Importing, working with, and exploring data
Week 2, Lecture 04:

Richard E.W. Berl

Spring 2019

Loading data

```
data("HairEyeColor")
hairEyeColor = as.data.frame(HairEyeColor)

gombe = read.csv(file="./data/gombe_128.csv", header=TRUE)

horseKicks = read.table(file="./data/HorseKicks.txt", header=TRUE, sep="\t")

library(readxl)
folktales = read_xlsx(path="./data/rsos150645supp1.xlsx",
                      sheet=1, range="A2:JP52")
folktales = as.data.frame(folktales)
colnames(folktales)[1] = "society"
```

Manipulating data

Subsetting

Subsetting is one of the most common tasks when you're dealing with data. There are multiple ways to subset data, specifically when it is in a data frame.

Using references

The first you've already done: just select the rows or columns you want.

```
horseKicks[,c(1:2)]
```

```
##    Year GC
## 1  1875  0
## 2  1876  2
## 3  1877  2
## 4  1878  1
## 5  1879  0
## 6  1880  0
## 7  1881  1
## 8  1882  1
## 9  1883  0
## 10 1884  3
## 11 1885  0
```

```
## 12 1886  2
## 13 1887  1
## 14 1888  0
## 15 1889  0
## 16 1890  1
## 17 1891  0
## 18 1892  1
## 19 1893  0
## 20 1894  1
```

You can do so by name:

```
horseKicks[c(1,5:10),c("C1","C2","C3")]
```

```
##    C1 C2 C3
## 1   0  0  0
## 5   0  0  1
## 6   3  2  1
## 7   0  0  2
## 8   2  0  0
## 9   0  1  2
## 10  0  1  0
```

You can also exclude rows or columns you **don't** want by putting a `-` before the vector. Here I use a sequence to exclude odd years:

```
horseKicks[-seq(1, nrow(horseKicks), 2),]
```

```
##    Year GC C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C14 C15
## 2  1876  2  0  0  0  1  0  0  0  0  0   0   0   1   1
## 4  1878  1  2  2  1  1  0  0  0  0  0   1   0   1   0
## 6  1880  0  3  2  1  1  1  0  0  0  2   1   4   3   0
## 8  1882  1  2  0  0  0  0  1  0  1  1   2   1   4   1
## 10 1884  3  0  1  0  0  0  0  1  0  0   2   0   1   1
## 12 1886  2  1  0  0  1  1  1  0  0  1   0   1   3   0
## 14 1888  0  1  1  0  0  1  1  0  0  0   0   1   1   0
## 16 1890  1  2  0  2  0  1  1  2  0  2   1   1   2   2
## 18 1892  1  3  2  0  1  1  3  0  1  1   0   1   1   0
## 20 1894  1  0  0  0  0  0  0  0  1  0   1   1   0   0
```

Using conditionals

Remember `which()`? You already used it for subsetting. If we just want the rows that match a certain condition, we put the statement in the row reference spot:

```
gombe[which(gombe$innov > 6),]
```

```
##     chimpcode sex kasekela      dom      sol     impl     symp     stbl
## 11       R475   1        1 5.666667 2.666667 5.333333 3.333333 5.000000
## 30       D152   0        1 6.000000 3.333333 3.333333 6.666667 6.000000
## 31       U376   1        1 7.000000 1.333333 6.333333 4.333333 3.000000
## 42       A383   1        1 3.666667 1.666667 6.333333 4.000000 4.000000
## 47       M171   1        1 5.666667 1.666667 5.666667 2.333333 4.666667
## 48       C141   1        1 3.500000 4.000000 3.000000 5.000000 3.000000
## 49       C133   0        1 2.666667 1.333333 5.000000 4.000000 4.333333
## 57       G103   1        1 6.333333 2.000000 5.000000 3.000000 5.666667
## 60       U464   1        1 1.000000 4.000000 4.000000 3.500000 4.000000
## 63       Q315   1        1 4.333333 4.666667 5.000000 6.000000 3.666667
```

```
## 100      I142   1        1 5.000000 4.666667 4.000000 4.333333 4.000000
## 107      Y440   1        1 4.000000 3.000000 3.333333 5.666667 3.333333
##         invt     depd      soc    thotl     help     exct     inqs
## 11  5.666667 4.000000 6.333333 4.000000 6.333333 4.666667 5.333333
## 30  5.000000 5.666667 6.666667 3.000000 7.000000 4.666667 5.333333
## 31  4.666667 4.333333 6.666667 4.000000 6.333333 6.666667 5.333333
## 42  6.000000 5.000000 6.333333 2.333333 5.666667 5.666667 6.333333
## 47  3.000000 6.666667 5.333333 2.333333 6.666667 5.333333 3.666667
## 48  2.500000 6.000000 6.000000 2.500000 6.000000 3.000000 4.000000
## 49  4.666667 6.333333 6.333333 1.333333 5.666667 3.000000 5.333333
## 57  5.333333 3.666667 6.666667 2.666667 4.666667 5.000000 5.000000
## 60  3.000000 4.500000 4.500000 1.000000 4.000000 2.500000 4.000000
## 63  4.333333 5.666667 6.000000 4.000000 6.333333 5.000000 5.333333
## 100 4.333333 5.666667 6.000000 4.000000 6.000000 3.666667 3.666667
## 107 4.666667 5.333333 5.666667 5.000000 5.666667 3.333333 3.000000
##         decs     indv    reckl     sens     unem      cur     vuln
## 11  6.000000 3.333333 4.666667 5.000000 4.000000 4.000000 6.333333
## 30  4.000000 5.000000 3.666667 5.333333 2.666667 5.333333 4.666667
## 31  5.666667 5.666667 3.333333 5.000000 2.666667 5.000000 4.333333
## 42  5.000000 4.666667 4.333333 5.333333 2.333333 6.000000 4.000000
## 47  5.000000 3.333333 6.333333 4.333333 2.333333 3.333333 5.333333
## 48  4.500000 2.500000 2.000000 2.000000 2.500000 3.000000 3.000000
## 49  5.000000 3.666667 4.000000 4.000000 2.666667 5.666667 4.666667
## 57  6.666667 3.000000 3.000000 6.666667 4.666667 4.333333 4.333333
## 60  2.500000 2.500000 1.500000 3.000000 2.500000 2.500000 2.500000
## 63  6.000000 4.333333 5.333333 5.333333 3.000000 5.000000 6.000000
## 100 4.000000 4.000000 4.333333 3.000000 3.333333 3.333333 4.666667
## 107 5.000000 4.333333 3.666667 3.000000 3.666667 2.666667 4.666667
##         actv     pred     conv     cool    innov dominance extraversion
## 11  3.666667 4.333333 5.333333 6.333333 6.333333  5.222222     5.000000
## 30  3.333333 3.666667 2.333333 6.333333 7.000000  4.111111     4.416667
## 31  5.000000 5.333333 4.000000 6.333333 7.000000  5.444444     5.166667
## 42  5.666667 3.333333 4.333333 4.666667 6.333333  3.888889     5.416667
## 47  6.333333 4.333333 4.000000 3.666667 6.333333  4.000000     5.666667
## 48  3.000000 3.000000 2.500000 6.500000 7.000000  3.333333     4.625000
## 49  5.666667 3.333333 3.333333 6.333333 6.333333  3.111111     5.750000
## 57  5.333333 2.000000 3.000000 5.666667 6.666667  5.777778     5.750000
## 60  1.500000 3.500000 3.500000 3.500000 6.500000  2.333333     3.875000
## 63  5.333333 4.666667 3.333333 5.666667 7.000000  4.222222     4.583333
## 100 4.333333 4.000000 4.000000 5.333333 6.666667  3.777778     4.416667
## 107 2.666667 3.000000 2.666667 4.000000 6.666667  3.888889     4.250000
##     conscientiousness agreeableness neuroticism openness
## 11           3.444444      4.888889    3.833333 5.333333
## 30           4.222222      6.333333    3.333333 5.666667
## 31           3.888889      5.222222    5.833333 5.500000
## 42           2.888889      5.000000    4.833333 6.166667
## 47           2.777778      4.444444    4.333333 4.083333
## 48           4.666667      4.333333    4.000000 4.125000
## 49           3.444444      4.555556    3.333333 5.500000
## 57           3.333333      4.777778    3.666667 5.333333
## 60           4.666667      3.500000    3.250000 4.000000
## 63           3.444444      5.888889    4.666667 5.416667
## 100          3.888889      4.444444    3.833333 4.500000
## 107          4.000000      4.777778    4.000000 4.250000
```

Good news is that you can do this without the `which()` function and it still works:

```
gombe[gombe$innov > 6,]
```

```
##     chimpcode sex kasekela      dom      sol     impl     symp     stbl
## 11       R475   1        1 5.666667 2.666667 5.333333 3.333333 5.000000
## 30       D152   0        1 6.000000 3.333333 3.333333 6.666667 6.000000
## 31       U376   1        1 7.000000 1.333333 6.333333 4.333333 3.000000
## 42       A383   1        1 3.666667 1.666667 6.333333 4.000000 4.000000
## 47       M171   1        1 5.666667 1.666667 5.666667 2.333333 4.666667
## 48       C141   1        1 3.500000 4.000000 3.000000 5.000000 3.000000
## 49       C133   0        1 2.666667 1.333333 5.000000 4.000000 4.333333
## 57       G103   1        1 6.333333 2.000000 5.000000 3.000000 5.666667
## 60       U464   1        1 1.000000 4.000000 4.000000 3.500000 4.000000
## 63       Q315   1        1 4.333333 4.666667 5.000000 6.000000 3.666667
## 100      I142   1        1 5.000000 4.666667 4.000000 4.333333 4.000000
## 107      Y440   1        1 4.000000 3.000000 3.333333 5.666667 3.333333
##         invt     depd      soc    thotl     help     exct     inqs
## 11  5.666667 4.000000 6.333333 4.000000 6.333333 4.666667 5.333333
## 30  5.000000 5.666667 6.666667 3.000000 7.000000 4.666667 5.333333
## 31  4.666667 4.333333 6.666667 4.000000 6.333333 6.666667 5.333333
## 42  6.000000 5.000000 6.333333 2.333333 5.666667 5.666667 6.333333
## 47  3.000000 6.666667 5.333333 2.333333 6.666667 5.333333 3.666667
## 48  2.500000 6.000000 6.000000 2.500000 6.000000 3.000000 4.000000
## 49  4.666667 6.333333 6.333333 1.333333 5.666667 3.000000 5.333333
## 57  5.333333 3.666667 6.666667 2.666667 4.666667 5.000000 5.000000
## 60  3.000000 4.500000 4.500000 1.000000 4.000000 2.500000 4.000000
## 63  4.333333 5.666667 6.000000 4.000000 6.333333 5.000000 5.333333
## 100 4.333333 5.666667 6.000000 4.000000 6.000000 3.666667 3.666667
## 107 4.666667 5.333333 5.666667 5.000000 5.666667 3.333333 3.000000
##         decs     indv    reckl     sens     unem      cur     vuln
## 11  6.000000 3.333333 4.666667 5.000000 4.000000 4.000000 6.333333
## 30  4.000000 5.000000 3.666667 5.333333 2.666667 5.333333 4.666667
## 31  5.666667 5.666667 3.333333 5.000000 2.666667 5.000000 4.333333
## 42  5.000000 4.666667 4.333333 5.333333 2.333333 6.000000 4.000000
## 47  5.000000 3.333333 6.333333 4.333333 2.333333 3.333333 5.333333
## 48  4.500000 2.500000 2.000000 2.000000 2.500000 3.000000 3.000000
## 49  5.000000 3.666667 4.000000 4.000000 2.666667 5.666667 4.666667
## 57  6.666667 3.000000 3.000000 6.666667 4.666667 4.333333 4.333333
## 60  2.500000 2.500000 1.500000 3.000000 2.500000 2.500000 2.500000
## 63  6.000000 4.333333 5.333333 5.333333 3.000000 5.000000 6.000000
## 100 4.000000 4.000000 4.333333 3.000000 3.333333 3.333333 4.666667
## 107 5.000000 4.333333 3.666667 3.000000 3.666667 2.666667 4.666667
##         actv     pred     conv     cool    innov dominance extraversion
## 11  3.666667 4.333333 5.333333 6.333333 6.333333  5.222222     5.000000
## 30  3.333333 3.666667 2.333333 6.333333 7.000000  4.111111     4.416667
## 31  5.000000 5.333333 4.000000 6.333333 7.000000  5.444444     5.166667
## 42  5.666667 3.333333 4.333333 4.666667 6.333333  3.888889     5.416667
## 47  6.333333 4.333333 4.000000 3.666667 6.333333  4.000000     5.666667
## 48  3.000000 3.000000 2.500000 6.500000 7.000000  3.333333     4.625000
## 49  5.666667 3.333333 3.333333 6.333333 6.333333  3.111111     5.750000
## 57  5.333333 2.000000 3.000000 5.666667 6.666667  5.777778     5.750000
## 60  1.500000 3.500000 3.500000 3.500000 6.500000  2.333333     3.875000
## 63  5.333333 4.666667 3.333333 5.666667 7.000000  4.222222     4.583333
## 100 4.333333 4.000000 4.000000 5.333333 6.666667  3.777778     4.416667
```

```
## 107 2.666667 3.000000 2.666667 4.000000 6.666667  3.888889     4.250000
##     conscientiousness agreeableness neuroticism openness
## 11           3.444444      4.888889    3.833333 5.333333
## 30           4.222222      6.333333    3.333333 5.666667
## 31           3.888889      5.222222    5.833333 5.500000
## 42           2.888889      5.000000    4.833333 6.166667
## 47           2.777778      4.444444    4.333333 4.083333
## 48           4.666667      4.333333    4.000000 4.125000
## 49           3.444444      4.555556    3.333333 5.500000
## 57           3.333333      4.777778    3.666667 5.333333
## 60           4.666667      3.500000    3.250000 4.000000
## 63           3.444444      5.888889    4.666667 5.416667
## 100          3.888889      4.444444    3.833333 4.500000
## 107          4.000000      4.777778    4.000000 4.250000
```

Why is this? Let's see:

```
which(gombe$innov > 6)
```

```
##  [1]  11  30  31  42  47  48  49  57  60  63 100 107
```

```
gombe$innov > 6
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
##  [45] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

The first returns the row indices that match that condition, and the second returns a logical vector for every row in the data frame. Remember that either can be used to reference a data frame.

You can get rows that must match more than one condition with the `&` operator:

