CAR: Chapter 1

Contents

This version rendered on January 19 2022 09:26
Based on scripts created for J. Fox and S. Weisberg (2019) An R Companion to

Applied Regression, 3rd Edition

Load the package written for this textbook:

```
library(car)
```

```
  Loading required package: carData
```

This script introduces a number of principles in R. Understanding these principles goes a long way towards helping you become a productive user of R. The name of each principle is shown in **bold**. You can search through The R Language Definition to pursue questions that arouse your curiosity.

Post questions and comments on Piazza.

1 Arithmetic and basic objects

Operators grouped in reverse order of **precedence**

```
2 + 3 # addition
```

```
[1] 5
```

```
2 - 3 # subtraction
```

```
[1] -1
```

```
-3    # unary minus (binary: two arguments, unary: one argument)
```

```
[1] -3
```

```
2*3   # multiplication
```

```
[1] 6
```

```
2/3   # division
```

```
[1] 0.6666667
```

```
2/0
```

```
[1] Inf
```

```
2/Inf
```

```
[1] 0
```

```
Inf - Inf
```

```
[1] NaN
```

```
2^3   # exponentiation
```

```
[1] 8
```

Examples

```
4^2 - 3*2
```

```
[1] 10
```

```
4 ^ 2-3 * 2  # are spaces important in arithmetic expressions?
```

```
[1] 10
```

```r
1 - 6 + 4  # with equal precedence operations
```

```
[1] -1
```

```r
           # are executed from left to right
1 + 4 - 6
```

```
[1] -1
```

```r
4^2 - 3*2
```

```
[1] 10
```

```r
(4^2) - (3*2)
```

```
[1] 10
```

```r
4^ (2 - 3)*2  # what happens here?
```

```
[1] 0.5
```

```
(4 + 3)^2
```

```
  [1] 49
```

```
4 + 3^2
```

```
  [1] 13
```

```
-2--3    # unary minus
```

```
  [1] 1
```

```
-2 - -3
```

```
  [1] 1
```

```
-2 - - 3
```

```
  [1] 1
```

```
-3*-2
```

```
  [1] 6
```

Which has higher precedence: exponentiation or unary minus?

Some systems, e.g. Excel, follow the opposite order of precedence

```
-3^2
```

```
  [1] -9
```

```
0-3^2
```

```
  [1] -9
```

```
(-3)^2
```

```
  [1] 9
```

Integer division

```
10 %/% 3
```

```
  [1] 3
```

Modulo arithmetic: remainder

```
10 %% 3
```

```
[1] 1
```

```
3 * (10 %/% 3) + 10 %% 3 # why?!
```

```
[1] 10
```

1.1 Extended arithmetic

```
1/0
```

```
[1] Inf
```

```
Inf - 1
```

```
[1] Inf
```

```
Inf + Inf
```

```
[1] Inf
```

```
Inf * 2
```

```
  [1] Inf
```

```
Inf - Inf
```

```
  [1] NaN
```

```
0/0
```

```
  [1] NaN
```

1.2 Complex numbers too

```
0i
```

```
  [1] 0+0i
```

```
1i
```

```
  [1] 0+1i
```

Euler's identity

```
exp(1)
```

```
[1] 2.718282
```

```
exp(1)^(1i*pi)
```

```
[1] -1+0i
```

```
exp(1)^(1i*pi) + 1  # complex machine 0
```

```
[1] 0+1.224647e-16i
```

1.3 How numbers are printed

```
1.23456789
```

```
[1] 1.234568
```

```
1.23456789 * 10^8
```

```
[1] 123456789
```

```
1.23456789e8  # same number in scientific notation
```

```
[1] 123456789
```

```
1.23456789 * 10^10
```

```
[1] 12345678900
```

```
1.23456789 * 10^12
```

```
[1] 1.234568e+12
```

```
1.23456789 * 10^-4
```

```
[1] 0.0001234568
```

```
1.23456789 * 10^-5
```

```
[1] 1.234568e-05
```

```
1.23456789 * 10^-10
```

```
[1] 1.234568e-10
```

1.4 Example of using options and restoring prior state

```
opts <- options(scipen = 15)  # penalty 'against' scientific notati
```

```
1.23456789
```

```
[1] 1.234568
```

```
1.23456789 * 10^8
```

```
[1] 123456789
```

```
1.23456789 * 10^10
```

```
[1] 12345678900
```

```
1.23456789 * 10^12
```

```
  [1] 1234567890000
```

```
1.23456789 * 10^-4
```

```
  [1] 0.0001234568
```

```
1.23456789 * 10^-5
```

```
  [1] 0.00001234568
```

```
1.23456789 * 10^-10
```

```
  [1] 0.0000000001234568
```

```
opts
```

```
  $scipen
  [1] 0
```

```
options(opts)  # restoring previous state
```

To see more options in base R use `?options`

1.5 Printing with `format`

```
format(1.23456789 * 10^10)
```

```
  [1] "12345678900"
```

```
format(1.23456789 * 10^10, big.mark = ',')
```

```
  [1] "12,345,678,900"
```

```
format(123456789.1234, big.mark = ',')
```

```
  [1] "123,456,789"
```

```
format(123456789.1234, big.mark = ',', nsmall = 2)
```

```
  [1] "123,456,789.12"
```

adding a dollar sign:

```
paste0('$',format(123456789.1234, big.mark = ',', nsmall = 2))
```

```
[1] "$123,456,789.12"
```

For more possibilities, see `?format` and `?prettyNum`. For people who know C, see `?formatC`

1.6 typeof

Every object in R has a 'type' that identifies its internal representation. You can find the type with the 'typeof' function.

Let's see the types of things we've seen so far:

```
typeof(1:4)
```

```
[1] "integer"
```

```
typeof(4)   # looks like an integer
```

```
[1] "double"
```

```r
typeof(4L)   # In 4L, L stands for L i.e. a 64 bit integer
```

```
[1] "integer"
```

```r
typeof(4L + 3L)
```

```
[1] "integer"
```

```r
typeof(4L + 3)
```

```
[1] "double"
```

```r
typeof(Inf)
```

```
[1] "double"
```

```r
typeof(options) # fancy term for a function with baggage
```

```
[1] "closure"
```

```r
typeof(1.23456789)
```

```
  [1] "double"
```

```
typeof(paste0)
```

```
  [1] "closure"
```

```
typeof('$')
```

```
  [1] "character"
```

```
typeof(typeof)
```

```
  [1] "closure"
```

```
typeof(opts)
```

```
  [1] "list"
```

2 Calling Functions

```
log(100)
```

```
[1] 4.60517
```

```
log(100, base = 10) # why is this different?
```

```
[1] 2
```

For the 'log' function, the second argument is optional because it has a default value.

What do you think the default value is?

```
log10(100) # equivalent
```

```
[1] 2
```

Functions have arguments.

```r
# the possible arguments and their names and defaults, if any
args("log") # the possible arguments and their
```

```
function (x, base = exp(1))
NULL
```

```r
            # names and defaults, if any
args("+") # operators are functions too
```

```
function (e1, e2)
NULL
```

```r
args("format")
```

```
function (x, ...)
NULL
```

```r
args("args")
```

```
function (name)
NULL
```

```
args("library")
```

```
function (package, help, pos = 2, lib.loc = NULL, character.only =
    logical.return = FALSE, warn.conflicts, quietly = FALSE,
    verbose = getOption("verbose"), mask.ok, exclude, include.only
    attach.required = missing(include.only))
NULL
```

- In a language like C, you need to supply every argument when you call a function. Thank goodness you don't need to do that in R.
- Arguments can be supplied by position and/or by name
- Names can be abbreviated as far as possible to avoid ambiguity

```
log(100, b=10) # 'b' will do. Why?
```

```
[1] 2
```

An argument can be optional even if it doesn't have a default value provided that value of the argument is not used as the function is evaluated.

```
log(100, 10) # arguments can be supplied by name or by position
```

```
[1] 2
```

3 Vectors and Variables

```
c(1, 2, 3, 4, NA, 6)  # c: combine or catenate or concatenate
```

```
[1]  1  2  3  4 NA  6
```

```
1:4 # integer sequence
```

```
[1] 1 2 3 4
```

```
4:1 # descending
```

```
[1] 4 3 2 1
```

```
-1:2 # negative to positive
```

```
  [1] -1  0  1  2
```

```
0-1:2 # why is this different? Could you explain why on a test???
```

```
  [1] -1 -2
```

```
seq(1, 4) # equivalent to 1:4
```

```
  [1] 1 2 3 4
```

```
seq(2, 8, by = 2) # specify interval between elements
```

```
  [1] 2 4 6 8
```

```
seq(0, 1, by = 0.1) # noninteger sequence
```

```
   [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
seq(0, 1, length = 11) # specify number of elements
```

```
   [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```r
rep(2, times = 10)  # 'rep' for 'repeat'
```

```
 [1] 2 2 2 2 2 2 2 2 2 2
```

```r
rep(c('a','b'), times= 5) # recycling repeat
```

```
 [1] "a" "b" "a" "b" "a" "b" "a" "b" "a" "b"
```

```r
rep(c('a','b'), length.out = 5)
```

```
[1] "a" "b" "a" "b" "a"
```

```r
rep(c('a','b'), each = 5)  # not recycling, each repeated 5 times
```

```
 [1] "a" "a" "a" "a" "a" "b" "b" "b" "b" "b"
```

```r
rep(c('a','b'), times = c(5,2)) # like 'each' different times
```

```
[1] "a" "a" "a" "a" "a" "b" "b"
```

```r
rep(c('a','b'), c(5,2)) # what's the default second argument?
```

```
[1] "a" "a" "a" "a" "a" "b" "b"
```

3.1 Operations on vectors

```
3 * c(1, 2, 3, 4)  # scalar multiplication as in linear algebra
```

```
[1]  3  6  9 12
```

```
c(1, 2, 3, 4) / 2
```

```
[1] 0.5 1.0 1.5 2.0
```

Dividing a vector by a vector: Is this like linear algebra?

```
c(1, 2, 3, 4) / c(4, 3, 2, 1)
```

```
[1] 0.2500000 0.6666667 1.5000000 4.0000000
```

What happened? **element-wise operations**. Many operations are applied element by element.

