

In-class exercise for tutorial012

Loops!

Introduction

All of what we think of as "statistics" is based upon repeating an experiment an infinite number of times. But rather than actually repeating the experiment, a bunch of calculus is used, plus assumptions to get the math to work. It may not seem obvious, but when we have been doing something as simple as compute the width of a sampling distribution from a set of data as s/\sqrt{n} , what we are really saying is:

"If we were to do this experiment an infinite number of times and make a distribution of the means from all the experiments, it would be a normal distribution and have a standard deviation of s/\sqrt{n} . (And, by the way, this formula is based on a bunch of math that we will never actually do!)"

One of the most important breakthroughs in statistics and data science was the realization that, with the repetition of a few simple operations (using computers), we can actually simulate experiments a "very large" number of times. And while it's true that "very large" is less than infinite, by using computers to repeat experiments many many times (say tenths of thousands), we free ourselves of the assumptions that had to be made in order to get the math underlying traditional statistics to work!

But how would we simulate repeating an experiment a number of times over in code?

You guessed it... **with a for loop!**

Load the data set

The data come from an online test of anxiety that – according to the sketchy website – was constructed such that the anxiety scores are **normally distributed** with a **mean of 50** and a **standard deviation of 10**.

Preliminaries of course...

```
In [9]: import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Load the data file "datasets/012_anxiety_data.npy" (assuming you put the file in your "datasets" folder – otherwise adjust path as necessary. Reminder: `np.load()` is your friend!

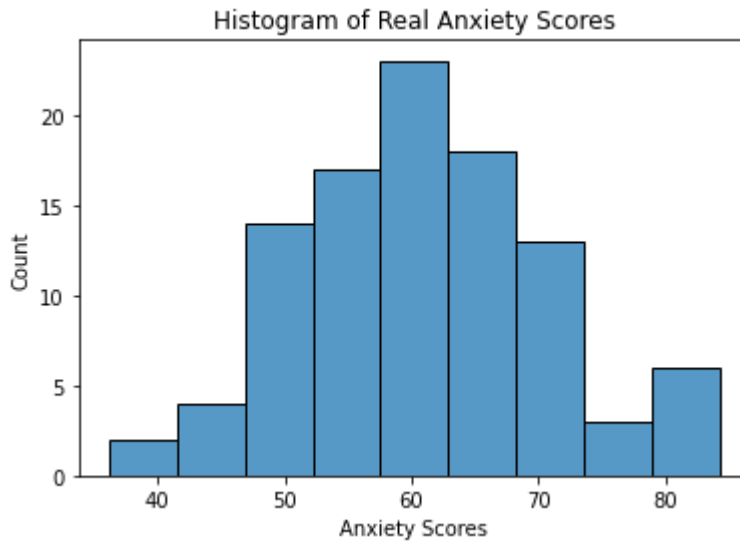
```
In [26]: real_data = np.load('datasets/012_anxiety_data.npy')
```

Now let's make sure we know our data set, `real_data`, well. Let's

- look at a histogram
- ditto with a kde
- compute the mean, median and standard deviation
- compute the standard error of the mean

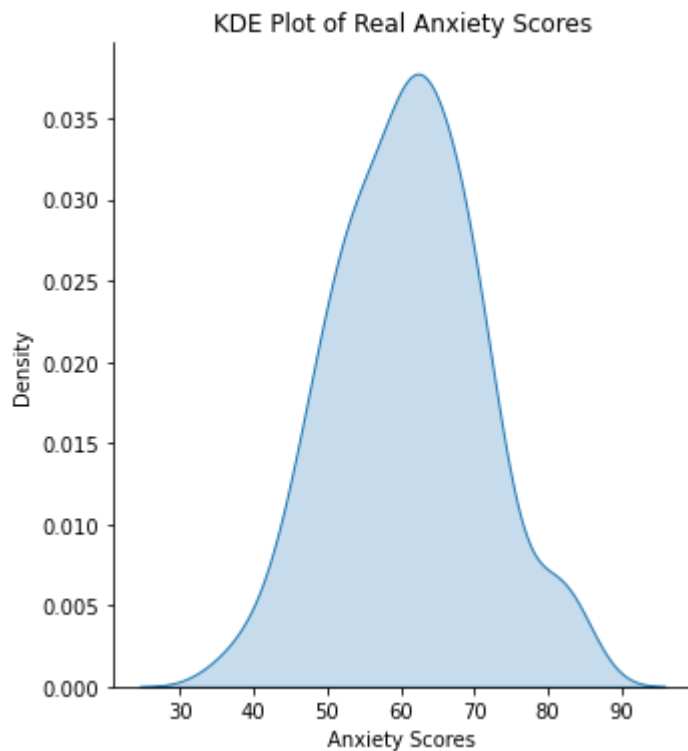
```
In [41]: # histogram
sns.histplot(real_data)
plt.xlabel('Anxiety Scores')
plt.title('Histogram of Real Anxiety Scores')
```