```
gombe[gombe$innov > 6 & gombe$sex == 1,]
```

```
##     chimpcode sex kasekela      dom      sol     impl     symp     stbl
## 11       R475   1        1 5.666667 2.666667 5.333333 3.333333 5.000000
## 31       U376   1        1 7.000000 1.333333 6.333333 4.333333 3.000000
## 42       A383   1        1 3.666667 1.666667 6.333333 4.000000 4.000000
## 47       M171   1        1 5.666667 1.666667 5.666667 2.333333 4.666667
## 48       C141   1        1 3.500000 4.000000 3.000000 5.000000 3.000000
## 57       G103   1        1 6.333333 2.000000 5.000000 3.000000 5.666667
## 60       U464   1        1 1.000000 4.000000 4.000000 3.500000 4.000000
## 63       Q315   1        1 4.333333 4.666667 5.000000 6.000000 3.666667
## 100      I142   1        1 5.000000 4.666667 4.000000 4.333333 4.000000
## 107      Y440   1        1 4.000000 3.000000 3.333333 5.666667 3.333333
##         invt     depd      soc    thotl     help     exct     inqs
## 11  5.666667 4.000000 6.333333 4.000000 6.333333 4.666667 5.333333
## 31  4.666667 4.333333 6.666667 4.000000 6.333333 6.666667 5.333333
## 42  6.000000 5.000000 6.333333 2.333333 5.666667 5.666667 6.333333
```

```
## 47  3.000000 6.666667 5.333333 2.333333 6.666667 5.333333 3.666667
## 48  2.500000 6.000000 6.000000 2.500000 6.000000 3.000000 4.000000
## 57  5.333333 3.666667 6.666667 2.666667 4.666667 5.000000 5.000000
## 60  3.000000 4.500000 4.500000 1.000000 4.000000 2.500000 4.000000
## 63  4.333333 5.666667 6.000000 4.000000 6.333333 5.000000 5.333333
## 100 4.333333 5.666667 6.000000 4.000000 6.000000 3.666667 3.666667
## 107 4.666667 5.333333 5.666667 5.000000 5.666667 3.333333 3.000000
##         decs     indv    reckl     sens     unem      cur     vuln
## 11  6.000000 3.333333 4.666667 5.000000 4.000000 4.000000 6.333333
## 31  5.666667 5.666667 3.333333 5.000000 2.666667 5.000000 4.333333
## 42  5.000000 4.666667 4.333333 5.333333 2.333333 6.000000 4.000000
## 47  5.000000 3.333333 6.333333 4.333333 2.333333 3.333333 5.333333
## 48  4.500000 2.500000 2.000000 2.000000 2.500000 3.000000 3.000000
## 57  6.666667 3.000000 3.000000 6.666667 4.666667 4.333333 4.333333
## 60  2.500000 2.500000 1.500000 3.000000 2.500000 2.500000 2.500000
## 63  6.000000 4.333333 5.333333 5.333333 3.000000 5.000000 6.000000
## 100 4.000000 4.000000 4.333333 3.000000 3.333333 3.333333 4.666667
## 107 5.000000 4.333333 3.666667 3.000000 3.666667 2.666667 4.666667
##         actv     pred     conv     cool    innov dominance extraversion
## 11  3.666667 4.333333 5.333333 6.333333 6.333333  5.222222     5.000000
## 31  5.000000 5.333333 4.000000 6.333333 7.000000  5.444444     5.166667
## 42  5.666667 3.333333 4.333333 4.666667 6.333333  3.888889     5.416667
## 47  6.333333 4.333333 4.000000 3.666667 6.333333  4.000000     5.666667
## 48  3.000000 3.000000 2.500000 6.500000 7.000000  3.333333     4.625000
## 57  5.333333 2.000000 3.000000 5.666667 6.666667  5.777778     5.750000
## 60  1.500000 3.500000 3.500000 3.500000 6.500000  2.333333     3.875000
## 63  5.333333 4.666667 3.333333 5.666667 7.000000  4.222222     4.583333
## 100 4.333333 4.000000 4.000000 5.333333 6.666667  3.777778     4.416667
## 107 2.666667 3.000000 2.666667 4.000000 6.666667  3.888889     4.250000
##     conscientiousness agreeableness neuroticism openness
## 11           3.444444      4.888889    3.833333 5.333333
## 31           3.888889      5.222222    5.833333 5.500000
## 42           2.888889      5.000000    4.833333 6.166667
## 47           2.777778      4.444444    4.333333 4.083333
## 48           4.666667      4.333333    4.000000 4.125000
## 57           3.333333      4.777778    3.666667 5.333333
## 60           4.666667      3.500000    3.250000 4.000000
## 63           3.444444      5.888889    4.666667 5.416667
## 100          3.888889      4.444444    3.833333 4.500000
## 107          4.000000      4.777778    4.000000 4.250000
```

Or return rows that match at least one of the conditions with the | (vertical pipe) "OR" operator:

```
gombe[gombe$innov >= 6.5 | gombe$invt >= 6.5,]
```

```
##     chimpcode sex kasekela      dom      sol     impl     symp     stbl
## 30       D152   0        1 6.000000 3.333333 3.333333 6.666667 6.000000
## 31       U376   1        1 7.000000 1.333333 6.333333 4.333333 3.000000
## 48       C141   1        1 3.500000 4.000000 3.000000 5.000000 3.000000
## 57       G103   1        1 6.333333 2.000000 5.000000 3.000000 5.666667
## 60       U464   1        1 1.000000 4.000000 4.000000 3.500000 4.000000
## 63       Q315   1        1 4.333333 4.666667 5.000000 6.000000 3.666667
## 100      I142   1        1 5.000000 4.666667 4.000000 4.333333 4.000000
## 107      Y440   1        1 4.000000 3.000000 3.333333 5.666667 3.333333
##         invt     depd      soc    thotl     help     exct     inqs
```

```
## 30  5.000000 5.666667 6.666667 3.000000 7.000000 4.666667 5.333333
## 31  4.666667 4.333333 6.666667 4.000000 6.333333 6.666667 5.333333
## 48  2.500000 6.000000 6.000000 2.500000 6.000000 3.000000 4.000000
## 57  5.333333 3.666667 6.666667 2.666667 4.666667 5.000000 5.000000
## 60  3.000000 4.500000 4.500000 1.000000 4.000000 2.500000 4.000000
## 63  4.333333 5.666667 6.000000 4.000000 6.333333 5.000000 5.333333
## 100 4.333333 5.666667 6.000000 4.000000 6.000000 3.666667 3.666667
## 107 4.666667 5.333333 5.666667 5.000000 5.666667 3.333333 3.000000
##         decs     indv    reckl     sens     unem      cur     vuln
## 30  4.000000 5.000000 3.666667 5.333333 2.666667 5.333333 4.666667
## 31  5.666667 5.666667 3.333333 5.000000 2.666667 5.000000 4.333333
## 48  4.500000 2.500000 2.000000 2.000000 2.500000 3.000000 3.000000
## 57  6.666667 3.000000 3.000000 6.666667 4.666667 4.333333 4.333333
## 60  2.500000 2.500000 1.500000 3.000000 2.500000 2.500000 2.500000
## 63  6.000000 4.333333 5.333333 5.333333 3.000000 5.000000 6.000000
## 100 4.000000 4.000000 4.333333 3.000000 3.333333 3.333333 4.666667
## 107 5.000000 4.333333 3.666667 3.000000 3.666667 2.666667 4.666667
##         actv     pred     conv     cool    innov dominance extraversion
## 30  3.333333 3.666667 2.333333 6.333333 7.000000  4.111111     4.416667
## 31  5.000000 5.333333 4.000000 6.333333 7.000000  5.444444     5.166667
## 48  3.000000 3.000000 2.500000 6.500000 7.000000  3.333333     4.625000
## 57  5.333333 2.000000 3.000000 5.666667 6.666667  5.777778     5.750000
## 60  1.500000 3.500000 3.500000 3.500000 6.500000  2.333333     3.875000
## 63  5.333333 4.666667 3.333333 5.666667 7.000000  4.222222     4.583333
## 100 4.333333 4.000000 4.000000 5.333333 6.666667  3.777778     4.416667
## 107 2.666667 3.000000 2.666667 4.000000 6.666667  3.888889     4.250000
##     conscientiousness agreeableness neuroticism openness
## 30           4.222222      6.333333    3.333333 5.666667
## 31           3.888889      5.222222    5.833333 5.500000
## 48           4.666667      4.333333    4.000000 4.125000
## 57           3.333333      4.777778    3.666667 5.333333
## 60           4.666667      3.500000    3.250000 4.000000
## 63           3.444444      5.888889    4.666667 5.416667
## 100          3.888889      4.444444    3.833333 4.500000
## 107          4.000000      4.777778    4.000000 4.250000
```

You can also use (or make) another vector that contains the names of the columns you want, and keep only the values that match. This uses the useful operator `%in%`, which asks "Is X `%in%` Y?" and returns `TRUE` or `FALSE` for each element in X.

```
varsToKeep = c("chimpcode","dom","soc")
head(gombe[,colnames(gombe) %in% varsToKeep])
```

```
##   chimpcode      dom      soc
## 1      E131 2.428571 4.571429
## 2       P70 4.666667 4.333333
## 3       G74 3.333333 5.666667
## 4      A364 1.666667 5.333333
## 5       B89 3.000000 6.000000
## 6       G19 4.000000 6.333333
```

Using `subset()`

Finally, sometimes the most straightforward way is to use the `subset()` function. You can pull up its help documentation and have a look at the syntax.

```
subset(hairEyeColor, Hair == "Black" & Eye == "Blue")
```

```
##     Hair  Eye    Sex Freq
## 5  Black Blue   Male   11
## 21 Black Blue Female    9
```

Notice that I didn't specify the names of the arguments here (`x=`, `subset=`). If you enter your arguments in the same order they appear in the function definition, you don't have to specify which is which. For functions like subset, it's easy to tell what each argument is referring to (and `x=Hair ==...` would look really confusing). Just make sure you're entering them in the right order, or you may get unexpected results.

Remember you need to **assign** your subset if you want to keep it! Otherwise your data frame is unchanged.

You should always keep your original data frame and assign subsets to new data frames, so that you can always go back and see the original or subset it differently.

Cleaning data

Tidy data

Three rules:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Visit: https://opendata.fcgov.com/Neighborhood-Livability-and-Social-Health/Fort-Collins-Shelter-Service-Data/u8nn-nj59

On the top right, click the "Export" button and then the "CSV" button to download the data. Place it in your class `/data` directory.

Read it into R:

```
focoShelter = read.csv(file="./data/Fort_Collins_Shelter_Service_Data.csv", header=TRUE)
```

Let's take a look at the data.

```
head(focoShelter)
```

```
##                    Month Rescue.Mission.Total.Served...Men
## 1 06/01/2017 12:00:00 AM                                1548
## 2 07/01/2017 12:00:00 AM                                1616
## 3 08/01/2017 12:00:00 AM                                1616
## 4 09/01/2017 12:00:00 AM                                1650
## 5 10/01/2017 12:00:00 AM                                2045
## 6 11/01/2017 12:00:00 AM                                2167
##   Rescue.Mission.Total.Served...Women
## 1                                 558
## 2                                 531
## 3                                 604
## 4                                 696
## 5                                 849
## 6                                 729
##   Rescue.Mission...of.Nights.with.Turnaways...Men
## 1                                              11
## 2                                               4
```

```
## 3                                                 10
## 4                                                  5
## 5                                                 13
## 6                                                  5
##   Rescue.Mission...of.Nights.with.Turnaways...Women
## 1                                                 2
## 2                                                 0
## 3                                                 6
## 4                                                 4
## 5                                                 5
## 6                                                 1
##   Rescue.Mission...Total...Turned.Away
## 1                                   41
## 2                                   10
## 3                                   37
## 4                                   21
## 5                                   55
## 6                                   15
##   Catholic.Charities.Total.Served...Men
## 1                                  1608
## 2                                  1582
## 3                                  1665
## 4                                  1402
## 5                                  1291
## 6                                  1739
##   Catholic.Charities.Total.Served...Women
## 1                                     588
## 2                                     615
## 3                                     625
## 4                                     475
## 5                                     487
## 6                                     602
##   Catholic.Charities.Total.Served...Families
## 1                                        134
## 2                                        142
## 3                                        148
## 4                                        129
## 5                                         96
## 6                                        118
##   Catholic.Charities...of.Nights.with.Turnaways
## 1                                            25
## 2                                            15
## 3                                            23
## 4                                            19
## 5                                             7
## 6                                             2
##   Catholic.Charities...Total...Turned.Away meta_row_index
## 1                                      127              0
## 2                                       56              1
## 3                                      105              2
## 4                                       64              3
## 5                                       26              4
## 6                                        6              5
##                               meta_row_id
```

```
## 1 d43fbe965d7aa17e8793776e1e03fe0a2322aa16
## 2 96d848fcfcbc3ff4369c88930f3170eba9f42e3c
## 3 0b30694d4ddef78c5b875cfef877192d50eace65
## 4 5507ebba2a6ae65b49b7961a1783995991dbff32
## 5 34350a8bd8a13ec601b7db4be332b005a476cb35
## 6 b97ecd23b4f9104a828de0f182d1482dfa9bcb3e
```

```
str(focoShelter)
```

```
## 'data.frame':    22 obs. of  13 variables:
##  $ Month                                          : Factor w/ 22 levels "01/01/2018 12:00:00 AM",.
##  $ Rescue.Mission.Total.Served...Men              : int  1548 1616 1616 1650 2045 2167 2036 2248 2
##  $ Rescue.Mission.Total.Served...Women            : int  558 531 604 696 849 729 679 740 539 648 .
##  $ Rescue.Mission...of.Nights.with.Turnaways...Men  : int  11 4 10 5 13 5 1 4 3 1 ...
##  $ Rescue.Mission...of.Nights.with.Turnaways...Women: int  2 0 6 4 5 1 1 1 0 0 ...
##  $ Rescue.Mission...Total...Turned.Away           : int  41 10 37 21 55 15 3 14 13 1 ...
##  $ Catholic.Charities.Total.Served...Men          : int  1608 1582 1665 1402 1291 1739 1495 1777 1
##  $ Catholic.Charities.Total.Served...Women        : int  588 615 625 475 487 602 459 509 402 592 .
##  $ Catholic.Charities.Total.Served...Families     : int  134 142 148 129 96 118 97 135 113 137 ...
##  $ Catholic.Charities...of.Nights.with.Turnaways  : int  25 15 23 19 7 2 0 1 3 5 ...
##  $ Catholic.Charities...Total...Turned.Away       : int  127 56 105 64 26 6 0 3 9 5 ...
##  $ meta_row_index                                 : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ meta_row_id                                    : Factor w/ 22 levels "0b30694d4ddef78c5b875cfef
```

```
colnames(focoShelter)
```

```
##  [1] "Month"
##  [2] "Rescue.Mission.Total.Served...Men"
##  [3] "Rescue.Mission.Total.Served...Women"
##  [4] "Rescue.Mission...of.Nights.with.Turnaways...Men"
##  [5] "Rescue.Mission...of.Nights.with.Turnaways...Women"
##  [6] "Rescue.Mission...Total...Turned.Away"
##  [7] "Catholic.Charities.Total.Served...Men"
##  [8] "Catholic.Charities.Total.Served...Women"
##  [9] "Catholic.Charities.Total.Served...Families"
## [10] "Catholic.Charities...of.Nights.with.Turnaways"
## [11] "Catholic.Charities...Total...Turned.Away"
## [12] "meta_row_index"
## [13] "meta_row_id"
```

So… are these data "tidy"?

No. Why not?

We've got variables spread across multiple columns. One variable is whether it was a Rescue Mission shelter or a Catholic Charities shelter. Another is whether it is referring to men, women, families, or all. The other variables are split apart by each of these.

So how do we fix it? Let's consult the Cheat Sheet.

Then we need to load the `tidyr` package. If you haven't installed the `tidyverse` package (which includes `tidyr`), you can do so now, or just install `tidyr` alone.