Applying a function to a vector:

```
log(c(0, 0.1, 1, 10, 100), base=10)
```

```
  [1] -Inf   -1    0    1    2
```

Many functions in R are **vectorized**. Sometimes for more than one argument:

```
log(c(2,10,100), base = c(2,10,100))
```

```
  [1] 1 1 1
```

What happens if you do something that doesn't make sense, like adding a long vector to a short vector:

```
c(1, 2, 3, 4) + c(4, 3) # no warning! What happens?
```

```
  [1] 5 5 7 7
```

R usually assumes that you must know what you are doing and tries to do something reasonable. Here R applied what's known as the **recycling** principle: recycle the shorter vector as many times as necessary to provide value to match the longer vector.

However, if the length of the longer vector is not a multiple of the length of the shorter vector, R starts to worry about you and gives you a warning:

```
c(1, 2, 3, 4) + c(4, 3, 2) # R thinks maybe you made a mistake
```

```
Warning in c(1, 2, 3, 4) + c(4, 3, 2): longer object length
is not a multiple of shorter object length
```

```
[1] 5 5 5 8
```

3.2 Creating and naming objects

```
x <- c(1, 2, 3, 4) # assignment -- does not print result
x # prints value of x
```

```
[1] 1 2 3 4
```

```
(x <- c(1, 2, 3, 4)) # assigns and prints at the same time
```

```
[1] 1 2 3 4
```

Why not = instead of <-? (pronouced 'gets').

We used = to assign arguments to parameters when calling functions.

Actually, = works ... most of the time. But stick with <-. Your fingers will get used to it.

R uses three symbols for three distinct operations that are all represented with = in most other languages. The third, which we will see soon, is logical = which is denoted == in R.

I still regularly make the mistake of typing = when I should have typed ==.

```
x/2  # equivalent to c(1, 2, 3, 4)/2
```

```
[1] 0.5 1.0 1.5 2.0
```

```
(y <- sqrt(x))  # assign AND print
```

```
[1] 1.000000 1.414214 1.732051 2.000000
```

3.3 Random number generation

```
set.seed(372291)  # for reproducibility (chosen randomly)
(x <- rnorm(100))  # 100 standard normal random numbers
```

```
  [1]  0.09833263 -0.89632065 -0.80499585 -0.23440421
  [5] -0.57098627 -0.12431051 -0.35938865  1.19509408
  [9] -0.51327643 -1.16888069  2.04501263  0.96774659
 [13] -0.33892531 -0.02385635 -0.13606928 -0.26846724
 [17]  0.06233524  0.86995856  1.13066988 -0.34642095
 [21]  0.35064032  1.11546736 -0.51577251  0.12023891
 [25]  0.26794257  0.81975768 -0.05561499  0.78947930
 [29]  0.24476899 -0.23868740  0.72776887  0.24950307
 [33]  0.29120391 -0.98124553  0.04353949 -0.85734282
 [37]  2.63618842 -1.25254955  0.83030163 -1.95090213
 [41] -2.68712959 -0.36168710 -0.58443713  0.81399991
 [45] -0.02585634 -1.11835697  1.11836461 -1.29533607
 [49]  0.19998422  0.51477512  0.53101946 -0.03766582
 [53]  0.98327288 -1.57294475  1.51531106  1.04962419
```

```
 [57] -0.88333128  2.13595738  1.44940962 -0.84459977
 [61] -1.80824601 -0.09673322 -1.00007308  1.63933460
 [65] -0.52798368 -0.24537308 -0.31443488  1.15314622
 [69] -0.91751726  1.26672906 -1.59600134  0.16465603
 [73]  0.85672451  0.82863616 -0.42284364  0.79206310
 [77] -0.02569929 -0.19991971  1.51607308  0.50764612
 [81]  1.28908616  1.03368892 -0.55391013 -1.29783452
 [85]  1.56289281  0.51518221  0.69617559 -0.86324452
 [89] -0.43990620  1.11514478 -1.40240347  0.10543920
 [93]  0.12556730  1.88108615  1.45922041 -0.33864564
 [97]  0.61101545 -0.23174691 -0.03237697  1.44987907
```

Many distributions are in base R: e.g. 'norm', 'exp', 'poisson', student', 't', 'cauchy', 'f', etc.

The 'extraDistr' package and others have many more.

To generate random numbers from a distribution, prepend the name with and **r**:

```
rf(10, df1 = 2, df2 = 10)
```

```
 [1] 1.31169785 0.07542327 0.44086623 4.30693205 9.62530417
 [6] 1.31581038 0.09651469 0.05892881 0.63146620 0.29580129
```

To get the density for a continuous distribution) or the probability for a discrete distribution, prepend the name with a **d** for density. Note that the probability in the case of a discrete distribution is indeed a density with respect to counting measure.

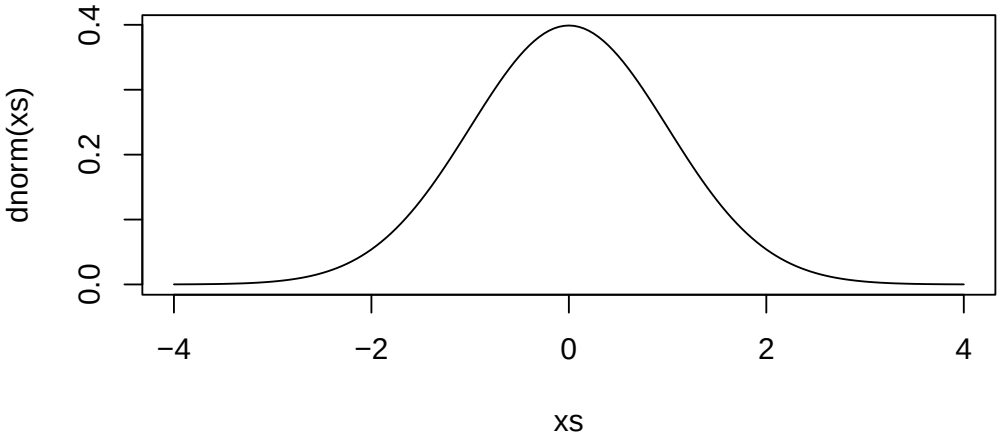
```
dnorm(0)
```

```
[1] 0.3989423
```

```
dnorm(seq(-3,3,1))
```

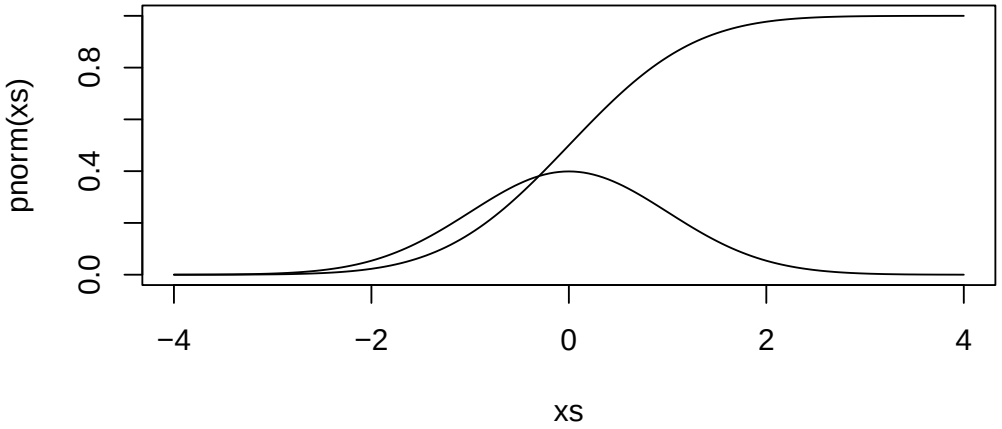
```
[1] 0.004431848 0.053990967 0.241970725 0.398942280
[5] 0.241970725 0.053990967 0.004431848
```

```
xs <- seq(-4, 4, by = .05)
plot(xs, dnorm(xs), type = 'l')
```



Prepend with a **p** for the cumulative distribution function

```
plot(xs, pnorm(xs), type = 'l')
lines(xs, dnorm(xs), type = 'l')
```



To get a quantile from a probability, the inverse CDF, prepend the distribution name with a **q**

Note the ubiquitous summary function:

```
summary(x)
```

```
      Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
-2.68713 -0.51390  0.05294  0.12372  0.83691  2.63619 
```

We will see much more about it when we take up OOP.

