Agenda for Week 3, Hr 1 (Tuesday, Jan 19)

Hour 1:

- Installing R and inputting data.

- Different tools for R: Notepad++ and RStudio.

- Basic commands: ?, ??, mean(), sd(), t.test(), lm(), plot()

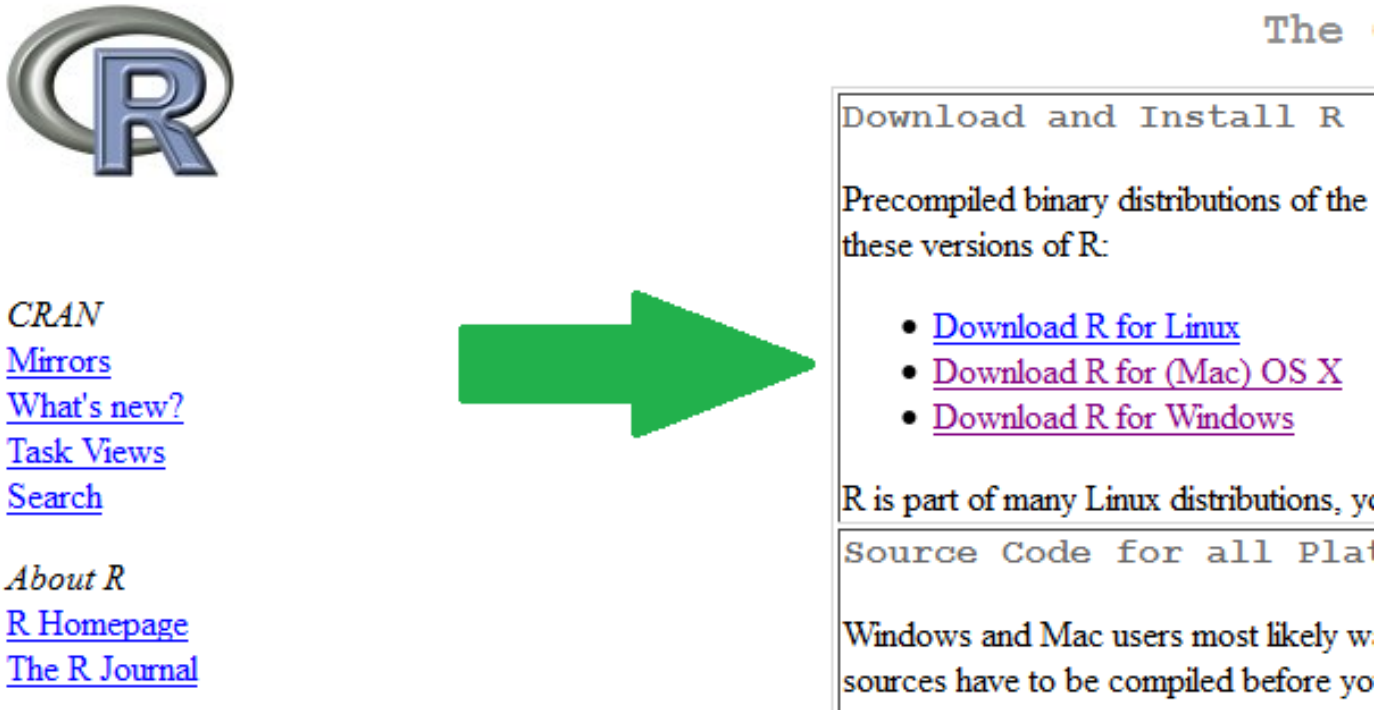
- t.test() and its output.

- cor() , cor.test() and their output

Hour 2: lm (regression), plot (scatterplots), cooks.distance and resid (diagnostics)

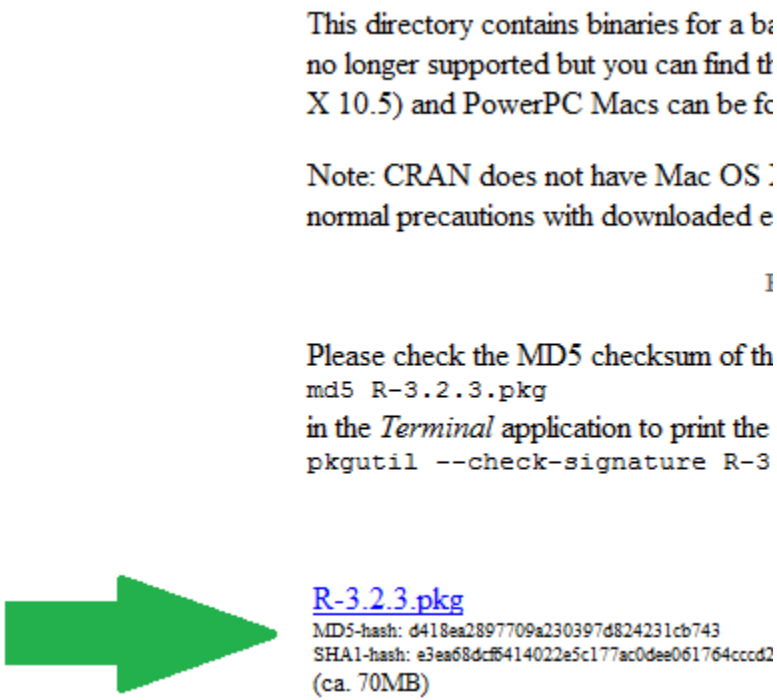
Installing R. http://cran.r-project.org/

