```
  Course: Statistical_Inference
  Lesson: Resampling

- Class: text
  Output: "Resampling. (Slides for this and other Data Science courses may be found at
  github https://github.com/DataScienceSpecialization/courses/. If you care to use
  them, they must be downloaded as a zip file and viewed locally. This lesson
  corresponds to 06_Statistical_Inference/13_Resampling.)"

- Class: text
  Output: In this lesson, you get a bonus! We'll talk about two topics in statistical
  inference, bootstrapping AND permutation testing. These both fall under the broader
  category of resampling methods. We'll start with bootstrapping.

- Class: text
  Output: The bootstrap is a handy tool for making statistical inferences. It is used
  in constructing confidence intervals and calculating standard errors for statistics
  that might be difficult for some reason (e.g., lack of data or no closed form).
  Wikipedia tells us that bootstrapping is a technique which "allows estimation of the
  sampling distribution of almost any statistic using very simple methods."  Simple is
  good, right?

- Class: text
  Output: The beauty of bootstrapping is that it avoids complicated mathematics and
  instead uses  simulation and computation to infer distributional properties you might
  not otherwise be able to determine.

- Class: text
  Output: It's relatively new, developed in 1979, by Bradley Efron, a  Stanford
  statistician.  The basic bootstrap principle uses OBSERVED data to construct an
  ESTIMATED population distribution using random sampling with replacement. From this
  distribution (constructed from the observed data) we can estimate the distribution of
  the statistic we're interested in.

- Class: mult_question
  Output: So, in bootstrapping the observed data substitutes for what?
  AnswerChoices: a population; a statistic; a hypothesis; observations
  CorrectAnswer: a population
  AnswerTests: omnitest(correctVal='a population')
  Hint: We'll used the observed data and sample it with replacement, just as we would
  do to find out information about some population.

- Class: mult_question
  Output: So, in bootstrapping if the observed data is the population, what would the
  random samplings correspond to?
  AnswerChoices: a population; a statistic; a hypothesis; observations
  CorrectAnswer: observations
  AnswerTests: omnitest(correctVal='observations')
  Hint: Sampling from a population would give us observations, right?

- Class: text
  Output:  In effect, the original observed sample  substitutes for the population. Our
  samplings become observations from which we estimate a statistic and get an idea
  about its distribution. This lets us better understand the underlying population
  (from which we didn't have enough data).

- Class: mult_question
  Output: Here's a critical point. In constructing the estimated distribution we sample
  the observed data WITH replacement. If the original sample is n long and we sampled n
  times without replacement what would we get?
  AnswerChoices: the original sample permuted; an entirely new sample; a better sample;
  a worse sample
  CorrectAnswer: the original sample permuted
  AnswerTests: omnitest(correctVal='the original sample permuted')
  Hint: Sampling without replacement n times permutes the original sample.

- Class: text
  Output: The motivating example from the slides involves computing the average of 50
  rolls of a die. Of course we can do this theoretically when we know that the die is
```

```
    fair. Remember, E(x) = Sum(x*p(x))  for x=1,2,...6, and p(x)=1/6 for all values of x.

  - Class: cmd_question
    Output:  For the heck of it, compute the expected die roll for a fair die.
    CorrectAnswer: 3.5
    AnswerTests: equiv_val(3.5)
    Hint: Type sum(1:6)/6 at the command prompt.

  - Class: figure
    Output: Theoretically, the average is 3.5. Here, we've run code and plotted a
    histogram after we took 1000 such averages, each of 50 dice rolls. Note the unusual
    y-axis scale. We're displaying this as a density function so the area of the
    salmon-colored region is theoretically 1. With this scale, though, all the heights of
    the bins actually add up to 5. So you have to multiply each height by .2 and add up
    all the results to get 1.
    Figure: plot1.R
    FigureType: new

  - Class: text
    Output: The point is, the empirical matches the theoretical. Yay! The highest bin is
    centered at 3.5 just as the math predicted. So what?

  - Class: figure
    Output: What if some joker wanted you to run the same experiment with a die he gave
    you and he warned you that the dice was loaded? In other words, it wasn't fair. It
    has some random distribution like this.
    Figure: plot2.R
    FigureType: new

  - Class: text
    Output: The outcomes aren't equally likely, are they? So when you do your 1000 runs
    of 50 rolls each, the density of the means looks different.

  - Class: cmd_question
    Output: We've done this for you and put the result in g2. Type print(g2) now to see
    the picture.
