Пищовче по R

Entering data with c
The most useful R command for quickly entering in small data sets is the c function. This function combines, or concatenates terms together. As an example, suppose we have the following count of the number of typos per page of these notes: 2 3 0 3 1 0 0 1 To enter this into an R session we do so with
typos = c(2,3,0,3,1,0,0,1)
typos
[1] 2 3 0 3 1 0 0 1

```
> typos.draft1 = c(2,3,0,3,1,0,0,1)
> typos.draft2 = c(0,3,0,3,1,0,0,1)
```

That is, the two typos on the first page were fixed. Notice the two different variable names. Unlike many other languages, the period is only used as punctuation. You can't use an _ (underscore) to punctuate names as you might in other programming languages so it is quite useful. [1]

sum(vector)
mean(vector)
median(vector)
max(vector)
length(vector)

x[day : (day+4)]

What is the mean, the variance, the standard deviation? Again, R makes these easy to answer:

```
> whale = c(74, 122, 235, 111, 292, 111, 211, 133, 156, 79)
> mean(whale)
[1] 152.4
> var(whale)
[1] 5113.378
> std(whale)
Error: couldn't find function "std"
> sqrt(var(whale))
[1] 71.50789
> sqrt( sum( (whale - mean(whale))^2 /(length(whale)-1)))
[1] 71.50789
```

Well, almost! First, one needs to remember the names of the functions. In this case `mean` is easy to guess, `var` is kind of obvious but less so, `std` is also kind of obvious, but guess what? It isn't there! So some other things were tried. First, we remember that the standard deviation is the square of the variance. Finally, the last line illustrates that R can almost exactly mimic the mathematical formula for the standard deviation:

$$\mathsf{SD}(X) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}.$$

Notice the sum is now `sum`, $\bar{X}$ is `mean(whale)` and `length(x)` is used instead of n.

Of course, it might be nice to have this available as a built-in function. Since this example is so easy, lets see how it is done:

```
> std = function(x) sqrt(var(x))
> std(whale)
[1] 71.50789
```

(above is same as sd(whale))

R Basics: Accessing Data

There are several ways to extract data from a vector. Here is a summary using both slicing and extraction by a logical vector. Suppose `x` is the data vector, for example `x=1:10`.

how many elements?	`length(x)`
ith element	`x[2]` ($i = 2$)
all *but* ith element	`x[-2]` ($i = 2$)
first k elements	`x[1:5]` ($k = 5$)
last k elements	`x[(length(x)-5):length(x)]` ($k = 5$)
specific elements.	`x[c(1,3,5)]` (First, 3rd and 5th)
all greater than some value	`x[x>3]` (the value is 3)
bigger than or less than some values	`x[ x< -2 | x > 2]`
which indices are largest	`which(x == max(x))`

Using tables

The `table` command allows us to look at tables. Its simplest usage looks like `table(x)` where `x` is a categorical variable.

Example: Smoking survey

A survey asks people if they smoke or not. The data is

```
Yes, No, No, Yes, Yes
```

We can enter this into `R` with the `c()` command, and summarize with the `table` command as follows

```
> x=c("Yes","No","No","Yes","Yes")
> table(x)
```



```
x
 No Yes
  2   3
```

The `table` command simply adds up the frequency of each unique value of the data.

Factors

Categorical data is often used to classify data into various levels or factors. For example, the smoking data could be part of a broader survey on student health issues. `R` has a special *class* for working with factors which is occasionally important to know as `R` will automatically adapt itself when it knows it has a factor. To make a factor is easy with the command `factor` or `as.factor`. Notice the difference in how `R` treats factors with this example

```
> x=c("Yes","No","No","Yes","Yes")
> x                             # print out values in x
[1] "Yes" "No"  "No"  "Yes" "Yes"
> factor(x)                     # print out value in factor(x)
[1] Yes No  No  Yes Yes
Levels:  No Yes                 # notice levels are printed.
```

Bar charts

A bar chart draws a bar with a a height proportional to the count in the table. The height could be given by the frequency, or the proportion. The graph will look the same, but the scales may be different.