1. Choose "Download R for [your operating system]"

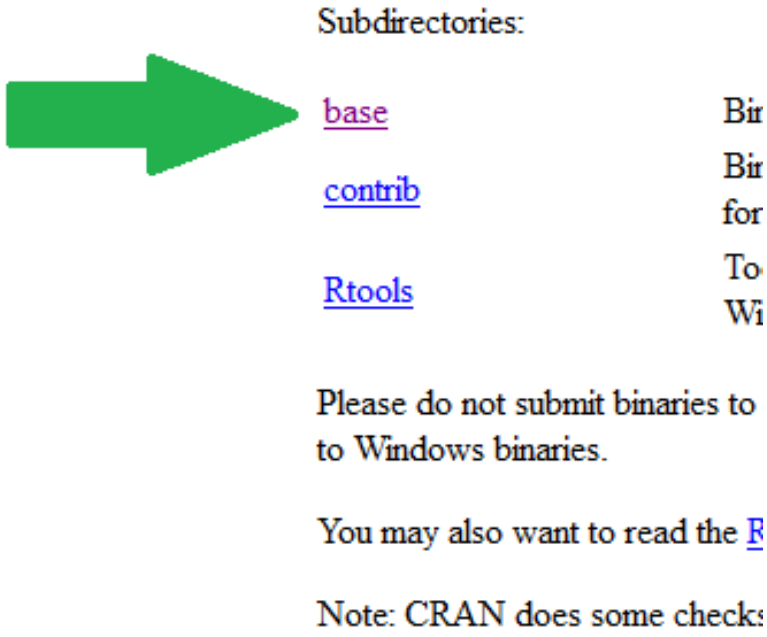


Installing R. http://cran.r-project.org/

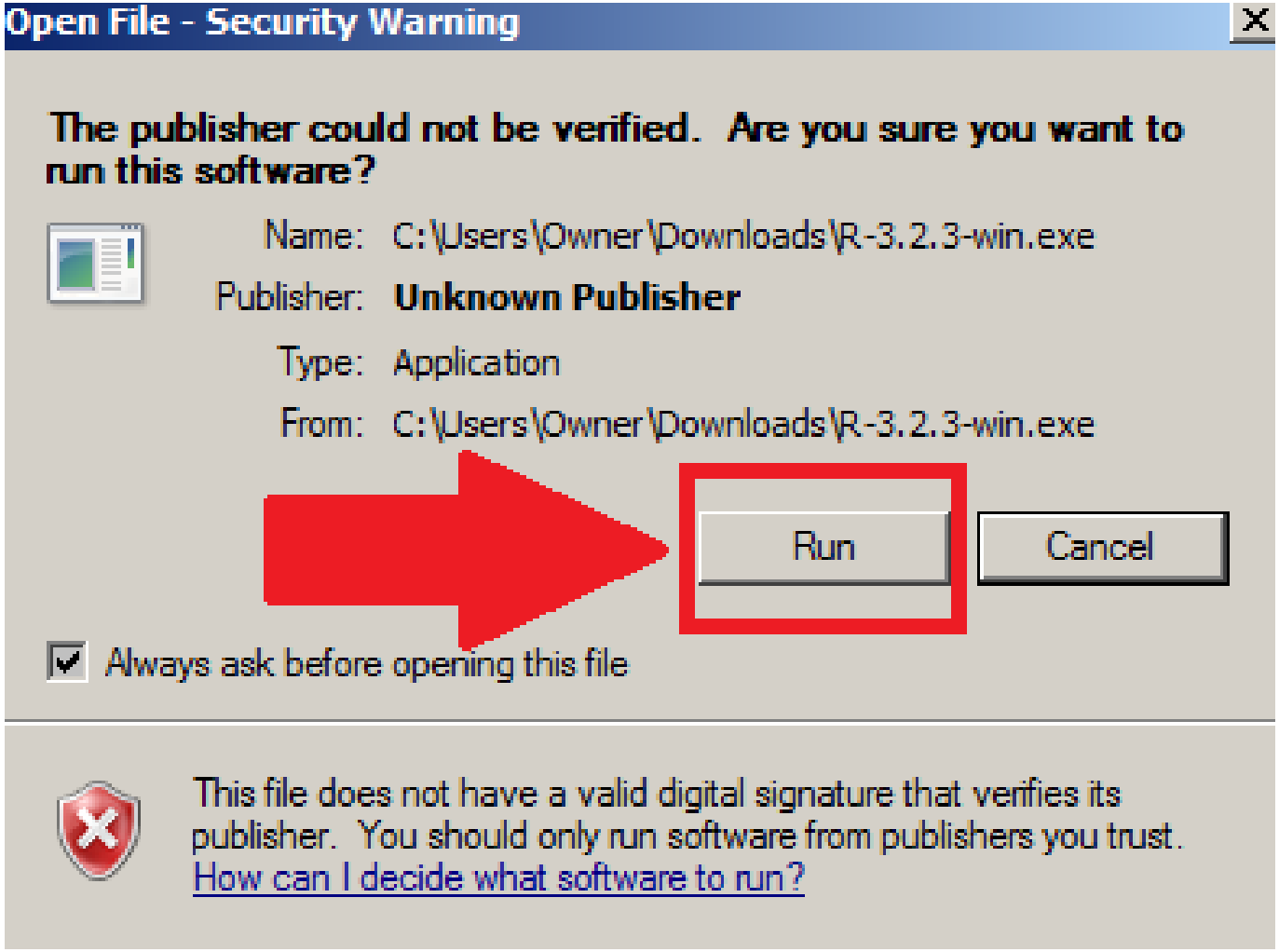
MacOS: Choose "R-3.2.3.pkg"



Windows: Choose "base"