```
Out[41]: Text(0.5, 1.0, 'Histogram of Real Anxiety Scores')
```



```
In [12]: # kde
sns.displot(real_data, kind='kde', fill=True)
plt.xlabel('Anxiety Scores')
plt.title('KDE Plot of Real Anxiety Scores')
```

```
Out[12]: Text(0.5, 1.0, 'KDE Plot of Real Anxiety Scores')
```



In [30]:

```
# mean, median and standard deviation
myMean = np.mean(real_data)
myMedian = np.median(real_data)
mySd = np.std(real_data)

print('Mean=', myMean)
print('Median=', myMedian)
print('Sd=', mySd)
```

```
Mean= 60.971860226088445
Median= 61.00404975776776
Sd= 9.789934489177513
```

In [31]:

```
# standard error
n = 100
mySe = sd/n**0.5

print('se=', mySe)
```

```
se= 0.9789934489177513
```

In a sentence or two of your own words, describe what the standard error of the mean is:

The standard error of the mean is the width of the sampling distribution.

Simulate a bunch of experimental replications

Imagine, we wanted to simulate many many repeats of the same experiments. For example, imagine that we wanted to appreciate the variability of the data obtained in the experiments, under certain conditions of noise and variability in the data.

How would we simulate a bunch of experiments? We obviously can't actually repeat the experiments in the real world. But, as data scientists, we do have a couple of options, both of which we can implement with `for` loops!

Monte Carlo Simulation

If we want to repeat the experiment a bunch of times, let's consider what we know! We know that the website claims that:

- the scores are normally distributed
- they have a mean of 50
- and a standard deviation of 10

So we should be able to use `numpy.random.randn()` to generate numbers that meet the first criterion. Then we just have to scale the standard deviation up by 10 and set the mean to 50. Luckily, we know how to multiply (`*`) and add (`+`), respectively.

So here's our mission:

- write a `for` loop that repeats `n_replications = 2000` times
- on each replication
 - compute the mean of the simulated experiment
 - store that mean in a `mc_means` numpy array
- do a histogram of the means
- make a kde also too
- compute the mean and standard deviation of the 2000 means
 - compare the "mean o' means" from your simulation with the data mean
 - compare the "standard deviation o' means" with the standard error of the data

The simulation via `for` loop:

```
In [39]: correct_mean = 50
correct_sd = 10
n_replications = 2000
anxiety_scores_per_exp = 100
mc_means = np.zeros((n_replications, 1))

for i in range(n_replications):
    correct_anxiety_scores = correct_mean + correct_sd*np.random.randn(anxiety_scores_p
    mc_means[i] = np.mean(correct_anxiety_scores)

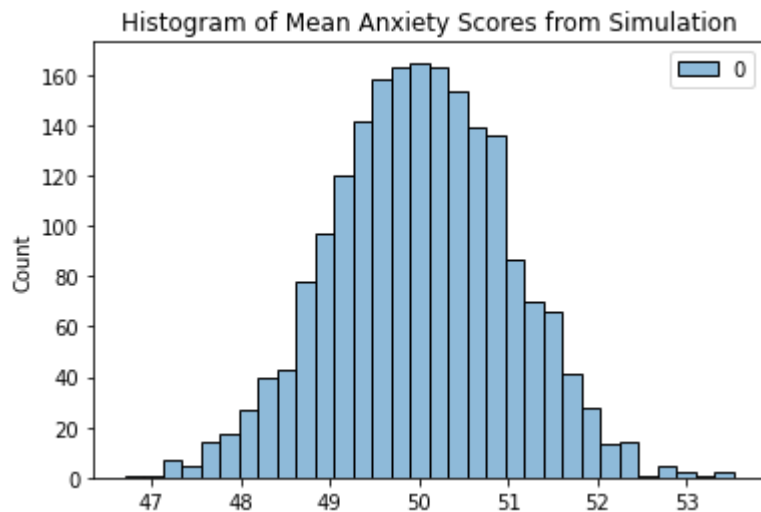
mc_means

Out[39]: array([[51.05437218],
               [48.05584343],
               [48.33883296],
               ...,
               [49.07865703],
               [49.20222711],
               [49.89744408]])
```

Histogram of the means:

