Week2

Introduction to Random Variable Assessemnt

Let's start by loading in the data that Rafa used in the video. Remember to click on "Raw" to download individual files from github.

```
dat = read.csv("femaleMiceWeights.csv")
```

The observed difference between high fat diet and control was calculated like so:

```
mean(dat[13:24,2]) - mean(dat[1:12,2])
```

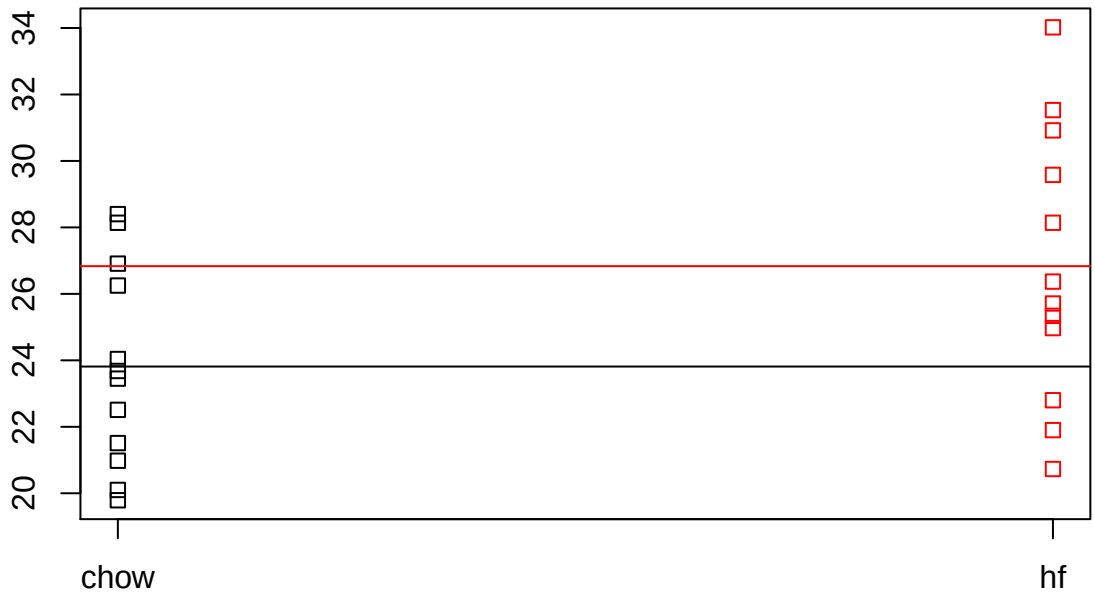
```
## [1] 3.020833
```

Let's make a plot of these two groups: a strip chart of the weights. We're going to use some of the functions that we used from the assessments in Week 1, split() and soon sapply().

Let's add the means to the plot as well:

```
s = split(dat[,2], dat[,1])
stripchart(s, vertical=TRUE, col=1:2)

abline(h=sapply(s, mean), col=1:2)
```



QUESTION 1.1

How many of the high fat mice weigh less than the mean of the control mice (chow)?

```
table(mean(dat[1:12,2])>dat[12:24,2])
```

```
##
## FALSE  TRUE
##    10     3
```

QUESTION 1.2

How many of the control mice (chow) weigh more than the mean of the high fat mice?

```
table(mean(dat[12:24,2])<dat[1:12,2])
```

```
##
## FALSE  TRUE
##     9     3
```

SAMPLE()

In the next video, Rafa will use the sample() function in R to generate random samples of a population.

Before watching that, let's try it out in your R session. Continuing from the code above where we split the weights by diet, s[["hf"]] or equivalently s$hf gives the weights for the high fat diet mice. Save that to a new vector 'highfat':

```
highfat = s[["hf"]]
```

Now print the values for highfat:

```
highfat
```

```
##  [1] 25.71 26.37 22.80 25.34 24.97 28.14 29.58 30.92 34.02 21.90 31.53
## [12] 20.73
```

Now try the following command:

```
sample(highfat, 6)
```

```
## [1] 25.34 21.90 24.97 22.80 20.73 34.02
```

You can press the UP arrow and ENTER to reproduce the command multiple times. The function sample() goes into the 'highfat' vector and chooses 6 values at random (6 is the second argument to sample, which is called the 'size' of the sample). With the default argument settings, sample() will only pick an observation once, so it's like drawing cards from a deck without putting the cards back in the deck. Because all of the value in 'highfat' are unique, they will only show up one time in the sample of 3 using the above line of code.

In the R console, type out `r ?sample` and hit ENTER. Read the argument descriptions for **x, size** and **replace**.

There is an argument to sample() called 'replace' which toggles whether or not an observation can be chosen more than once (whether to replace the observations back in the population after they are chosen once). The default setting for 'replace' is set to FALSE, so if we don't say anything, the sample() function will not choose an observation more than once.

Try the following command a few times:

```
sample(highfat, 6, replace=TRUE)
```

```
## [1] 29.58 20.73 25.71 30.92 28.14 25.71
```

You should notice that some of the time, there are repeated values in the sample. This is because we changed the setting of 'replace' to allow for multiple draws of the same obseration.

In the following video, we will be using sample() with replace=FALSE.

QUESTION 1.3

Finally, we have a short problem showing a trick with calculating proportions. If we have a logical vector, for example produced by the logical expression,

```
highfat  > 30
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE
## [12] FALSE
```

. . . a convenient trick is to use this logical vector directly in functions which usually work on numeric values. What happens inside of the R function is that the TRUE is turned into a 1, and the FALSE is turned into a 0. This happens automatically, so you don't need to convert the vector type yourself:

```
as.numeric(highfat > 30)
```

```
##  [1] 0 0 0 0 0 0 0 1 1 0 1 0
```

For example if we want to know the number of the high fat diet mice that weigh over 30:

```
sum(highfat > 30)
```

```
## [1] 3
```

The proportion of high fat diet mice over 30 is the sum of high fat diet mice over 30 divided by the number of high fat diet mice, in other words, the mean of a vector of 1s and 0s. What is the proportion of high fat diet mice over 30?

```
mean(highfat > 30)
```

```
## [1] 0.25
```