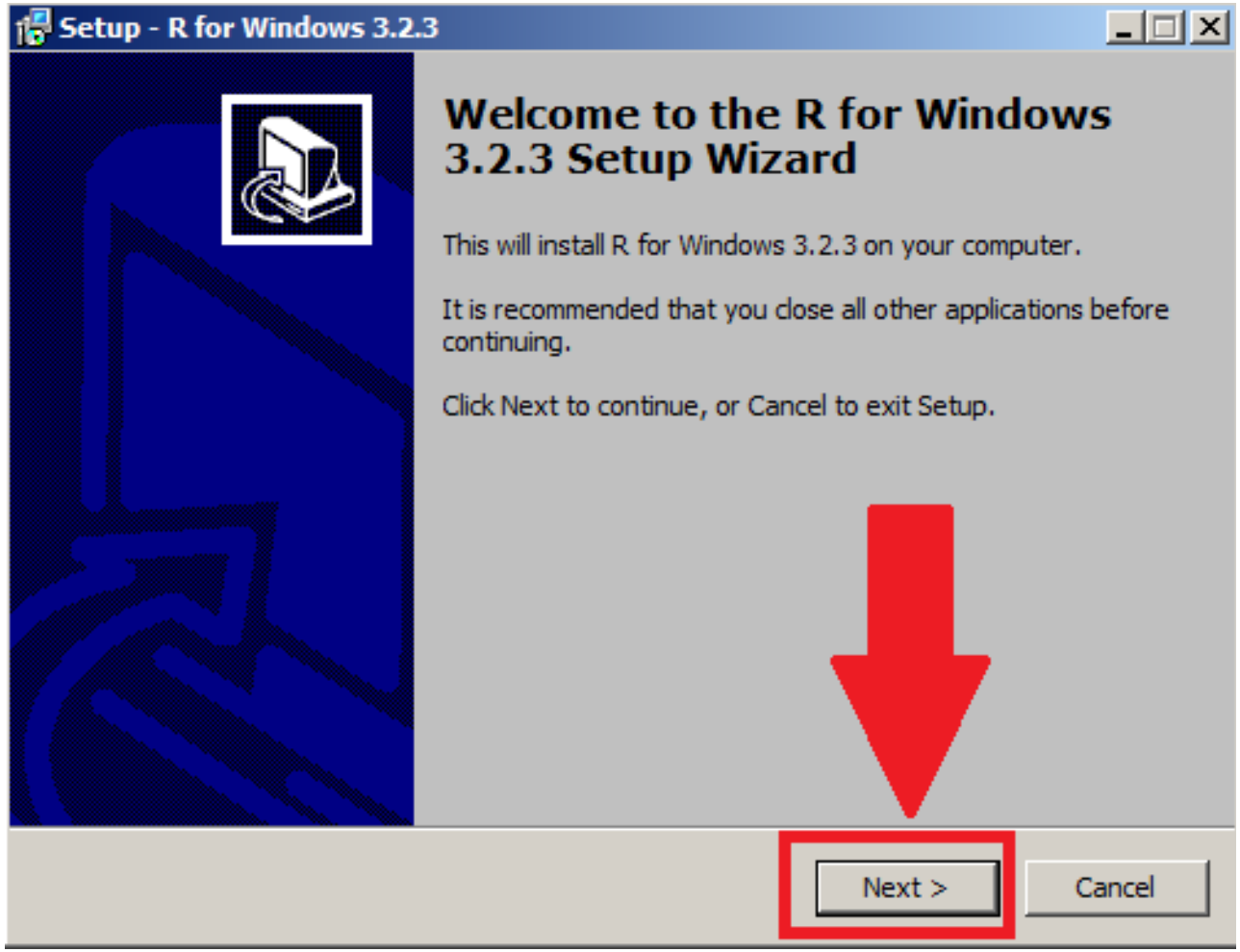


The version number may be higher than 3.2.3 in the future.