Suppose, a group of 25 people are surveyed as to their beer-drinking preference. The categories were (1) Domestic can, (2) Domestic bottle, (3) Microbrew and (4) import. The raw data is

```
3 4 1 1 3 4 3 3 1 3 2 1 2 1 2 3 2 3 1 1 1 1 4 3 1
```

Let's make a barplot of both frequencies and proportions. First, we use the `scan` function to read in the data then we plot (figure 1)

```
> beer = scan()
1: 3 4 1 1 3 4 3 3 1 3 2 1 2 1 2 3 2 3 1 1 1 1 4 3 1
26:
Read 25 items
> barplot(beer)                 # this isn't correct
> barplot(table(beer))          # Yes, call with summarized data
> barplot(table(beer)/length(beer)) # divide by n for proportion
```

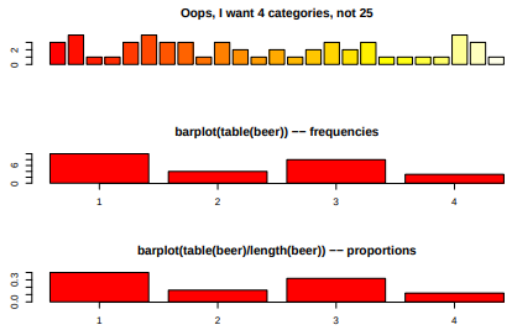

Figure 1: Sample barplots

The idea of a quantile generalizes this median. The p quantile, (also known as the 100p%-percentile) is the point in the data where 100p% is less, and 100(1-p)% is larger. If there are n data points, then the p quantile occurs at the position $1+(n-1)p$ with weighted averaging if this is between integers. For example the .25 quantile of the numbers 10,17,18,25,28,28 occurs at the position 1+(6-1)(.25) = 2.25. That is 1/4 of the way between the second and third number which in this example is 17.25.

The .25 and .75 quantiles are denoted the *quartiles*. The first quartile is called Q_1, and the third quartile is called Q_3. (You'd think the second quartile would be called Q_2, but use "the median" instead.) These values are in the R function
RCodesummary. More generally, there is a `quantile` function which will compute any quantile between 0 and 1. To find the quantiles mentioned above we can do

```
> data=c(10, 17, 18, 25, 28, 28)
> summary(data)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.00   17.25   21.50   21.00   27.25   28.00
> quantile(data,.25)
  25%
17.25
> quantile(data,c(.25,.75))     # two values of p at once
  25%   75%
17.25 27.25
```

Example: Making numeric data categorical

Categorical variables can come from numeric variables by aggregating values. For example. The salaries could be placed into broad categories of 0-1 million, 1-5 million and over 5 million. To do this using R one uses the `cut()` function and the `table()` function.

Suppose the salaries are again

```
12 .4 5 2 50 8 3 1 4 .25
```

And we want to break that data into the intervals

$$[0,1],(1,5],(5,50]$$

To use the cut command, we need to specify the cut points. In this case 0,1,5 and 50 (=`max(sals)`). Here is the syntax

```
> sals = c(12, .4, 5, 2, 50, 8, 3, 1, 4, .25) # enter data
> cats = cut(sals,breaks=c(0,1,5,max(sals)))  # specify the breaks
> cats                              # view the values
 [1] (5,50] (0,1]  (1,5]  (1,5]  (5,50] (5,50] (1,5]  (0,1]  (1,5]  (0,1]
Levels:  (0,1] (1,5] (5,50]
> table(cats) # organize
cats
 (0,1]  (1,5] (5,50]
     3      4      3
> levels(cats) = c("poor","rich","rolling in it") # change labels
> table(cats)
cats
         poor          rich rolling in it
            3             4             3
```

Bivariate data

We see that there may be some relationship[7]

What would be nice to have are the marginal totals and the proportions. For example, what proportion of smokers study 5 hours or less. We know that this is $3/(3+2+1) = 1/2$, but how can we do this in R?

The command `prop.table` will compute this for us. It needs to be told the table to work on, and a number to indicate if you want the row proportions (a 1) or the column proportions (a 2) the default is to just find proportions.

```
> tmp=table(smokes,amount)      # store the table
> old.digits = options("digits") # store the number of digits
> options(digits=3)             # only print 3 decimal places
> prop.table(tmp,1)             # the rows sum to 1 now
      amount
smokes   1     2     3
     N 0.0 0.500 0.500
     Y 0.5 0.333 0.167
> prop.table(tmp,2)             # the columns sum to 1 now
      amount
smokes 1   2     3
     N 0 0.5 0.667
     Y 1 0.5 0.333
> prop.table(tmp)
      amount                    # all the numbers sum to 1
smokes   1   2   3
     N 0.0 0.2 0.2
     Y 0.3 0.2 0.1
> options(digits=old.digits)    # restore the number of digits
```

