double check that you have completed all of the free response questions as the auto-grader does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this lab. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

[ ]:
```
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('lab11.ipynb', sorted(glob.glob('tests/q*.py'))))
```

```
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75425.38701254
Population SD: 51697.0349864653
SD of sample means: 5196.05192736159
Sample size:  400
Population mean: 75463.91814023031
Average of sample means:  75426.2673366325
Population SD: 51697.0349864653
SD of sample means: 2580.2878965420646
Sample size:  625
Population mean: 75463.91814023031
Average of sample means:  75449.7245769888
Population SD: 51697.0349864653
SD of sample means: 2041.2592524521547
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75476.51650400001
Population SD: 51697.0349864653
SD of sample means: 5261.648031977751
Sample size:  100
```

```
Population mean: 75463.91814023031
Average of sample means:  75465.6925244
Population SD: 51697.0349864653
SD of sample means: 5165.149397184802
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75486.96885064001
Population SD: 51697.0349864653
SD of sample means: 5234.2551993928855
Sample size:  100
Population mean: 75463.91814023031
Average of sample means:  75504.07067897999
Population SD: 51697.0349864653
SD of sample means: 5173.615254214384
Sample SD:  56021.01769363527
Sample size:  10
Population mean: 77878.28799999999
Average of sample means:  78649.09346399999
Population SD: 56021.01769363527
SD of sample means: 17654.503243139865
Sample SD:  47589.29362472614
Sample size:  200
Population mean: 66737.64165
Average of sample means:  66911.24487485
Population SD: 47589.29362472614
SD of sample means: 3402.4423501487004
Sample SD:  51868.21695669012
Sample size:  1000
Population mean: 74088.29583
Average of sample means:  74104.09860530999
Population SD: 51868.21695669012
SD of sample means: 1605.9924185447844
['tests/q1_2.py', 'tests/q1_4.py', 'tests/q1_5.py', 'tests/q2_3.py',
'tests/q2_4.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7f8feb09fc90>

Question 2:

<gofer.ok.OKTestsResult at 0x7f8feb09f050>

Question 3:

<gofer.ok.OKTestsResult at 0x7f8feb189810>

Question 4:

<gofer.ok.OKTestsResult at 0x7f8feb521050>

Question 5:
```

```
<gofer.ok.OKTestsResult at 0x7f8feb57a0d0>
```

```
1.0
```

Name: Allan Gongora

Section: 0131