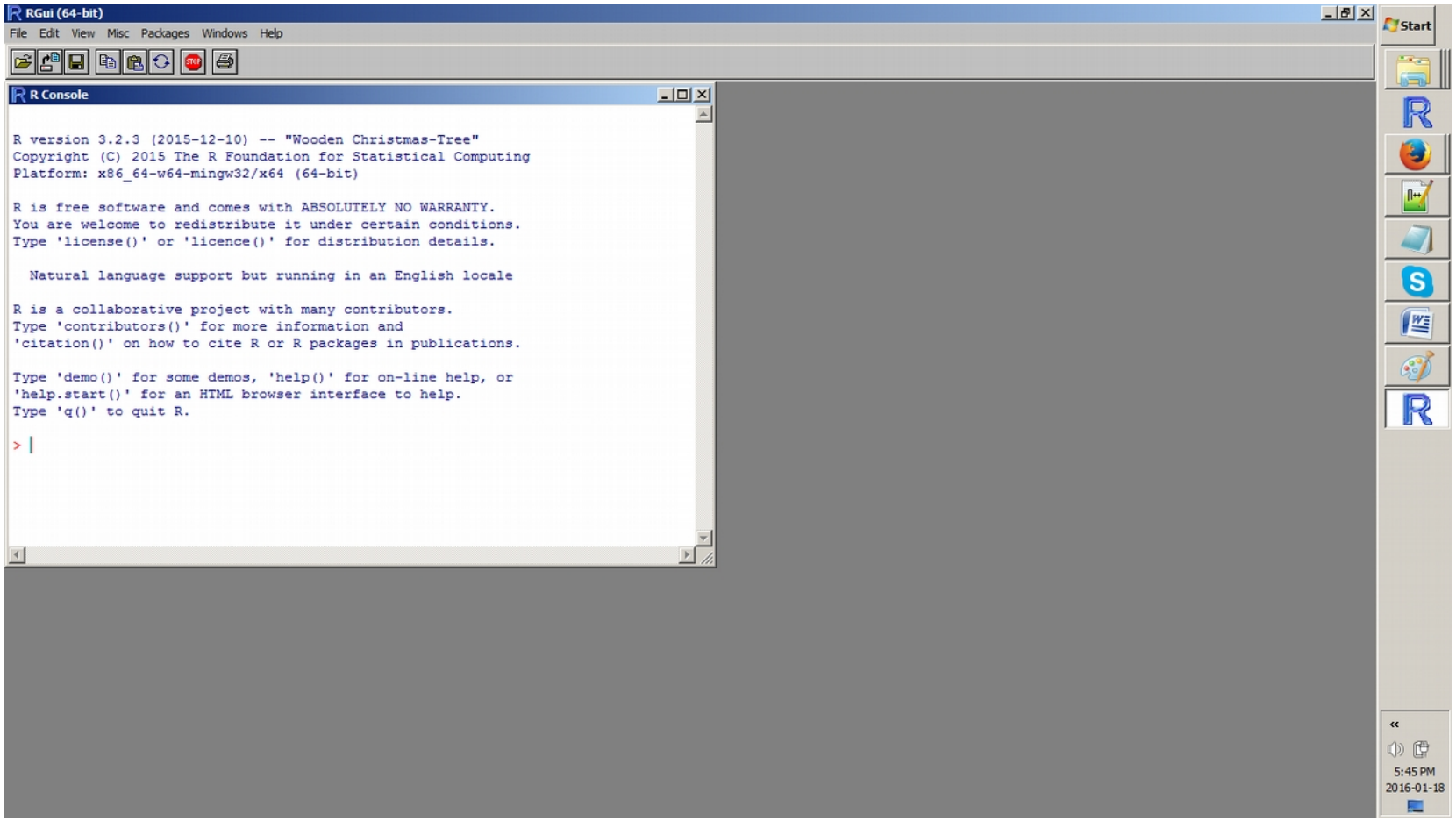
After it has downloaded (about 5 mins), click “run” or the Mac equivalent. Say ‘yes’ if it asks for user access, then choose your preferred language.

The default for everything is fine, so press ‘next’ through everything in the installation wizard.



…and you’re done!

Opening the program “R x64 3.2.3” will give you a window that looks like this:



The white part is the “console”. The Mac version doesn’t have this gray space.

Base R works by copy-pasting things into this console window and pressing enter.

Try these scientific calculator functions to see what R does:

Input	**Expected Output**

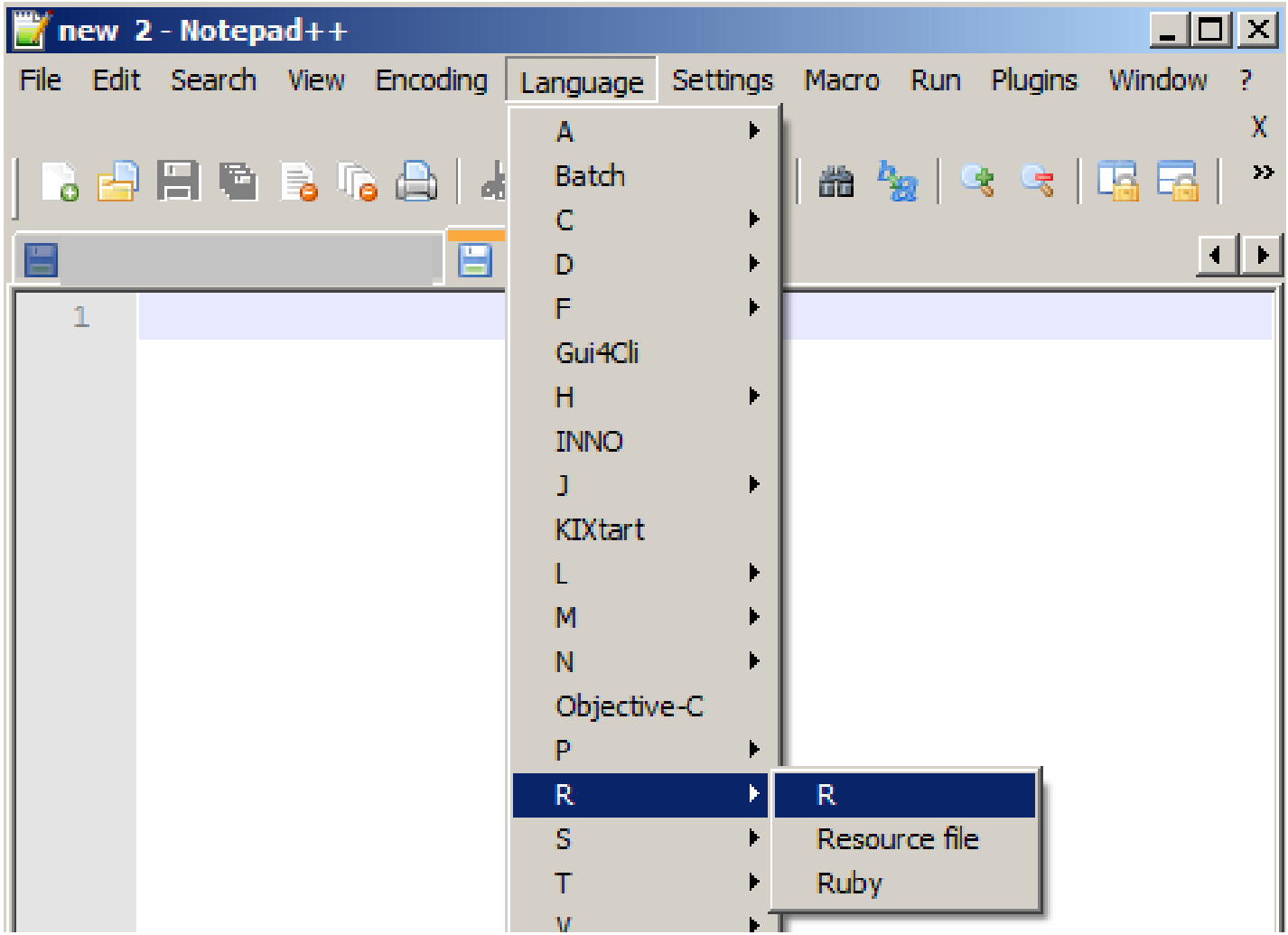
Notepad++ (optional, windows only)

