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 then infinite, by using computers to repeat experiments many many times (say tenths of thousands), we free ourselves of the assumptions that had to 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 [12]:
```
import numpy as np
import seaborn as sns
```

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 [13]:
```
mdff = np.load("datasets/012_anxiety_data.npy")
display(mdff)
```

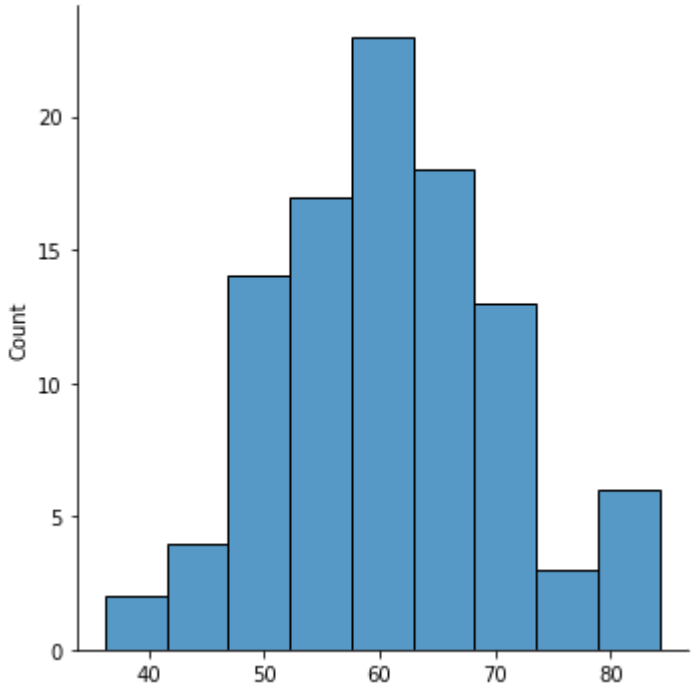
```
array([66.15598705, 60.99347947, 73.95178076, 69.44786909, 66.18546847,
       46.87309342, 55.4822878 , 61.01462004, 53.8560988 , 70.26684944,
       51.41494246, 74.40727401, 57.36228612, 81.16455062, 59.54616334,
       49.19403488, 84.20749803, 60.21862833, 47.004287  , 60.62489583,
       65.62010043, 66.66460096, 69.92435732, 71.93795517, 62.08777851,
       58.38746687, 53.38311807, 36.19697643, 70.21563628, 44.98299005,
       67.9994991 , 55.82578757, 61.36715595, 55.43705844, 64.19714952,
       40.19598403, 70.78236851, 59.7730611 , 67.30582054, 61.05911192,
       52.47104034, 44.52717704, 73.52349749, 64.82501164, 70.01136376,
       55.02199822, 52.94748241, 50.20134902, 60.59984295, 49.23329907,
       41.99194972, 80.92696942, 53.81366322, 66.94236944, 55.5589867 ,
       62.98725634, 48.97524177, 56.11617858, 55.56544834, 60.1595137 ,
       47.41608974, 82.58214288, 65.19048143, 65.93283347, 45.51017657,
       65.95559035, 66.78266307, 52.79709827, 48.18003522, 59.52598521,
       65.24045299, 66.97949535, 55.82678371, 60.38729762, 54.81340829,
       69.99306229, 62.70126315, 68.67203462, 62.56824652, 59.50985961,
       64.21744252, 62.42381687, 71.57609653, 62.30253591, 54.44716293,
       68.05096356, 52.035678  , 49.45510412, 82.16040963, 80.48932072,
       73.77966276, 68.80533689, 61.40617446, 61.93634883, 59.78807273,
       51.98860418, 72.48991155, 60.79358897, 52.04206527, 51.31701497])
```

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

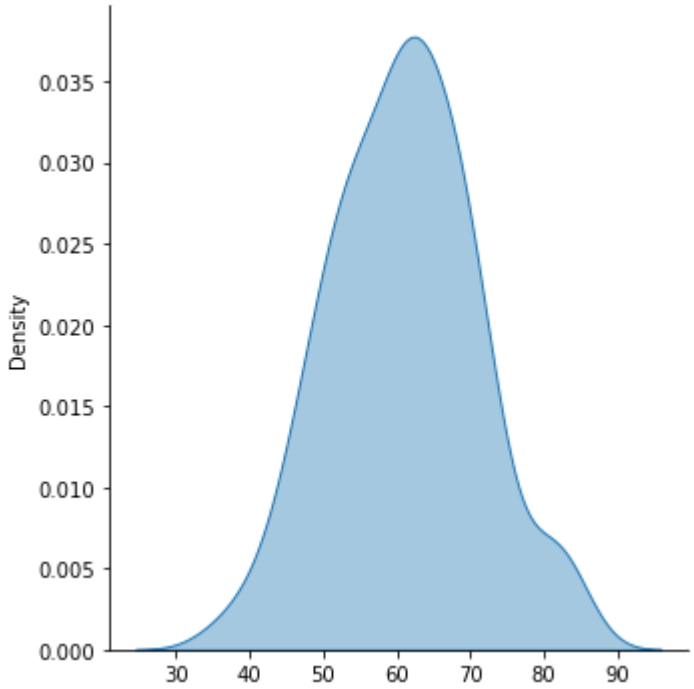
In [14]:
```
# histogram
sns.displot(mdff, kind='hist')
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x227e5a65550>



In [18]:
```
# kde
sns.displot(mdff, kind='kde', fill=True, alpha=.4)
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x227eabb6400>