```
library(tidyr)
```

First, let's get rid of the last two columns since we don't need them.

```
focoShelter = focoShelter[,-c(12:13)]
```

Right now our data are in what's called **wide** format. It's called this because all of your variables are columns, so if you have a lot of variables then your data frame will stretch out horizontally. Because of Tidy Data Rule #1, most "tidy" data will be wide.

Our goal is to get these data cleaned up and back into the wide format. But first we have to change it to **tall** format to split some of our variables apart, and then convert it back to wide.

To get our data into the **tall** format, we use the `gather()` function from the Cheat Sheet. Prior to the `tidyr` package being developed, this was called "melting" (by using the `melt()` function from `reshape2`).

```
focoShelterTidy = gather(focoShelter, "variable", "value", 2:11)
```

We can use `unique()` to see all the unique values of the `variable` variable, to make sure all of our columns made it.

```
unique(focoShelterTidy$variable)
```

```
##  [1] "Rescue.Mission.Total.Served...Men"
##  [2] "Rescue.Mission.Total.Served...Women"
##  [3] "Rescue.Mission...of.Nights.with.Turnaways...Men"
##  [4] "Rescue.Mission...of.Nights.with.Turnaways...Women"
##  [5] "Rescue.Mission...Total...Turned.Away"
##  [6] "Catholic.Charities.Total.Served...Men"
##  [7] "Catholic.Charities.Total.Served...Women"
##  [8] "Catholic.Charities.Total.Served...Families"
##  [9] "Catholic.Charities...of.Nights.with.Turnaways"
## [10] "Catholic.Charities...Total...Turned.Away"
```

They did!

But we can see from our variable names that not all of them are split by "Men," "Women," and "Families." Some of them represent total numbers.

So, for our next step, let's split our data frame into two: one that contains only the "Men," "Women," and "Families" variables; and another that has all the others.

One way we can do this is to use `grepl()`, the logical version of the `grep()` function, which searches for partial character matches. We use it to search for any values of `variable` that contain the text "Men," "Women," or "Families," and then we subset the data frame to the rows that contain any of them (with the `|` "OR" operator).

```
focoShelterMWF = focoShelterTidy[grepl("Men", focoShelterTidy$variable, fixed=T) |
                                 grepl("Women", focoShelterTidy$variable, fixed=T) |
                                 grepl("Families", focoShelterTidy$variable, fixed=T),]
```

For our second subset that contains all the other rows, all we have to do is put a `!` before the `grepl()` functions to negate the logical vector that comes out of it (all "TRUE"s become "FALSE" and vice versa). This gives us the complementary set of rows to what we subset before.

```
focoShelterNot = focoShelterTidy[!grepl("Men", focoShelterTidy$variable, fixed=T) &
                                 !grepl("Women", focoShelterTidy$variable, fixed=T) &
                                 !grepl("Families", focoShelterTidy$variable, fixed=T),]
```

Now let's make sure both of our subsets add up to the whole.

```
nrow(focoShelterTidy)
```

```
## [1] 220
```

```
nrow(focoShelterMWF) + nrow(focoShelterNot)
```

```
## [1] 220
```

And we can take a look at our "MWF" data frame.

```
head(focoShelterMWF)
```

```
##                    Month                           variable value
## 1 06/01/2017 12:00:00 AM Rescue.Mission.Total.Served...Men  1548
## 2 07/01/2017 12:00:00 AM Rescue.Mission.Total.Served...Men  1616
## 3 08/01/2017 12:00:00 AM Rescue.Mission.Total.Served...Men  1616
## 4 09/01/2017 12:00:00 AM Rescue.Mission.Total.Served...Men  1650
## 5 10/01/2017 12:00:00 AM Rescue.Mission.Total.Served...Men  2045
## 6 11/01/2017 12:00:00 AM Rescue.Mission.Total.Served...Men  2167
```

```
str(focoShelterMWF)
```

```
## 'data.frame':    154 obs. of  3 variables:
##  $ Month   : Factor w/ 22 levels "01/01/2018 12:00:00 AM",..: 9 11 13 15 17 19 21 1 3 5 ...
##  $ variable: chr  "Rescue.Mission.Total.Served...Men" "Rescue.Mission.Total.Served...Men" "Rescue.Mis
##  $ value   : int  1548 1616 1616 1650 2045 2167 2036 2248 2023 1909 ...
```

```
colnames(focoShelterMWF)
```

```
## [1] "Month"    "variable" "value"
```

Our next task is to split the "Sex" element off of each variable. We can do this easily by using the "..." part of each variable name just before the sex it's referencing. The trouble is that some of the variable names have another "..." earlier in the character string and some don't.

One way to deal with this would be to just rename the variables (you could use `recode()` or `gsub()`) to make them easier to split.

Here, I use a nasty bit of regex ("regular expressions") to tell the function that I want it to separate based on the last occurrence of "..." for each variable name.

Some functions, like `grepl()` above, use regex but also allow you to just type in the match you want (with the `fixed=TRUE` argument). Unfortunately, `separate()` doesn't, but will often still work with plain text.

```
focoShelterMWF = separate(focoShelterMWF, 2, into=c("variable","Sex"),
                          sep="\\.\\.\\.(?!.*\\.\\.\\.)")
```

After doing that, we want to merge our two subset data frames back together. But to do that, they both need to have the same structure: the same columns in the same order.

So, for the "Not" data frame, we need to add the "Sex" column and, when we do, we can assign a value of "Total" for all of those variables, since they all refer to the total number of people served regardless of sex or gender.

Then, we reorder the columns to match the "MWF" data frame.

```
focoShelterNot$Sex = "Total"
focoShelterNot = focoShelterNot[,c(1,2,4,3)]
```

Let's check them out to make sure they look compatible.

```
head(focoShelterMWF)
```

```
##                    Month                    variable Sex value
## 1 06/01/2017 12:00:00 AM Rescue.Mission.Total.Served Men  1548
## 2 07/01/2017 12:00:00 AM Rescue.Mission.Total.Served Men  1616
## 3 08/01/2017 12:00:00 AM Rescue.Mission.Total.Served Men  1616
## 4 09/01/2017 12:00:00 AM Rescue.Mission.Total.Served Men  1650
## 5 10/01/2017 12:00:00 AM Rescue.Mission.Total.Served Men  2045
## 6 11/01/2017 12:00:00 AM Rescue.Mission.Total.Served Men  2167
```

```
head(focoShelterNot)
```

```
##                      Month                                variable   Sex value
## 89 06/01/2017 12:00:00 AM Rescue.Mission...Total...Turned.Away Total    41
## 90 07/01/2017 12:00:00 AM Rescue.Mission...Total...Turned.Away Total    10
## 91 08/01/2017 12:00:00 AM Rescue.Mission...Total...Turned.Away Total    37
## 92 09/01/2017 12:00:00 AM Rescue.Mission...Total...Turned.Away Total    21
## 93 10/01/2017 12:00:00 AM Rescue.Mission...Total...Turned.Away Total    55
## 94 11/01/2017 12:00:00 AM Rescue.Mission...Total...Turned.Away Total    15
```

They do. So now we can use `rbind()` to join them together row-wise (the second data frame underneath the first). If we needed to join extra columns, we could use `cbind()`, or there are a number of `dplyr` solutions for joining variables from different data sets on the Cheat Sheet.

```
focoShelterTidy = rbind(focoShelterMWF, focoShelterNot)
str(focoShelterTidy)
```

```
## 'data.frame':    220 obs. of  4 variables:
##  $ Month   : Factor w/ 22 levels "01/01/2018 12:00:00 AM",..: 9 11 13 15 17 19 21 1 3 5 ...
##  $ variable: chr  "Rescue.Mission.Total.Served" "Rescue.Mission.Total.Served" "Rescue.Mission.Total.S
##  $ Sex     : chr  "Men" "Men" "Men" "Men" ...
##  $ value   : int  1548 1616 1616 1650 2045 2167 2036 2248 2023 1909 ...
```

Looks good. Our last step is to convert it all back to wide format again.

```
focoShelterTidy = spread(focoShelterTidy, variable, value)
head(focoShelterTidy)
```

```
##                      Month      Sex
## 1 01/01/2018 12:00:00 AM Families
## 2 01/01/2018 12:00:00 AM      Men
## 3 01/01/2018 12:00:00 AM    Total
## 4 01/01/2018 12:00:00 AM    Women
## 5 01/01/2019 12:00:00 AM Families
## 6 01/01/2019 12:00:00 AM      Men
##   Catholic.Charities...of.Nights.with.Turnaways
## 1                                            NA
## 2                                            NA
## 3                                             1
## 4                                            NA
## 5                                            NA
## 6                                            NA
##   Catholic.Charities...Total...Turned.Away Catholic.Charities.Total.Served
## 1                                       NA                             135
## 2                                       NA                            1777
## 3                                        3                              NA
## 4                                       NA                             509
## 5                                       NA                             191
## 6                                       NA                            2311
##   Rescue.Mission...of.Nights.with.Turnaways
## 1                                        NA
## 2                                         4
## 3                                        NA
## 4                                         1
## 5                                        NA
## 6                                         9
##   Rescue.Mission...Total...Turned.Away Rescue.Mission.Total.Served
```

```
## 1                                    NA                       NA
## 2                                    NA                     2248
## 3                                    14                       NA
## 4                                    NA                      740
## 5                                    NA                       NA
## 6                                    NA                     2351
```

Our data are now tidy! There are a couple of final details we could take care of, though. Our `Month` variable and `Sex` variable have gotten out of order compared to the original data frame. We can fix them by reassigning them as factors and specifying the order of the levels. For `Month`, we can use the order given in the original data frame. For `Sex` we can write it out manually.

Then, we sort the data frame by those two columns. One way, used below, is to use the `with()` and `order()` functions, but there are others.

```
focoShelterTidy$Month = factor(focoShelterTidy$Month,
                               levels=as.character(focoShelter$Month))
focoShelterTidy$Sex = factor(focoShelterTidy$Sex,
                             levels=c("Men","Women","Families","Total"))
focoShelterTidy = focoShelterTidy[with(focoShelterTidy, order(Month, Sex)),]

head(focoShelterTidy)
```

```
##                      Month      Sex
## 34 06/01/2017 12:00:00 AM      Men
## 36 06/01/2017 12:00:00 AM    Women
## 33 06/01/2017 12:00:00 AM Families
## 35 06/01/2017 12:00:00 AM    Total
## 42 07/01/2017 12:00:00 AM      Men
## 44 07/01/2017 12:00:00 AM    Women
##    Catholic.Charities...of.Nights.with.Turnaways
## 34                                            NA
## 36                                            NA
## 33                                            NA
## 35                                            25
## 42                                            NA
## 44                                            NA
##    Catholic.Charities...Total...Turned.Away
## 34                                       NA
## 36                                       NA
## 33                                       NA
## 35                                      127
## 42                                       NA
## 44                                       NA
##    Catholic.Charities.Total.Served
## 34                            1608
## 36                             588
## 33                             134
## 35                              NA
## 42                            1582
## 44                             615
##    Rescue.Mission...of.Nights.with.Turnaways
## 34                                       11
## 36                                        2
## 33                                       NA
## 35                                       NA
```

```
## 42                                          4
## 44                                          0
##    Rescue.Mission...Total...Turned.Away Rescue.Mission.Total.Served
## 34                                   NA                       1548
## 36                                   NA                        558
## 33                                   NA                         NA
## 35                                   41                         NA
## 42                                   NA                       1616
## 44                                   NA                        531
```

```
str(focoShelterTidy)
```

```
## 'data.frame':    88 obs. of  8 variables:
##  $ Month                                        : Factor w/ 22 levels "06/01/2017 12:00:00 AM",..: 1
##  $ Sex                                          : Factor w/ 4 levels "Men","Women",..: 1 2 3 4 1 2 3
##  $ Catholic.Charities...of.Nights.with.Turnaways: int  NA NA NA 25 NA NA NA 15 NA NA ...
##  $ Catholic.Charities...Total...Turned.Away     : int  NA NA NA 127 NA NA NA 56 NA NA ...
##  $ Catholic.Charities.Total.Served              : int  1608 588 134 NA 1582 615 142 NA 1665 625 ...
##  $ Rescue.Mission...of.Nights.with.Turnaways    : int  11 2 NA NA 4 0 NA NA 10 6 ...
##  $ Rescue.Mission...Total...Turned.Away         : int  NA NA NA 41 NA NA NA 10 NA NA ...
##  $ Rescue.Mission.Total.Served                  : int  1548 558 NA NA 1616 531 NA NA 1616 604 ...
```

There is more we could do to these data, but we will leave it for now. For one, we can see there are a lot of missing data values. This is just because none of the variables had all of those levels defined in the first place. It would probably make sense for us to sum the other values of `Sex` to the level `Total` for variables that don't have `Total` defined. That way we could use that level for analysis, and look at the other levels if we wanted finer breakdowns by `Men`, `Women`, and `Families` for the variables that have them.

Problems with data

Missing data

Finding missing data

As we have seen, some data sets can have a lot of missing data, and this can be a problem when we want to run analyses.

For another example, we can load the built-in `airquality` data set.

```
data("airquality")
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

We can see right away that `airquality` has some `NA` values.