Notepad++ is a programmer's notepad. It works like notepad in windows or textwriter in mac, in that it is a basic word processor.

However, this one has options like automatic colour coding and autofill that make writing and reading R code much easier.

http://notepad-plus-plus.org/

When using, Notepad++, make sure to use Language → R → R to get the proper colour coding.



Thanks for your understanding. I'm a little hoarse.

Learning how to Learn, the ? and ?? commands.

We can use ??*<term>* to search for commands related to a specific term. For example, entering

```
??histograms
```

…will bring up a list of commands related to histograms, like this:

```
Help pages:
```

ggplot2::geom_histogram	Histogram
graphics::hist.POSIXt	Histogram of a Date or Date-Time Object
graphics::hist	Histograms
graphics::plot.histogram	Plot Histograms
grDevices::nclass.Sturges	Compute the Number of Classes for a Histogram
KernSmooth::dpih	Select a Histogram Bin Width
lattice::histogram	Histograms and Kernel Density Plots
lattice::panel.histogram	Default Panel Function for histogram

Help pages:

ggplot2::geom_histogram	Histogram
graphics::hist.POSIXt	Histogram of a Date or Date-Time Object
graphics::hist	Histograms
graphics::plot.histogram	Plot Histograms
grDevices::nclass.Sturges	Compute the Number of Classes for a Histogram
KernSmooth::dpih	Select a Histogram Bin Width
lattice::histogram	Histograms and Kernel Density Plots
lattice::panel.histogram	Default Panel Function for histogram

The first part is the name of the package that has the command, and the second part, after the ::, is code to do that command. We're going to try the command **`hist`** in the graphics package.

You can click on graphics::hist to get information the hist command, including examples.

You can also use the ?<command> to get information about single command. For example,

```
?hist
```

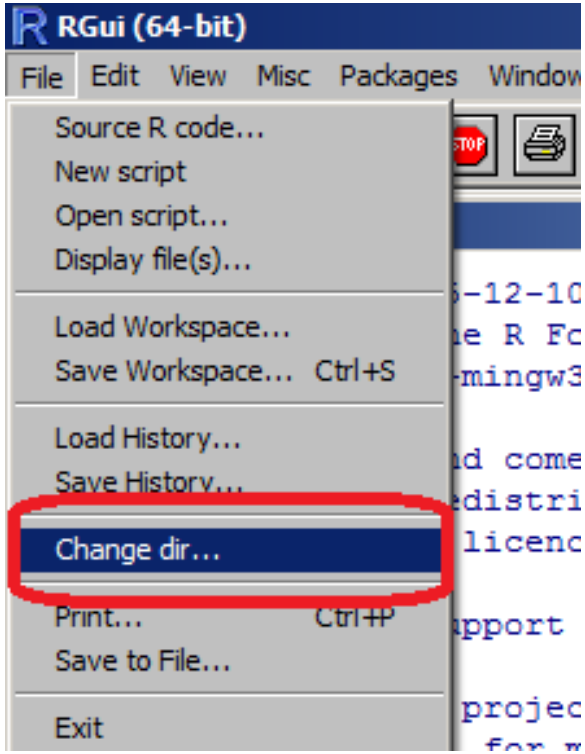
brings up information about the hist() histogram command.

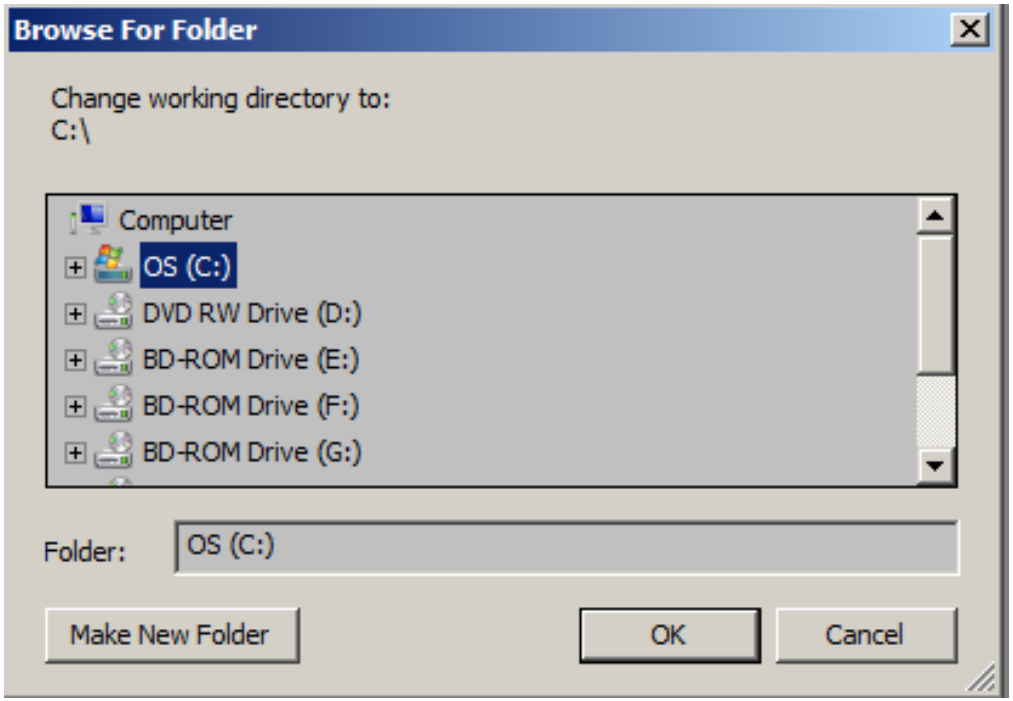
`?lm` **and** `?anova` bring up information about regression (linear models) and about AnOVa respectively.