In [24]:
```
# mean, median and standard deviation
mean = np.mean(mdff)
median = np.median(mdff)
stddev = np.std(mdff)

print("The mean of the data is", mean)
print("The median of the data is", median)
print("The standard deviation of the data is", stddev)
```

```
The mean of the data is 60.971860226088445
The median of the data is 61.00404975776776
The standard deviation of the data is 9.789934489177513
```

In [28]:
```
# standard error
standardError = stddev / np.sqrt(np.size(mdff))

print("The standard error of the data is",standardError)
```

```
The standard error of the data is 0.9789934489177513
```

--- The standard error tells us how spread out our sample data is from the mean. This also gives us insight to the likelihood our sample data is accurate compared to the population

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

Type *Markdown* and LaTeX: α^2

Simulate a bunch of experimental replications

Imagine, we wanted to simulate many many repeates of the same experiments. Fpr examp,e 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 experment 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 critereon. 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:

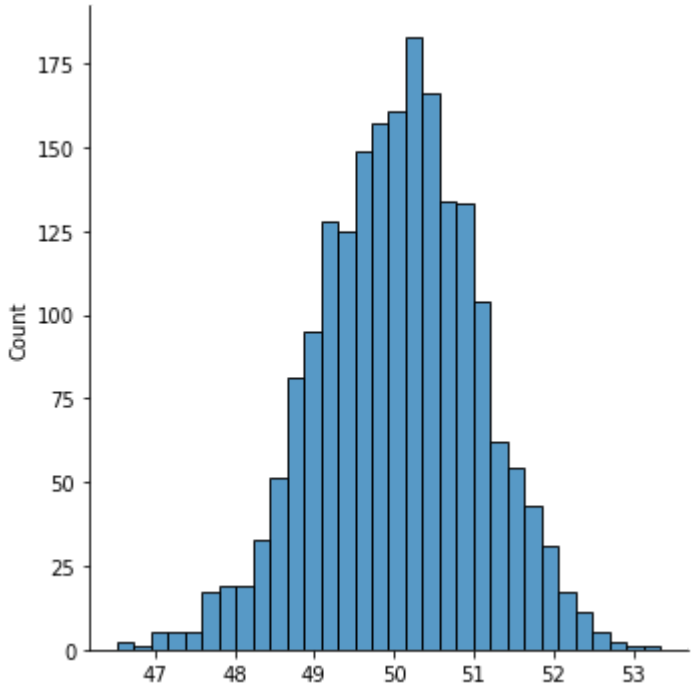
In [37]:
```
nReplications = 2000
mc_means = np.zeros((nReplications))
x = np.random.randn(100,1)
sampleMean, sampleStd = 50, 10


for i in range(nReplications):
    thisData = sampleMean + np.random.randn(100,1)*sampleStd
    mc_means[i] = np.mean(thisData)

mc_means
```

Out[37]:
```
array([52.44530299, 50.68394047, 50.28377307, ..., 50.29769468,
       48.76033437, 53.33699283])
```

Histogram of the means:

In [41]:
```
sns.displot(mc_means, kind='hist')
```

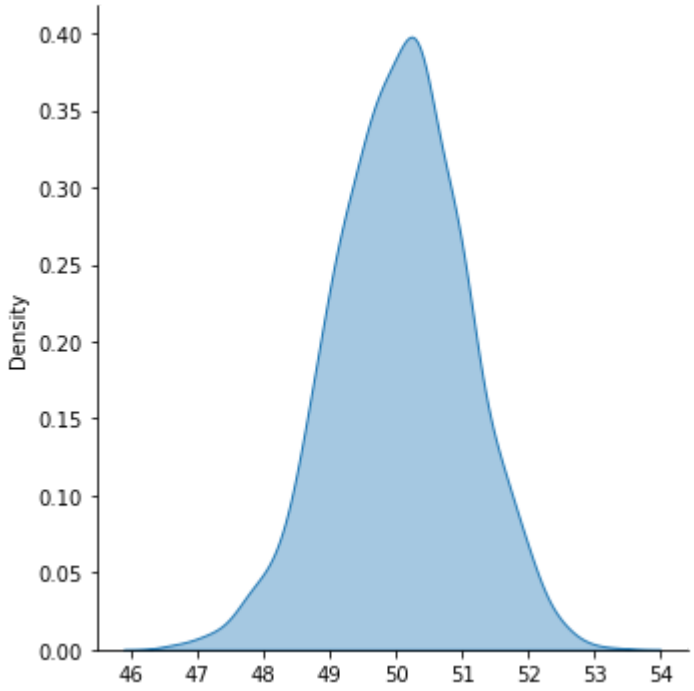
Out[41]:
```
<seaborn.axisgrid.FacetGrid at 0x227eae35cd0>
```



