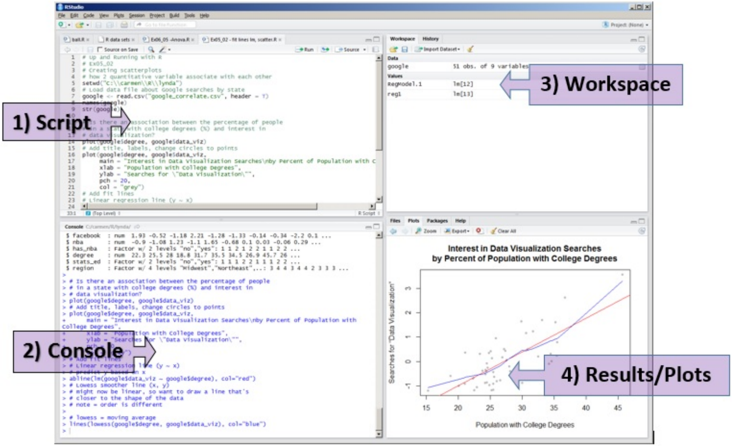
Introduction to R

November 21, 2018

R and RStudio



What R can do

What R can do

Everything.[1,2]

1 Except think about your science

2 Occasionally in a non efficient way

What R can do

Everything.[1,2]

1 Except think about your science

2 Occasionally in a non efficient way

What about RStudio?

- Makes your life easier
- Many handy tricks
  - Autocomplete suggestion
  - Ctrl-Enter to send command to R
  - str() and View() objects in Environment
  - Files, packages, help selectors
  - Version control...

Calculating a mean: Arithmetic and assignment

```r
(2 + 3 + 5 + 1) / 4
```

```
[1] 2.75
```

Calculating a mean: Arithmetic and assignment

```
  (2 + 3 + 5 + 1) / 4

[1] 2.75
```

```
  a <- 2
  b <- 3
  c <- 5
  d <- 1

  (a + b + c + d) / 4

[1] 2.75
```

Calculating a mean: Arithmetic and assignment

```
(2 + 3 + 5 + 1) / 4

[1] 2.75
```

```
a <- 2
b <- 3
c <- 5
d <- 1

(a + b + c + d) / 4

[1] 2.75
```

```
a <- 45
(a + b + c + d) / 4

[1] 13.5
```

Calculating a mean: using vectors

```r
c(2,3,5,1) # c is for concatenate
```

```
[1] 2 3 5 1
```

Calculating a mean: using vectors

```
c(2,3,5,1) # c is for concatenate

[1] 2 3 5 1
```

```
mydata <- c(2,3,5,1) # save the vector
```

Calculating a mean: using vectors

```
c(2,3,5,1) # c is for concatenate
```

```
[1] 2 3 5 1
```

```
mydata <- c(2,3,5,1) # save the vector
```

```
mydata <- (2,3,5,1) # c is missing => error!
```

```
Error:  <text>:1:15:  unexpected ','
1:  mydata <- (2,
                  ^
```

Why bother with vectors?

Substitution:

```
  mydata[2] <- 4
  mydata

[1] 2 4 5 1
```

Vectorized operations:

```
  mydata*5 + 2

[1] 12 22 27  7
```

Exercise

```
-5:20

 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[24] 18 19 20
```

Calculating a mean: using functions

How to use a function?

```
?mean
```

Or use **tab**

Calculating a mean: using functions

How to use a function?

```
?mean
```

Or use **tab**

```
mean(c(2,4,5,1))

[1] 3

mean(mydata)

[1] 3

mean(x = mydata)

[1] 3
```

Loading data

```r
trees <- read.csv("trees.csv")
```

Loading data

```
trees <- read.csv("trees.csv")
```

```
str(trees)

'data.frame': 31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: int  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ..
```

Try also `summary`, `class`, `head`, `tail`

Access

Bracket-syntax

- Row: dataframe[row,]
- Column: dataframe[, column]
- Element: dataframe[row, column]

Access

Bracket-syntax

- Row: dataframe[row,]
- Column: dataframe[, column]
- Element: dataframe[row, column]

```
trees[,1]
trees[1:8,]
trees[c(2,1,2), 3]
trees[, "Height"]
```

Dollar-syntax

- Column `dataframe$column_name`
- Element `dataframe$column_name[row]`

```
trees$Height
```

Time to think a tiny bit!