A quick way to see how many missing values our data has is to use the `summary()` function, which lists the number of `NA`s in each column below some summary statistics.

```
summary(airquality)
```

```
##      Ozone           Solar.R           Wind             Temp
```

```
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00
##  NA's   :37       NA's   :7
##      Month            Day
##  Min.   :5.000   Min.   : 1.0
##  1st Qu.:6.000   1st Qu.: 8.0
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
##
```

We can also do this for the `focoShelter` data that we just tidied.

```
summary(focoShelterTidy)
```

```
##                     Month          Sex
##  06/01/2017 12:00:00 AM: 4   Men     :22
##  07/01/2017 12:00:00 AM: 4   Women   :22
##  08/01/2017 12:00:00 AM: 4   Families:22
##  09/01/2017 12:00:00 AM: 4   Total   :22
##  10/01/2017 12:00:00 AM: 4
##  11/01/2017 12:00:00 AM: 4
##  (Other)                :64
##  Catholic.Charities...of.Nights.with.Turnaways
##  Min.   : 0.000
##  1st Qu.: 2.000
##  Median : 8.000
##  Mean   : 9.182
##  3rd Qu.:14.500
##  Max.   :25.000
##  NA's   :66
##  Catholic.Charities...Total...Turned.Away Catholic.Charities.Total.Served
##  Min.   :  0.00                           Min.   :  44.0
##  1st Qu.:  5.25                           1st Qu.: 139.8
##  Median : 29.00                           Median : 587.0
##  Mean   : 39.73                           Mean   : 766.6
##  3rd Qu.: 62.00                           3rd Qu.:1393.0
##  Max.   :127.00                           Max.   :2311.0
##  NA's   :66                               NA's   :22
##  Rescue.Mission...of.Nights.with.Turnaways
##  Min.   : 0.000
##  1st Qu.: 1.000
##  Median : 3.500
##  Mean   : 4.636
##  3rd Qu.: 6.250
##  Max.   :17.000
##  NA's   :44
##  Rescue.Mission...Total...Turned.Away Rescue.Mission.Total.Served
##  Min.   : 1.00                        Min.   : 531.0
##  1st Qu.:10.75                        1st Qu.: 694.5
```

```
##  Median :20.00                       Median :1215.0
##  Mean   :23.64                       Mean   :1301.7
##  3rd Qu.:35.00                       3rd Qu.:1924.2
##  Max.   :58.00                       Max.   :2351.0
##  NA's   :66                          NA's   :44
```

There are other ways to find missing values in a data set. For specific columns, we can subset using the `is.na()` function, which returns `TRUE` if it finds an `NA` value.

```
airquality[is.na(airquality$Ozone),]
```

```
##     Ozone Solar.R Wind Temp Month Day
## 5      NA      NA 14.3   56     5   5
## 10     NA     194  8.6   69     5  10
## 25     NA      66 16.6   57     5  25
## 26     NA     266 14.9   58     5  26
## 27     NA      NA  8.0   57     5  27
## 32     NA     286  8.6   78     6   1
## 33     NA     287  9.7   74     6   2
## 34     NA     242 16.1   67     6   3
## 35     NA     186  9.2   84     6   4
## 36     NA     220  8.6   85     6   5
## 37     NA     264 14.3   79     6   6
## 39     NA     273  6.9   87     6   8
## 42     NA     259 10.9   93     6  11
## 43     NA     250  9.2   92     6  12
## 45     NA     332 13.8   80     6  14
## 46     NA     322 11.5   79     6  15
## 52     NA     150  6.3   77     6  21
## 53     NA      59  1.7   76     6  22
## 54     NA      91  4.6   76     6  23
## 55     NA     250  6.3   76     6  24
## 56     NA     135  8.0   75     6  25
## 57     NA     127  8.0   78     6  26
## 58     NA      47 10.3   73     6  27
## 59     NA      98 11.5   80     6  28
## 60     NA      31 14.9   77     6  29
## 61     NA     138  8.0   83     6  30
## 65     NA     101 10.9   84     7   4
## 72     NA     139  8.6   82     7  11
## 75     NA     291 14.9   91     7  14
## 83     NA     258  9.7   81     7  22
## 84     NA     295 11.5   82     7  23
## 102    NA     222  8.6   92     8  10
## 103    NA     137 11.5   86     8  11
## 107    NA      64 11.5   79     8  15
## 115    NA     255 12.6   75     8  23
## 119    NA     153  5.7   88     8  27
## 150    NA     145 13.2   77     9  27
```

If we want to see the rows that have ANY missing values across all columns, we can use `rowSums()` with `is.na()` to return rows where at least one `NA` value exists.

```
airquality[rowSums(is.na(airquality)) > 0,]
```

```
##     Ozone Solar.R Wind Temp Month Day
## 5      NA      NA 14.3   56     5   5
```

```
## 6      28      NA 14.9   66     5   6
## 10     NA     194  8.6   69     5  10
## 11      7      NA  6.9   74     5  11
## 25     NA      66 16.6   57     5  25
## 26     NA     266 14.9   58     5  26
## 27     NA      NA  8.0   57     5  27
## 32     NA     286  8.6   78     6   1
## 33     NA     287  9.7   74     6   2
## 34     NA     242 16.1   67     6   3
## 35     NA     186  9.2   84     6   4
## 36     NA     220  8.6   85     6   5
## 37     NA     264 14.3   79     6   6
## 39     NA     273  6.9   87     6   8
## 42     NA     259 10.9   93     6  11
## 43     NA     250  9.2   92     6  12
## 45     NA     332 13.8   80     6  14
## 46     NA     322 11.5   79     6  15
## 52     NA     150  6.3   77     6  21
## 53     NA      59  1.7   76     6  22
## 54     NA      91  4.6   76     6  23
## 55     NA     250  6.3   76     6  24
## 56     NA     135  8.0   75     6  25
## 57     NA     127  8.0   78     6  26
## 58     NA      47 10.3   73     6  27
## 59     NA      98 11.5   80     6  28
## 60     NA      31 14.9   77     6  29
## 61     NA     138  8.0   83     6  30
## 65     NA     101 10.9   84     7   4
## 72     NA     139  8.6   82     7  11
## 75     NA     291 14.9   91     7  14
## 83     NA     258  9.7   81     7  22
## 84     NA     295 11.5   82     7  23
## 96     78      NA  6.9   86     8   4
## 97     35      NA  7.4   85     8   5
## 98     66      NA  4.6   87     8   6
## 102    NA     222  8.6   92     8  10
## 103    NA     137 11.5   86     8  11
## 107    NA      64 11.5   79     8  15
## 115    NA     255 12.6   75     8  23
## 119    NA     153  5.7   88     8  27
## 150    NA     145 13.2   77     9  27
```

With smaller data sets, it can also be easy to pick out `NA` values by scanning the data frame in a spreadsheet format, with `View()` (note the capital "V"):

```
View(airquality)
```

Fixing missing data

Now that we've found our missing values, what do we do with them? Well, there usually isn't a good answer, and it depends on your data and what you plan to do with it.

Removal

If you need to do something about them because your analyses require you to, the most straightforward answer is to remove them. However, if you don't have many data points or they were especially expensive or

difficult to collect, you may not want to throw them away.

If we want to remove values from just one variable, we can negate the statement that we used before to find the missing values to instead return the inverse where there are no missing values, using `!`.

(Note: I use `head()` below just to avoid flooding the screen with the full data set every time. It is not required for the task.)

```
head(airquality[!is.na(airquality$Ozone),])
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 6    28      NA 14.9   66     5   6
## 7    23     299  8.6   65     5   7
```

Likewise, we can return only rows that have zero `NA` values by slightly modifying our statement from before:

```
head(airquality[rowSums(is.na(airquality)) == 0,])
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```

There is also a function called `complete.cases()` that will return only the observations that have no missing values:

```
head(airquality[complete.cases(airquality)])
```

```
## Error in `[.data.frame`(airquality, complete.cases(airquality)): undefined columns selected
```

Oops. I forgot to include the `,` after my `complete.cases()` expression to indicate that I'm subsetting by rows and want to keep all the columns. If I fix that, it should work:

```
head(airquality[complete.cases(airquality),])
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```

Another way to return *only* missing values is to negate this condition:

```
airquality[!complete.cases(airquality),]
```

```
##     Ozone Solar.R Wind Temp Month Day
## 5      NA      NA 14.3   56     5   5
## 6      28      NA 14.9   66     5   6
## 10     NA     194  8.6   69     5  10
## 11      7      NA  6.9   74     5  11
## 25     NA      66 16.6   57     5  25
## 26     NA     266 14.9   58     5  26
## 27     NA      NA  8.0   57     5  27
```

```
## 32     NA     286  8.6   78     6   1
## 33     NA     287  9.7   74     6   2
## 34     NA     242 16.1   67     6   3
## 35     NA     186  9.2   84     6   4
## 36     NA     220  8.6   85     6   5
## 37     NA     264 14.3   79     6   6
## 39     NA     273  6.9   87     6   8
## 42     NA     259 10.9   93     6  11
## 43     NA     250  9.2   92     6  12
## 45     NA     332 13.8   80     6  14
## 46     NA     322 11.5   79     6  15
## 52     NA     150  6.3   77     6  21
## 53     NA      59  1.7   76     6  22
## 54     NA      91  4.6   76     6  23
## 55     NA     250  6.3   76     6  24
## 56     NA     135  8.0   75     6  25
## 57     NA     127  8.0   78     6  26
## 58     NA      47 10.3   73     6  27
## 59     NA      98 11.5   80     6  28
## 60     NA      31 14.9   77     6  29
## 61     NA     138  8.0   83     6  30
## 65     NA     101 10.9   84     7   4
## 72     NA     139  8.6   82     7  11
## 75     NA     291 14.9   91     7  14
## 83     NA     258  9.7   81     7  22
## 84     NA     295 11.5   82     7  23
## 96     78      NA  6.9   86     8   4
## 97     35      NA  7.4   85     8   5
## 98     66      NA  4.6   87     8   6
## 102    NA     222  8.6   92     8  10
## 103    NA     137 11.5   86     8  11
## 107    NA      64 11.5   79     8  15
## 115    NA     255 12.6   75     8  23
## 119    NA     153  5.7   88     8  27
## 150    NA     145 13.2   77     9  27
```

Let's go ahead and remove all rows with any missing values from `airquality` and see how many rows were removed.

```
airqualityRem = airquality[complete.cases(airquality),]
nrow(airquality)
```

```
## [1] 153
```

```
nrow(airqualityRem)
```

```
## [1] 111
```

```
nrow(airquality) - nrow(airqualityRem)
```

```
## [1] 42
```

```
(nrow(airquality) - nrow(airqualityRem)) / nrow(airquality)
```

```
## [1] 0.2745098
```

We lost 42 rows, or 27.4% of the original data set. That's a pretty sizeable chunk. So, again, keep in mind what you're throwing away when you remove missing data casewise.

Replacement

Another option is replacement. Sometimes this is a good idea if you need to fix missing values by inserting the value that should have been there.

```
airqualityFix = airquality
airqualityFix[is.na(airqualityFix$Ozone),][1,1] = 17
airqualityFix[is.na(airqualityFix$Solar.R),][1,2] = 142
head(airqualityFix)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    17     142 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Usually, though, you don't want to replace missing values with some other arbitrary value, because R won't treat them as missing values anymore.

```
airqualityRep = airquality
airqualityRep[is.na(airqualityRep) == T] = 0
head(airqualityRep)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5     0       0 14.3   56     5   5
## 6    28       0 14.9   66     5   6
```

Imputation

Another more advanced method for dealing with missing data, usually when you really need to be able to use those values for analysis, is called imputation. There are a variety of imputation methods that calculate a value to substitute in for each missing value.

One of the most basic is to use the mean of the variable:

```
airqualityImp = airquality
mean(airqualityImp$Ozone, na.rm=T)
```

```
## [1] 42.12931
```

```
airqualityImp$Ozone[is.na(airqualityImp$Ozone)] = mean(airqualityImp$Ozone, na.rm=T)
airqualityImp$Ozone
```

```
##   [1]  41.00000  36.00000  12.00000  18.00000  42.12931  28.00000  23.00000
##   [8]  19.00000   8.00000  42.12931   7.00000  16.00000  11.00000  14.00000
##  [15]  18.00000  14.00000  34.00000   6.00000  30.00000  11.00000   1.00000
##  [22]  11.00000   4.00000  32.00000  42.12931  42.12931  42.12931  23.00000
##  [29]  45.00000 115.00000  37.00000  42.12931  42.12931  42.12931  42.12931
##  [36]  42.12931  42.12931  29.00000  42.12931  71.00000  39.00000  42.12931
##  [43]  42.12931  23.00000  42.12931  42.12931  21.00000  37.00000  20.00000
##  [50]  12.00000  13.00000  42.12931  42.12931  42.12931  42.12931  42.12931
##  [57]  42.12931  42.12931  42.12931  42.12931  42.12931 135.00000  49.00000
##  [64]  32.00000  42.12931  64.00000  40.00000  77.00000  97.00000  97.00000
##  [71]  85.00000  42.12931  10.00000  27.00000  42.12931   7.00000  48.00000
```

```
##  [78]  35.00000  61.00000  79.00000  63.00000  16.00000  42.12931  42.12931
##  [85]  80.00000 108.00000  20.00000  52.00000  82.00000  50.00000  64.00000
##  [92]  59.00000  39.00000   9.00000  16.00000  78.00000  35.00000  66.00000
##  [99] 122.00000  89.00000 110.00000  42.12931  42.12931  44.00000  28.00000
## [106]  65.00000  42.12931  22.00000  59.00000  23.00000  31.00000  44.00000
## [113]  21.00000   9.00000  42.12931  45.00000 168.00000  73.00000  42.12931
## [120]  76.00000 118.00000  84.00000  85.00000  96.00000  78.00000  73.00000
## [127]  91.00000  47.00000  32.00000  20.00000  23.00000  21.00000  24.00000
## [134]  44.00000  21.00000  28.00000   9.00000  13.00000  46.00000  18.00000
## [141]  13.00000  24.00000  16.00000  13.00000  23.00000  36.00000   7.00000
## [148]  14.00000  30.00000  42.12931  14.00000  18.00000  20.00000
```

Whether this is a statistically valid solution is another question. My inclination is that it's usually a bad idea. The saying goes: "garbage in, garbage out." If you're running analyses using imputed data, you can't really trust the results that come out, because your data were fabricated in the first palce. This is especially questionable if at least a third of your data are imputed. Below that, you might be able to get away with it, but I still think it's usually a bad idea.

More sophisticated imputation methods are available in the `mice` package.

Collinearity and confounding

Collinearity and confounding can be huge problems in your data that you should be aware of. Sometimes they're more of a theoretical concern in that they may not throw errors in your analyses, but will give you spurious results whether you're aware of them or not.

Collinearity is usually only an issue when you get to running an ANOVA or multiple regression and throw in a bunch of variables to see what best predicts some outcome variable. Collinear variables are a problem because they are highly correlated and represent the same information.