KDE of the means

In [44]:
```
sns.displot(mc_means, kind='kde', fill=True, alpha=.4)
```

Out[44]: <seaborn.axisgrid.FacetGrid at 0x227eb0ad7f0>



Compute the mean value of your simulation means:

In [45]:
```
simulationMean = np.mean(mc_means)

print("The mean value of the simulation means is", simulationMean)
```

```
The mean value of the simulation means is 50.042626272434724
```

Compare it with the original data mean:

Type *Markdown* and LaTeX: α^2

Compute the standard deviation of your simulation means:

In [47]:
```
simulationStd = np.std(mc_means)

print("The standard deviation of the simulation means is", simulationStd)
```

```
The standard deviation of the simulation means is 0.9982826772303605
```

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

In [49]:
```
simulationStdErr = simulationStd / np.sqrt(np.size(mc_means))

print("The standard error of the simulation means is", simulationStdErr)
```

```
The standard error of the simulation means is 0.022322279270475674
```

--- The standard error in our simulation is singificantly smaller than our sample data, implying that our simulation more accurately represents the population

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

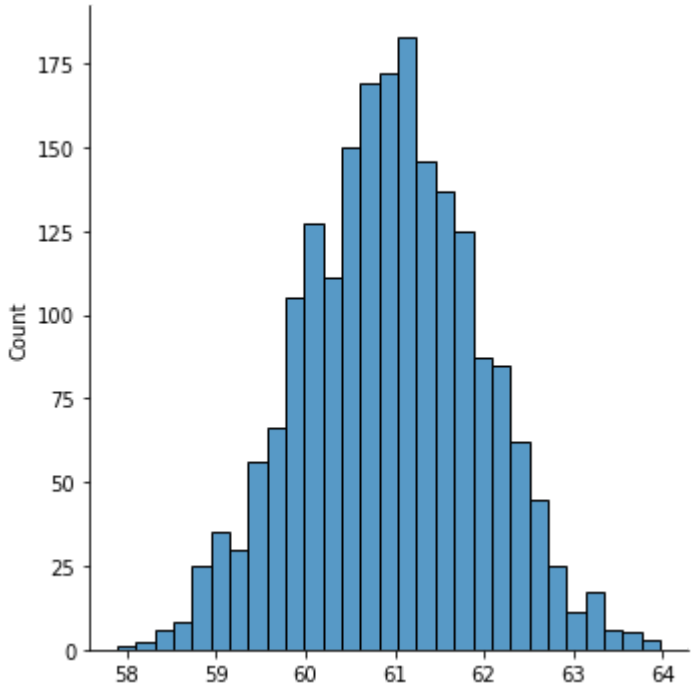
Proceed!

In [51]:
```
mdffMean = np.mean(mdff)
mdffStd = np.std(mdff)

nReplications = 2000
mc_means = np.zeros((nReplications))

for i in range(nReplications):
    x = np.random.randn(100,1)
    thisData = mdffMean + x*mdffStd
    mc_means[i] = np.mean(thisData)

mc_means
```

Out[51]:
```
array([60.83344517, 59.46825254, 60.23012598, ..., 60.69506381,
       63.02024979, 60.56445745])
```

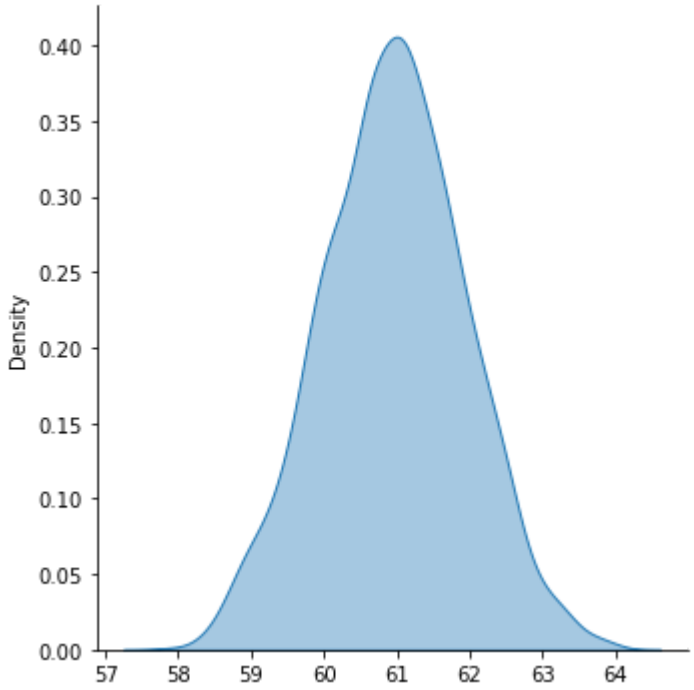
In [52]:
```
sns.displot(mc_means, kind="hist")
```

Out[52]: <seaborn.axisgrid.FacetGrid at 0x227eb15b250>



In [53]:
```
sns.displot(mc_means, kind='kde', fill=True, alpha=.4)
```

Out[53]: <seaborn.axisgrid.FacetGrid at 0x227eafd8790>



In [ ]:
```

```