    CorrectAnswer: print(g2)
    AnswerTests: omnitest(correctExpr='print(g2)')
    Hint: Type print(g2) at the command prompt.

  - Class: text
    Output: Picture's a little different, right? Although this example is a bit
    contrived, it illustrates an important concept. We really want a distribution of
    means and we  have only one set of observations. (In this case it was the empirical
    distribution associated with the unfair die - the big blue picture.) We used that one
    distribution, to "create" many (1000) distributions by sampling with replacement from
    the given one. We sampled 50000 times so we created 1000 distributions of 50 rolls
    each.

  - Class: text
    Output:  We then calculated the mean of each of our created distributions and got a
    distribution of means. Sampling the one distribution many times gives us some
    variability in the resulting statistics we calculate. We can then calculate the
    standard error  and confidence intervals associated with the statistic.

  - Class: mult_question
    Output: Before we go on to more theory, here's another example in which we'll try to
    find a distribution of medians of a population. Do you recall what a median is?
    AnswerChoices: 50th percentile; a point halfway between rare and well-done; the most
    frequent outcome; a person who talks to spirits
    CorrectAnswer: 50th percentile
    AnswerTests: omnitest(correctVal='50th percentile')
    Hint: Half the outcomes are above it and half below it.

  - Class: cmd_question
    Output:  Recall the father and son height data. Once again, we've loaded it for you.
    We've placed the height of the sons  in the vector sh and the length of this vector
    is stored in the variable nh. Use the R command head to look at the first few entries
    of sh.
```

```
    CorrectAnswer: head(sh)
    AnswerTests: omnitest(correctExpr='head(sh)')
    Hint: Type head(sh) at the command prompt.

  - Class: cmd_question
    Output:  Now look at nh to see how long sh is.
    CorrectAnswer: nh
    AnswerTests: omnitest(correctExpr='nh')
    Hint: Type nh at the command prompt.

  - Class: text
    Output:  Now we'll create 1000 distributions of the same length as the original sh.
    We'll do this by sampling sh with replacement  1000*nh times and store the results in
    an array with 1000 rows, each with nh entries. Then we'll take the median of each row
    and plot the result.

  - Class: text
    Output: Note that every time we draw from the empirical distribution sh, each of its
    nh data points is equally likely to be pulled, therefore the probability of drawing
    any one is 1/nh. The 1000 samples we create will vary from the original.

  - Class: figure
    Output: Here's the resulting density curve. This estimates the distribution of
    medians. The thick vertical line shows where the  median of the original, observed
    data sh lies.
    Figure: fatherson.R
    FigureType: new

  - Class: cmd_question
    Output:  We stored the 1000  medians of the resampled sets in the vector
    resampledMedians. Use the R function median to compute the median of numbers in this
    vector.
    CorrectAnswer: median(resampledMedians)
    AnswerTests: obliterate("resamples"); omnitest(correctExpr='median(resampledMedians)')
    Hint: Type median(resampledMedians) at the command prompt.

  - Class: cmd_question
    Output:  Now compute the median of the original sample sh.
    CorrectAnswer: median(sh)
    AnswerTests: omnitest(correctExpr='median(sh)')
    Hint: Type median(sh) at the command prompt.


  - Class: text
    Output: Pretty close, right? Now back to theory. Suppose you have a statistic that
    estimates some population parameter, but you don't know its sampling distribution.
    The bootstrap principle uses the distribution defined by the observed data to
    approximate the sampling distribution of that statistic.

  - Class: text
    Output: The nice thing about bootstrapping is that you can always do it with
    simulation. The general procedure follows by first simulating B complete data sets
    from the observed data by sampling with replacement. Make sure B is large and that
    you're sampling WITH replacement to create data sets the same size as the original.

  - Class: text
    Output: This approximates drawing from the sampling distribution of that statistic,
    at least as well as the data approximates the true population distribution. By
    calculating the statistic for each simulated data set and using these simulated
    statistics we can either define a confidence interval (e.g. find the 2.5 and the 97.5
    percentiles) or take the standard deviation to estimate a standard error of that
    statistic.

  - Class: text
    Output: Notice that this process doesn't use any fancy math or asymptotics. The only
    assumption behind it is that the observed sample is representative of the underlying
    population.

  - Class: text
```

```
    Output: We've created the vector fh for you which contains the fathers' heights from
    the father son data we've been working with. It's the same length as the sons' data
    (1078) which is stored in nh. B, the number of bootstraps we want has been set to
    1000. We'll do an example now in small steps.

  - Class: cmd_question
    Output:  Our one sample of observed data is in the vector fh. Use the R function
    sample to sample fh nh*B times. Set the argument replace to TRUE. Put the result in
    the variable sam.