```
cor(gombe$dom, gombe$dominance)
```

```
## [1] 0.7885871
```

In the `gombe` data set, both of these variables represent dominance (`dom` is an individual attitudinal rating and `dominance` is a combination of several variables). If you were running analyses, you'd want to pick one and use it, not both.

Confounding is, in my opinion, one of the most important problems to understand about data, especially when you're analyzing social data.

The short definition of confounding is when you have variables that influence your predictors and your outcome, so that you think there is a relationship between your variables that is really caused by another factor.

Some good (hilarious) examples are available at: http://www.tylervigen.com/spurious-correlations

Not all of these are examples of confounding, but it's easy to imagine how some third factor might be causing the trends in both correlated variables.

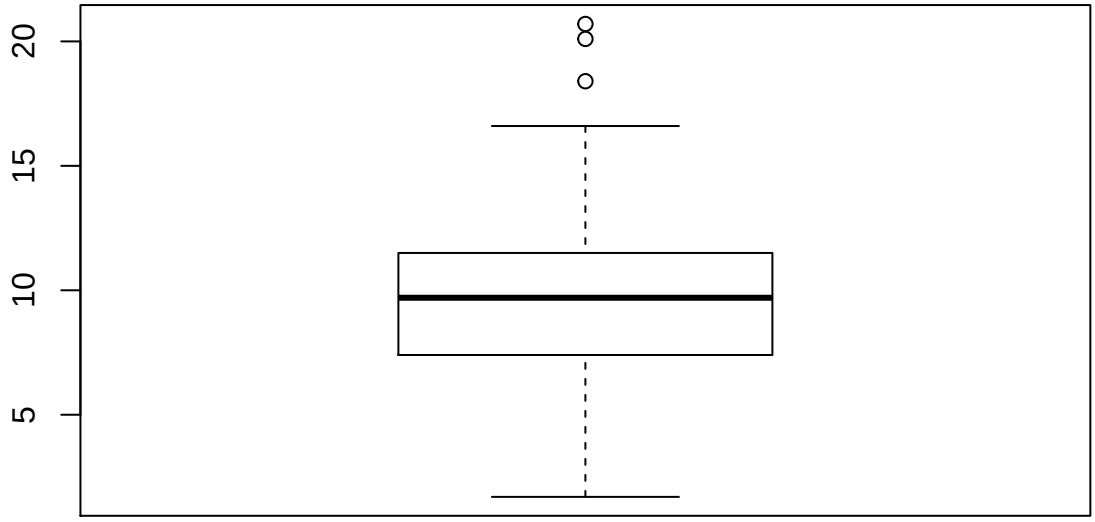
A classic example is that homicide rates are highly correlated with ice cream sales. Why? Does ice cream make people kill each other? No, of course not! It's because both homicide rates and ice cream sales are higher in the summer months.

Confounding is common in social science studies when people are looking for significant relationships between social variables (like genes that cause "success"), and forget to think about things like ethnicity and socioeconomic status that are associated with many other demographic and life history variables.

Outliers

```
# windows()  # Command to open a new plot window on Windows
# quartz()  # Command to open a new plot window on Mac
# X11()  # Command to open a new plot window on Linux
```

```
boxplot(airquality$Wind)
```




```
boxplot.stats(airquality$Wind)
```

```
## $stats
## [1]  1.7  7.4  9.7 11.5 16.6
##
## $n
## [1] 153
##
## $conf
## [1]  9.176285 10.223715
##
## $out
## [1] 20.1 18.4 20.7
```

```
names(boxplot.stats(airquality$Wind))
```

```
## [1] "stats" "n"     "conf"  "out"
```

```
boxplot.stats(airquality$Wind)$out
```

```
## [1] 20.1 18.4 20.7
```

"Tukey's Fences"

Quartile plus or minus 1.5 times the interquartile range:

```
Lower outlier: < Q1 - 1.5 (Q3 - Q1)
Upper outlier: > Q3 + 1.5 (Q3 - Q1)
```

```
quantile(airquality$Wind)
```

```
##   0%  25%  50%  75% 100%
##  1.7  7.4  9.7 11.5 20.7
```

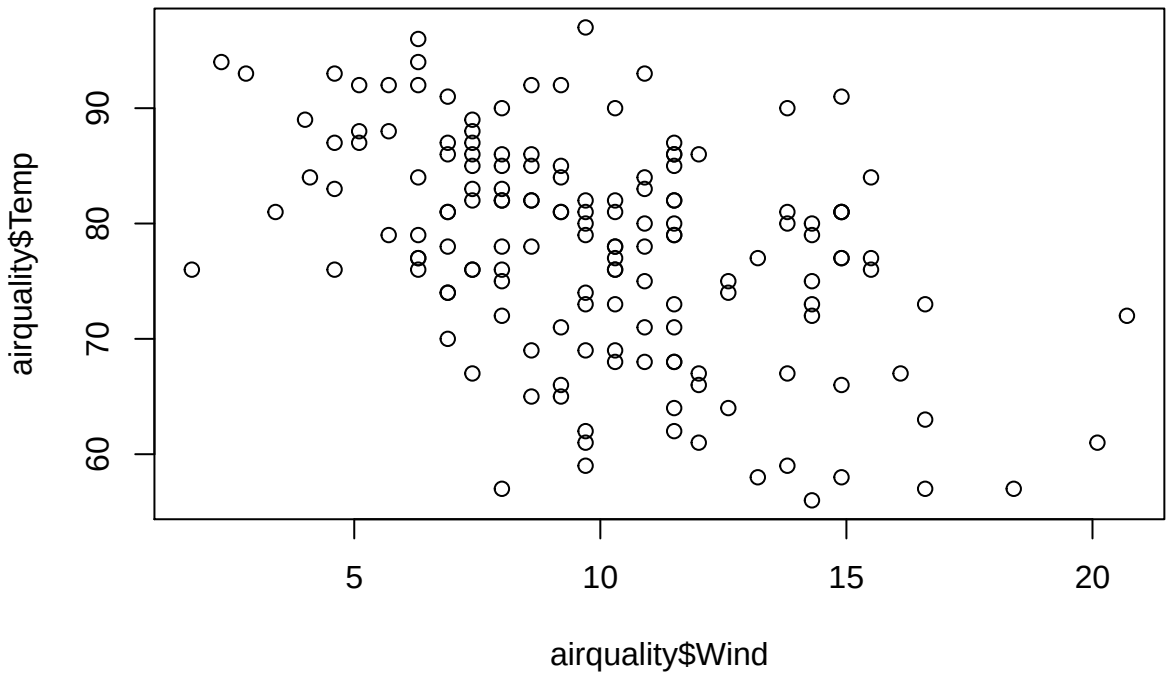
```
IQR(airquality$Wind)
```

```
## [1] 4.1
```

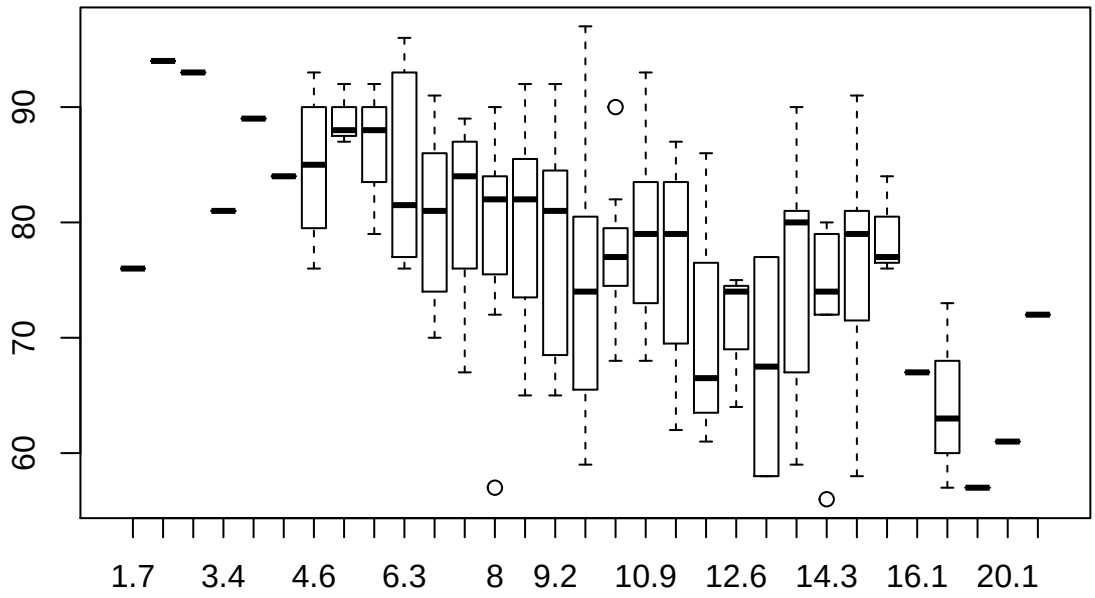
```
airquality$Wind[airquality$Wind < (quantile(airquality$Wind)[[2]] - 1.5 * IQR(airquality$Wind)) |
                  airquality$Wind > (quantile(airquality$Wind)[[4]] + 1.5 * IQR(airquality$Wind))]
```

```
## [1] 20.1 18.4 20.7
```

```
plot(airquality$Wind, airquality$Temp)
```




```
boxplot(Temp ~ Wind, data=airquality)
```



Identifying outliers when you're looking at a distribution of more than one variable is known as multivariate outlier detection. One package that handles this is `mvoutlier`.

Exploratory data analysis

EDA is NOT "fishing," "data dredging," or "*p*-hacking." EDA is an important part of understanding your data.

Descriptive statistics

```
summary(gombe)
```

```
##    chimpcode        sex            kasekela           dom        
##  A100   :  1   Min.   :0.0000   Min.   :0.0000   Min.   :1.000  
##  A341   :  1   1st Qu.:0.0000   1st Qu.:1.0000   1st Qu.:2.333  
##  A364   :  1   Median :0.0000   Median :1.0000   Median :3.000  
##  A383   :  1   Mean   :0.4375   Mean   :0.8188   Mean   :3.355  
##  A412   :  1   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.298  
##  B24    :  1   Max.   :1.0000   Max.   :1.0000   Max.   :7.000  
##  (Other):122                                                    
##       sol              impl            symp             stbl       
##  Min.   :1.000   Min.   :1.333   Min.   :1.333   Min.   :1.667  
##  1st Qu.:2.500   1st Qu.:2.667   1st Qu.:3.643   1st Qu.:3.000  
##  Median :3.333   Median :3.333   Median :4.333   Median :3.750  
##  Mean   :3.407   Mean   :3.504   Mean   :4.335   Mean   :3.865  
```

```
##  3rd Qu.:4.333   3rd Qu.:4.333   3rd Qu.:5.333   3rd Qu.:4.333
##  Max.   :7.000   Max.   :6.333   Max.   :6.750   Max.   :6.667
##
##       invt            depd            soc            thotl
##  Min.   :1.500   Min.   :1.333   Min.   :2.000   Min.   :1.000
##  1st Qu.:3.333   1st Qu.:3.667   1st Qu.:4.333   1st Qu.:2.000
##  Median :3.667   Median :4.333   Median :5.292   Median :2.500
##  Mean   :3.893   Mean   :4.306   Mean   :5.019   Mean   :2.603
##  3rd Qu.:4.333   3rd Qu.:5.000   3rd Qu.:6.000   3rd Qu.:3.000
##  Max.   :6.333   Max.   :6.667   Max.   :7.000   Max.   :5.000
##
##       help            exct            inqs            decs
##  Min.   :2.000   Min.   :1.333   Min.   :1.333   Min.   :2.000
##  1st Qu.:4.333   1st Qu.:2.667   1st Qu.:3.000   1st Qu.:4.333
##  Median :5.000   Median :3.417   Median :3.667   Median :5.000
##  Mean   :4.960   Mean   :3.619   Mean   :3.714   Mean   :4.910
##  3rd Qu.:5.700   3rd Qu.:4.350   3rd Qu.:4.333   3rd Qu.:5.667
##  Max.   :7.000   Max.   :6.667   Max.   :6.333   Max.   :6.800
##
##       indv           reckl           sens            unem
##  Min.   :1.333   Min.   :1.000   Min.   :2.000   Min.   :1.333
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:3.625   1st Qu.:2.667
##  Median :3.708   Median :2.667   Median :4.417   Median :3.333
##  Mean   :3.833   Mean   :3.003   Mean   :4.302   Mean   :3.402
##  3rd Qu.:4.667   3rd Qu.:3.786   3rd Qu.:5.000   3rd Qu.:4.000
##  Max.   :6.667   Max.   :6.333   Max.   :6.667   Max.   :5.333
##
##       cur             vuln            actv            pred
##  Min.   :1.667   Min.   :1.333   Min.   :1.500   Min.   :1.667
##  1st Qu.:3.000   1st Qu.:3.000   1st Qu.:4.000   1st Qu.:3.333
##  Median :3.667   Median :3.750   Median :4.667   Median :3.856
##  Mean   :3.787   Mean   :3.786   Mean   :4.565   Mean   :3.885
##  3rd Qu.:4.667   3rd Qu.:4.333   3rd Qu.:5.333   3rd Qu.:4.667
##  Max.   :6.333   Max.   :6.333   Max.   :7.000   Max.   :6.333
##
##       conv            cool           innov         dominance
##  Min.   :2.000   Min.   :1.667   Min.   :1.667   Min.   :2.111
##  1st Qu.:3.000   1st Qu.:4.000   1st Qu.:3.667   1st Qu.:3.444
##  Median :3.500   Median :4.667   Median :4.467   Median :3.830
##  Mean   :3.652   Mean   :4.604   Mean   :4.494   Mean   :3.986
##  3rd Qu.:4.298   3rd Qu.:5.333   3rd Qu.:5.333   3rd Qu.:4.514
##  Max.   :6.667   Max.   :6.500   Max.   :7.000   Max.   :6.000
##
##   extraversion   conscientiousness agreeableness    neuroticism
##  Min.   :2.500   Min.   :2.444     Min.   :2.444   Min.   :1.500
##  1st Qu.:4.236   1st Qu.:4.054     1st Qu.:4.111   1st Qu.:3.333
##  Median :4.612   Median :4.537     Median :4.516   Median :3.833
##  Mean   :4.586   Mean   :4.459     Mean   :4.532   Mean   :3.877
##  3rd Qu.:5.083   3rd Qu.:4.889     3rd Qu.:5.111   3rd Qu.:4.333
##  Max.   :5.917   Max.   :6.000     Max.   :6.417   Max.   :5.833
##
##     openness
##  Min.   :2.167
##  1st Qu.:3.417
```

```
##  Median :3.854
##  Mean   :3.972
##  3rd Qu.:4.462
##  Max.   :6.167
##
```

```
mean(gombe$extraversion)
```

```
## [1] 4.58585
```

```
median(gombe$extraversion)
```

```
## [1] 4.612228
```

Note: There is no built-in function to calculate the mode in R, but you can find one online or in a variety of packages.

```
range(gombe$extraversion)
```

```
## [1] 2.500000 5.916667
```

```
min(gombe$extraversion)
```

```
## [1] 2.5
```

```
max(gombe$extraversion)
```

```
## [1] 5.916667
```

```
quantile(gombe$extraversion)
```

```
##       0%      25%      50%      75%     100%
## 2.500000 4.236111 4.612228 5.083333 5.916667
```

```
sd(gombe$extraversion)
```

```
## [1] 0.6635367
```

```
var(gombe$extraversion)
```

```
## [1] 0.440281
```

We can verify that variance is the SD squared, and that SD is the square root of the variance.