Calculate the mean for all three variables in trees, excluding the last (31st) record.

Solution for one column

Calculate the mean for all three variables in trees, excluding the last (31st) record.

```
mean(trees$Girth[1:30])
mean(trees[1:30, "Girth"])
mean(trees$Girth[-31])
mean(trees[-31, "Girth"])
```

How to get the row means?

```
mean(trees[1,])
mean(trees[2,])
mean(trees[...,])
```

How to get the row means?

```
mean(trees[1,])
mean(trees[2,])
mean(trees[...,])
```

:


How to get the row means? For-loops

```
for (i in 1:N)
{
  something as a function of i
}
```

How to get the row means? For-loops

```
for (i in 1:N)
{
  something as a function of i
}
```

```
ResultMean <- vector() # we will store the results there
for (i in 1:31)
{
  ResultMean[i] <- mean(as.numeric(trees[i,]))
}
```

For-loops: your turn!

Load rock data

```
rock <- read.csv("rock.csv")
```

Use a for loop to obtain column averages

Solution

Load rock data.

```
rock <- read.csv("rock.csv")
```

Use a for loop to obtain column averages

```
storage <- vector(length = ncol(rock))
for (i in 1:ncol(rock))
{
  storage[i] <- mean(rock[,i])
}
```

More concise alternative: apply functions

```
apply(X = dataframe, MARGIN = 1 (row) or 2 (col), FUN = function)
```

More concise alternative: apply functions

```
apply(X = dataframe, MARGIN = 1 (row) or 2 (col), FUN = function)
```

```r
apply(X = rock, MARGIN = 1, FUN = mean)#by row (not meaningful)
apply(X = rock, MARGIN = 2, FUN = mean)#by column
```

Even better (worse)...

```
colMeans(rock)
rowMeans(rock)
```

Even better (worse)...

```
colMeans(rock)
rowMeans(rock)
```

Trade-off concision / flexibility

- colMeans shortest, but does only means
- apply very flexible, but does only array/matrix/data-frame
- for-loop looks complex, but infinitely flexible
- (NB: your computer does a for-loop whether you see it or not)

While-loop: idea

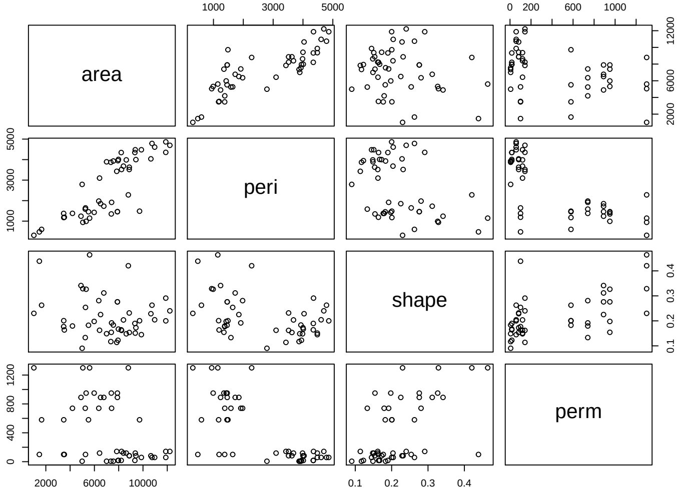
Less common than for-loops
Stop the loop after a condition is met

What is the smallest reproductive rate necessary to obtain a growing population?

```
library(popbio)
mat <- matrix(c(0,0.8,1,0), nrow = 2)
lambda(mat)

[1] 0.8944272

cond <- 1
while(lambda(mat) < 1 )
{
  mat[1,2] <- mat[1,2]+0.001
}
mat[1,2]

[1] 1.251
```