    CorrectAnswer: sam <- sample(fh,nh*B,replace=TRUE)
    AnswerTests: expr_creates_var('sam'); omnitest(correctExpr='sam <-
    sample(fh,nh*B,replace=TRUE)')
    Hint: Type sam <- sample(fh,nh*B,replace=TRUE) at the command prompt.

  - Class: cmd_question
    Output:  Now form sam into a matrix with B rows and nh columns. Use the R function
    matrix and put the result in resam
    CorrectAnswer: resam <- matrix(sam,B,nh)
    AnswerTests: expr_creates_var('resam'); omnitest(correctExpr='resam <-
    matrix(sam,B,nh)')
    Hint: Type resam <- matrix(sam,B,nh) at the command prompt.

  - Class: cmd_question
    Output:  Now use the R function apply to take the median (third argument) of each row
    of resam (first argument). Put the result in meds. The second argument, the number 1,
    specifies that the application of the function is to the rows of the first argument.
    CorrectAnswer: meds <- apply(resam,1,median)
    AnswerTests: obliterate("sam"); expr_creates_var('meds'); omnitest(correctExpr='meds
    <- apply(resam,1,median)')
    Hint: Type meds <- apply(resam,1,median) at the command prompt.

  - Class: cmd_question
    Output:  Now look at the difference between the median of fh and the median of meds.
    CorrectAnswer: median(meds)-median(fh)
    AnswerTests: ANY_of_exprs("median(meds)-median(fh)","median(fh)-median(meds)")
    Hint: Type median(meds)-median(fh) or median(fh)-median(meds) at the command prompt.

  - Class: cmd_question
    Output:  Pretty close, right? Now use the R function sd to estimate the standard
    error of the vector meds.
    CorrectAnswer: sd(meds)
    AnswerTests: obliterate("resam"); omnitest(correctExpr='sd(meds)')
    Hint: Type sd(meds) at the command prompt.

  - Class: cmd_question
    Output:  We previously did this same process for the sons' data and stored the
    resampled medians in the 1000-long vector resampledMedians. Find the standard error
    of resampledMedians.
    CorrectAnswer: sd(resampledMedians)
    AnswerTests: omnitest(correctExpr='sd(resampledMedians)')
    Hint: Type sd(resampledMedians) at the command prompt.

  - Class: cmd_question
    Output:  Now we'll find a 95% confidence interval for the sons' data with the R
    function quantile. The first argument is the vector of resampledMedians and the
    second is the expression c(.025,.975). Do this now.
    CorrectAnswer: quantile(resampledMedians,c(.025,.975))
    AnswerTests: omnitest(correctExpr='quantile(resampledMedians,c(.025,.975))')
    Hint: Type quantile(resampledMedians,c(.025,.975)) at the command prompt.

  - Class: cmd_question
    Output:  Pretty close quantiles, right? Now do the same thing for the fathers' data.
    Recall that it's stored in the vector meds.
    CorrectAnswer: quantile(meds,c(.025,.975))
    AnswerTests: omnitest(correctExpr='quantile(meds,c(.025,.975))')
    Hint: Type quantile(meds,c(.025,.975)) at the command prompt.

  - Class: text
    Output: Another pair of close quantiles, but notice that these quantiles of the
```

```
    fathers' medians differ from those of the sons.

  - Class: text
    Output: Bootstrapping is a very diverse and complicated topic and we just skimmed the
    surface here. The technique we showed you is nonparametric, that is, it's not based
    on any parameterized family of probability distributions. We  used only one set of
    observations that we assumed to be representative of the population.

  - Class: text
    Output: Finally, the confidence intervals we calculated might not perform very well
    because of biases but the  R package bootstrap provides an easy fix for this problem.

  - Class: text
    Output: Now, to permutation testing, another handy tool used in group comparisons. As
    bootstrapping did, permutation testing samples a single dataset a zillion times and
    calculates a statistic based on these samplings.

  - Class: text
    Output: Permutation testing, however, is based on the idea of exchangability of group
    labels. It measures whether or not outcomes are independent of group identity. Our
    zillion samples simply permute group labels associated with outcomes. We'll see an
    example of this.

  - Class: figure
    Output: Here's a picture from the dataset InsectSprays which contains counts of the
    number of bugs killed by  six different sprays.