```
sd(gombe$extraversion)^2
```

```
## [1] 0.440281
```

```
sqrt(var(gombe$extraversion))
```

```
## [1] 0.6635367
```

Univariate plots

Histograms and density plots

Let's look at all the values from the `horseKicks` data together, regardless of column.

```
horseKicks
```

```
##    Year GC C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C14 C15
## 1  1875  0  0  0  0  0  0  0  1  1  0   0   0   1   0
## 2  1876  2  0  0  0  1  0  0  0  0  0   0   0   1   1
## 3  1877  2  0  0  0  0  0  1  1  0  0   1   0   2   0
## 4  1878  1  2  2  1  1  0  0  0  0  0   1   0   1   0
## 5  1879  0  0  0  1  1  2  2  0  1  0   0   2   1   0
```

```
## 6  1880  0  3  2  1  1  1  0  0  0  2   1   4   3   0
## 7  1881  1  0  0  2  1  0  0  1  0  1   0   0   0   0
## 8  1882  1  2  0  0  0  0  1  0  1  1   2   1   4   1
## 9  1883  0  0  1  2  0  1  2  1  0  1   0   3   0   0
## 10 1884  3  0  1  0  0  0  0  1  0  0   2   0   1   1
## 11 1885  0  0  0  0  0  0  1  0  0  2   0   1   0   1
## 12 1886  2  1  0  0  1  1  1  0  0  1   0   1   3   0
## 13 1887  1  1  2  1  0  0  3  2  1  1   0   1   2   0
## 14 1888  0  1  1  0  0  1  1  0  0  0   0   1   1   0
## 15 1889  0  0  1  1  0  1  1  0  0  1   2   2   0   2
## 16 1890  1  2  0  2  0  1  1  2  0  2   1   1   2   2
## 17 1891  0  0  0  1  1  1  0  1  1  0   3   3   1   0
## 18 1892  1  3  2  0  1  1  3  0  1  1   0   1   1   0
## 19 1893  0  1  0  0  0  1  0  2  0  0   1   3   0   0
## 20 1894  1  0  0  0  0  0  0  0  1  0   1   1   0   0
```

```
hist(horseKicks[,c(2:ncol(horseKicks))])
```

```
## Error in hist.default(horseKicks[, c(2:ncol(horseKicks))]): 'x' must be numeric
```

Weird. Okay, let's try to coerce it to numeric.

```
hist(as.numeric(horseKicks[,c(2:ncol(horseKicks))]))
```

```
## Error in hist(as.numeric(horseKicks[, c(2:ncol(horseKicks))])): (list) object cannot be coerced to ty
```

For some strange reason, it's stored as a list. To get it out, we can use `unlist()`:

```
unlist(horseKicks[,c(2:ncol(horseKicks))])
```

```
##   GC1   GC2   GC3   GC4   GC5   GC6   GC7   GC8   GC9  GC10  GC11  GC12
##     0     2     2     1     0     0     1     1     0     3     0     2
##  GC13  GC14  GC15  GC16  GC17  GC18  GC19  GC20   C11   C12   C13   C14
##     1     0     0     1     0     1     0     1     0     0     0     2
##   C15   C16   C17   C18   C19  C110  C111  C112  C113  C114  C115  C116
##     0     3     0     2     0     0     0     1     1     1     0     2
##  C117  C118  C119  C120   C21   C22   C23   C24   C25   C26   C27   C28
##     0     3     1     0     0     0     0     2     0     2     0     0
##   C29  C210  C211  C212  C213  C214  C215  C216  C217  C218  C219  C220
##     1     1     0     0     2     1     1     0     0     2     0     0
##   C31   C32   C33   C34   C35   C36   C37   C38   C39  C310  C311  C312
##     0     0     0     1     1     1     2     0     2     0     0     0
##  C313  C314  C315  C316  C317  C318  C319  C320   C41   C42   C43   C44
##     1     0     1     2     1     0     0     0     0     1     0     1
##   C45   C46   C47   C48   C49  C410  C411  C412  C413  C414  C415  C416
##     1     1     1     0     0     0     0     1     0     0     0     0
##  C417  C418  C419  C420   C51   C52   C53   C54   C55   C56   C57   C58
##     1     1     0     0     0     0     0     0     2     1     0     0
##   C59  C510  C511  C512  C513  C514  C515  C516  C517  C518  C519  C520
##     1     0     0     1     0     1     1     1     1     1     1     0
##   C61   C62   C63   C64   C65   C66   C67   C68   C69  C610  C611  C612
##     0     0     1     0     2     0     0     1     2     0     1     1
##  C613  C614  C615  C616  C617  C618  C619  C620   C71   C72   C73   C74
##     3     1     1     1     0     3     0     0     1     0     1     0
##   C75   C76   C77   C78   C79  C710  C711  C712  C713  C714  C715  C716
##     0     0     1     0     1     1     0     0     2     0     0     2
##  C717  C718  C719  C720   C81   C82   C83   C84   C85   C86   C87   C88
##     1     0     2     0     1     0     0     0     1     0     0     1
```

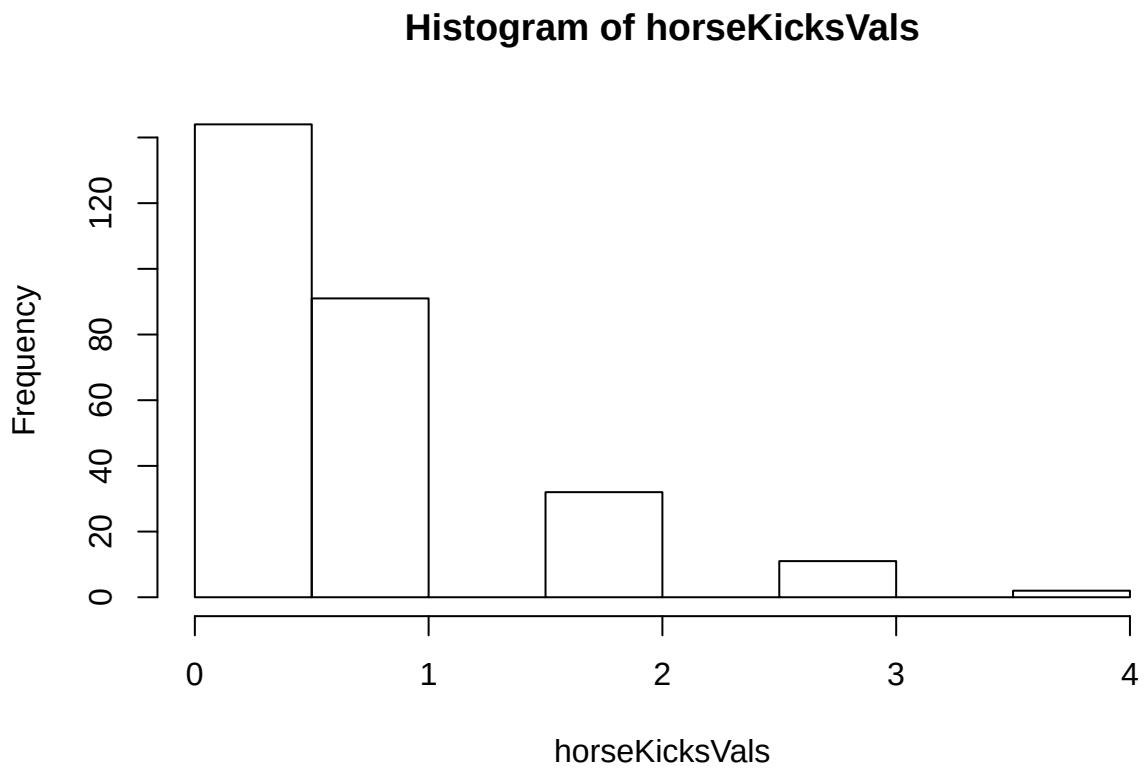
```
##   C89  C810  C811  C812  C813  C814  C815  C816  C817  C818  C819  C820
##     0     0     0     0     1     0     0     0     1     1     0     1
##   C91   C92   C93   C94   C95   C96   C97   C98   C99  C910  C911  C912
##     0     0     0     0     0     2     1     1     1     0     2     1
##  C913  C914  C915  C916  C917  C918  C919  C920  C101  C102  C103  C104
##     1     0     1     2     0     1     0     0     0     0     1     1
##  C105  C106  C107  C108  C109 C1010 C1011 C1012 C1013 C1014 C1015 C1016
##     0     1     0     2     0     2     0     0     0     0     2     1
## C1017 C1018 C1019 C1020  C111  C112  C113  C114  C115  C116  C117  C118
##     3     0     1     1     0     0     0     0     2     4     0     1
##  C119 C1110 C1111 C1112 C1113 C1114 C1115 C1116 C1117 C1118 C1119 C1120
##     3     0     1     1     1     1     2     1     3     1     3     1
##  C141  C142  C143  C144  C145  C146  C147  C148  C149 C1410 C1411 C1412
##     1     1     2     1     1     3     0     4     0     1     0     3
## C1413 C1414 C1415 C1416 C1417 C1418 C1419 C1420  C151  C152  C153  C154
##     2     1     0     2     1     1     0     0     0     1     0     0
##  C155  C156  C157  C158  C159 C1510 C1511 C1512 C1513 C1514 C1515 C1516
##     0     0     0     1     0     1     1     0     0     0     2     2
## C1517 C1518 C1519 C1520
##     0     0     0     0
```

And we can use `unname()` to get just the values of the vector without the column names:

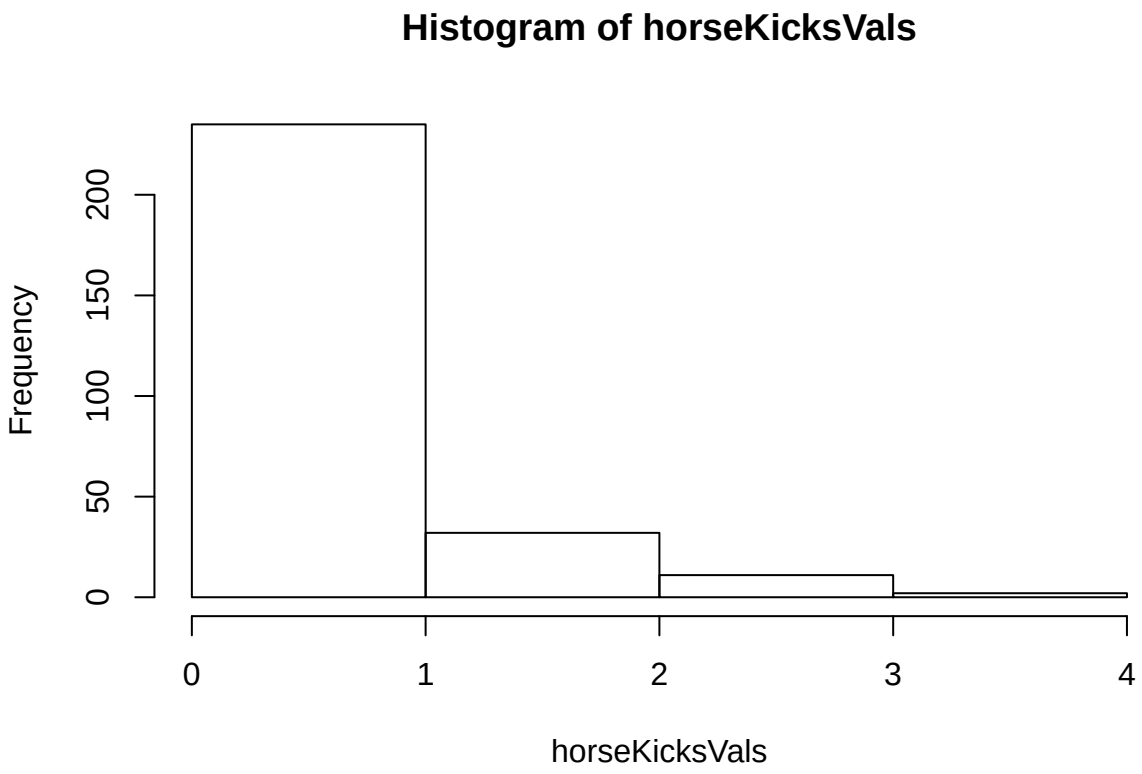
```
horseKicksVals = unname(unlist(horseKicks[,c(2:ncol(horseKicks))]))
horseKicksVals
```

```
##   [1] 0 2 2 1 0 0 1 1 0 3 0 2 1 0 0 1 0 1 0 1 0 0 0 2 0 3 0 2 0 0 0 1 1 1 0
##  [36] 2 0 3 1 0 0 0 0 2 0 2 0 0 1 1 0 0 2 1 1 0 0 2 0 0 0 0 0 1 1 1 2 0 2 0
##  [71] 0 0 1 0 1 2 1 0 0 0 0 1 0 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 2
## [106] 1 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 1 0 2 0 0 1 2 0 1 1 3 1 1 1 0 3 0 0
## [141] 1 0 1 0 0 0 1 0 1 1 0 0 2 0 0 2 1 0 2 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0
## [176] 0 1 1 0 1 0 0 0 0 0 2 1 1 1 0 2 1 1 0 1 2 0 1 0 0 0 0 1 1 0 1 0 2 0 2
## [211] 0 0 0 0 2 1 3 0 1 1 0 0 0 0 2 4 0 1 3 0 1 1 1 1 2 1 3 1 3 1 1 1 2 1 1
## [246] 3 0 4 0 1 0 3 2 1 0 2 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 2 2 0 0 0 0
```