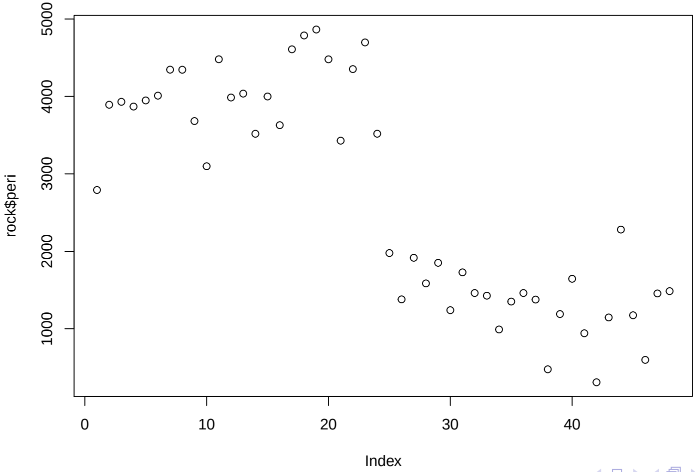
plot function

```
plot(rock)
```



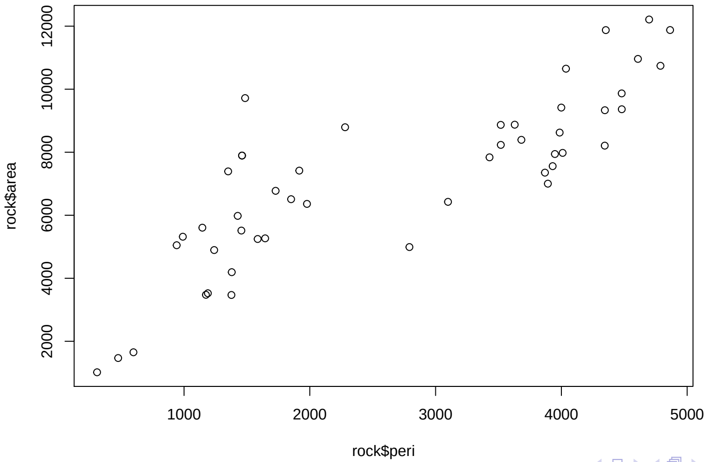
plot function

```r
plot(rock$peri)
```



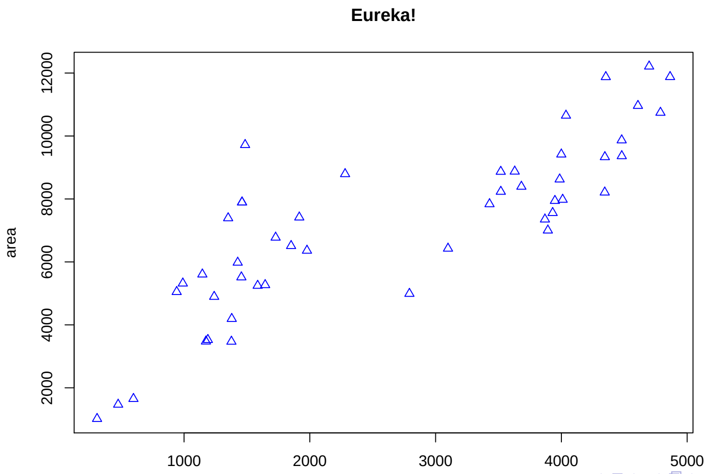
plot function

```
plot(x = rock$peri, y = rock$area)
```



plot function

```
plot(x = rock$peri, y = rock$area, main = "Eureka!",
     xlab = "Perimeter", ylab = "area", col="blue", pch=2)
```

