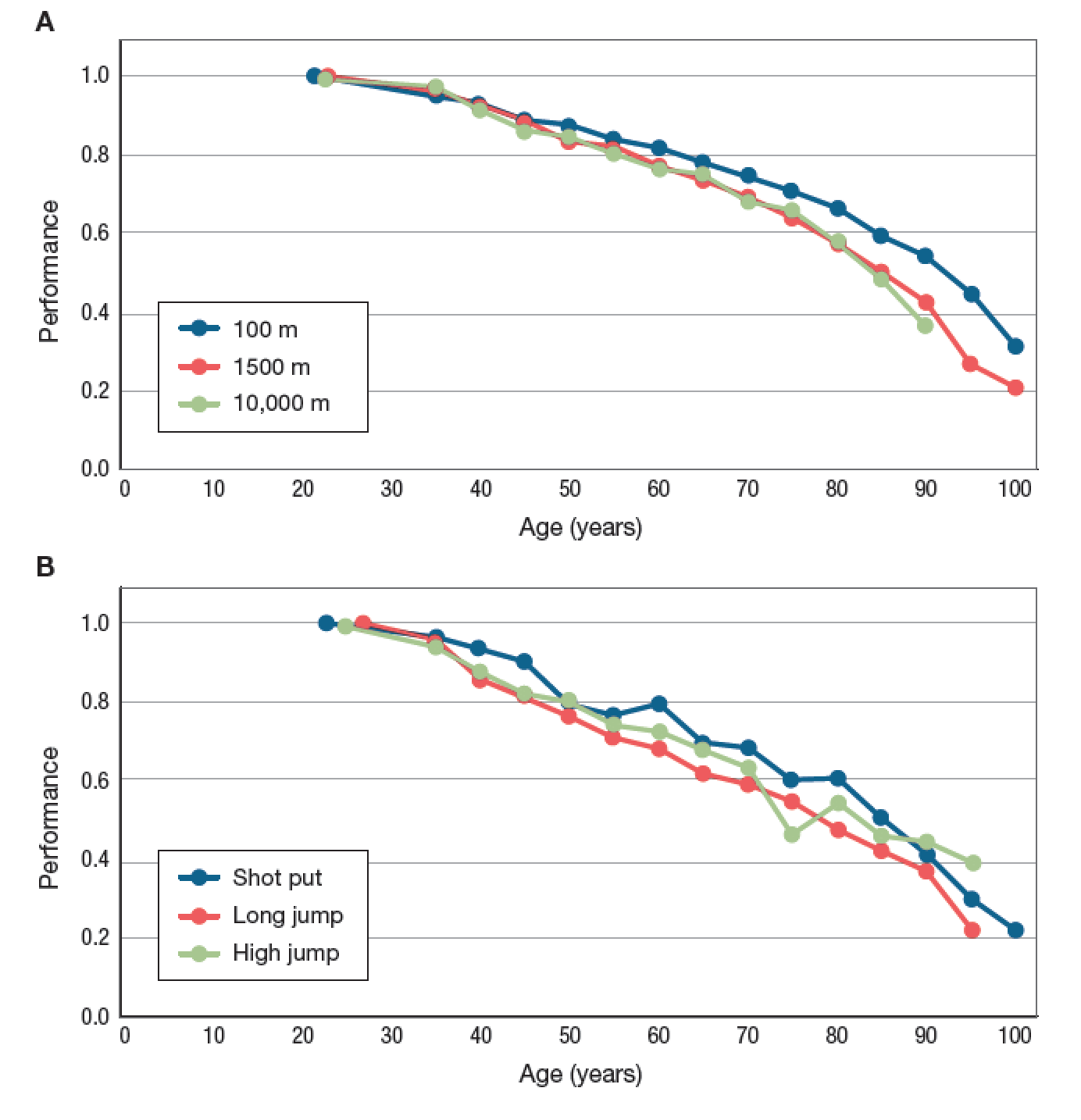
**Track and Field Case Study**

An introduction to sampling from a distribution with R, inspired by [Calling Bull](https://callingbullshit.org/)’s [Track and Field Case Study](https://callingbullshit.org/case_studies/case_study_track_records.html), by Dr. Diaz Eaton, Bates College, for use in DCS 105A.