Linear Regression

```
> data(home);attach(home)
> x = old                          # use generic variable names
> y = new                          # for illustration only.
> plot(x,y)
> abline(lm(y ~ x))
> detach(home)
```

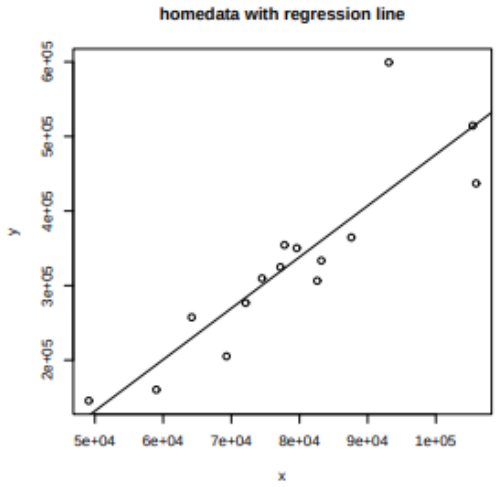

Figure 13: Home data with regression line

Correlation coefficients

A valuable numeric summary of the strength of the linear relationship is the Pearson correlation coefficient, R, defined by

$$R = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - X)^2 \sum(Y_i - Y)^2}}$$

This is a scaled version of the covariance between X and Y. This measures how one variable varies as the other does. The correlation is scaled to be in the range $[-1, 1]$. Values or R^2 close to 1 indicate a strong linear relationship, values close to 0 a weak one. (There still may be a relationship, just not a linear one.) In R the correlation coefficient is found with the `cor` function


```
> cor(x,y)                         # to find R
[1] 0.881
> cor(x,y)^2                       # to find R^2
[1] 0.776
```

Data frames

```
> weight = c(150, 135, 210, 140)
> height = c(65, 61, 70, 65)
> gender = c("Fe","Fe","M","Fe")
> study = data.frame(weight,height,gender) # make the data frame
> study
  weight height gender
```



```
1    150     65     Fe
2    135     61     Fe
3    210     70      M
4    140     65     Fe
```

Notice, the columns inherit the variable names. Different names are possible if desired. Try

```
> study = data.frame(w=weight,h=height,g=gender)
```

```
> study
  weight height gender
1    150     65     Fe
2    135     61     Fe
3    210     70      M
4    140     65     Fe
> rm(weight)                    # clean out an old copy
> weight
Error: Object "weight" not found
> attach(study)
> weight
[1] 150 135 210 140
```

To access as a list A list is a more general storage concept than a data frame. A list is a set of objects, each of which can be any other object. A data frame is a list, where the objects are the columns as vectors.

To access a list, one uses either a dollar sign, $, or double brackets and a number or name. So for our `study` variable we can access the weight (the first column) as a list all of these ways

```
> study$weight                  # using $
[1] 150 135 210 140
> study[['weight']]             # using the name.
> study[['w']]                  # unambiguous shortcuts are okay
> study[[1]]                    # by position
```

These two can be combined as in this example. To get just the females information. These are the rows where gender is 'Fe' so we can do this

```
> study[study$gender == 'Fe', ]  # use $ to access gender via a list
      weight height gender
Mary     150     65     Fe
Alice    135     61     Fe
Judy     140     65     Fe
```

In probability theory and statistics, the **chi-squared distribution** (also **chi-square** or **χ^2-distribution**) with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables.

Problem

Assuming that the data in quine follows the normal distribution, find the 95% confidence interval estimate of the difference between the female proportion of Aboriginal students and the female proportion of Non-Aboriginal students, each within their own ethnic group.

Solution

We apply the prop.test function to compute the difference in female proportions. The Yates's continuity correction is disabled for pedagogical reasons.

```
> prop.test(table(quine$Eth, quine$Sex), correct=FALSE)

        2-sample test for equality of proportions
        without continuity correction

data:  table(quine$Eth, quine$Sex)
X-squared = 0.0041, df = 1, p-value = 0.949
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.15642  0.16696
sample estimates:
 prop 1  prop 2
0.55072 0.54545
```

Answer

The 95% confidence interval estimate of the difference between the female proportion of Aboriginal students and the female proportion of Non-Aboriginal students is between -15.6% and 16.7%.