(Commands are often written with () at the end to show that there are functions, the input or parameters go inside the () usually.)

Inputting data from a file.

First, make sure you have the Beardies.txt file downloaded before continuing. Use the menu at the top of the R window and choose File → Change Dir and navigate to the folder that contains "Beardies.txt" (data about some bearded dragons) and click OK.





Then enter the following into the console.

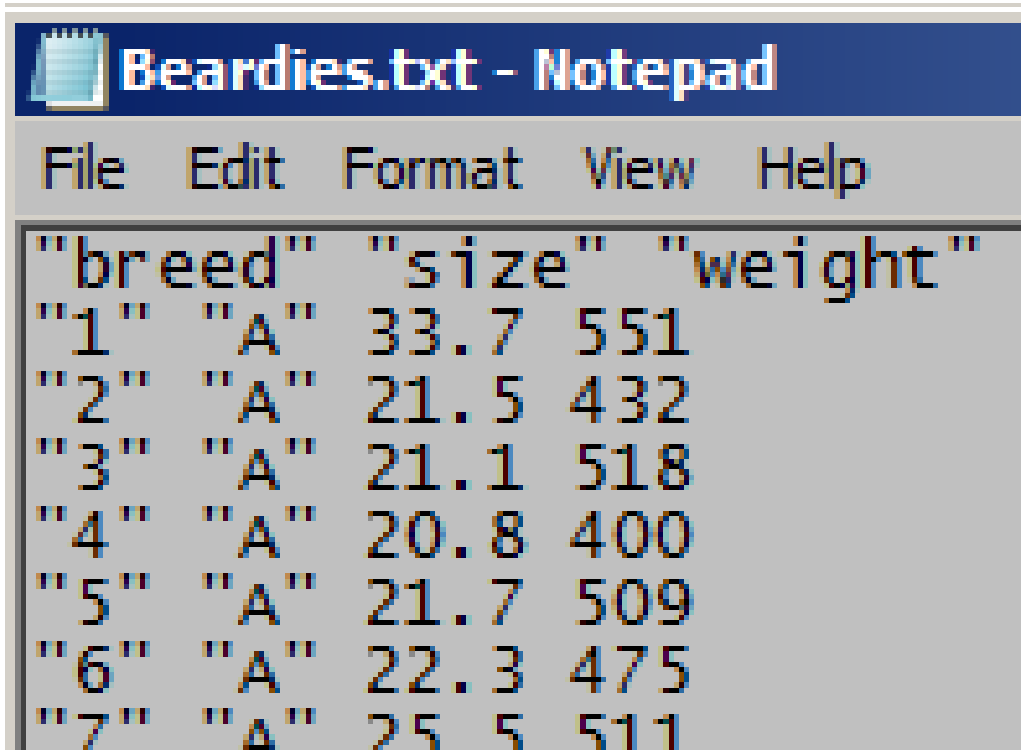
```
Beard = read.table("Beardies.txt", sep=' ')
```

- **read.table** is the command.
- **"Beardies.txt"** is the filename that contains the info you want.
- `sep=' '` tells the console to look for a space between the entries. Sometimes this is a tab or a comma, so it helps to open new datasets in notepad to see how they're set up before putting them in R.
- **Beard** is the name the dataset when it's in R. This can be any name without spaces and starting with a letter.

CAPITALS MATTER!!!

On the left is ‘Beardies.txt’ in notepad.

On the right is the output from the **head()** command in R.




```
> head(beard)
  breed size weight
1     A 33.7    551
2     A 21.5    432
3     A 21.1    518
4     A 20.8    400
5     A 21.7    509
6     A 22.3    475
>
```

You can refer to specific variables in the dataset with $.

Beard$breed will give you the breeds of all the bearded dragons in this set.

Beard$size will list the sizes.

This raw data isn't the most useful, we can also make a frequency table.