Learning Objectives:

* Explore how the track and field figure was debunked, using a null model.
* Practice R syntax and RStudio environment for installing and loading packages, mathematical functions, ? and args().
* Provide optional practice efficient code implementations using sapply & for loops.

**Testing for deviations from an expected distribution**

In a few of the case studies, particularly the scientific ones, we’ve talked about some kind of measure of statistical difference, such as a p-value. Recall one definition of the p-value: The **p-value** tells us how likely our observation is a result of chance alone (sampling from the null distribution - the distribution that the null hypothesis and the statistical test would assume).

While there are some derivations of mathematical formulas that can simplify this determination, one way to explore the influence of probability and and underlying distribution is to perform simulation experiments.

**Discuss**

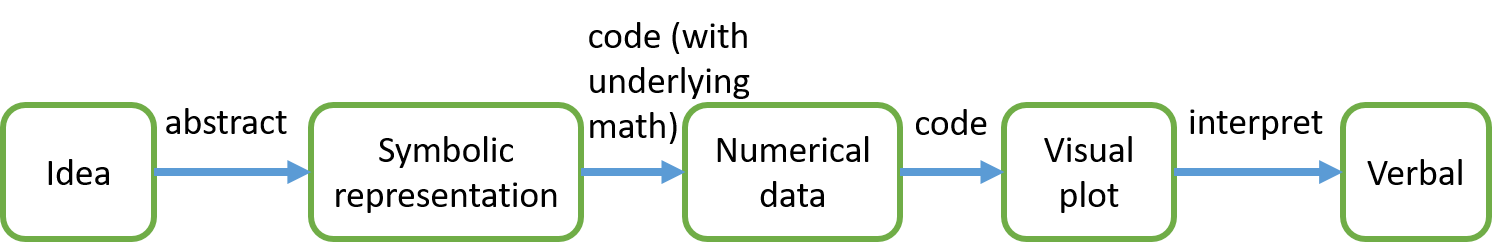
The figure above was originally meant to exhibit senescence in track and field, the “physical decline humans undergo with age.” From the Calling Bull Case Study, the alternative hypothesis is: "*The problem with that graph is that the sample sizes are very different for different ages..* *How many 30-year-olds run the 1500 meters competitively? How many 80-year-olds? Far, far fewer. Moreover, these world records are outliers. So from the data provided, we don't really know how much of the decline in performance is due to decreases in sample size, and how much of the decline is due to actual senescence.*"

Discuss this alternative hypothesis for the data presented. How would you test this? What would you need to know or to simulate?

**MODELING/PROGRAMMING**

**Modeling** and **programming** are broad terms that encompass a process including elements such as planning, implementation, testing, revision, feedback, etc., for some purpose. The particular type of model we will construct here is a **null model**.

In the case we are considering below, our process of modeling is:



**Abstracted idea in context**

Perhaps the overall track and field athletic performance is not dropping due to age, but just due to a drop in the number of participating athletes. We will simulate picking from a random distribution of athletes, and then assume that the record that year is the maximum value of that sample. When we plot those maximum values, that will give us a visual of the alternative hypothesis for the data observed.

**PLAN**

**\* Coding professional practice - Intentionally plan your code before you start typing.**

We will follow the procedure set forth in the Calling Bull Case study:

“To convince ourselves that the argument is plausible, we can simulate this effect, drawing scores from the same distribution but for populations of size 106, 105, etc., down to 101. For each population, we take the highest score (again scaled relative to the highest score overall) and plot it. In the figure below, I've run a simulation using normally distributed, exponentially, and log-normally distributed scores.”

Create a plan from that description for drawing from a normal distribution:

1. Prepare workspace by...

3. Plot...
4. Repeat 2-4, but drawing the sample from an exponential and a log-normal distribution.

**\*Coding professional practice - sketch out your plan in psuedocode before implementing. Psuedocode is really the wireframe or structure that you will use, but perhaps not the perfect syntax yet (***e.g.* plot with the x-axis as exponents of the sample size and the y axis as the max value**). This may also prompt you to identify some commands you may have to look up. Remember to Google, use ?command, and args().**

**Discuss**

Sketch out some psuedocode for the plan above. Compare and contrast approaches with your group members. What commands will you need to know in order to carry out your plan? Will this require any particular use of structures? What packages will you need to install to use these commands?