3.4 Character objects

```
(words <- c("To", "be", "or", "not", "to", "be"))
```

```
[1] "To"  "be"  "or"  "not" "to"  "be" 
```

```
typeof(words)
```

```
[1] "character"
```

```r
paste(words, collapse=" ")
```

```
[1] "To be or not to be"
```

```r
paste(words)
```

```
[1] "To"  "be"  "or"  "not" "to"  "be"
```

```r
paste(words, words)
```

```
[1] "To To"   "be be"   "or or"   "not not" "to to"
[6] "be be"
```

```r
paste('Variable', 1:10)
```

```
 [1] "Variable 1"  "Variable 2"  "Variable 3"  "Variable 4"
 [5] "Variable 5"  "Variable 6"  "Variable 7"  "Variable 8"
 [9] "Variable 9"  "Variable 10"
```

```r
paste('Variable', 1:10, sep = '_')
```

```
 [1] "Variable_1"  "Variable_2"  "Variable_3"  "Variable_4"
 [5] "Variable_5"  "Variable_6"  "Variable_7"  "Variable_8"
 [9] "Variable_9"  "Variable_10"
```

```
paste('Var', 1:10, sep = '')
```

```
 [1] "Var1"  "Var2"  "Var3"  "Var4"  "Var5"  "Var6"  "Var7"
 [8] "Var8"  "Var9"  "Var10"
```

```
paste0('Var', 1:10)   # same as sep = ''
```

```
 [1] "Var1"  "Var2"  "Var3"  "Var4"  "Var5"  "Var6"  "Var7"
 [8] "Var8"  "Var9"  "Var10"
```

Take a column of $ amounts and format accordingly

```
(amts <- c(123.45,123456.78,12345678.912))
```

```
 [1]       123.45    123456.78 12345678.91
```

```
paste0('$', format(amts, big.mark = ',', nsmall = 2))
```

```
  [1] "$       123.45" "$   123,456.78" "$12,345,678.91"
```

OOPS: Look up help on format with '?format' and discover the 'trim' argument.

```
paste0('$', format(amts, big.mark = ',', nsmall = 2, trim = TRUE))
```

```
  [1] "$123.45"        "$123,456.78"    "$12,345,678.91"
```

That's better!

3.5 Logical values

```
(logical.values <- c(TRUE, TRUE, FALSE, TRUE))
```

```
  [1]  TRUE  TRUE FALSE  TRUE
```

```
typeof(logical.values)
```

```
  [1] "logical"
```

Actually R uses '3-valued' logic

```
(logical.values <- c(TRUE, TRUE, FALSE, NA, TRUE))
```

```
[1]  TRUE  TRUE FALSE    NA  TRUE
```

```
!logical.values # unary not
```

```
[1] FALSE FALSE  TRUE    NA FALSE
```

```
logical.values | FALSE  # binary or -- note that FALSE gets recycle
```

```
[1]  TRUE  TRUE FALSE    NA  TRUE
```

```
logical.values & FALSE  # binary and
```

```
[1] FALSE FALSE FALSE FALSE FALSE
```

```
logical.values | TRUE  # binary or -- note that FALSE gets recycled
```

```
[1] TRUE TRUE TRUE TRUE TRUE
```

```
logical.values & TRUE  # binary and
```

```
  [1]  TRUE  TRUE FALSE    NA  TRUE
```

```
logical.values == FALSE  # equals
```

```
  [1] FALSE FALSE  TRUE    NA FALSE
```

```
logical.values != FALSE  # not equals
```

```
  [1]  TRUE  TRUE FALSE    NA  TRUE
```

```
logical.values == TRUE  # equals
```

```
  [1]  TRUE  TRUE FALSE    NA  TRUE
```

```
logical.values != TRUE  # not equals
```

```
  [1] FALSE FALSE  TRUE    NA FALSE
```

Did the 'NA" always result in 'NA' in the corresponding position? If not, why not? Could you explain why on a test?

3.6 Truth tables

```
y <- c(TRUE, FALSE, NA)
y
```

```
  [1]  TRUE FALSE    NA
```

```
names(y)   # the names attribute of x is NULL
```

```
  NULL
```

```
names(y) <- as.character(y) # use 'names' replacement function
                            # to give y names
y
```

```
   TRUE FALSE  <NA>
   TRUE FALSE    NA
```

Truth tables

```
outer(y, y, '|')   # note the value of "TRUE | NA"
```

```
       TRUE FALSE <NA>
  TRUE  TRUE  TRUE TRUE
  FALSE TRUE FALSE   NA
  <NA>  TRUE    NA   NA
```

```
outer(y, y, '&')
```

```
        TRUE FALSE  <NA>
  TRUE   TRUE FALSE    NA
  FALSE FALSE FALSE FALSE
  <NA>     NA FALSE    NA
```

```
outer(y, y, '==')
```

```
        TRUE FALSE <NA>
  TRUE   TRUE FALSE   NA
  FALSE FALSE  TRUE   NA
  <NA>     NA    NA   NA
```

```
outer(y, y, '!=')
```

```
        TRUE FALSE <NA>
  TRUE  FALSE  TRUE   NA
  FALSE  TRUE FALSE   NA
  <NA>     NA    NA   NA
```

Example of logical expressions

```
1 == 2
```

```
  [1] FALSE
```

```
1 != 2
```

```
  [1] TRUE
```

```
1 <= 2
```

```
  [1] TRUE
```

```
1 < 1:3   # recycling
```

```
  [1] FALSE  TRUE  TRUE
```

```
3:1 > 1:3
```

```
  [1]  TRUE FALSE FALSE
```

```
3:1 >= 1:3
```

```
  [1]  TRUE  TRUE FALSE
```

```
TRUE & c(TRUE, FALSE)     # logical AND
```

```
  [1]  TRUE FALSE
```

```
c(TRUE, FALSE, FALSE) | c(TRUE, TRUE, FALSE)  # logical OR
```

```
  [1]  TRUE  TRUE FALSE
```

Logical vectors used in 'ifelse' statement

```r
(z <- x[1:10]) # first 10 elements of x
```

```
 [1]  0.09833263 -0.89632065 -0.80499585 -0.23440421
 [5] -0.57098627 -0.12431051 -0.35938865  1.19509408
 [9] -0.51327643 -1.16888069
```

```r
z < -0.5 # is each element less than -0.5?
```

```
 [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE
[10]  TRUE
```

```r
z > 0.5  # is each element greater than 0.5
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
[10] FALSE
```

```r
z < -0.5 | z > 0.5  #  < and > are of higher precedence than |
```

```
 [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
[10]  TRUE
```

```
abs(z) > 0.5  # absolute value, equivalent to last expression
```

```
 [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
[10]  TRUE
```

The 'ifelse' function has three arguments:

- all three are vectors
- the first is a logical vector
- the second gives values for positions where the first is true
- the third gives values where the first is false

```
ifelse( abs(z) > 0.5, z, 0) # note that 0 get recycled
```

```
 [1]  0.0000000 -0.8963206 -0.8049958  0.0000000 -0.5709863
 [6]  0.0000000  0.0000000  1.1950941 -0.5132764 -1.1688807
```

We can also use a logical vector to **select** elements of a vector:

```
abs(z) > 0.5 # which values of z satisfy |z| > 0.5
```

```
 [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
[10]  TRUE
```

```
z
```

```
 [1]  0.09833263 -0.89632065 -0.80499585 -0.23440421
 [5] -0.57098627 -0.12431051 -0.35938865  1.19509408
 [9] -0.51327643 -1.16888069
```

```
z[abs(z) > 0.5] # select values of z for which |z| > 0.5
```

```
[1] -0.8963206 -0.8049958 -0.5709863  1.1950941 -0.5132764
[6] -1.1688807
```

```
z[!(abs(z) > 0.5)] # values z for which |z| <= 0.5
```

```
[1]  0.09833263 -0.23440421 -0.12431051 -0.35938865
```

3.7 Special logical operators && and ||

- Work only on single expressions, not vectors

- Only evaluate what they need to determine the result

```
TRUE && FALSE   # works only on single expressions, not vectors
```

```
[1] FALSE
```

```
TRUE || FALSE   # only evaluates what it needs to determine result
```

```
[1] TRUE
```

QUESTION: Explain what happens here

```
log(-1)
```

```
Warning in log(-1): NaNs produced
```

```
[1] NaN
```

```
TRUE || log(-1)
```

```
[1] TRUE
```

```
FALSE || log(-1)
```

```
Warning in log(-1): NaNs produced
```

```
[1] NA
```

3.8 How coercion works in R

What happens when you try to do something that isn't quite right with an object?

```
sum(c(T,F,F,T)) # logical gets coerced to numeric
```

```
[1] 2
```

TRUE becomes 1 and FALSE becomes 0

```
sum(!c(T,F,F,T))
```

```
[1] 2
```

This is very useful if you want the count the number of elements of a vector that

satisfy a condition. For example, how many numbers in 'x' are greater than 1.96 in absolute value:

```
sum(abs(x) > 1.96)
```

```
[1] 4
```

... and what proportion?

```
sum(abs(x) > 1.96) / length(x)
```

```
[1] 0.04
```

What happens when you catenate things that are of different types? Vectors can only contain elements of the same type.
We'll soon see how to use a **list** that can contain elements of different types.

```
c("A", FALSE, 3.0)
```

```
[1] "A"     "FALSE" "3"
```

```
c(10, FALSE, -6.5, TRUE)
```

```
[1] 10.0  0.0 -6.5  1.0
```

