Week 2 Part 1

Random Variables Exercises

```
library(downloader)
url <- "https://raw.githubusercontent.com/genomicsclass/dagdata/master/inst/extdata/femaleControlsPopula
filename <- basename(url)
download(url, destfile=filename)
x <- unlist( read.csv(filename) )
```

```
RNGkind("Mersenne-Twister", "Inversion", "Rejection")
```

Random Variables Exercises #1

```
mean(x)
```

```
## [1] 23.89338
```

Random Variables Exercises #2

Take a random sample of size 5. What is the absolute value (use abs()) of the difference between the average of the sample and the average of all the values?

```
set.seed(1)
rand_sample <- sample(x,size = 5)

abs_diff <- abs((mean(rand_sample)-mean(x)))
abs_diff
```

```
## [1] 0.3293778
```

Random Variables Exercises #3

After setting the seed at 5, set.seed(5), take a random sample of size 5. What is the absolute value of the difference between the average of the sample and the average of all the values?

```
set.seed(5)
rand_sample <- sample(x,size = 5)

abs_diff <- abs((mean(rand_sample)-mean(x)))
abs_diff
```

```
## [1] 0.3813778
```

Null Distributions Exercises

```
library(downloader)
url <- "https://raw.githubusercontent.com/genomicsclass/dagdata/master/inst/extdata/femaleControlsPopula
filename <- basename(url)
download(url, destfile=filename)
x <- unlist( read.csv(filename) )
```

Here x represents the weights for the entire population.

Null Distributions Exercises #1

Set the seed at 1, then using a for-loop take a random sample of 5 mice 1,000 times. Save these averages.

What proportion of these 1,000 averages are more than 1 gram away from the average of x ?

```
set.seed(1)
n <- 1000
avg_mice <- vector('numeric',n)
for (k in 1:n) {
  control <- sample(x,size=5)
  avg_mice[k] <- mean(control)
}

mean(abs(avg_mice-mean(x))>1)
```

```
## [1] 0.503
```

Null Distributions Exercises #2

We are now going to increase the number of times we redo the sample from 1,000 to 10,000. Set the seed at 1, then using a for-loop take a random sample of 5 mice 10,000 times. Save these averages.

What proportion of these 10,000 averages are more than 1 gram away from the average of x ?

```
set.seed(1)
n <- 10000
avg_mice <- vector('numeric',n)
for (k in 1:n) {
  control <- sample(x,size = 5)
  avg_mice[k] <- mean(control)
}

mean(abs(avg_mice-mean(x))>1)
```

```
## [1] 0.5084
```

Probability Distributions Exercises #1

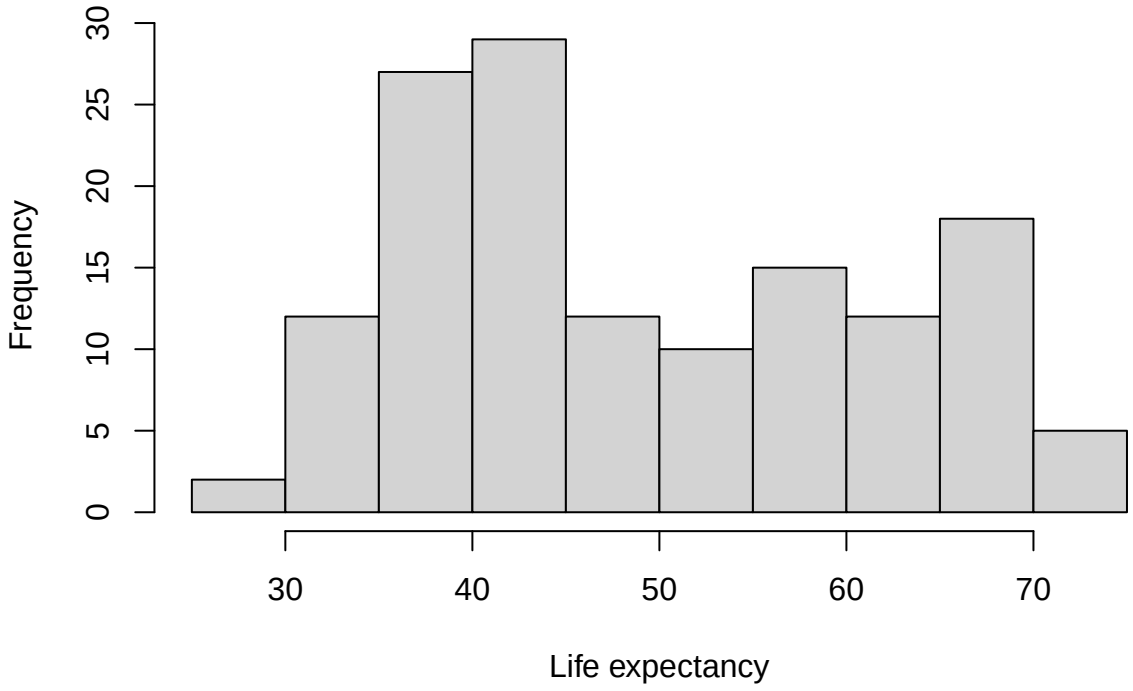
```
library(gapminder)
data(gapminder)
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Create a vector x of the life expectancies of each country for the year 1952. Plot a histogram of these life expectancies to see the spread of the different countries.