**IMPLEMENT**

The steps below may not be the same as yours. That is okay. You may use your plan if you like, but I wanted to give you suggestions for potential commands, etc. Optional challenges are given at each step and focus on code efficiency.

**Step 1:**

Start a new R-script (in RStudio, go to **File > New File > R-script**).

Add the file information to the top of your script.

# Diaz Eaton, DCS 105 A Calling Bull

# Track and Field Case Study, Feb 27, 2019

# Team member names

# Clear out any old variable names with the first code line

rm(list=ls())

#Install and load any packages needed.

#In this case, lets install ggplot2. It was used in the Intermediate R DataCamp tutorial to do qplot. In the past, we have also just used a plot command, which could also work, but since the next DataCamp module is ggplot2, this makes it a logical choice.

# Install and load ggplot2

install.packages(“ggplot2”)

library(ggplot2)

**Step 2:**

**\*Coding professional practice - name the parameters close to what they mean. Use # (aka “comment out”) before descriptors that explain what you are doing and/or the units, or where the numbers you are using come from.**

#Make 6 sample from the normal distribution and store each in a vector.

#sample1, sample2, …, sample6 where samplei is sampling 10^(ith)

sample1 <- abs(rnorm(10)) #use abs because bellcurve is + and -

sample2 <- abs(rnorm(10^2))

sample3 <-

sample4 <-

sample5 <-

sample6 <-

# You fill in the rest!

#Try using ?rnorm to see how to use rnorm. What kind of distribution is this sampling from and what are the assumptions? Use args(rnorm) to see the possible inputs and the defaults.

**\*Coding professional practice - build in ways to check your code to make sure it is functioning as intended.**

Look at your environment window in RStudio. Are the variables storing as expected?

#Optional: To practice coding efficiency, when you are done, consider how you might make the same code in one line by storing the sample vectors into a data.frame or list. Hint: To make this vector assignment faster, check out [this stack overflow thread](https://stackoverflow.com/questions/14399868/how-to-make-a-vector-using-a-for-loop). You could try a for loop (why not practice your for loop DataCamp skills!), or you could harness the power of the vector (for expert points).

**Step 3:**

#Declare y variable for plotting

# Y vector: find the max of each and store that in a vector. consider that the first data point should be the max of the 10^6 sample

max\_samp <- rev(c(max(sample1), max(sample2), max(sample3), max(sample4), max(sample5), max(sample6)))

# Scale to a 0 to 1 by dividing by the highest performance

max\_samp <- max\_samp/max(max\_samp)

# Optional: If you did the optional challenge in step one, now consider how you might make the step 2 process faster for the Y vector by using one of the apply commands.

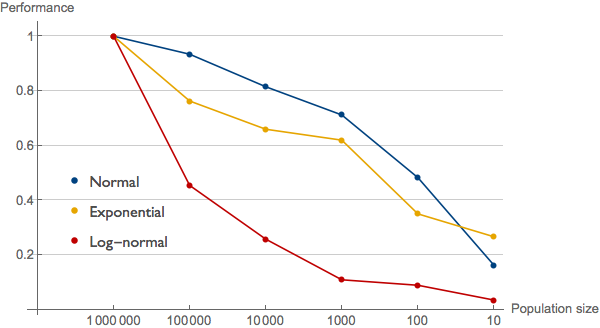
**Step 4:**

**#**Plot the data using qplot which is a function in ggplot2. Use ? and args() to figure out the how to use qplot. Think about the best practices in graphing that you talked about in class. Here’s a [cool tutorial](http://www.sthda.com/english/wiki/qplot-quick-plot-with-ggplot2-r-software-and-data-visualization#scatter-plots-with-smoothed-line).