3.9 The hierarchy of atomic types: promotion and demotion

- logical (lowest)
- numeric:
  - integer
  - double
  - complex
- character (highest)

All the elements of a vector must be of the same type so if you try to mix types in a vector, the 'lower' types get **coerced** (i.e. **promoted**) to the 'higher' types

```
c(TRUE, 1, 'one') # all get coerced to character
```

```
[1] "TRUE" "1"    "one"
```

```
c(TRUE,1i)        # all get coerced to complex
```

```
[1] 1+0i 0+1i
```

```
c(TRUE,1i, 'one') # what should happen here?
```

```
[1] "TRUE" "0+1i" "one"
```

QUESTION: Try to guess the difference between the result evaluated from the following 2 lines. Note that the innermost expressions need to be evaluated first.

```
c(c(TRUE, 2), 'dog')
c(TRUE, c(2, 'dog'))
```

3.10 Explicit coercion

```
as.logical(2)
```

```
[1] TRUE
```

```
as.character(2)
```

```
[1] "2"
```

```
as.integer(2.5)
```

```
[1] 2
```

```
as.integer(2.9)   # truncation, not rounding
```

```
[1] 2
```

```
round(2.5)   # What's happening?
```

```
[1] 2
```

```
round(3.5)
```

```
[1] 4
```

```
typeof(round(2.5))
```

```
  [1] "double"
```

```
as.complex(2)
```

```
  [1] 2+0i
```

```
as.numeric('2')
```

```
  [1] 2
```

```
as.numeric('two')
```

```
  Warning: NAs introduced by coercion
  [1] NA
```

3.11 Implicit coercion

```
4 + TRUE
```

```
  [1] 5
```

'TRUE' got **promoted** to numeric

```
4 & FALSE
```

```
[1] FALSE
```

4 got **demoted** to logical

```
1 == TRUE # does 1 get demoted, or TRUE get promoted?
```

```
[1] TRUE
```

4 Selecting elements of a vector

4.1 Selecting by position with positive numbers

```
x[12]              # 12th element
```

```
[1] 0.9677466
```

```
words[2]          # second element
```

```
[1] "be"
```

```
logical.values[3] # third element
```

```
[1] FALSE
```

```
x[6:15]       # elements 6 through 15
```

```
 [1] -0.12431051 -0.35938865  1.19509408 -0.51327643
 [5] -1.16888069  2.04501263  0.96774659 -0.33892531
 [9] -0.02385635 -0.13606928
```

```
x[c(1, 3, 5)] # 1st, 3rd, 5th elements
```

```
[1]  0.09833263 -0.80499585 -0.57098627
```

QUESTION: What happens if you go too far?

```
words[10]
```

```
[1] NA
```

```
x[100000]
```

```
[1] NA
```

```
logical.values[9]
```

```
[1] NA
```

Although all these NA's look the same, they are subtly different

4.2 Selecting by omission of positions with negative numbers

```
x[-(11:100)] # omit elements 11 through 100
```

```
 [1]  0.09833263 -0.89632065 -0.80499585 -0.23440421
 [5] -0.57098627 -0.12431051 -0.35938865  1.19509408
 [9] -0.51327643 -1.16888069
```

```
letters  # this object comes with R
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
letters[-c(1,5,9,15,21)] # what have we done?
```

```
 [1] "b" "c" "d" "f" "g" "h" "j" "k" "l" "m" "n" "p" "q" "r"
[15] "s" "t" "v" "w" "x" "y" "z"
```

4.3 Selecting with a logical vector

```
v <- 1:4
v[c(TRUE, FALSE, FALSE, TRUE)]
```

```
[1] 1 4
```

```
vowels <- c('a','e','i','o','u')
letters %in% vowels  # a very useful operator
```

```
 [1]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
[10] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
[19] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
letters[!(letters %in% vowels)]
```

```
 [1] "b" "c" "d" "f" "g" "h" "j" "k" "l" "m" "n" "p" "q" "r"
[15] "s" "t" "v" "w" "x" "y" "z"
```

```
`[`(v,c(T,F,T,F))   # Everything that happens is a function!
```

```
[1] 1 3
```

```
`[`(v,c(T,F))   # What's happening here??
```

```
[1] 1 3
```

```
v[c(T,F)]
```

```
[1] 1 3
```

4.4 Selecting with names

Vectors can get named:

```
(age <- c(10, 9, 15, 16))
```

```
[1] 10  9 15 16
```

```
names(age) <- c('Paula','Imran','Angela','Jiwon') # replacement fn
age
```

```
  Paula  Imran Angela  Jiwon
     10      9     15     16
```

Selecting by name

```
age['Angela']
```

```
Angela
    15
```

```
age['George']
```

```
<NA>
  NA
```

```
age[c('Jiwon','Paula')]
```

```
Jiwon Paula
   16    10
```

```
age[c(NA,'Paula')]
```

```
 <NA> Paula
   NA    10
```

```
names(age)
```

```
[1] "Paula"  "Imran"  "Angela" "Jiwon"
```

```
names(age) %in% c('Bob', 'Paula', 'Imran')
```

```
  [1]  TRUE  TRUE FALSE FALSE
```

```
age[ names(age) %in% c('Bob', 'Paula', 'Imran') ]
```

```
  Paula Imran
     10     9
```

```
age[ !names(age) %in% c('Bob', 'Paula', 'Imran') ]
```

```
  Angela  Jiwon
      15     16
```

Example of a **regular expression**

'^A' matches a capital A at the beginning of a string

'grepl' stands for global regular expression print logical

```
grepl('^A', names(age))
```

```
  [1] FALSE FALSE  TRUE FALSE
```

```
age[ grepl('^A', names(age)) ]
```

```
  Angela
      15
```

```
sort(names(age))
```

```
  [1] "Angela" "Imran"  "Jiwon"  "Paula"
```

```
age[ sort(names(age)) ]
```

```
  Angela  Imran  Jiwon  Paula
      15      9     16     10
```

4.5 Selecting elements of a matrix

A **matrix** is like a vector but with two dimensions. If the dimension is higher, it's called an **array**.

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
mat # notice that it got filled column by column
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

A matrix can have rownames and columns names

```
rownames(mat) <- c('Algebra','Analysis','Geometry')
colnames(mat) <- c('2013','2014','2015','2016')
mat
```

```
         2013 2014 2015 2016
Algebra     1    4    7   10
Analysis    2    5    8   11
Geometry    3    6    9   12
```

```
dim(mat)
```

```
[1] 3 4
```

```
nrow(mat)
```

```
[1] 3
```

```
ncol(mat)
```

```
[1] 4
```

```
sum(mat)
```

```
[1] 78
```

Selecting elements is much like vector except that we have two dimensions. Look at the following carefully to see what's happening. Notice what happens if a dimension is blank.

```
mat[2,4]
```

```
  [1] 11
```

```
mat[2,5]  # different from vectors
```

```
  Error in mat[2, 5]: subscript out of bounds
```

```
mat[c(2,3),]  # blank means: take everything
```

```
          2013 2014 2015 2016
  Analysis    2    5    8   11
  Geometry    3    6    9   12
```

```
mat[c(2,3), 1:4]  # blank means: take everything
```

```
          2013 2014 2015 2016
  Analysis    2    5    8   11
  Geometry    3    6    9   12
```

```
mat[c(2,3),3] # Oops! a dimension got dropped
```

```
  Analysis Geometry
         8        9
```

Dropping a dimension might seem like no big deal but it's one of the things that early designers regret adopting as a default. If your matrix is buried in a function and it drops a dimension that might spell hidden trouble when the function tries to multiply it by another matrix.

```
mat[c(2,3), 3, drop = FALSE]
```

```
           2015
  Analysis    8
  Geometry    9
```

Using 'drop' is the safe way within functions where you don't know whether the selecting vector might have length 1 or 0 when the function is called.

```
mat[c('Algebra','Geometry'),c(2014,2015)] # Need characters
```

```
  Error in mat[c("Algebra", "Geometry"), c(2014, 2015)]: subscript o
```

```
mat[c('Algebra','Geometry'),c('2014','2015')] # Good
```

```
           2014 2015
```

```
  Algebra     4    7
  Geometry    6    9
```

```
mat[c('Algebra','Geometry'),c(NA,'2014','2015')] # NA tilts
```

```
  Error in mat[c("Algebra", "Geometry"), c(NA, "2014", "2015")]: sub
```

```
mat[c('Algebra','Geometry'),-3]
```

```
           2013 2014 2016
  Algebra     1    4   10
  Geometry    3    6   12
```

```
mat[c('Algebra','Geometry'),-(1:4)] # a columnless matrix
```

```
  Algebra
  Geometry
```

```
mat[-(1:3),]
```

```
       2013 2014 2015 2016
```

```r
mat[-(1:3),-(1:4)]
```

```
<0 x 0 matrix>
```

5 Operators are functions

Chambers' dictum (main early creator of S at Bell Labs)

- **Everything that exists is an object**
- **Everything that happens is a function call**

```r
1 + 2 # `+` is a binary operator
```

```
[1] 3
```

```r
`+`(1,2)  # but really a function with two arguments
```

```
[1] 3
```

Note: to call an object with a weird name, just put its names in backticks.

6 Valid object names in R

What names are valid?

You can use:

- letters, case-sensitive
- numbers
- underline
- period
- must start with a letter or period but not a number or underline
- the first non-period character must not be a number

QUESTION: Which of the following assignments use valid names?