Random Variable Assessemnt II

```
population = read.csv("femaleControlsPopulation.csv")
```

Because this is just one column of data, as you can see with head(), let's just extract the one column as a vector:

```
population = population[,1]
```

QUESTION 2.1

What's the control population mean?

```
mean(population)
```

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

The idea of a random variable, is that we pick at random some samples from the population and then calculate the mean. So depending on which samples we pick, the mean changes, and therefore we say that the mean of a sample of the population is a random variable. In R, we can make a random sample using the sample() function:

```
sample(population, 12)
```

```
##  [1] 22.51 21.60 23.45 20.54 26.14 28.36 23.68 19.90 23.96 20.30 24.77
## [12] 23.53
```

We can calculate the mean for a sample in the normal way:

```
mean(sample(population, 12))
```

```
## [1] 25.61167
```

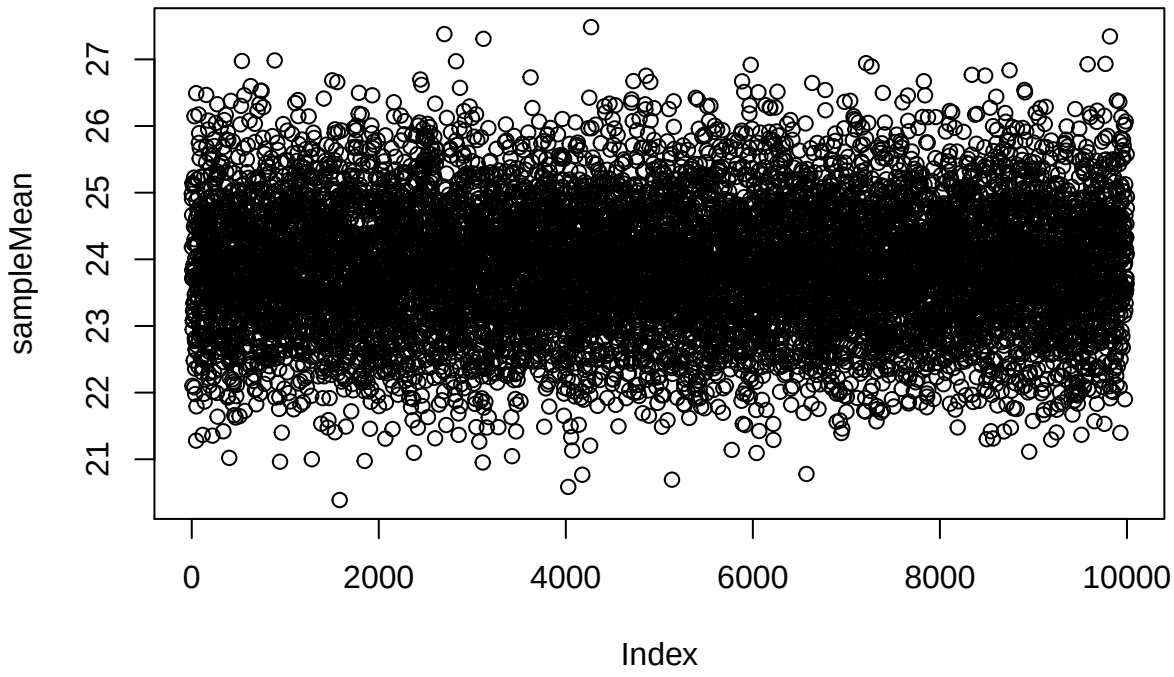
In the video Rafa showed how to perform the whole experiment (pick two groups of 12 and calculate the difference in means) 10,000 times using a for loop. Here we'll try a more concise way to do many random samples, using the **replicate()** function in R. The replicate() function takes two arguments, which are the number of times to replicate, and then an expression: the command you want to replicate. First let's try out making a random sample of one group of 12 mice from the population and calculating the mean:

```
sampleMean = replicate(10000, mean(sample(population, 12)))
head(sampleMean)
```

```
## [1] 24.98333 24.18333 24.90750 24.66583 23.71250 23.83833
```

We can't quiz you on the values you get, because these are random and will be different on everyone's computer! (Although there is an extra command in R to make sure we all get the same result from a random sample command.) Let's plot the different means that we got, spreading them out one at a time along the x axis:

```
plot(sampleMean)
```



We can also use replicate() to perform the same operation that Rafa did in the video: calculating the difference between two random samples of 12 from the control mice:

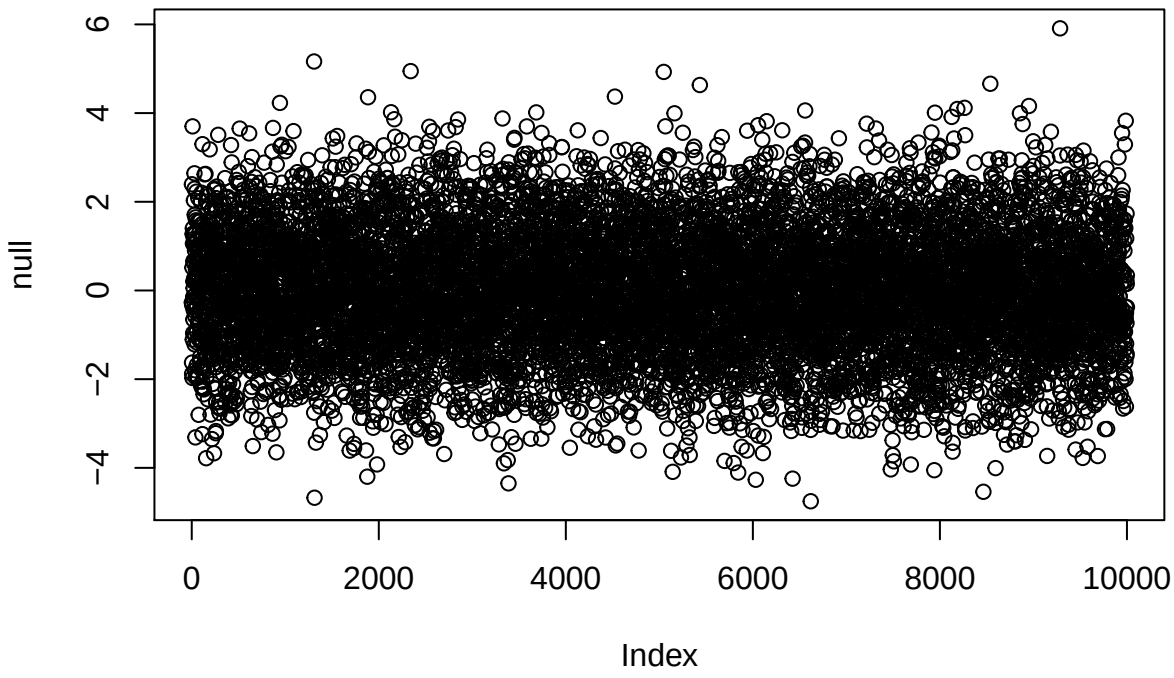
```
null = replicate(10000, mean(sample(population, 12)) - mean(sample(population, 12)))
```