```
In [43]: sns.histplot(mc_means)
plt.xlabel = ('Mean Anxiety Scores')
plt.title('Histogram of Mean Anxiety Scores from Simulation')
```

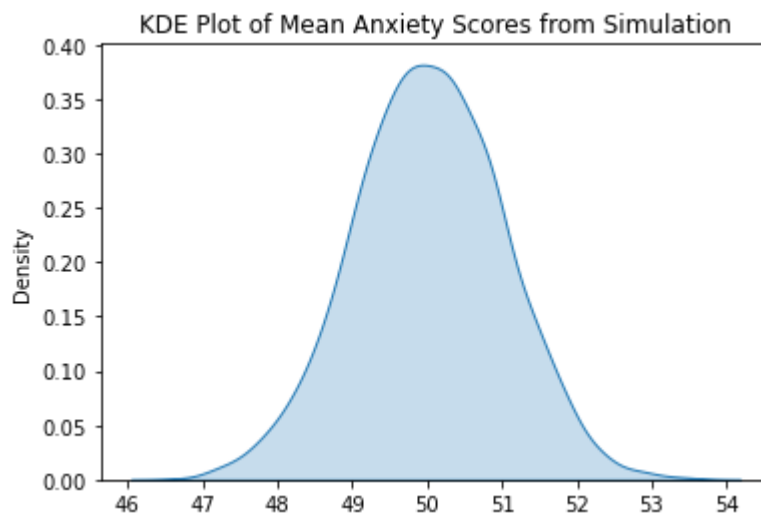
```
Out[43]: Text(0.5, 1.0, 'Histogram of Mean Anxiety Scores from Simulation')
```



KDE of the means

```
In [83]: mykde = sns.kdeplot(x=mc_means[:,0], fill=True)
mykde.set_xlabel = ('Mean Anxiety Scores')
mykde.set_title('KDE Plot of Mean Anxiety Scores from Simulation')
```

```
Out[83]: Text(0.5, 1.0, 'KDE Plot of Mean Anxiety Scores from Simulation')
```



Compute the mean value of your simulation means:

```
In [48]: mc_means_mean = np.mean(mc_means)
print('The mean value of the simulation means is', mc_means_mean)
```

The mean value of the simulation means is 50.01158390319385

Compare it with the original data mean:

The mean value of my simulation means is 50.0 and range of the means from the simulation goes

from about 46 to 54. However, the mean from the online data was 61.0, which is quite a bit higher than the expected mean of about 50.

Compute the standard deviation of your simulation means:

```
In [49]: mc_means_sd = np.std(mc_means)
print('The standard deviation of the simulation means is', mc_means_sd)
```

The standard deviation of the simulation means is 0.9983823605811377

Compare it with the standard error you computed from the original data:

```
In [52]: print('The standard error of the original data is', mySe,
          'whereas the standard deviation of the simulation means (which should be the same
          print('So these numbers are actually pretty similar, unlike the means')
```

The standard error of the original data is 0.9789934489177513 whereas the standard deviation of the simulation means (which should be the same number) is 0.9983823605811377
So these numbers are actually pretty similar, unlike the means

Bonus (not required)

If you knocked the above out with time to spare – congratulations – and let's think about this: you not only have the information given above as clues to the true state of the world. You also have:

- the data themselves (or the histogram thereof that you made)
- the actual mean of the original data
- the actual standard deviation of the original data

So rather than do a simulation based on the claimed mean of the sketchy website, you could base a new simulation on the data you actually have!

Note that, if you wrote your code reasonably well above, you should only have to change the values of two variables to do this new simulation!

Proceed!