1. `a_very_long_name <- 0`
2. `_tmp <- 2`
3. `.tmp <- 2`
4. `..val <- 3`
5. `.2regression <- TRUE`
6. `._2_val <- 'a'`

7 Writing your own functions

It's easy in R to write your own functions and to build up a toolbox over time. You can document your functions in a package and share it with other users.

```
mean(x)
```

```
[1] 0.123724
```

```
sum(x)/length(x)
```

```
[1] 0.123724
```

Here's a very simple function that duplicates an existing one:

```
myMean <- function(x){
    sum(x)/length(x)
}
myMean(x)
```

```
[1] 0.123724
```

```r
y # defined earlier as sqrt(c(1, 2, 3, 4))
```

```
   TRUE FALSE  <NA>
   TRUE FALSE    NA
```

```r
myMean(y)
```

```
  [1] NA
```

```r
myMean(1:100)
```

```
  [1] 50.5
```

```r
myMean(sqrt(1:100))
```

```
  [1] 6.714629
```

```r
mySD <- function(x){
    sqrt(sum((x - myMean(x))^2)/(length(x) - 1))
}
mySD(1:100)
```

```
  [1] 29.01149
```

```
sd(1:100) # check
```

```
  [1] 29.01149
```

```
typeof(mySD)
```

```
  [1] "closure"
```

Functions are 'first-class' objects in R

```
mySD
```

```
  function(x){
      sqrt(sum((x - myMean(x))^2)/(length(x) - 1))
  }
```

```
myMean
```

```
  function(x){
      sum(x)/length(x)
```

```
}
<bytecode: 0x55a57e9ab910>
```

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
mySD(letters)
```

```
Error in sum(x): invalid 'type' (character) of argument
```

8 Quick overview of basic regression and plots in R

'Duncan' is a data frame (the term used in R to refer to the kind of object used to contain a data set) in the 'car' package. Since we loaded the 'car' package (with 'library(car)') we can use 'Duncan' typing its name.

```
head(Duncan, n=10) # U.S. data set from the 50s, first 10 lines
```

```
             type income education prestige
  accountant prof     62        86       82
  pilot      prof     72        76       83
  architect  prof     75        92       90
  author     prof     55        90       76
  chemist    prof     64        86       90
  minister   prof     21        84       87
  professor  prof     64        93       93
  dentist    prof     80       100       90
  reporter     wc     67        87       52
  engineer   prof     72        86       88
```

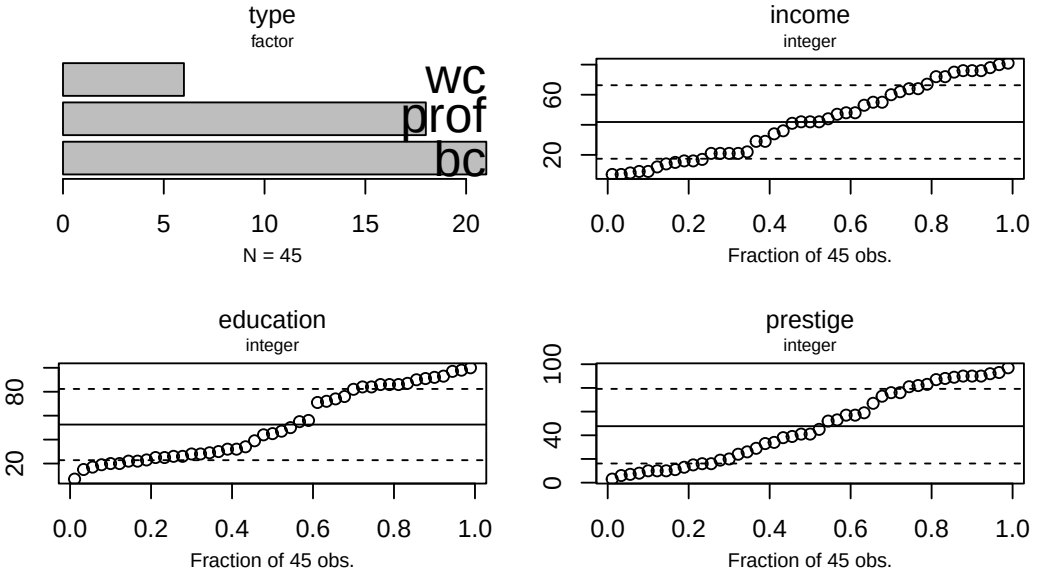
```
dim(Duncan)
```

```
  [1] 45  4
```

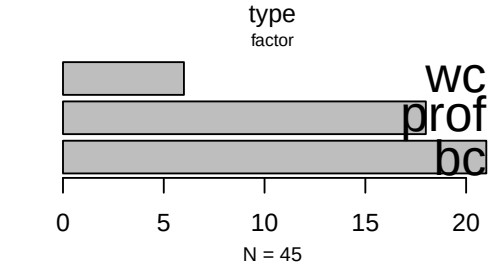
```
summary(Duncan)
```

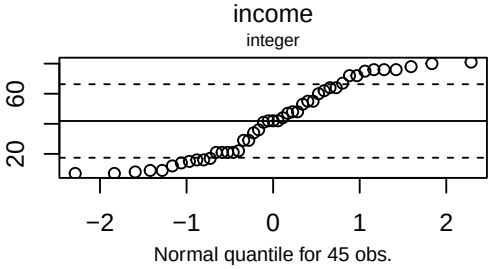
```
    type        income          education         prestige
 bc   :21   Min.   : 7.00   Min.   :  7.00   Min.   : 3.00
 prof:18   1st Qu.:21.00   1st Qu.: 26.00   1st Qu.:16.00
 wc   : 6   Median :42.00   Median : 45.00   Median :41.00
            Mean   :41.87   Mean   : 52.56   Mean   :47.69
            3rd Qu.:64.00   3rd Qu.: 84.00   3rd Qu.:81.00
            Max.   :81.00   Max.   :100.00   Max.   :97.00
```

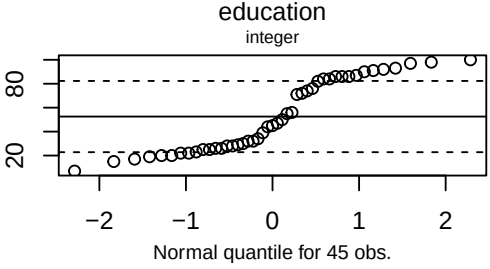
```
library(spida2)  # install with devtools::install_github('gmonette/
xqplot(Duncan)   # uniform quantile plots
```

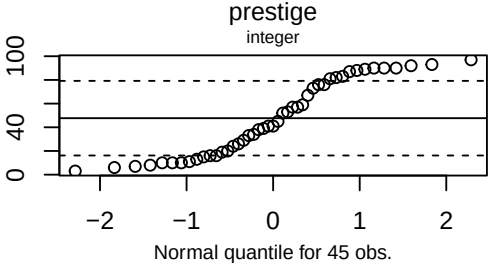



```
xqplot(Duncan, ptype = 'n')    # normal quantile plots
```


type
factor
wc
prof
bc
0
5
10
15
20
N = 45


income
integer
20
60
−2
−1
0
1
2
Normal quantile for 45 obs.


education
integer
20
80
−2
−1
0
1
2
Normal quantile for 45 obs.

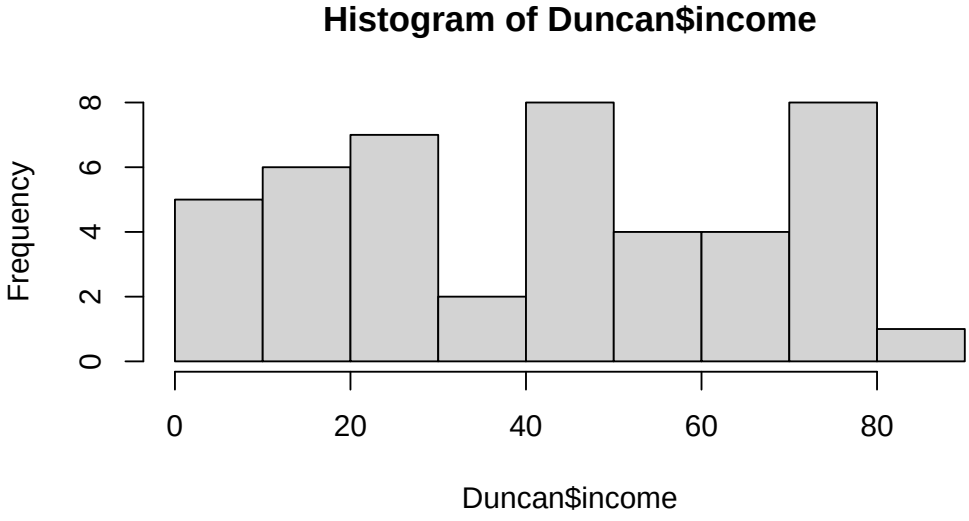

prestige
integer
0
40
100
−2
−1
0
1
2
Normal quantile for 45 obs.

8.1 Referencing variables in data frames

We will see this again later

Fully qualified name