Take a look at a few of these 10,000 differences, and plot them along the x-axis, like we did previously for the mean of 12 random mice:

```
head(null)
```

```
## [1] -0.2716667  2.3950000 -1.6266667 -1.8800000 -0.3300000 -0.2816667
```

```
plot(null)
```



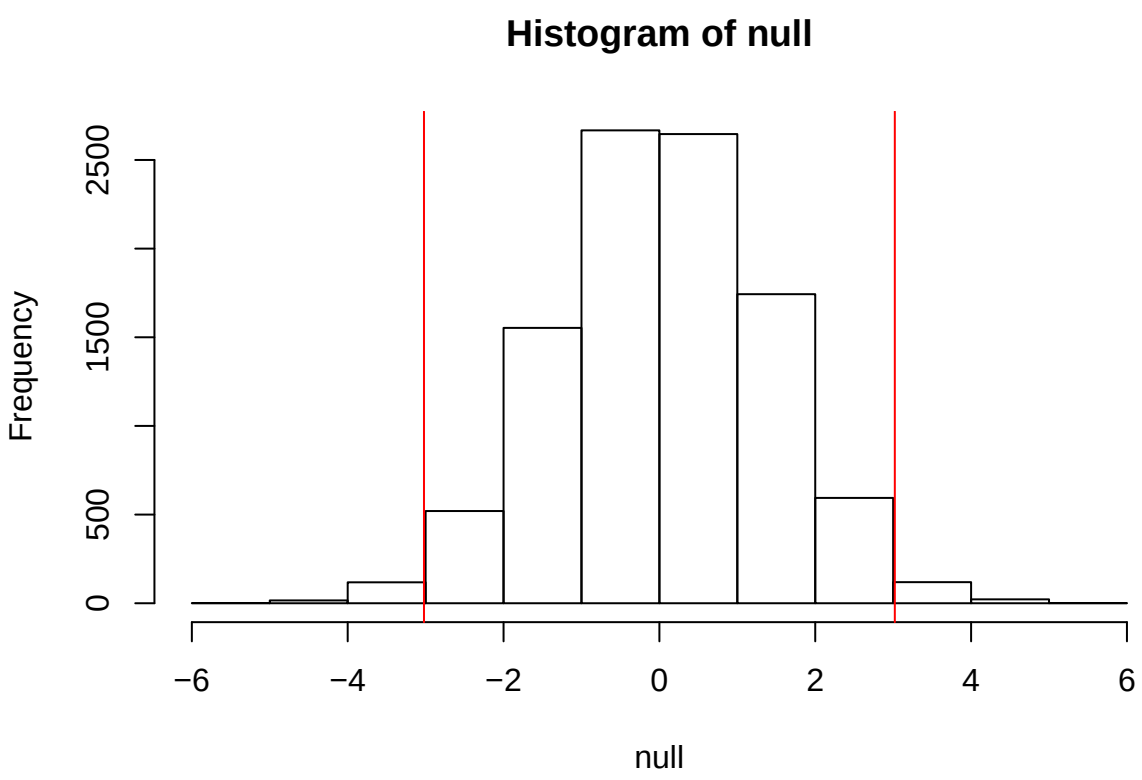
Introduction to Null Distribution Assessment

In the previous assessments, we created a vector of differences between means of random samples from the control population. This gives us a sense of the null distribution of differences if there is no true effect of a high fat diet. Let's recreate that vector:

```
null = replicate(10000, mean(sample(population, 12)) - mean(sample(population, 12)))
```

The simple visualization of stacking the values which are close, in order to see the spread, is incredibly useful. Instead of actually writing the values on the plot, a more common visualization (which we will see again in Week 4), is a histogram. The histogram also puts the values into bins along the x-axis, but instead of writing the value, we just draw a vertical bar with the height equal to the number of values that fell in that bin:

```
hist(null)
diff = mean(dat[13:24,2]) - mean(dat[1:12,2])
abline(v=diff, col="red")
abline(v=-diff, col="red")
```



Let's return to the original difference we observed between the mice fed high fat diets and control mice:

```
diff = mean(dat[13:24,2]) - mean(dat[1:12,2])
```

Now what do we see when we add this difference to the histogram:

```
abline(v=diff, col="red")
```

If we look for the number of null distribution values to the right of the red line, we would say "we calculated the probability of observing a larger difference from the null distribution". This is sometimes called a "one-tailed" probability, because we only look at one "tail" of the histogram (the left and right sides where the bars become short).

We can also add the negative of the difference:

```
abline(v=-diff, col="red")
```

By looking at the tails on both sides of the histogram, we can say "we calculated the probability of observing as extreme a difference from the null distribution". This is sometimes called a "two-tailed" probability. And as Rafa said in the video, this probability is commonly referred to as a p-value.

QUESTION 3.1

Because everyone's random samples are different on different computers, and this was done 10,000 times, everyone's null vector will be slightly different. Therefore, the following questions which involve the null distribution have an allowable range. We have calculated the range such that all students will have answers which fall in the range.