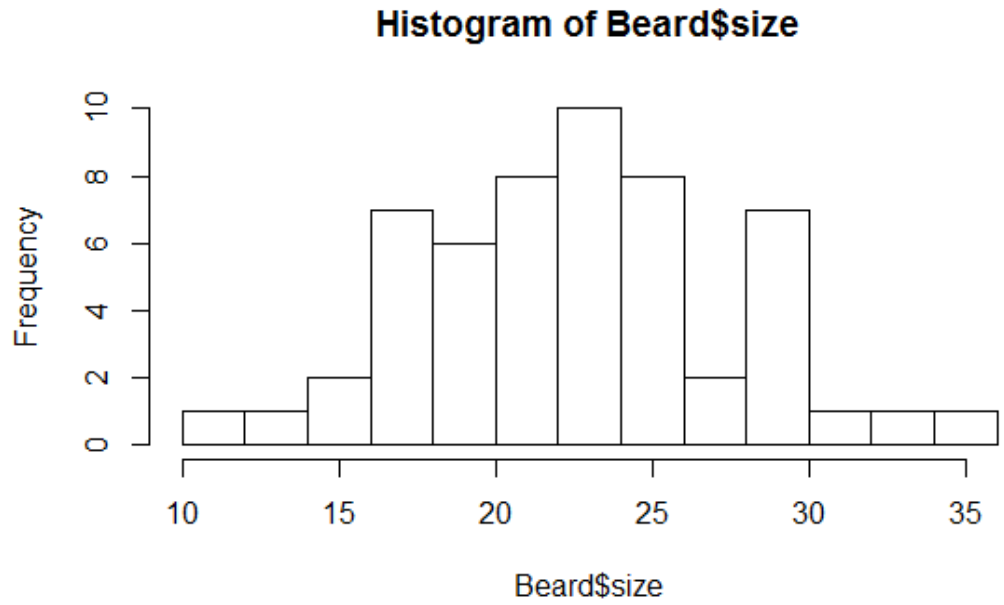
```
table( Beard$breed )
table( Beard$size )
```

Frequency tables aren't useful for continuous data like size, but we can take a histogram instead.

```
hist( Beard$size )
```

We could change the number of bars that are shown by setting the parameter n.

```
hist( Beard$size, n=10)
```

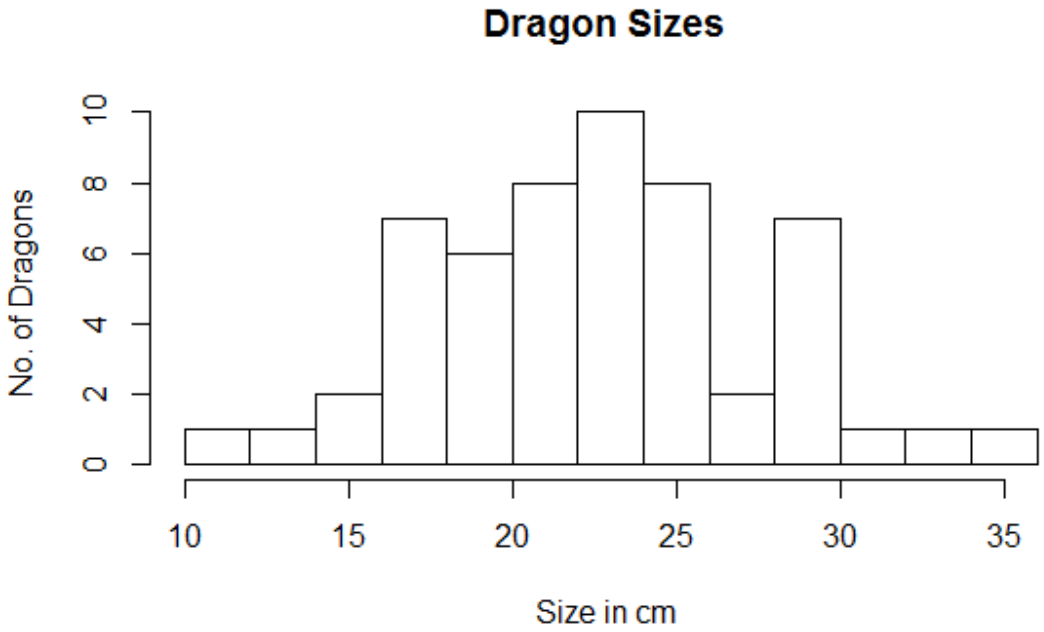


We could also add a title with the parameter **main**…

```
hist(Beard$size, n=10, main="Dragon Sizes")
```

…and labels on the x and y axes with **xlab and ylab**.

```
hist(Beard$size,n=10, main="Dragon Sizes",
xlab="Size in cm", ylab="No. of Dragons")
```



`mean(Beard$size)` gives the mean

`sd(Beard$weight)` gives the standard deviation

`median(Beard$size)` gives the median

`length(Beard$size)` gives the number of elements, also known as the sample size.

`summary(Beard)` gives summary information of each variable. (use summary(Beard$size) for just one variable)

R be free, matey. No software piracy be needed.

T-Tests

The basic command for a t-test in R is **t.test()** .

We can test the hypothesis that the mean size of all the beardies is 20 (A one-sample, two-sided test)

```
t.test(Beard$size, mu=20)
```

```
> t.test(Beard$size, mu=20)

        One Sample t-test

data:  Beard$size
t = 4.1381, df = 54, p-value = 0.0001234
alternative hypothesis: true mean is not equal to 20
95 percent confidence interval:
 21.41625 24.07830
sample estimates:
mean of x
 22.74727
```

The output setup isn't too bad

[Name of calculation or test]

data: [whatever you entered.]

t =**[t-score]**, df = **[degrees of freedom]** (n-2), p-value = **[p-value]**

alternative hypothesis: [whatever you entered]

95 percent confidence interval:

[lower part of CI] [upper part of CI]

sample estimates:

mean of x (possibly) mean of y

[mean of x] [(possibly) mean of y]

We can change the alternative to either side

```
t.test(Beard$size, mu=20, alternative="less")
```

```
> t.test(Beard$size, mu=20, alternative="less")

        One Sample t-test

data:  Beard$size
t = 4.1381, df = 54, p-value = 0.9999
alternative hypothesis: true mean is less than 20
95 percent confidence interval:
     -Inf 23.85834
sample estimates:
mean of x
 22.74727
```

Note: Same t-score, df, and mean.