```
hist(Duncan$income)
```

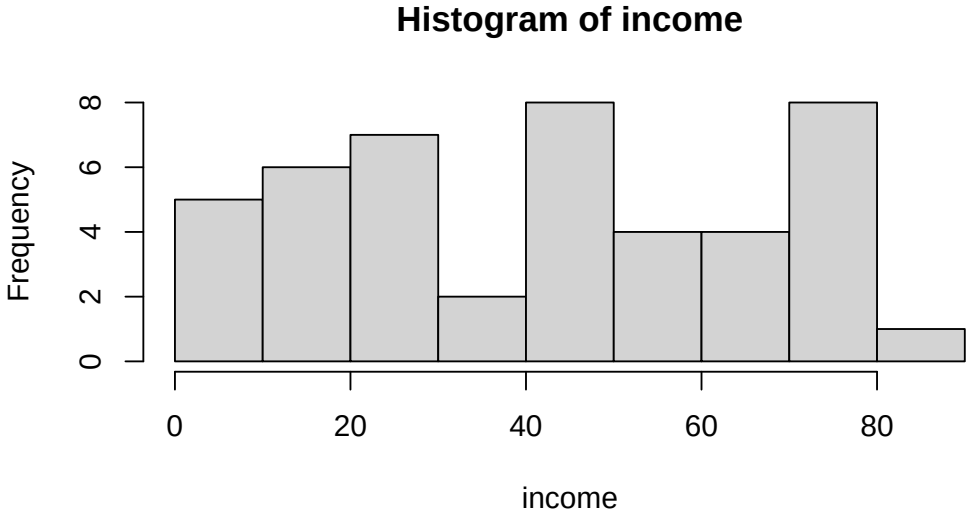


using the 'with' function: the second argument is evaluated in the data frame

```
with(Duncan, hist(income))
```

using ‘attach’ – ___ very highly deprecated___ e.g. this blog post

```
attach(Duncan)
hist(income)
```

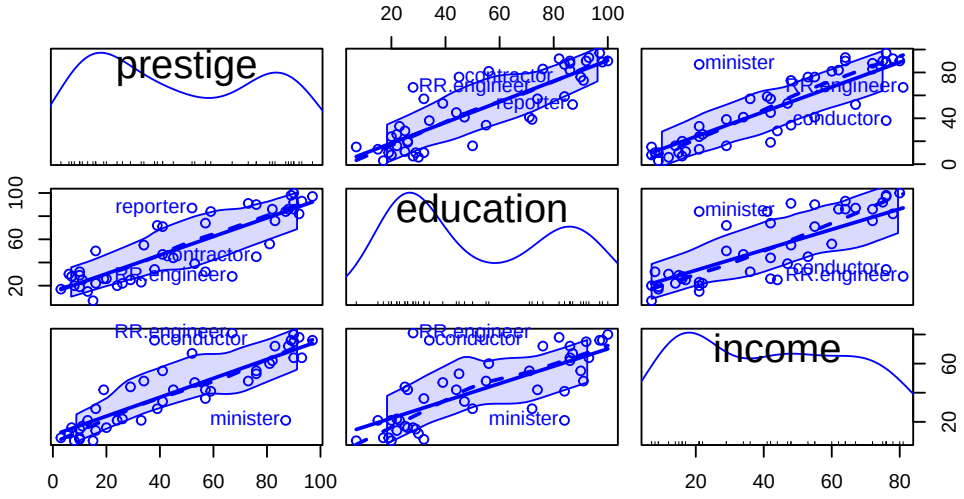



```
detach(Duncan)
```

Many 'recent' (past 25 years) modelling functions and graphic functions in R use

formulas. When a model or a graph is defined by a formula, the data frame in which the terms of the formula are to be found is an argument of the function.

```
scatterplotMatrix( ~ prestige + education + income,
                   data = Duncan, id = list(n=3))
```



Fitting a least-squares regression model

```
(Duncan.model <- lm(prestige ~ education + income,
                    data = Duncan))
```

```
Call:
lm(formula = prestige ~ education + income, data = Duncan)

Coefficients:
(Intercept)    education       income
    -6.0647       0.5458       0.5987
```

Estimated coefficients and other results of the regression:

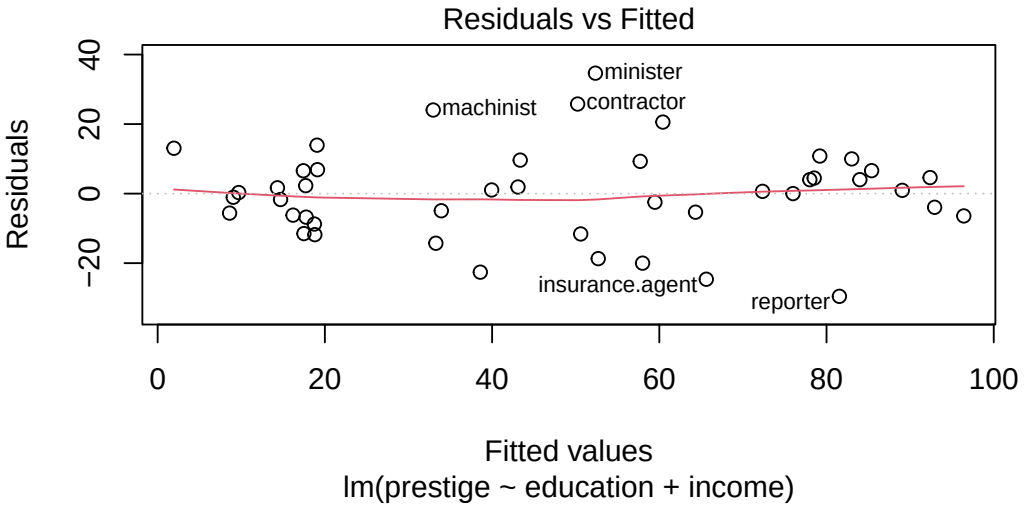
```
summary(Duncan.model)
```

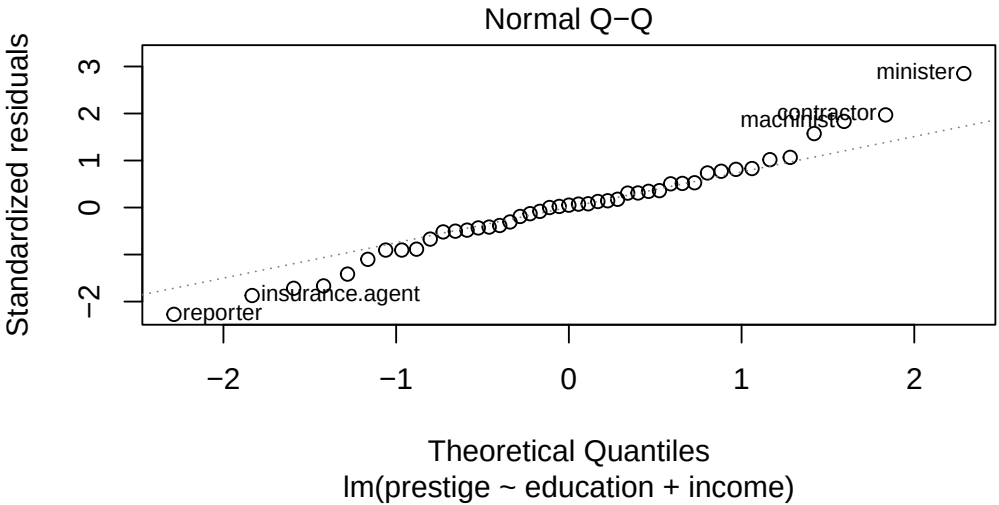
```
Call:
lm(formula = prestige ~ education + income, data = Duncan)

Residuals:
```

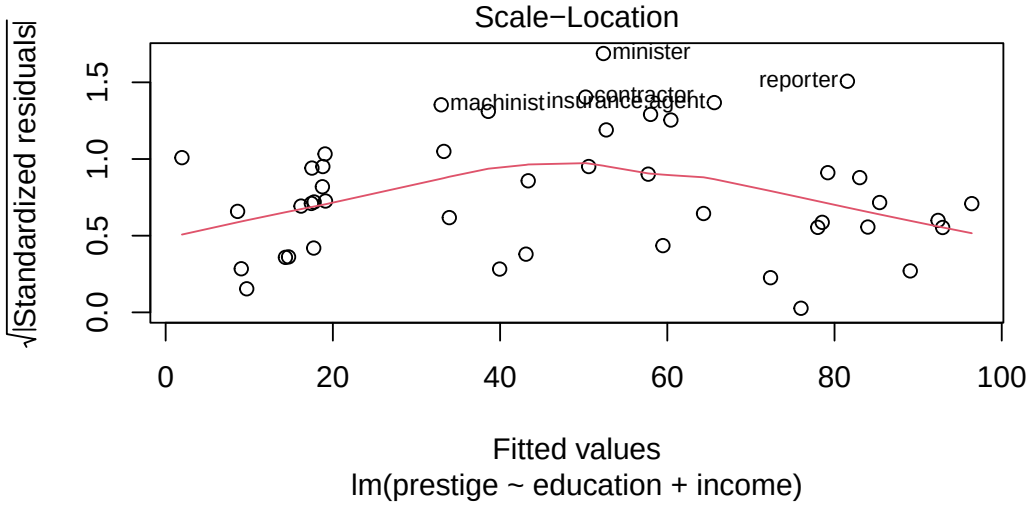
```
    Min      1Q  Median      3Q     Max
-29.538  -6.417   0.655   6.605  34.641