What is the one-tailed probability of seeing as big a difference as we observed, calculated from your null distribution?

```
mean(null>abs(diff))
```

```
## [1] 0.014
```

QUESTION 3.2

What is the two-tailed probability of seeing as big a difference as we observed, calculated from your null distribution?

```
mean(abs(null)>abs(diff))
```

```
## [1] 0.0271
```

Probability Distribution Assessment

The data set called "Gapminder" which is available as an R-package on Github. This data set contains the life expectancy, GDP per capita, and population by country, every five years, from 1952 to 2007. It is an excerpt of a larger and more comprehensive set of data available on Gapminder.org, and the R package of this dataset was created by the statistics professor Jennifer Bryan.

First, install the gapminder data set using the devtools R-package.

```
devtools::install_github("jennybc/gapminder")
```

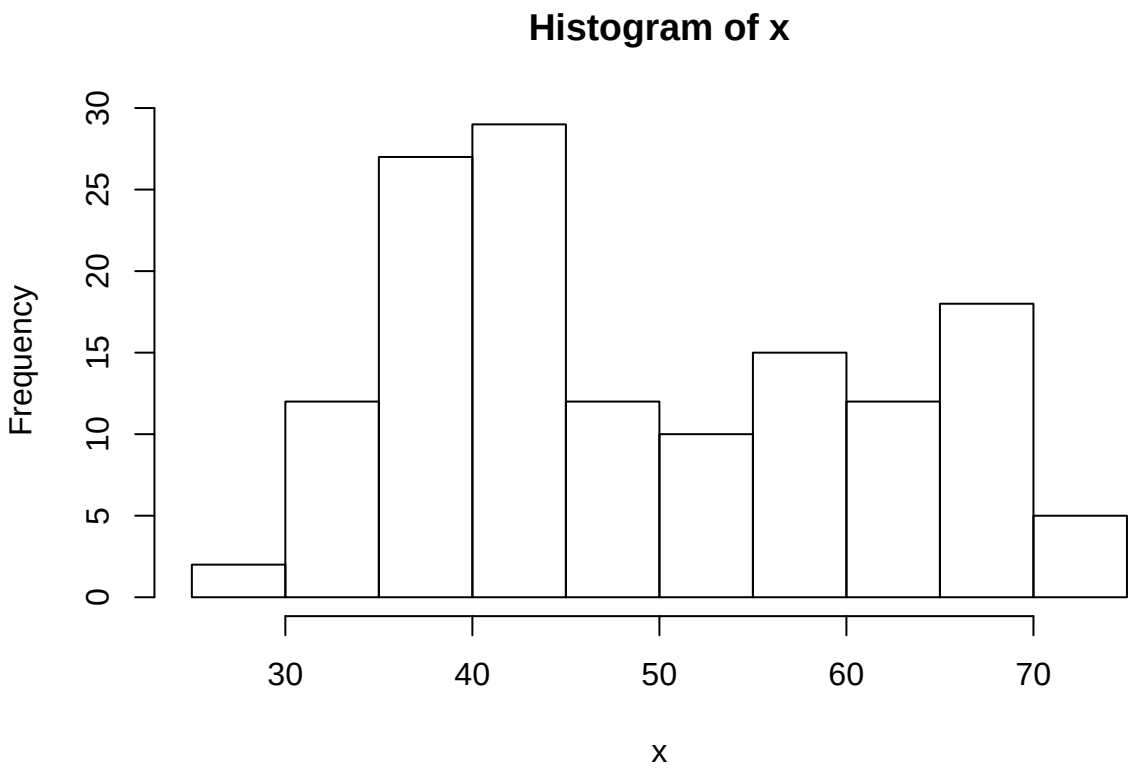
Next, load the gapminder data set. To find out more information about the data set, use `?gapminder` which will bring up a help file. To return the first few lines of the data set, use the function head().

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

```
##       country continent year lifeExp      pop gdpPercap
## 1 Afghanistan      Asia 1952  28.801  8425333  779.4453
## 2 Afghanistan      Asia 1957  30.332  9240934  820.8530
## 3 Afghanistan      Asia 1962  31.997 10267083  853.1007
## 4 Afghanistan      Asia 1967  34.020 11537966  836.1971
## 5 Afghanistan      Asia 1972  36.088 13079460  739.9811
## 6 Afghanistan      Asia 1977  38.438 14880372  786.1134
```

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$lifeExp[which(gapminder$year==1952)]
hist(x)
```



QUESTION 1.1

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 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<=40)
```

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

QUESTION 1.2

What is the proportion of countries in 1952 that have a life expectancy between 40 and 60 years? Hint: this is the proportion that have a life expectancy less than or equal to 60 years, minus the proportion that have a life expectancy less than or equal to 40 years.

```
mean(x<=60)-mean(x<=40)
```

```
## [1] 0.4647887
```

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)
```

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

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

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

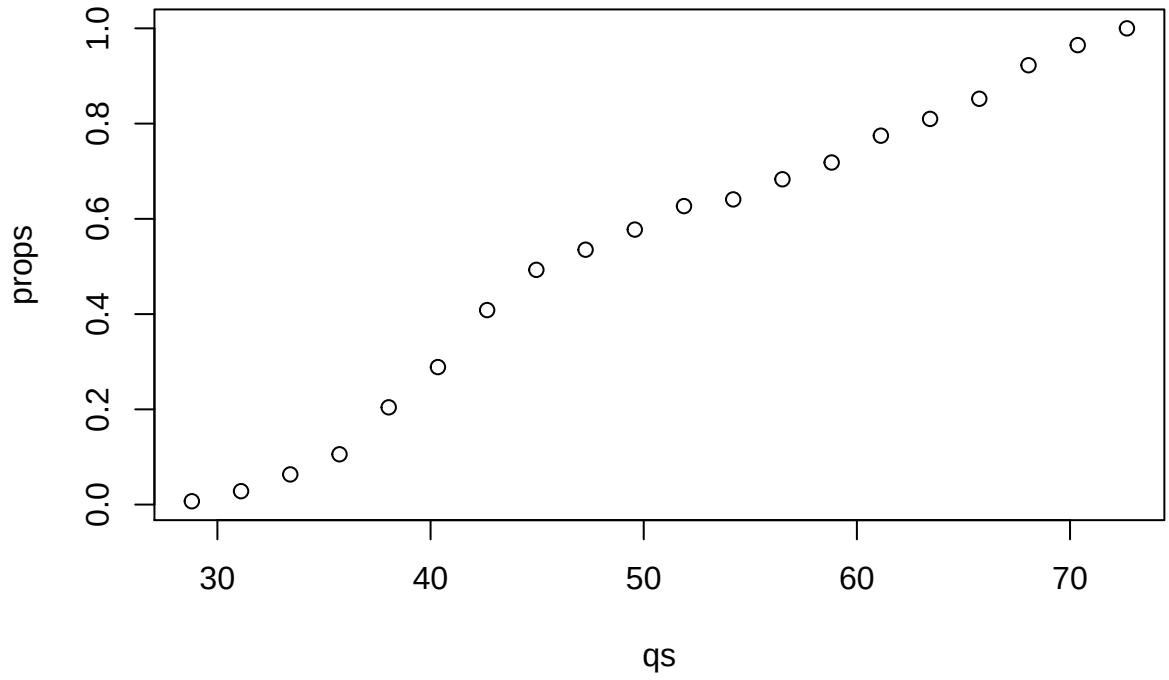
```
##  [1] 28.80100 31.10989 33.41879 35.72768 38.03658 40.34547 42.65437
##  [8] 44.96326 47.27216 49.58105 51.88995 54.19884 56.50774 58.81663
## [15] 61.12553 63.43442 65.74332 68.05221 70.36111 72.67000
```

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)
```