```
In [53]: #creating a sampling distribution that is based on the mean and sd of the website data
myMean #mean from the website data
mySd #sd from the website data
n_replications = 2000
anxiety_scores_per_exp = 100
website_means = np.zeros((n_replications, 1))

for i in range(n_replications):
    correct_anxiety_scores = myMean + mySd*np.random.randn(anxiety_scores_per_exp, 1) #
    website_means[i] = np.mean(correct_anxiety_scores)

website_means
```

```
Out[53]: array([[61.34638231],
               [59.66651275],
               [62.42074514],
```

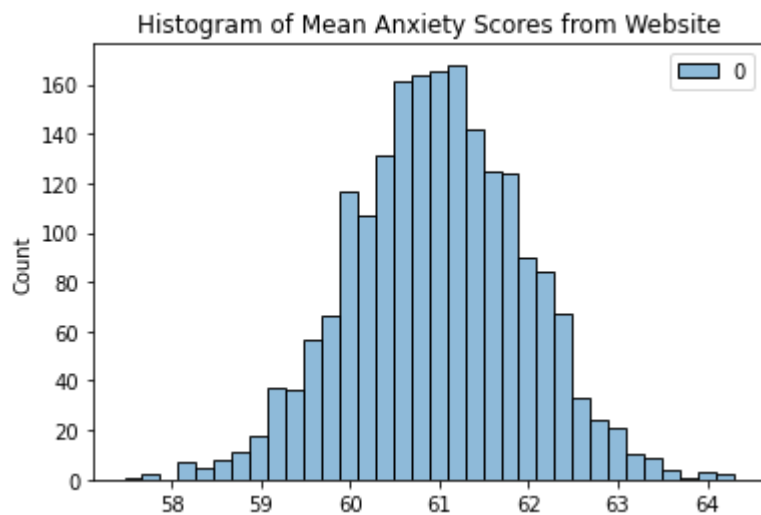
```
...,
[60.61538925],
[60.37243353],
[60.42279015]])
```

In [66]: `website_means.shape`

Out[66]: (2000, 1)

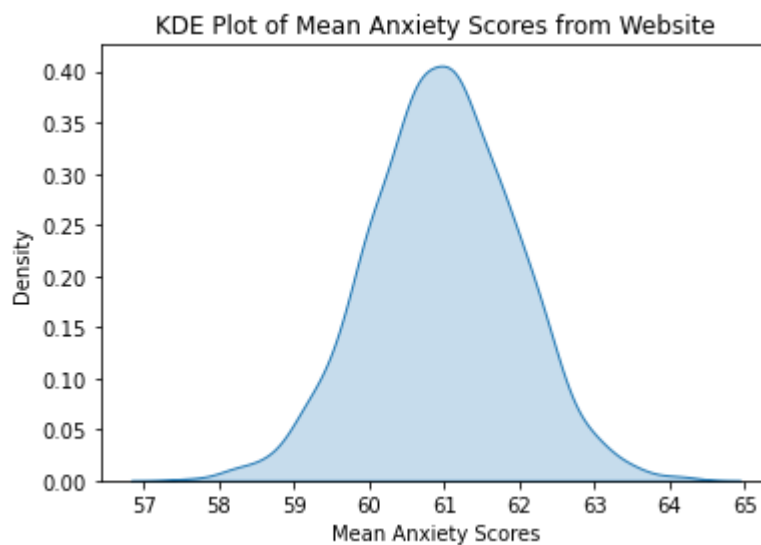
In [54]: `sns.histplot(website_means)`
`plt.xlabel = ('Mean Anxiety Scores')`
`plt.title('Histogram of Mean Anxiety Scores from Website')`

Out[54]: Text(0.5, 1.0, 'Histogram of Mean Anxiety Scores from Website')



In [69]: `mykde = sns.kdeplot(x=website_means[:,0], fill=True)`
`mykde.set_xlabel('Mean Anxiety Scores')`
`mykde.set_title('KDE Plot of Mean Anxiety Scores from Website')`

Out[69]: Text(0.5, 1.0, 'KDE Plot of Mean Anxiety Scores from Website')



In [72]:

```
websitesVsSimulation = np.hstack((mc_means, website_means))  
websitesVsSimulation
```

```
Out[72]: array([[51.05437218, 61.34638231],  
       [48.05584343, 59.66651275],  
       [48.33883296, 62.42074514],  
       ...,  
       [49.07865703, 60.61538925],  
       [49.20222711, 60.37243353],  
       [49.89744408, 60.42279015]])
```

```
In [79]: mykde = sns.kdeplot(x=mc_means[:,0], fill=True)  
mkkde = sns.kdeplot(x=website_means[:,0], fill=True)  
mykde.set_title('Original Simulation Means vs Website Simulation Means')  
mykde.set_xlabel('Mean Anxiety Scores')
```

```
Out[79]: Text(0.5, 0, 'Mean Anxiety Scores')
```