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.06466    4.27194  -1.420    0.163
education    0.54583    0.09825   5.555 1.73e-06 ***
income       0.59873    0.11967   5.003 1.05e-05 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.37 on 42 degrees of freedom
Multiple R-squared:  0.8282,  Adjusted R-squared:   0.82
F-statistic: 101.2 on 2 and 42 DF,  p-value: < 2.2e-16
```

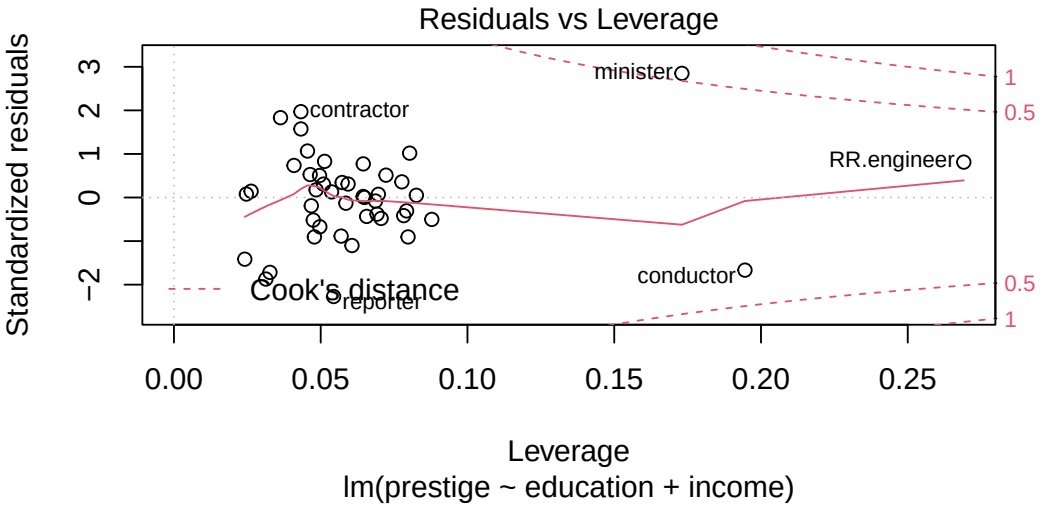
Four important plots to check the regression:

```
plot(Duncan.model, id.n = 5, ask = FALSE)
```



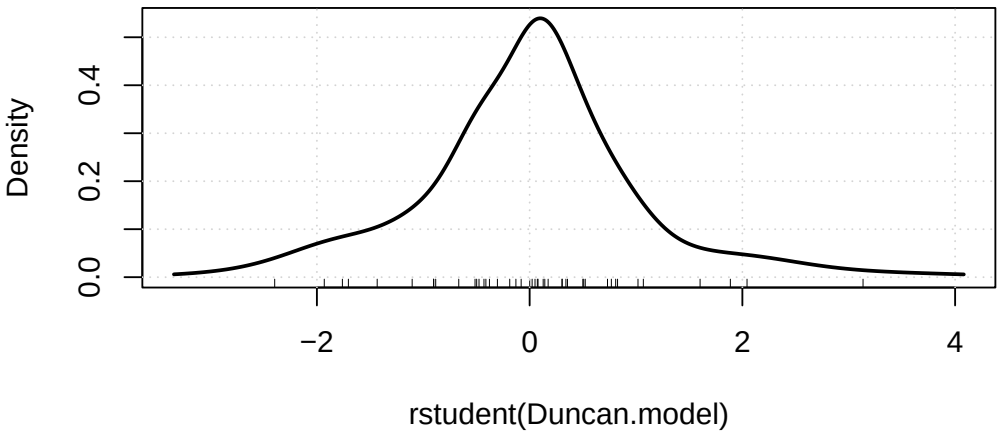

Normal Q−Q
Standardized residuals
3
2
1
0
−2
minister
contractor
machinist
insurance.agent
reporter
−2
−1
0
1
2
Theoretical Quantiles
lm(prestige ~ education + income)


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
minister
reporter
machinist
insurance.agent
contractor
0
20
40
60
80
100
Fitted values
lm(prestige ~ education + income)



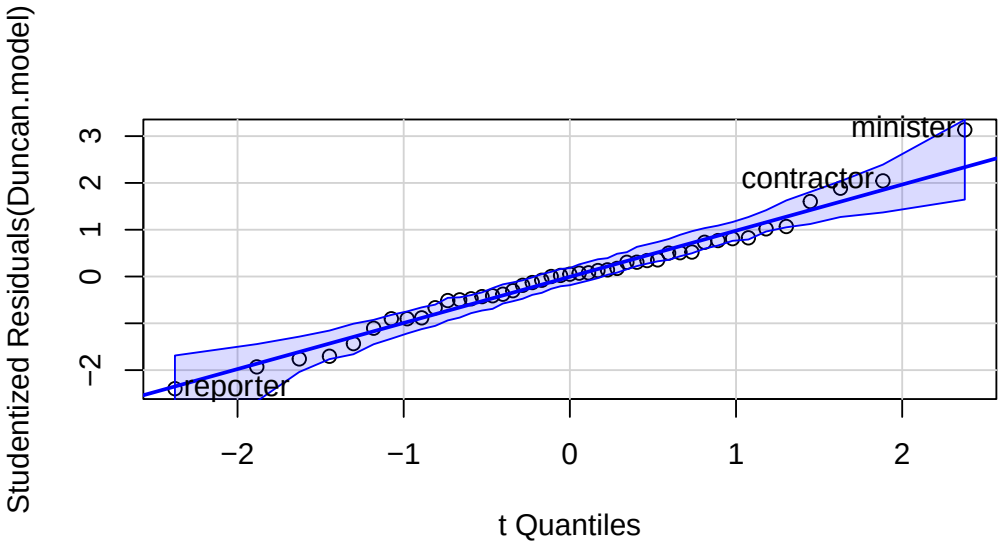
The density of the studentized residuals:

```
densityPlot(rstudent(Duncan.model))
```



Quantile plot of residuals:

```
qqPlot(Duncan.model, id = list(n = 3))
```



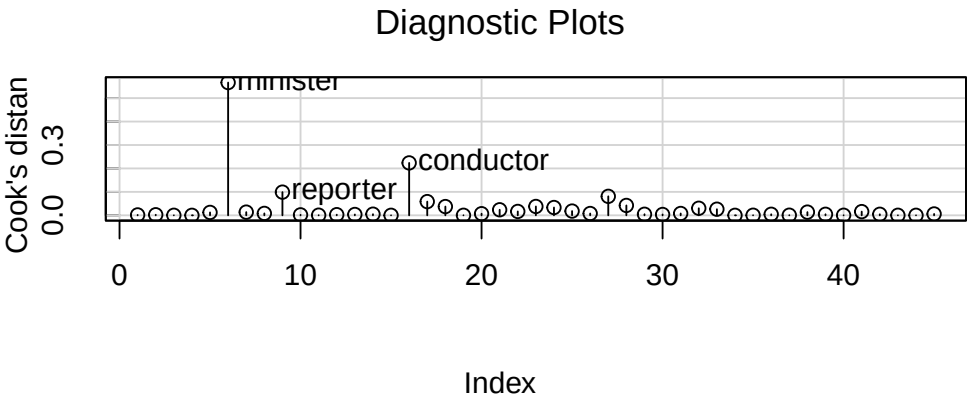
```
    minister   reporter contractor
           6          9         17
```

Outliers:

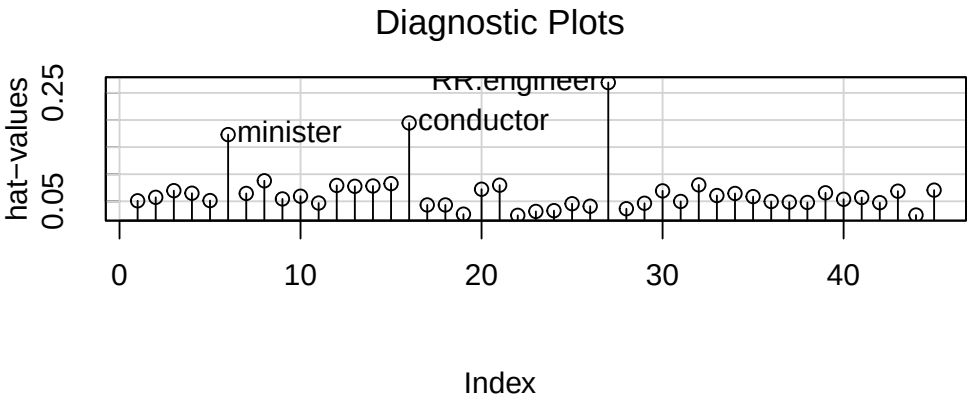
```
outlierTest(Duncan.model)
```

```
  No Studentized residuals with Bonferroni p < 0.05
  Largest |rstudent|:
           rstudent unadjusted p-value Bonferroni p
  minister 3.134519          0.0031772      0.14297
```