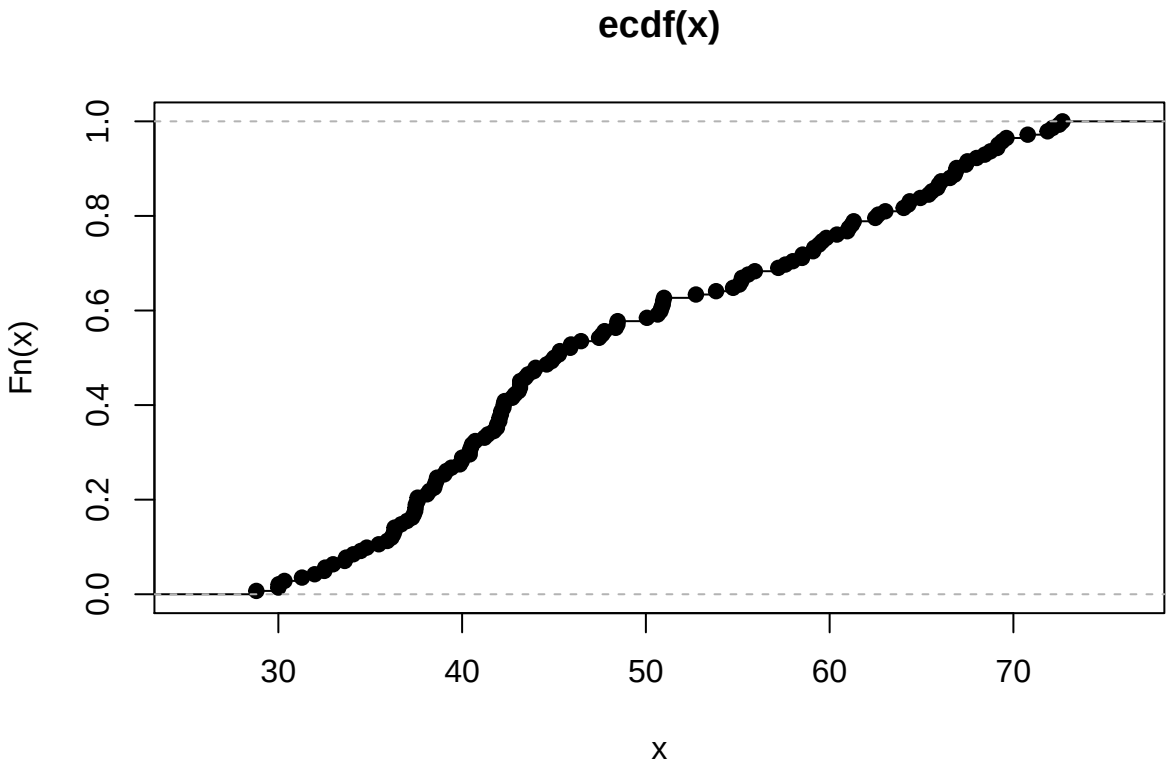


Note that we could also have written this in one line, by defining the 'prop' function but without naming it:

```
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))
```



##The Normal Distribution Assessment Previously, we introduced the concept of summarizing data using empirical distributions. In this tutorial, we introduce one of the most commonly occuring and most commonly used distributions: the normal distribution. The normal distribution (also known as the Gaussian distribution or the bell curve) is a distribution that is characterized by two parameters: mu (the mean) and sigma (the standard deviation).

If the distribution of a data set is approximated by a normal distribution and we know mu and sigma, then we can approximate a lot about this distribution. For example, for any interval e.g. [a, b], we can calculate the proportion of observations that are in that interval. We will compute this in a question below.

We will continue using the Gapminder dataset of country data which was used in the previous assessment.

Reminder: load this dataset with:

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

QUESTION 2.1

Create a vector which gives the population sizes of the countries in 1952. Examine the histogram of these population sizes (it might help to increase the number of 'breaks'). Now examine the histogram of the log10 of these population sizes. We will try to find parameters for a Normal distribution which match the logarithm (base 10) of population sizes.

Note: you need to use the log10() function to look at the log10 of population sizes. Do not use the log() function, which gives you logarithm using a base of 'e' (~2.718).