```
hist(horseKicksVals)
```



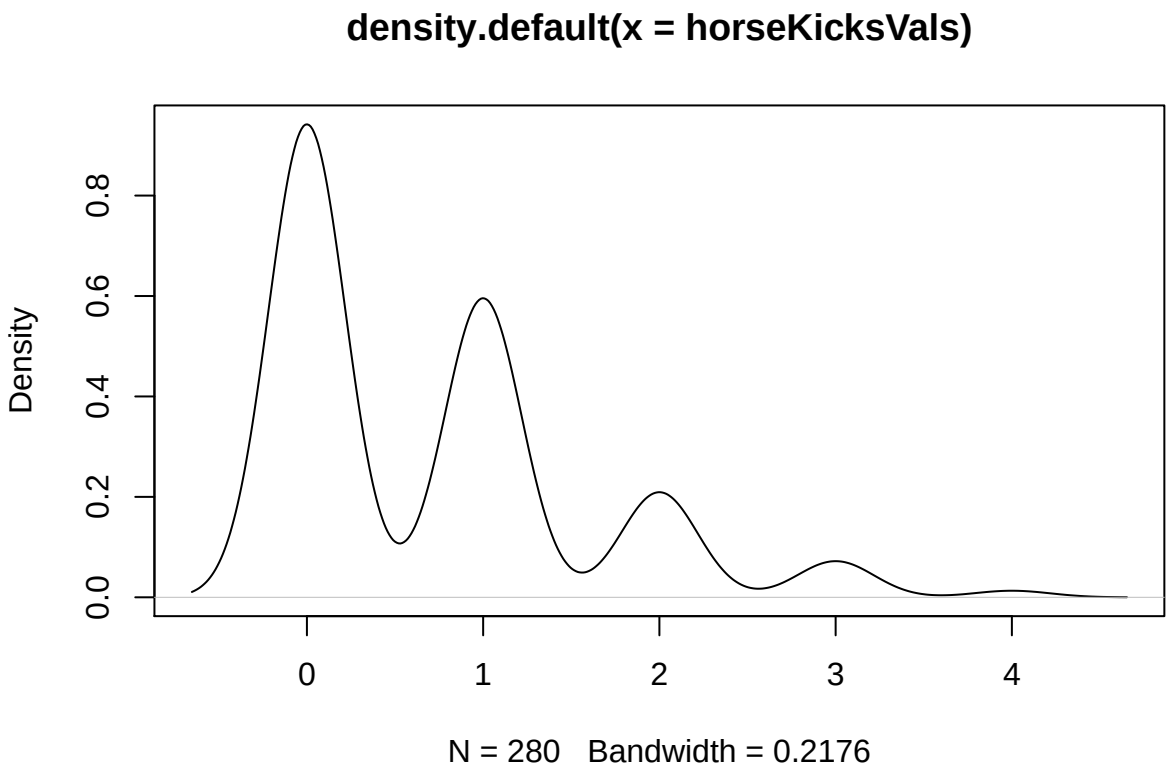

```
hist(horseKicksVals, breaks=4)
```




```
density(horseKicksVals)
```

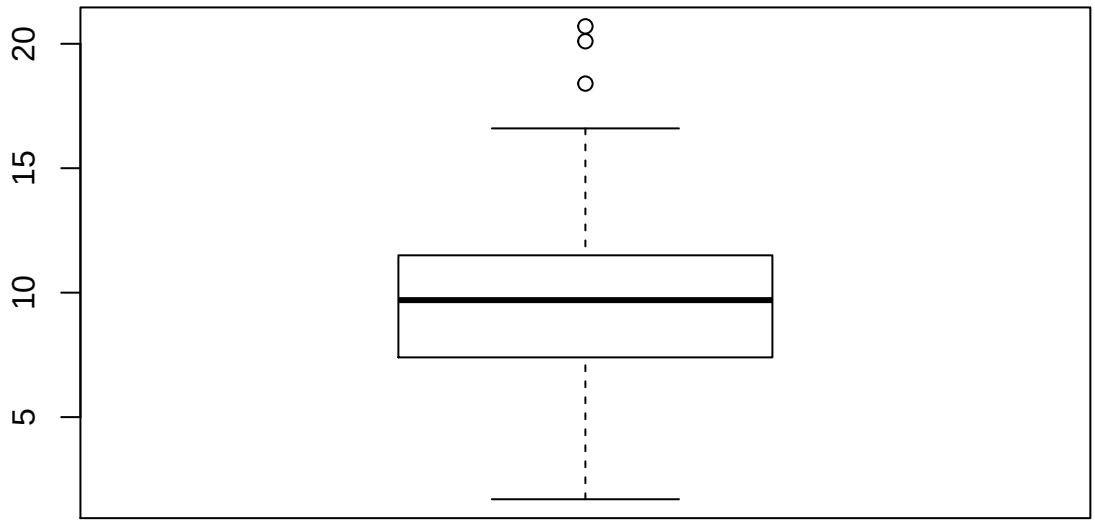
```
##
## Call:
##  density.default(x = horseKicksVals)
##
## Data: horseKicksVals (280 obs.); Bandwidth 'bw' = 0.2176
##
##        x                y
##  Min.   :-0.6529   Min.   :0.0001476
##  1st Qu.: 0.6736   1st Qu.:0.0169635
##  Median : 2.0000   Median :0.0714833
##  Mean   : 2.0000   Mean   :0.1881658
##  3rd Qu.: 3.3264   3rd Qu.:0.2342777
##  Max.   : 4.6529   Max.   :0.9418414
```

```
plot(density(horseKicksVals))
```



Box plots and violin plots

```
boxplot(airquality$Wind)
```



A violin plot displays the same data, but shows the distribution of values on the horizontal (X) axis.

```
install.packages("vioplot")
```

```
library(vioplot)
```

```
## Warning: package 'vioplot' was built under R version 3.5.3
```

```
## Loading required package: sm
```

```
## Warning: package 'sm' was built under R version 3.5.3
```

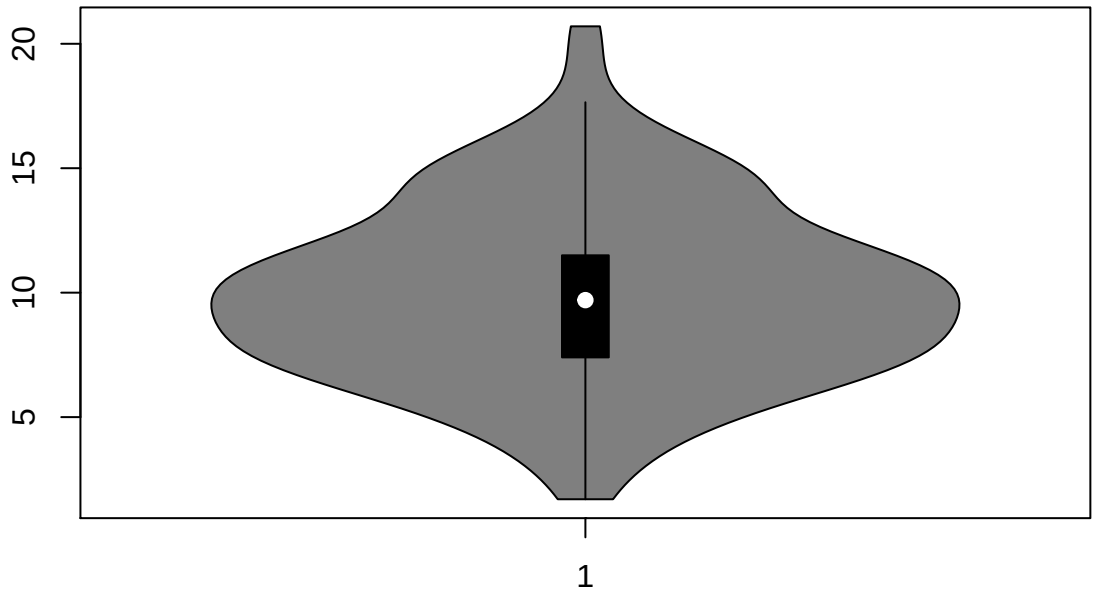
```
## Package 'sm', version 2.2-5.6: type help(sm) for summary information
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

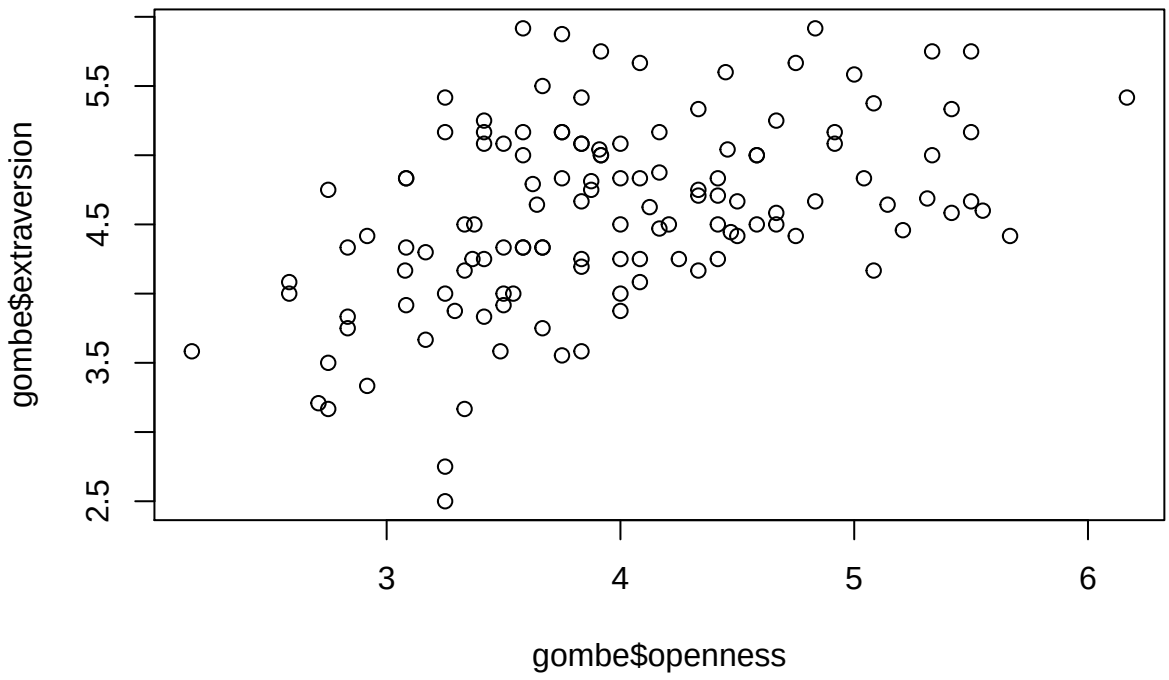
```
vioplot(airquality$Wind)
```



Bivariate plots

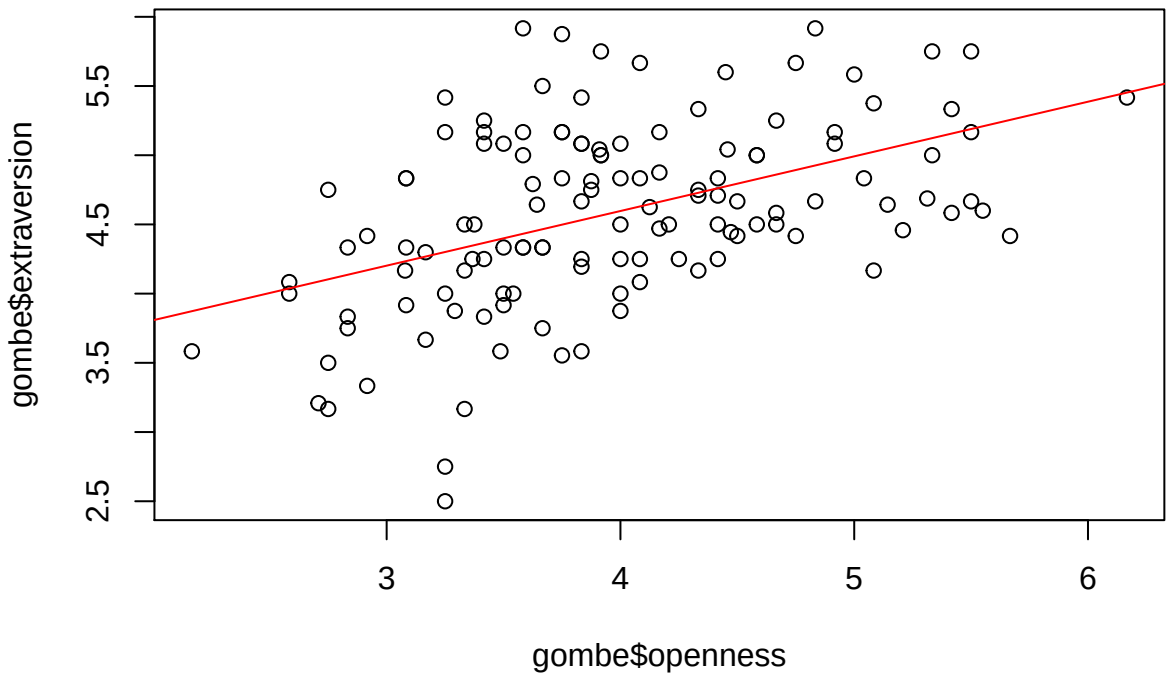
Scatterplots

```
plot(gombe$openness, gombe$extraversion)
```



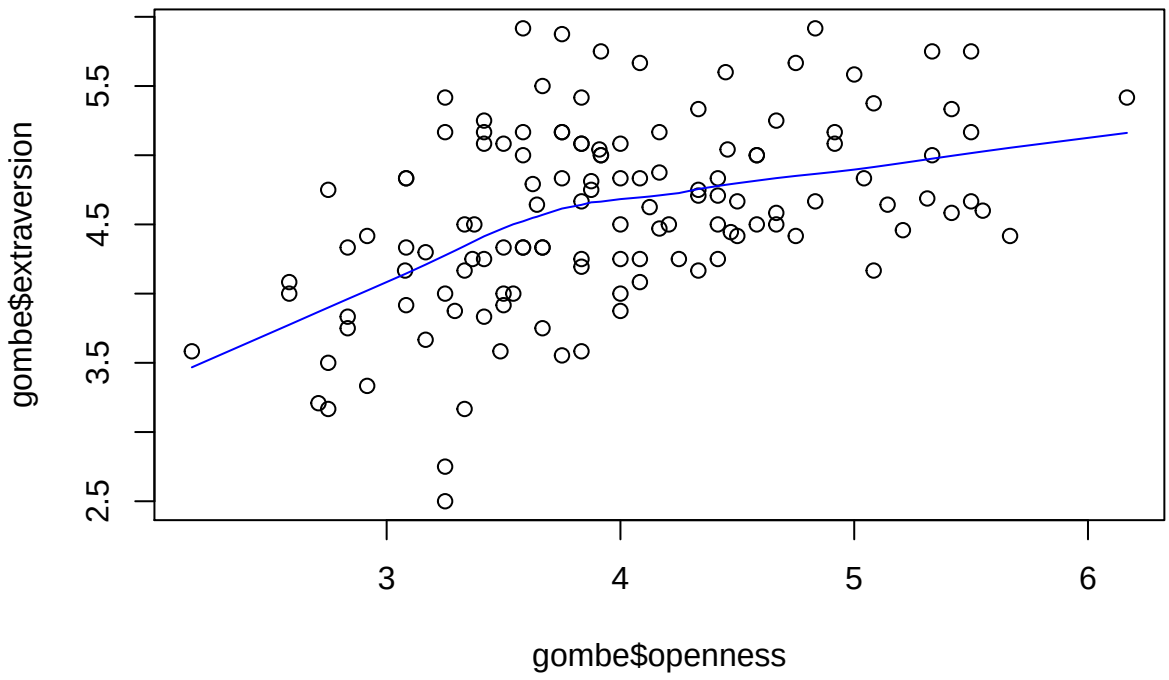
We can add a regression line:

```
plot(gombe$openness, gombe$extraversion)
abline(lm(gombe$extraversion ~ gombe$openness), col="red")
```