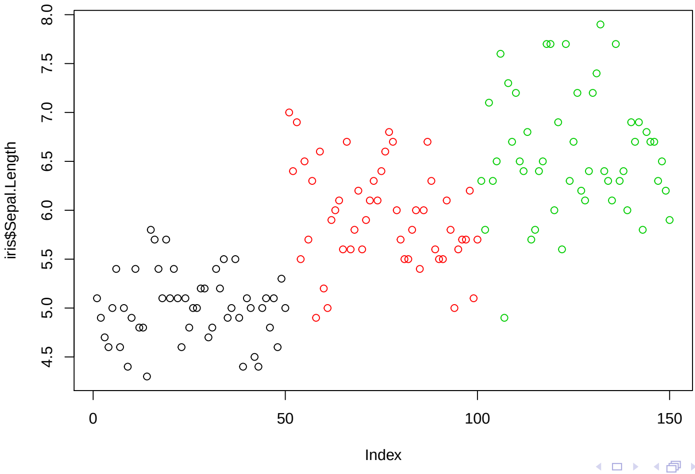


plot function: back to the mean

```
data("iris")
```

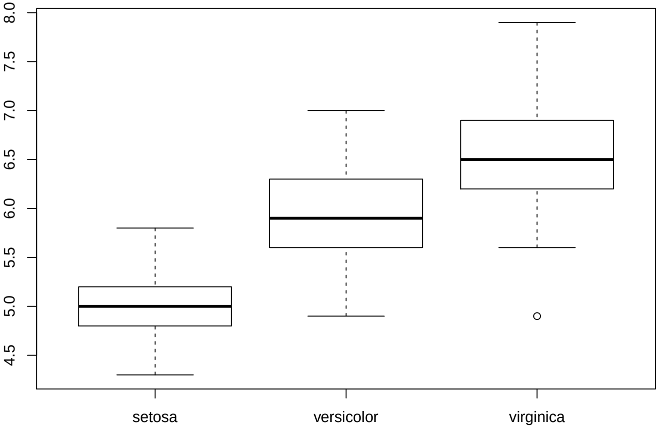
plot function: back to the mean

```
plot(iris$Sepal.Length, col=iris$Species)
```



boxplots

```
boxplot(iris$Sepal.Length ~ iris$Species)
```



Student's T.test introduction

```
?t.test
```

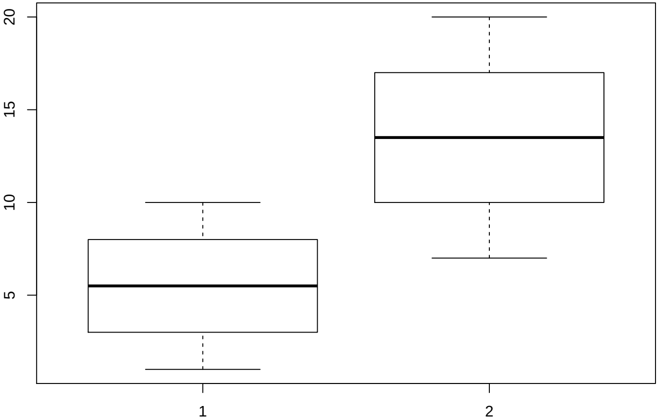
Student's T.test introduction

```
?t.test
```

```
t.test(1:10, y = c(7:20))


Welch Two Sample t-test

data:  1:10 and c(7:20)
t = -5.4349, df = 21.982, p-value = 1.855e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.052802  -4.947198
sample estimates:
mean of x mean of y
      5.5      13.5
```

T.test introduction

```
boxplot(c(1:10, 7:20) ~ c(rep(1,10), rep(2, 14)))
```



Are irises different?

Use t-tests to compare species in the iris dataset

Are irises different? Solution

Use t-tests to compare species in the iris dataset
Sorry, I was mean and forgot to tell about subsetting, which you needed here.
Subset to the species *setosa*:

```
iris[iris$Species == "setosa", ]
```

One t-test for sepal length between *setosa* and *versicolor*:

```
t.test(x = iris$Sepal.Length[iris$Species == "setosa"],
      y = iris$Sepal.Length[iris$Species == "versicolor"])
```