    Figure: insectSprays.R
    FigureType: new

  - Class: text
    Output: We'll use permutation testing to compare Spray B with Spray C.

  - Class: cmd_question
    Output:  Use the R command dim to find the dimensions of InsectSprays.
    CorrectAnswer: dim(InsectSprays)
    AnswerTests: omnitest(correctExpr='dim(InsectSprays)')
    Hint: Type dim(InsectSprays) at the R prompt.

  - Class: cmd_question
    Output:  Now use the R command names to find what the two columns of InsectSprays
    contain.
    CorrectAnswer: names(InsectSprays)
    AnswerTests: omnitest(correctExpr='names(InsectSprays)')
    Hint: Type names(InsectSprays) at the R prompt.

  - Class: text
    Output: We'll use permutation testing to compare Spray B with Spray C. We subsetted
    data for these two sprays into a data frame subdata. Moreover, the two data frames
    Bdata and Cdata contain the data for their respective sprays.

  - Class: cmd_question
    Output:  Now use the R command range on Bdata$count to find the minimum and maximum
    counts for Spray B.
    CorrectAnswer: range(Bdata$count)
    AnswerTests: omnitest(correctExpr='range(Bdata$count)')
    Hint: Type range(Bdata$count) at the R prompt.

  - Class: cmd_question
    Output:  The picture makes more sense now, right? Now do the same for Spray C. Its
    data  is in Cdata.
    CorrectAnswer: range(Cdata$count)
    AnswerTests: omnitest(correctExpr='range(Cdata$count)')
    Hint: Type range(Cdata$count) at the R prompt.

  - Class: text
    Output: From the ranges (as well as the picture), the sprays look a lot different.
    We'll test the (obviously false) null hypothesis that their means are the same.

  - Class: cmd_question
```

```
    Output:  To make the analysis easier we've defined two arrays for you, one holding
    the counts for sprays B and C. It's call BCcounts. Look at it now.
    CorrectAnswer: BCcounts
    AnswerTests: omnitest(correctExpr='BCcounts')
    Hint: Type BCcounts at the command prompt.

  - Class: cmd_question
    Output:  The second array we've defined holds the spray identification and it's
    called group. These two arrays line up with each other, that is, the first 12 entries
    of counts are associated with spray B and the last 12 with spray C.   Look at group
    now.
    CorrectAnswer: group
    AnswerTests: omnitest(correctExpr='group')
    Hint: Type group at the command prompt.

  - Class: cmd_question
    Output:  We've also defined for you a one-line function testStat which takes two
    parameters, an array of counts and an array of associated identifiers. It assumes all
    the counts come from group B or group C. It subtracts the mean of the counts from
    group C from the mean of the counts of group B. Type testStat with no parentheses and
    no arguments to see how it's defined.
    CorrectAnswer: testStat
    AnswerTests: omnitest(correctExpr='testStat')
    Hint: Type testStat at the command prompt.

  - Class: cmd_question
    Output:  Now set a variable obs by invoking testStat with the arguments BCcounts and
    group and assigning the result to obs.
    CorrectAnswer: obs <- testStat(BCcounts,group)
    AnswerTests: expr_creates_var('obs'); omnitest(correctExpr='obs <-
    testStat(BCcounts,group)')
    Hint: Type obs <- testStat(BCcounts,group) at the command prompt.

  - Class: cmd_question
    Output:  Take a peek at obs now.
    CorrectAnswer: obs
    AnswerTests: omnitest(correctExpr='obs')
    Hint: Type obs at the command prompt.

  - Class: cmd_question
    Output:  Pretty big difference, right? You can check this by using mean on
    Bdata$count and on Cdata$count and subtracting the latter from the former.
    Equivalently, you can just apply mean to Bdata$count-Cdata$count. Do either one now.
    CorrectAnswer: mean(Bdata$count)-mean(Cdata$count)
    AnswerTests:
    ANY_of_exprs("mean(Bdata$count)-mean(Cdata$count)","mean(Bdata$count-Cdata$count)")
    Hint: Type mean(Bdata$count)-mean(Cdata$count) at the command prompt.

  - Class: mult_question
    Output: So, mean(Bdata$count)-mean(Cdata$count) equals mean(Bdata$count-Cdata$count)
    because ?
    AnswerChoices: the data is special;  mean is linear; mathemagic;
    CorrectAnswer:  mean is linear
    AnswerTests: omnitest(correctVal='mean is linear')
    Hint: Recall the mean or average is linear so the mean of a sum is the sum of the
    means.