What is the standard deviation of the log10 of population size of the countries in 1952? Use R's built-in function for the standard deviation: sd()

```
pop1952=gapminder$pop[which(gapminder$year==1952)]
sd(log10(pop1952))
```

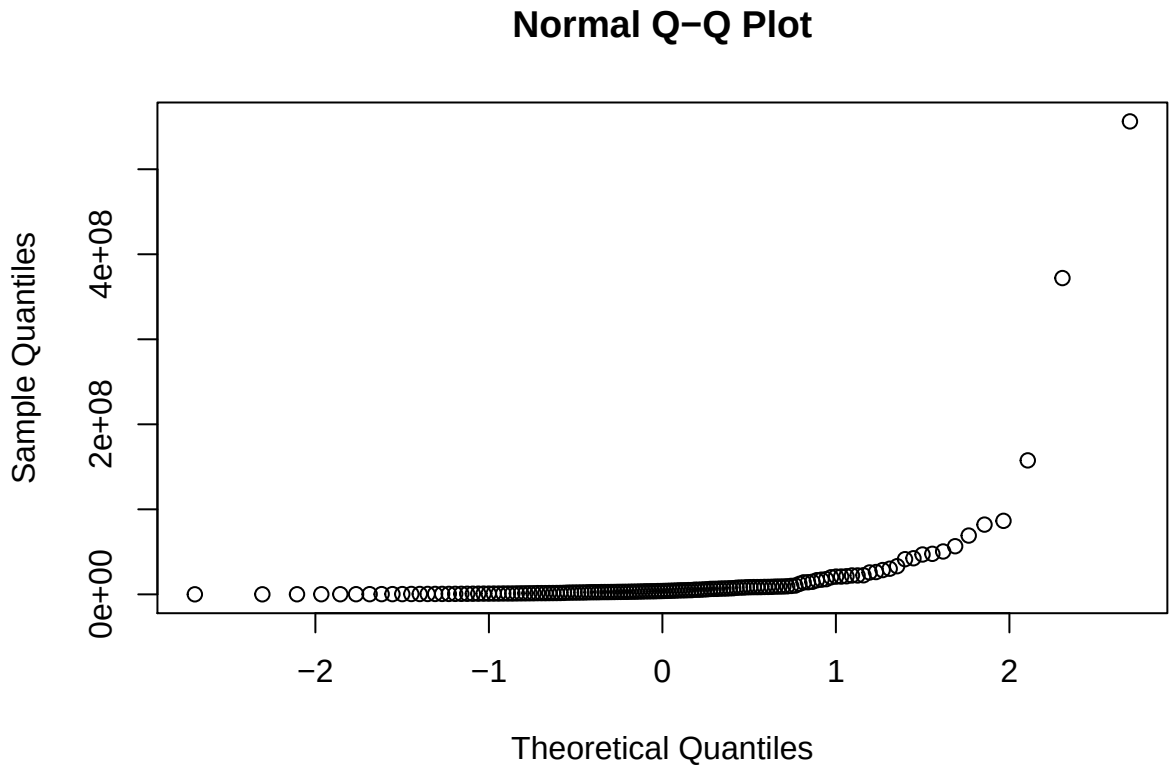
```
## [1] 0.7070292
```

QUESTION 2.2

We want to further explore the log10 of the 1952 population sizes, and compare to a Normal distribution. Create a vector 'x' of the log10 of the 1952 population sizes.

Examine a Q-Q plot of this vector, comparing to the theoretical quantiles of a Normal distribution. You can use qqnorm() directly on the vector to do this. Note that, while the observations are generally along a line, the sample quantiles are not centered on 0.

```
qqnorm(pop1952)
```

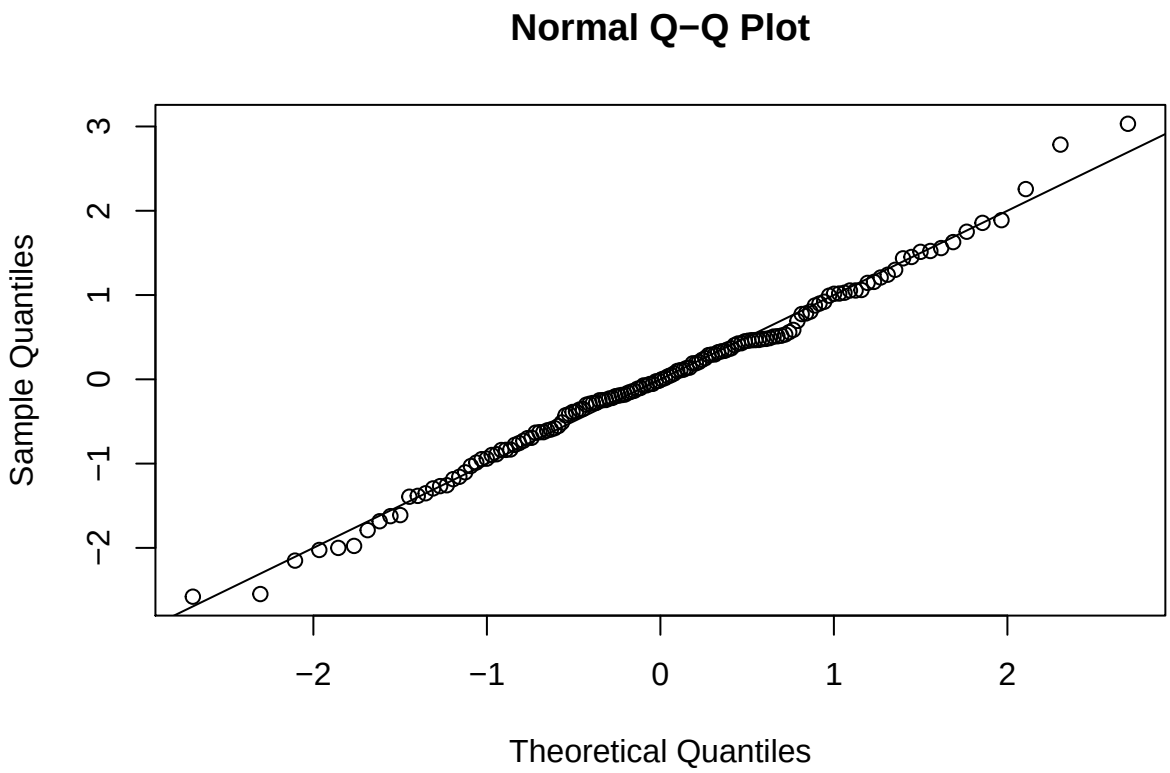


Standardize the log10 population size vector, by subtracting the mean and dividing by the standard deviation. Save this to a new vector 'z'. You have to make sure to wrap the subtraction operation in parentheses before dividing, to ensure that this operation happens first, as in: (a - b)/c.

```
pop1952log10=log10(pop1952)
z=(pop1952log10-mean(pop1952log10))/sd(pop1952log10)
```