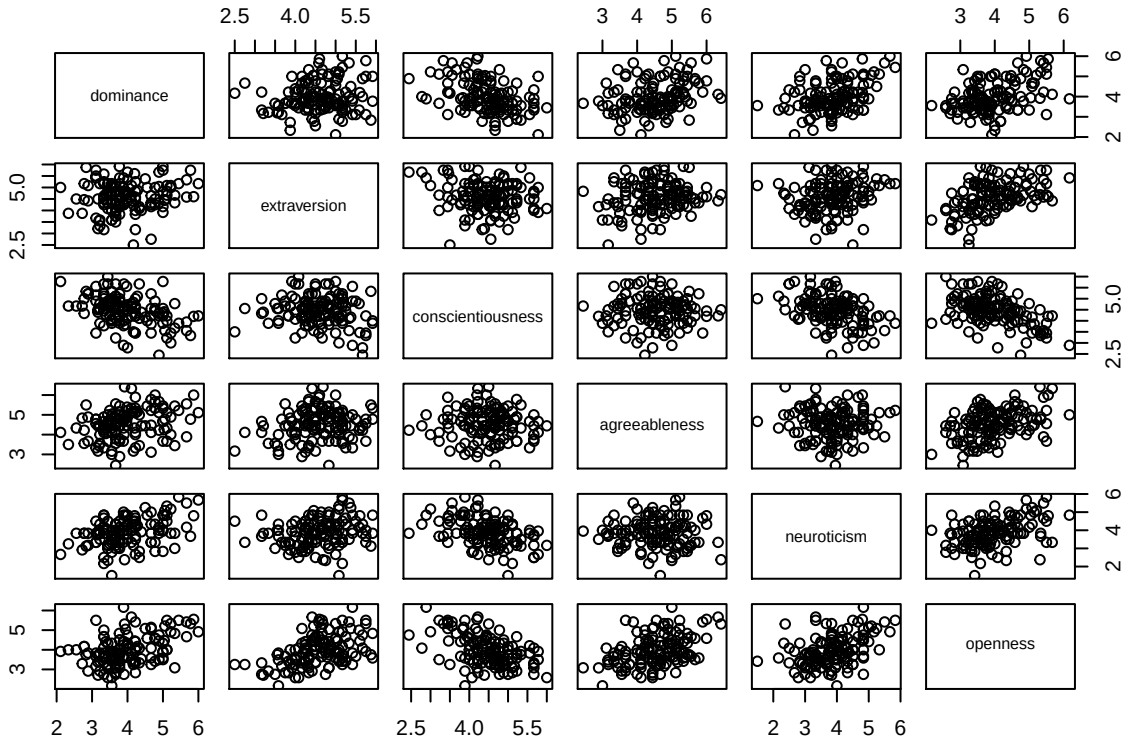


Or a smoothed LOESS/LOWESS line:

```
plot(gombe$openness, gombe$extraversion)
lines(lowess(gombe$openness, gombe$extraversion), col="blue")
```

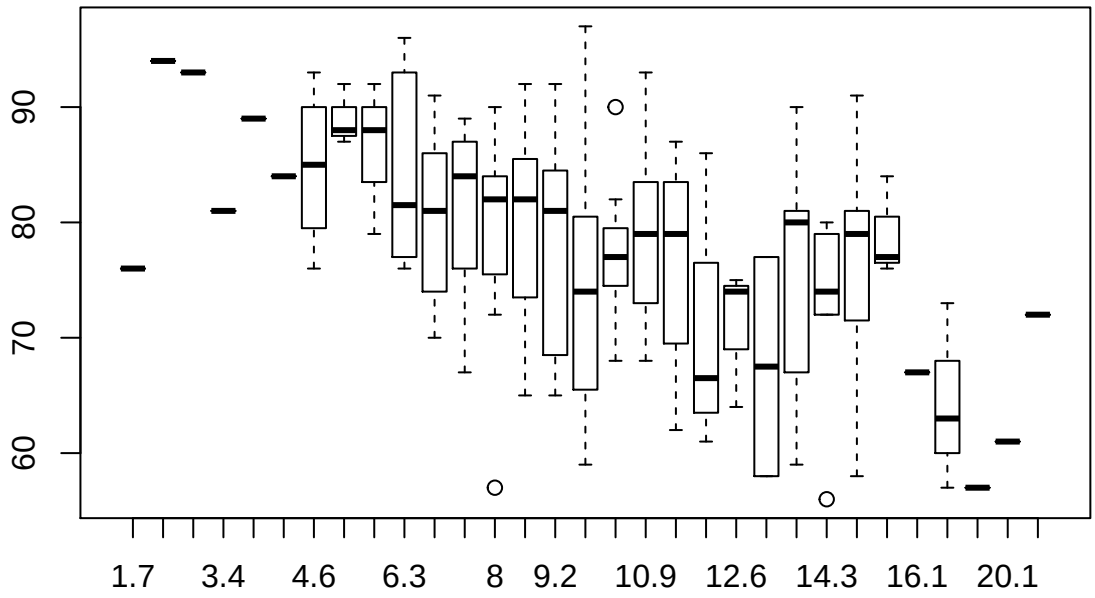



```
pairs(~ dominance + extraversion + conscientiousness +
        agreeableness + neuroticism + openness,
      data=gombe)
```

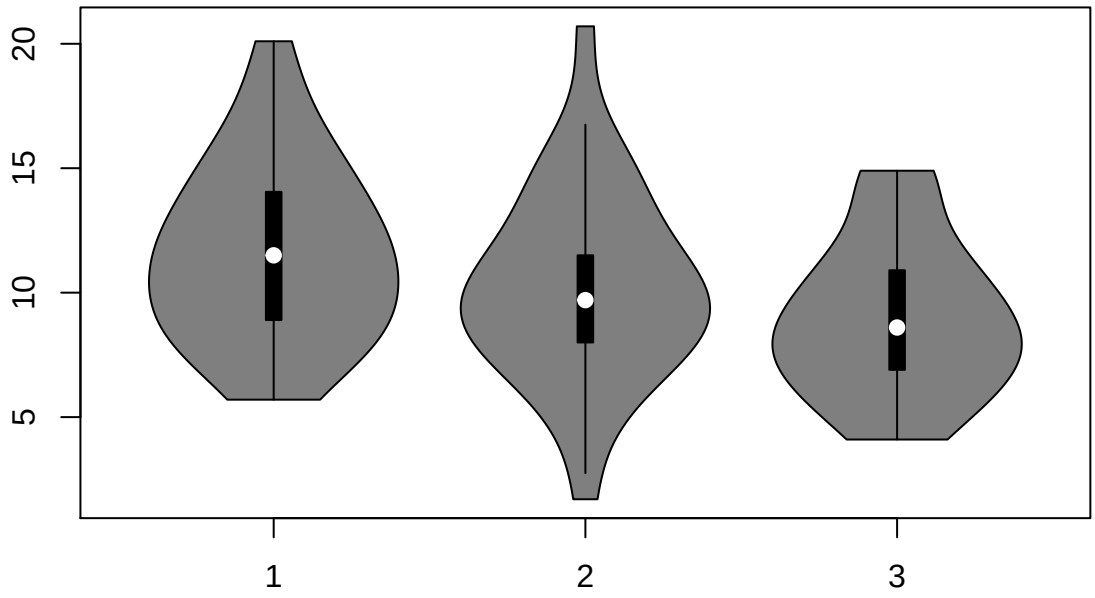


Box plots and violin plots

```
boxplot(Temp ~ Wind, data=airquality)
```




```
vioplot(airquality$Wind[airquality$Month == 5],
        airquality$Wind[airquality$Month == 6],
        airquality$Wind[airquality$Month == 7])
```



Association

```
cor(gombe$openness, gombe$extraversion)
```

```
## [1] 0.4681147
```

```
cor.test(gombe$openness, gombe$extraversion)
```

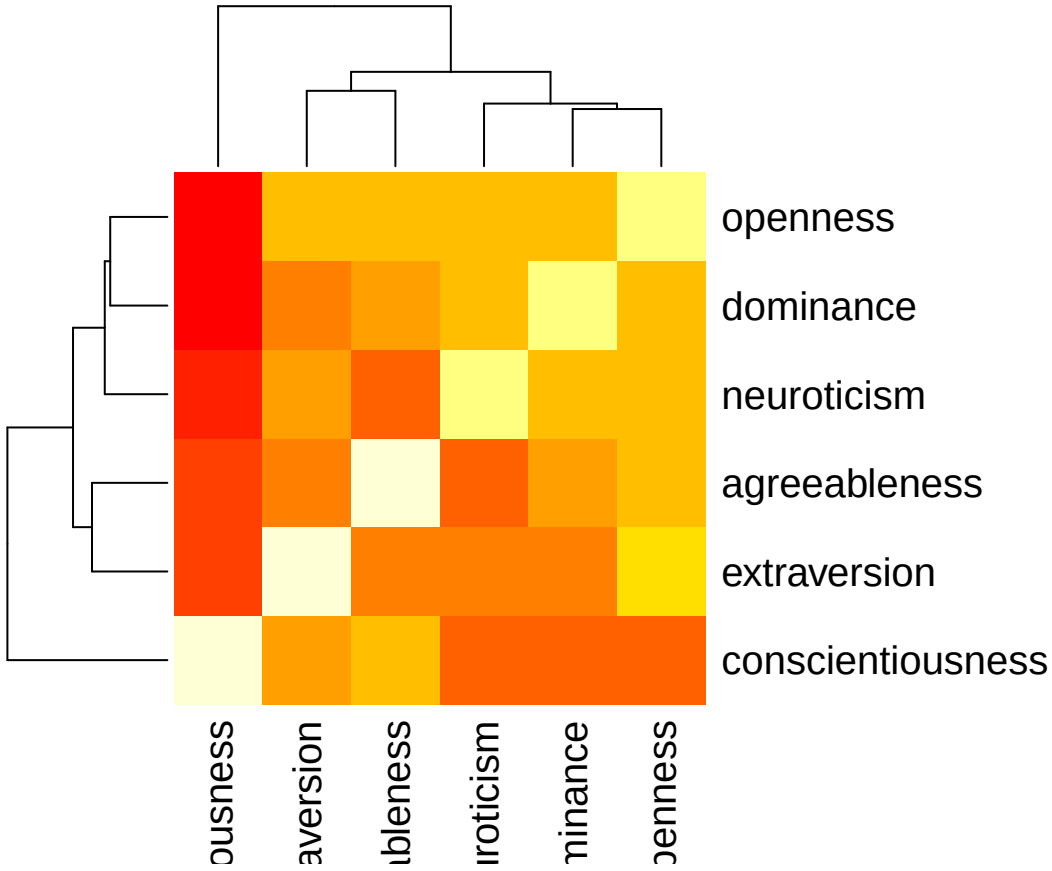
```
##
##  Pearson's product-moment correlation
##
## data:  gombe$openness and gombe$extraversion
## t = 5.9463, df = 126, p-value = 2.521e-08
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3206297 0.5934389
## sample estimates:
##       cor
## 0.4681147
```

```
cor(gombe[,28:33])
```

```
##                   dominance extraversion conscientiousness agreeableness
## dominance         1.0000000    0.1364398       -0.41875624    0.29389987
## extraversion      0.1364398    1.0000000       -0.14321625    0.18180931
## conscientiousness -0.4187562   -0.1432162        1.00000000   -0.02349069
## agreeableness     0.2938999    0.1818093       -0.02349069    1.00000000
```

```
## neuroticism         0.3942695    0.1884500       -0.41641012   -0.01586568
## openness            0.4142172    0.4681147       -0.49466090    0.41866159
##                   neuroticism   openness
## dominance          0.39426953  0.4142172
## extraversion       0.18845003  0.4681147
## conscientiousness -0.41641012 -0.4946609
## agreeableness     -0.01586568  0.4186616
## neuroticism        1.00000000  0.4048034
## openness           0.40480340  1.0000000
```

```
gombeCorMat = cor(gombe[,28:33])
heatmap(gombeCorMat)
```



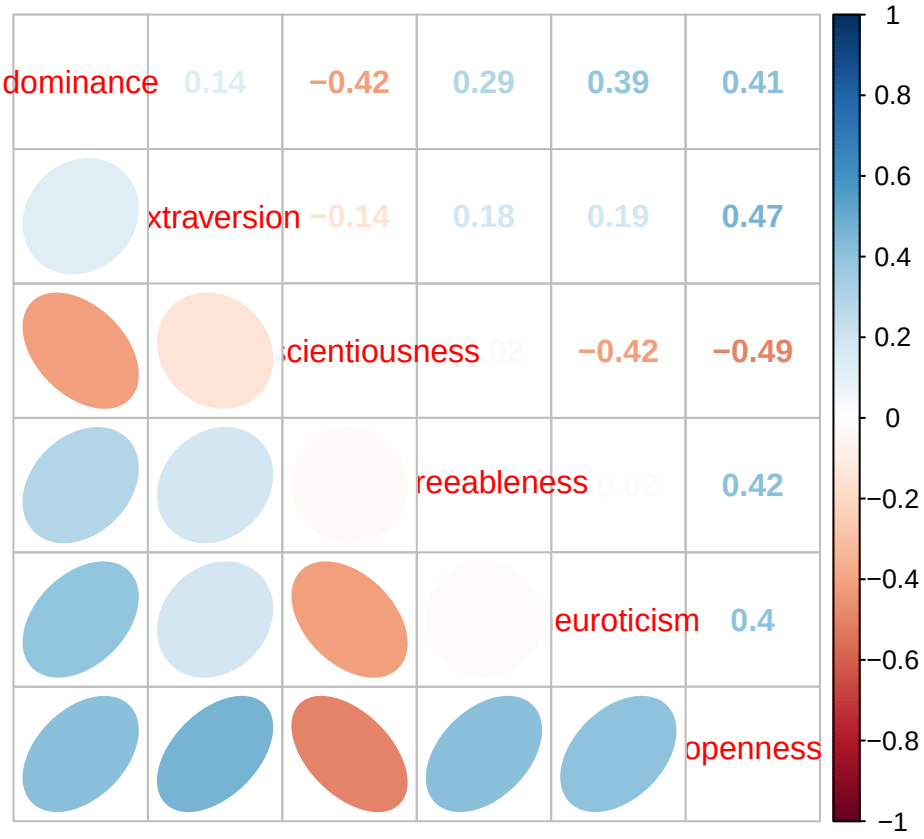

```
install.packages("corrplot")
```

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 3.5.3
```

```
## corrplot 0.84 loaded
```

```
corrplot.mixed(gombeCorMat, lower="ellipse", upper="number")
```


dominance
0.14
−0.42
0.29
0.39
0.41
xtraversion
−0.14
0.18
0.19
0.47
cientiousness
02
−0.42
−0.49
reeableness
0.02
0.42
euroticism
0.4
openness
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

(pdf / Rmd)