```
x <- gapminder[gapminder$year == 1952, ]

hist(x$lifeExp,xlab = 'Life expectancy',main = '')
```



In statistics, the empirical cumulative distribution function (or empirical cdf or empirical distribution function) is the function F(a) for any a, which tells you the proportion of the values which are less than or equal to a.

We can compute F in two ways: the simplest way is to type mean(x <= a). This calculates the number of values in x which are less than or equal to a, divided by the total number of values in x, in other words the proportion of values less than or equal to a.

The second way, which is a bit more complex for beginners, is to use the ecdf() function. This is a bit complicated because this is a function that doesn't return a value, but a function.

Let's continue, using the simpler mean() function.

What is the proportion of countries in 1952 that have a life expectancy less than or equal to 40?

```
mean(x[,4] <= 40)
```

```
## [1] 0.2887324
```

sapply() on a custom function

Suppose we want to plot the proportions of countries with life expectancy q for a range of different years. R has a built in function for this, plot(ecdf(x)), but suppose we didn't know this. The function is quite easy to build, by turning the code from question 1.1 into a custom function, and then using sapply(). Our custom function will take an input variable q, and return the proportion of countries in x less than or equal to q. The curly brackets, { and }, allow us to write an R function which spans multiple lines:

```
prop = function(q) {
  mean(x <= q)
}
```

Try this out for a value of q: prop(40)

```
x <- x[,4]
prop(40)
```

```
## [1] 0.2887324
```

Now let's build a range of qs that we can apply the function to:

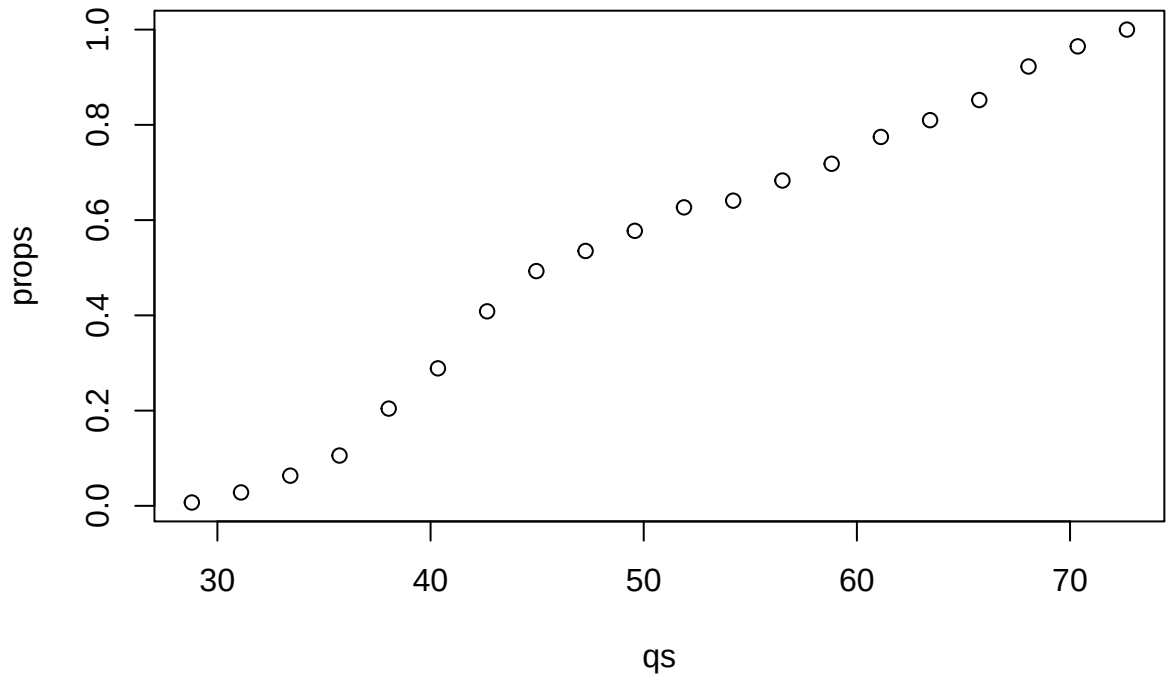
```
qs = seq(from=min(x), to=max(x), length=20)
```

Print qs to the R console to see what the seq() function gave us. Now we can use sapply() to apply the prop function to each element of qs:

```
props = sapply(qs, prop)
```

Take a look at props, either by printing to the console, or by plotting it over qs:

```
plot(qs, props)
```

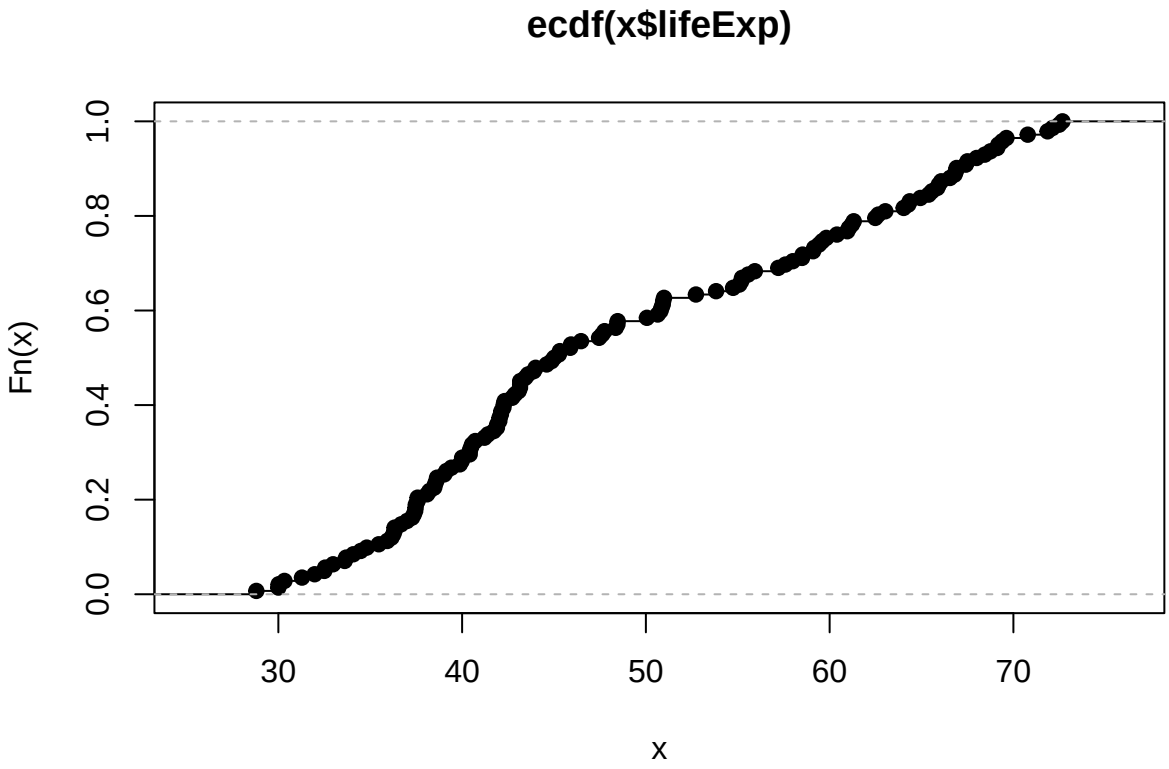


Note that we could also have written this in one line, by defining the prop function inside of sapply() but without naming it:

```
props = sapply(qs, function(q) mean(x <= q))
```

This last style is called using an "inline" function or an "anonymous" function. Let's compare our homemade plot with the pre-built one in R:

```
plot(ecdf(x$lifeExp))
```


ecdf(x$lifeExp)
Fn(x)
0.0
0.2
0.4
0.6
0.8
1.0
30
40
50
60
70
x