Different confidence interval and p-value. Why?

About that alternative…

```
t.test(Beard$size, mu=20, alternative="less")
```

…this means we are testing the null hypothesis

$H_0 : \mu \geq 20$ against $H_A : \mu < 20$.

The sample mean is 22.747, so there is no evidence that the true mean is ***less*** than 20. Hence, the p-value is very large.

p-value: 0.9999.

The confidence interval is also one-sided, which is rarely useful.

We can also do two-sample tests, but we need to break the sample up first. The subset() command allows up to break a dataset up. ***Notice the double = sign after breed.*** In programming, == means 'is this true?'.

```
onlyA = subset(Beard, breed == "A")
```

The dataset onlyA now contains the 20 observations of Breed A dragons.

```
onlyB = subset(Beard, breed == "B")
```

Likewise, onlyB contains 20 observations from Breed B.

We can test if dragons in these groups have the same mean size (assuming normality within each group)

```
t.test(onlyA$size, onlyB$size)
```

```
> t.test(onlyA$size, onlyB$size)

        Welch Two Sample t-test

data:  onlyA$size and onlyB$size
t = -2.2574, df = 37.616, p-value = 0.02987
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.0421839 -0.3278161
sample estimates:
mean of x mean of y
   21.930    25.115
```

Notice that df is NOT 40 -2, it's less. (Slightly unequal variance)

We could include the assumption of equal variance

`t.test(onlyA$size, onlyB$size, var.equal=TRUE)`

```
> t.test(onlyA$size, onlyB$size, var.equal=TRUE)

        Two Sample t-test

data:  onlyA$size and onlyB$size
t = -2.2574, df = 38, p-value = 0.02981
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -6.0412267 -0.3287733
sample estimates:
mean of x mean of y
   21.930    25.115
```

Very minor changes (df, p-value, confidence interval).

So “equal variance” is a reasonable assumption.

… but we COULD test the hypothesis of equal variance.

```
var.test(onlyA$size, onlyB$size)
```

```
> var.test(onlyA$size, onlyB$size)

        F test to compare two variances

data:  onlyA$size and onlyB$size
F = 0.81653, num df = 19, denom df = 19, p-value = 0.6631
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.3231907 2.0629114
sample estimates:
ratio of variances
         0.8165254
```

p-value is 0.6631, so there is ***no evidence against*** equal variance. Also note that F is near 1.

Spelling counts. Don’t confuse ‘snack’ and ‘snake’.

Correlation in R

The function to find the Pearson correlation coefficient between two samples is cor().

```
cor(Beard$size, Beard$weight)
```

```
> cor(Beard$size, Beard$weight)
[1] 0.7872019
>
>  cor(Beard$weight, Beard$size)
[1] 0.7872019
>
> cor(Beard$size, Beard$size)
[1] 1
```

Things to notice:

1. The order doesn't matter in correlation. The correlation of (X,Y) is the same as the correlation of (Y,X). Correlation is just a measure of how well a line describes their relationship, so it shouldn't matter which variable goes first.

2. The correlation with any variable and itself is 1. Always.

```
> cor(Beard$breed, Beard$size)
Error in cor(Beard$breed, Beard$size) : 'x' must be numeric
>
```

3. Correlation only works when ***BOTH*** variables are numeric.

Pearson correlation is just the default. We can change that with the "method" setting.

```
cor(Beard$size, Beard$weight,
method="spearman")
```

```
> cor(Beard$size, Beard$weight, method="spearman")
[1] 0.7753143
> cor(Beard$size, Beard$weight, method="pearson")
[1] 0.7872019
>
```

Only a small difference between the two correlation coefficients.

Non-linearity is unlikely to be a problem.

For more details about a correlation, we can use cor.test()

```
cor.test(Beard$size, Beard$weight)
```

```
> cor.test(Beard$size, Beard$weight)

        Pearson's product-moment correlation

data:  Beard$size and Beard$weight
t = 9.2929, df = 53, p-value = 1.025e-12
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.6596723 0.8706669
sample estimates:
      cor
0.7872019
```

The output setup is a lot like the t-test

[Name of calculation or test]

data: [whatever you entered.]

t =**[t-score]**, df = **[degrees of freedom]** (n-2), p-value = **[p-value]**

alternative hypothesis: [whatever you entered]

95 percent confidence interval:

[lower part of CI] [upper part of CI]

sample estimates:

cor

[correlation coefficient]

Notice about the confidence intervals:

```
95 percent confidence interval:
 0.6596723 0.8706669
sample estimates:
      cor
0.7872019
```

The confidence intervals of correlation coefficients are NOT symmetric around the estimate.

The method to find the confidence interval involves an extra step called a transformation. This is needed because the t-distribution spans from (-infinity, to +infinity) and correlations span ***[-1,+1]***.

About that p-value…

```
p-value = 1.025e-12
```

"1.025e-12" is shorthand for scientific notation.

1.025e-12 = $\boldsymbol{1.025 \times 10^{-12}}$, or 0.000000000001025

Looking at the rest of the information (t = 9.29, a confidence interval of in (0.659, 0.870), it shouldn't be surprising that we have very strong evidence that the correlation isn't zero.

Finally, we can check specifically for positive or for negative correlations:

```
cor.test(Beard$size, Beard$weight,
alternative="greater")
```

We can test using the Spearman correlation:

```
cor.test(Beard$size, Beard$weight,
method="spearman")
```

…and we can change the level of the confidence interval.

```
cor.test(Beard$size, Beard$weight,
conf.level=0.99)
```

Don’t overload yourself. Take a break.

Next hour: lm (regression), plot (scatterplots), cooks.distance and resid (diagnostics)