#For this plot, I’m not going to use the sample size as the actual x value, just because it requires additional work to flip the x axis to decreasing (the default is to plot data as increasing along the x-axis, not in the order of the points) and to deal with log scaling it. [One alone isn’t too difficult to do](http://www.sthda.com/english/wiki/ggplot2-axis-scales-and-transformations#log-and-sqrt-transformations), but [both become complicated](https://stackoverflow.com/questions/12570270/apply-two-transformations-on-one-axis). Therefore we will only flip the axis, and the x-value should be interpreted as the log of the sample size (or the exponent of 10^) instead of log-scaling.

#scatterplot, x is 6,5,4,3,2,1, y is max\_samp, which makes each (x,y) data point, geom tells us how to plot, scale\_x\_reverse flips the axis.

qplot(6:1, max\_samp, xlab = "Log(Sample Size)", ylab = "Relative Performance", geom=c("point", "line")) + scale\_x\_reverse()



Track and Field Case Study Figure from Calling Bull. Your code should replicate the blue line/data points as samples are being drawn from a normal distribution.

#Optional: Beside reducing the number of lines in a code, another way to increase efficiency is to refrain from storing values that will not be useful later. We generated a vector of length 1 million only to use the maximum value alone for the rest of the analysis. What if we took the max at the point when we generated the original list?

**ANALYZE**

**Discuss**

What result did you get? Same as Calling Bull? Different? Is the code working properly?

**PLAY**

Can you get the line to be blue instead of black? Play around a bit with the graphing command to see what happens.

**MODIFY (if time)**

1. **Discuss & Plan:** Performance might be picked from other distribution types. How would you modify the code you wrote in the script to pull from an exponential or a log-normal distribution?
2. **Implement**
3. **Analyze output:** 
   1. What answer did you get?
   2. Does it agree with the Calling Bull Case Study Figure?
   3. What does it mean?

**FAQ:**

Q: What does it mean that you are drawing from a distribution?

A: Imagine that you had a bunch of runners. We stacked them up in bins of some measurement of performance level, just as we stacked up the number of hands that graded the visualization Monday as a 7 or as a 8 into a distribution. Now if we go back and chose a random square from that distribution, we read the score, record it as another entry in our vector, and put it back. If you do this 10 times, you get a vector of length 10. If you do it 100 times, you get a vector of length 100, etc. The computer simulates this process. If you use rnorm(10) it will do it 10 times, pulling from a bell-shaped curve with mean 0 and standard deviation 1 (means that about 2/3rd of your numbers will probably be between -1 and 1).

Q: How does your distribution affect the outcome?

A: Imagine that you had the kind of population that was mostly between 0 and 1, and then only a couple at a 2 performance or a 3. You'd mostly be drawing 0 and 1s and rarely hit a 3 or greater. Contrast that with a distribution that has all values equally likely between 0 and 4. Getting a 3 or more is no longer a rare event. So the underlying distribution you assume, matters. For a quick set of pictures with underlying distributions see: [link](https://docs.google.com/document/d/1LJREvvYDgPStMq6riFZp0jUOo0bGoJB9rW2-T1ftEks/edit?usp=sharing).

Q: How does sample size affect the outcome?

A: In the case of a uniform distribution, where all values between 0 and 4 are equally likely, there might be little effect. The maximum of each of the samples is probably close. However, in the case of the distributions considered in the project, getting a 3 or larger was a rare event. So if you have a small sample size, you are unlikely to experience that event - the maximum is likely to be only very high when your sample size is very large.

Q: When I looked at drawing for the normal distribution and the exponential or log-normal, they look similar. Also, the graph is not always concave down. Why is that? Did Carl feed us more BS while refuting BS?

A: There are a few things going on. 1) Because we are taking the absolute value - because I'm not sure what negative performance curves mean, we aren't really using the normal distribution like Carl does in his simulations - we are using the "folded normal." It is possible this contributes - you can remove the absolute value to see. 2) When we actually run stochastic simulation models, because of the sampling, we usually run 100 or 1000 and take the average of all of those simulations - wrap a for loop around your code! 3) The y axis auto adjusts, so I added a ylim = c(0,1) to qplot so that I'd be better able to compare trends across graphs. The result? Well if you use the average result, we get something much closer to the smooth concave down graph that Carl gets. and the difference between using rnorm and abs(rnorm) is minimal. See the pictures below. I challenge you also to reproduce the result - this is a great application of your Intermediate R skills!