Examine a Q-Q plot of 'z' against the Normal distribution using qqnorm(). You can add a diagonal "identity line" with abline(0,1).

```
qqnorm(z)
abline(0,1)
```



What is the z-score of the country with the largest population size? (you can use max(z) to see the very last and largest value).

```
max(z)
```

```
## [1] 3.03194
```

QUESTION 2.3

Now we will use a Normal distribution approximation of the real distribution to ask questions about the expected proportions.

```
x<-pop1952log10
```

We will use the vector 'x' of the log10 of the 1952 population sizes for each country. We are going to create a function which gives back the Normal distribution cumulative density function for a Normal with the same mean and standard deviation as the vector x. This is accomplished by writing a one line function which uses `pnorm()`. `pnorm()` takes a value 'q', and returns the proportion of a Normal distribution which is less than or equal to 'q', for a Normal with a given mean and standard deviation.

```
F = function(q) pnorm(q, mean=mean(x), sd=sd(x))
```

Now, we can use our Normal approximation function to get the proportion of countries that have a log10 1952 population less than or equal to any number. For example, the approximate proportion of countries that have less than or equal to 1 million people in 1952 is given by F(6). This relies on knowing the fact that 1 million = 10^6.

```
F(6)
```

```
## [1] 0.1974126
```

Suppose we want to know the number of countries, not the proportion. This is the proportion times the total number of countries. So, the next step is to store the total number of countries to a value 'n':

```
n = length(x)
```

Finally, using the Normal approximation, estimate the number of countries that should have a log10 1952 population between 6 and 7 (i.e., between 1 million and 10 million people)? The answer should be the proportion from the normal approximation times the total number of countries.

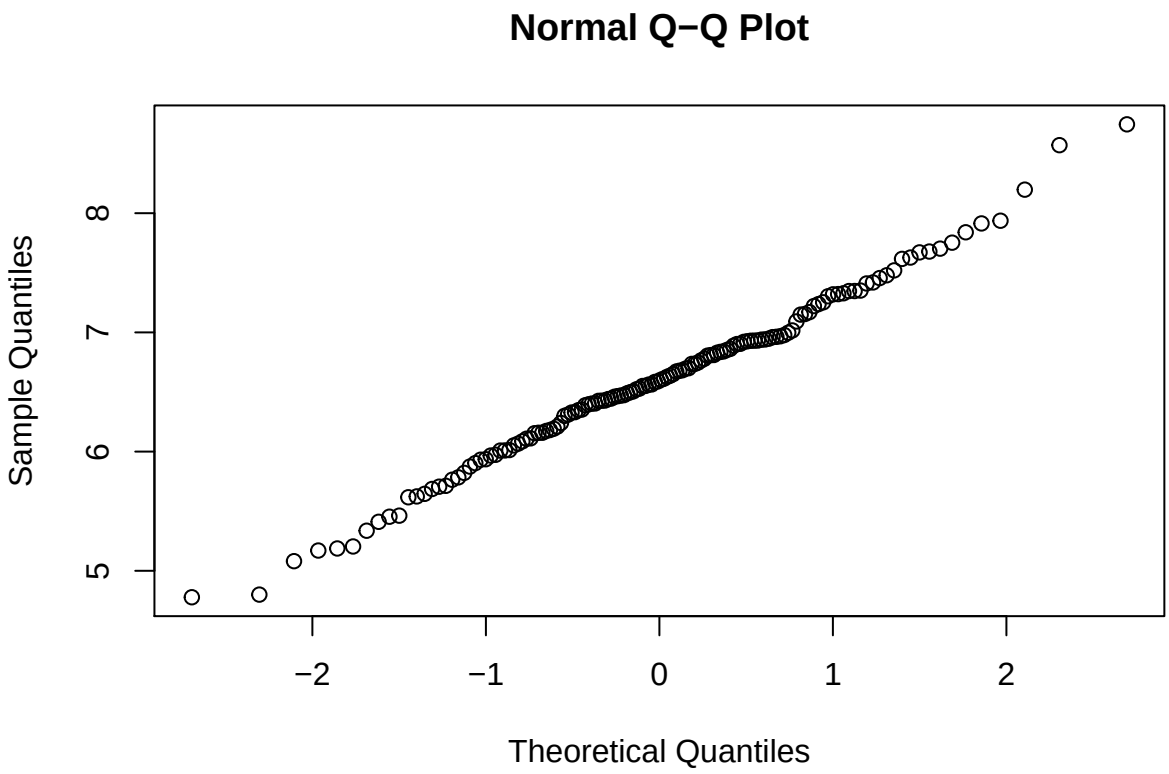
```
(F(7)-F(6))*n
```

```
## [1] 73.27622
```

QUESTION 2.4 (1 point possible)

In the previous video and in problem 2.2 above, we used a QQ plot, as a diagnostic to see if a sample was approximately normally distributed. Here, we will recreate such a plot from scratch. The plot we want to create is:

```
qqnorm(x)
```



. . . which shows the quantiles of a standard normal distribution (mean 0, standard deviation 1) on the x-axis, and the sample quantiles on the y-axis. If the points fall along a line, then the sample is approximately normally distributed. We can build this plot using the quantiles of a standard normal, and the same quantiles of our sample. First we need to find the number of observations in our sample:

```
n = length(x)
```

The sorted values of x represent n quantiles of the sample distribution spread from 0.0 to 1.0, however the quantiles do not include 0 and 1. Therefore, the sorted values of x are considered the 0.5/n, 1.5/n, 2.5/n, . . . , (n - 0.5)/n quantiles of the sample distribution. Or in code, we want to find the quantiles of the standard normal distribution which are associated with the following probabilities:

```
ps = ((1:n) - 0.5)/n
```

The matching quantiles of the standard normal distribution can be found by plugging ps into qnorm().

```
qnorm(ps)
```