  - Class: text
    Output: Now this is where the permutation testing starts to involve resampling. We're
    going to test whether or not the particular group association of the counts affects
    the difference of the means.

  - Class: text
    Output: We'll keep the same array of counts, just randomly relabel them, by permuting
    the group array. R makes this process very easy. Calling the function sample (which
    we've used several times in this lesson) with one argument, an array name, will
    simply permute the elements of that array.
```

```
- Class: cmd_question
  Output:  Call sample now on the array group to see what happens.
  CorrectAnswer: sample(group)
  AnswerTests: omnitest(correctExpr='sample(group)')
  Hint: Type sample(group) at the command prompt.

- Class: text
  Output: "The labels are all mixed up now. We'll do this permuting of labels and then
  we'll recalculate the difference of the means of the two \"new\" (really newly
  labelled) groups."

- Class: cmd_question
  Output:  "We'll relabel and calculate the difference of means 10000 times and store
  the differences (of means) in the array perms. Here's what the code looks like perms
  <- sapply(1 : 10000, function(i) testStat(BCcounts, sample(group))). Try it now."
  CorrectAnswer: "perms <- sapply(1 : 10000, function(i) testStat(BCcounts,
  sample(group)))"
  AnswerTests: "expr_creates_var('perms'); omnitest(correctExpr='perms <- sapply(1 :
  10000, function(i) testStat(BCcounts, sample(group)))')"
  Hint: "Type perms <- sapply(1 : 10000, function(i) testStat(BCcounts, sample(group)))
  at the command prompt."

- Class: cmd_question
  Output: We can take the mean of the virtual array of the boolean expression perms >
  obs. Do this now.
  CorrectAnswer: mean(perms>obs)
  AnswerTests: omnitest(correctExpr='mean(perms>obs)')
  Hint: Type mean(perms>obs) at the command prompt.

- Class: text
  Output: So on average 0 of the permutations had a difference greater than the
  observed. That means we would reject the null hypothesis that the means of the two
  sprays were equal.


- Class: figure
  Output: Here's a histogram of the difference of the means. Looks pretty normal,
  right? We can see that the distribution runs roughly between -10 and +10 and it's
  centered around 0. The vertical line shows where the observed difference of means was
  and we see that it's pretty far away from the distribution of the resampled
  permutations. This means that group identification did matter and sprays B and C were
  quite different.
  Figure: insectHisto.R
  FigureType: new

- Class: figure
  Output: Here's the picture of the InsectSprays again. Suppose we run the same
  experiment, this time comparing sprays D and E, which look more alike. We've
  redefined testStat to look at these sprays and subtract the mean of spray E from the
  mean of spray D.
  Figure: insectSprays2.R
  FigureType: new

- Class: cmd_question
  Output: We've also stored off the D and E data in DEcounts and the group labels in
  group. Run testStat now with DEcounts and group.
  CorrectAnswer: testStat(DEcounts,group)
  AnswerTests: omnitest(correctExpr='testStat(DEcounts,group)')
  Hint: Type testStat(DEcounts,group) at the command prompt.

- Class: cmd_question
  Output: "We've  stored off this value, 1.416667, in the variable obs for you. Now run
  the permutation command, with DEcounts. Here it is, perms <- sapply(1 : 10000,
  function(i) testStat(DEcounts, sample(group)))"
  CorrectAnswer: "perms <- sapply(1 : 10000, function(i) testStat(DEcounts,
  sample(group)))"
  AnswerTests: "expr_creates_var('perms'); omnitest(correctExpr='perms <- sapply(1 :
  10000, function(i) testStat(DEcounts, sample(group)))')"
  Hint: "Type perms <- sapply(1 : 10000, function(i) testStat(DEcounts, sample(group)))
```

```
    at the command prompt."

  - Class: figure
    Output: Finally, we can plot the histogram of the distribution of the difference of
    the means. We see that with these sprays the observed difference of means (the
    vertical line) is closer to the mean of the permuted labels. This indicates that
    sprays D and E are quite similar and we fail to reject the null hypothesis that the
    means were equal.
    Figure: insectHisto.R
    FigureType: new

  - Class: text
    Output: Congrats! We hope you weren't bugged too much by this lesson and feel like
    you've pulled yourself up by your bootstraps.

  - Class: mult_question
    Output: "Would you like to receive credit for completing this course on
      Coursera.org?"
    CorrectAnswer: NULL
    AnswerChoices: Yes;No
    AnswerTests: coursera_on_demand()
    Hint: ""
```