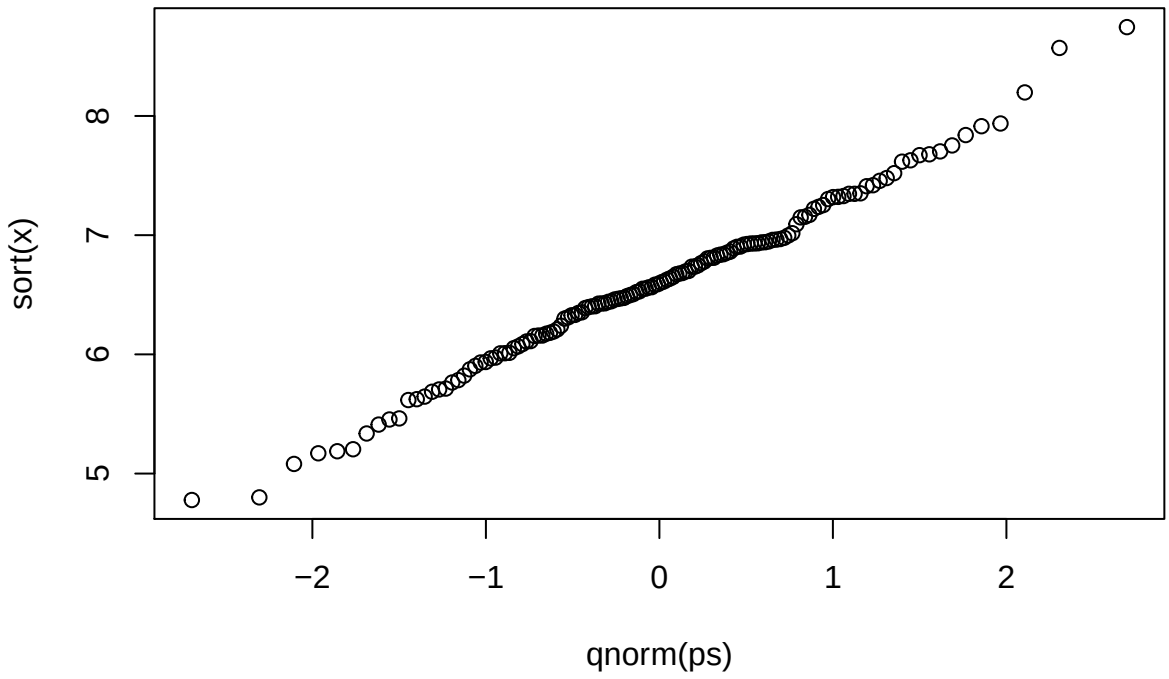
What is the quantile of the standard normal distribution which matches to the smallest number in x (the first element of sort(x))?

```
qnorm(ps[1])
```

```
## [1] -2.69484
```

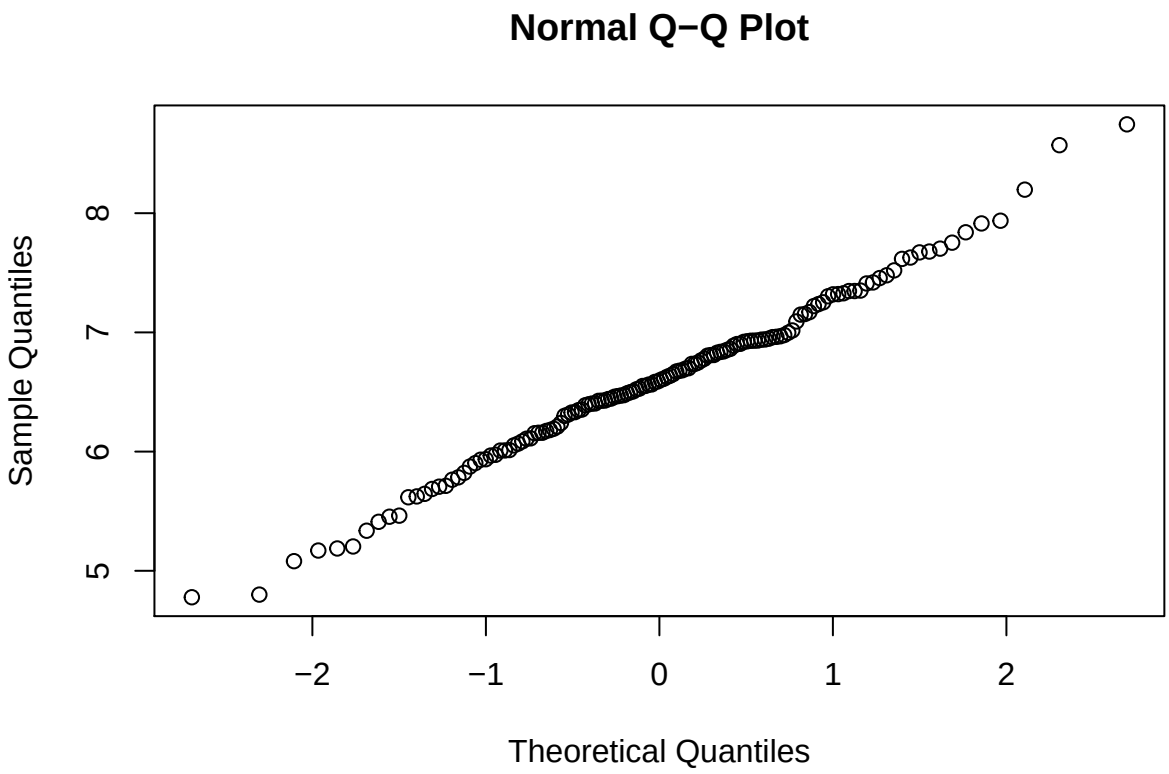
We can then construct our plot, as we have the quantiles of the sample distribution, and can plug 'ps' into qnorm to get the matching sample quantiles of a normal distribution:

```
plot(qnorm(ps), sort(x))
```



Compare the homemade plot to qqnorm(x).

```
qqnorm(x)
```


Normal Q−Q Plot
Sample Quantiles
5
6
7
8
−2
−1
0
1
2
Theoretical Quantiles