```
influenceIndexPlot(Duncan.model, vars = "Cook",
    id = list(n=3))
```



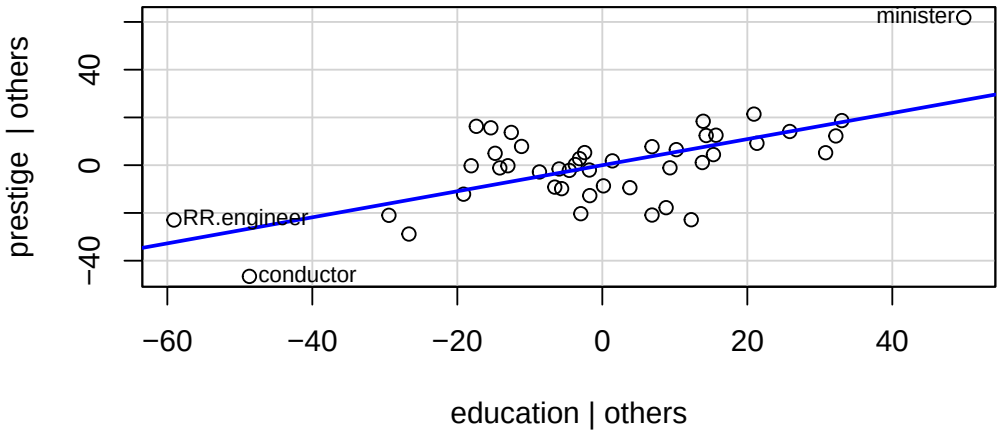

```
influenceIndexPlot(Duncan.model, vars = "hat",
    id = list(n=3))
```


Diagnostic Plots
hat−values
0.05
0.25
minister
conductor
RR.engineer
0
10
20
30
40
Index

8.2 Added-variable plots

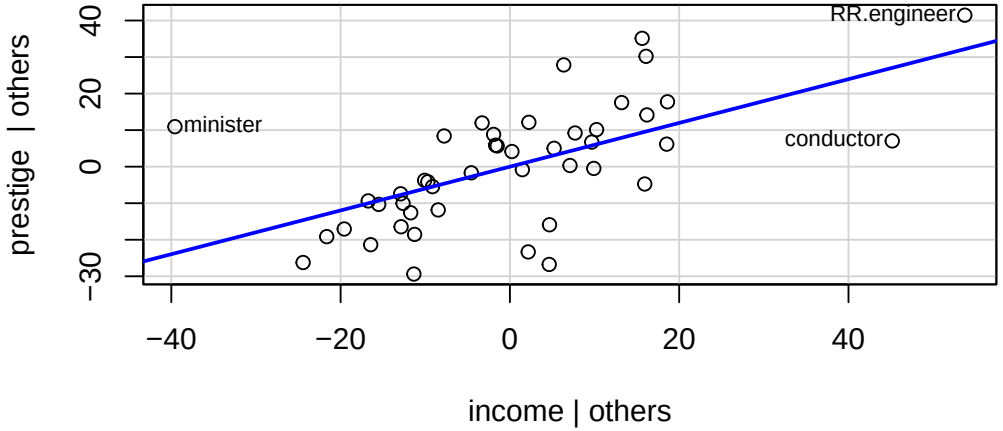
Also know in PROC REG in SAS as 'partial regression leverage plots' not to be confused with partial residual plots (better described as 'component plus residual plots') although the are often called that.

```
avPlots(Duncan.model, ~ education,
    id=list(cex=0.75, n=3, method="mahal"))
```



Imagine the following plot is a simple scatterplot between two variables. What do you see?

```
avPlots(Duncan.model, ~ income,
    id=list(cex=0.75, n=3, method="mahal"))
```

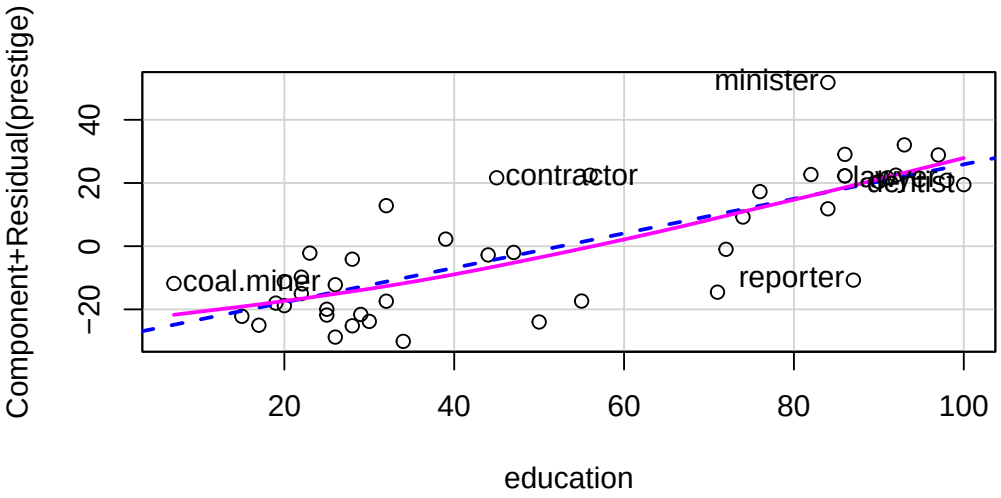


In multiple regression, the AVP is the closest you come to seeing what drives the estimation of each regression coefficient.

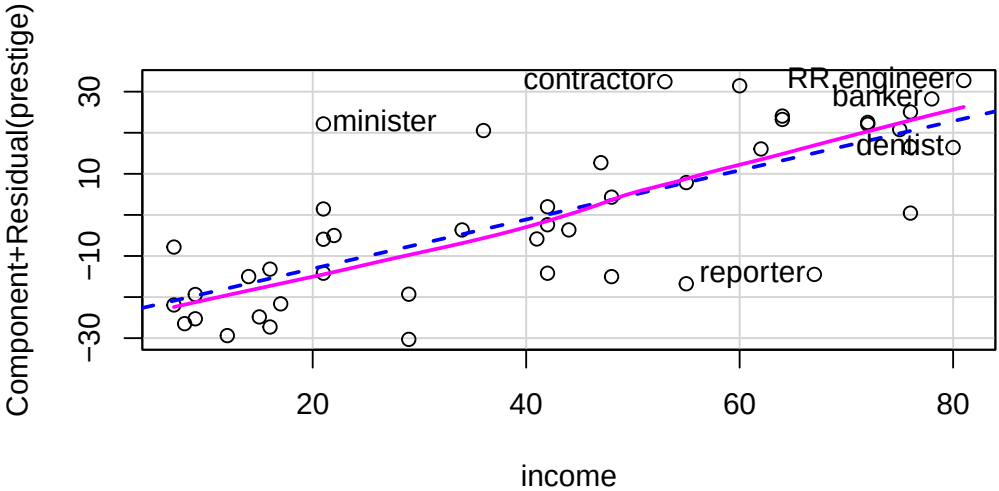
8.3 Component-plus-residual plots

Also known as partial residual plots. These tend to be more widely used but often less informative than AVPs

```
crPlots(Duncan.model, ~ education, id = list(n=3))
```




```
crPlots(Duncan.model, ~ income, id = list(n=3))
```



Non-constant variance test

```
ncvTest(Duncan.model)
```

```
  Non-constant Variance Score Test
  Variance formula: ~ fitted.values
  Chisquare = 0.3810967, Df = 1, p = 0.53702
```

```
ncvTest(Duncan.model, var.formula = ~ income + education)
```

```
  Non-constant Variance Score Test
  Variance formula: ~ income + education
  Chisquare = 0.6976023, Df = 2, p = 0.70553
```

Position of row names

```
whichNames(c("minister", "conductor"), Duncan)
```

```
   minister conductor
          6         16
```

```
duncan.model.2 <- update(Duncan.model, subset = -c(6, 16))
```

An alternative approach that uses the '%in%' operator to creae a logical vector

```
outliers <- names(Duncan) %in% c("minister", "conductor")
```

```
duncan.model.2 <- update(Duncan.model, subset = !outliers)
```

```
summary(duncan.model.2)
```

```
Call:
lm(formula = prestige ~ education + income, data = Duncan, subset

Residuals:
    Min      1Q  Median      3Q     Max
-29.538  -6.417   0.655   6.605  34.641

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.06466    4.27194  -1.420    0.163
education    0.54583    0.09825   5.555 1.73e-06 ***
```

```
income       0.59873    0.11967   5.003 1.05e-05 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.37 on 42 degrees of freedom
Multiple R-squared:  0.8282,  Adjusted R-squared:   0.82
F-statistic: 101.2 on 2 and 42 DF,  p-value: < 2.2e-16
```

Comparing coefficients with and without outliers:

```
compareCoefs(duncan.model, duncan.model.2)
```

```
Error in compareCoefs(duncan.model, duncan.model.2): object 'dunca
```

Type of predictor

```
summary(Duncan$type)
```

```
   bc prof   wc
   21   18    6
```

```
summary(Duncan$prestige)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00   16.00   41.00   47.69   81.00   97.00
```

```
summary(Duncan)
```

```
     type         income          education         prestige
 bc   :21   Min.   : 7.00   Min.   :  7.00   Min.   : 3.00
 prof:18   1st Qu.:21.00   1st Qu.: 26.00   1st Qu.:16.00
 wc   : 6   Median :42.00   Median : 45.00   Median :41.00
            Mean   :41.87   Mean   : 52.56   Mean   :47.69
            3rd Qu.:64.00   3rd Qu.: 84.00   3rd Qu.:81.00
            Max.   :81.00   Max.   :100.00   Max.   :97.00
```

Model with interaction

```
summary(lm(prestige ~ education + income, data = Duncan))
```

```
Call:
lm(formula = prestige ~ education + income, data = Duncan)

Residuals:
    Min      1Q  Median      3Q     Max
-29.538  -6.417   0.655   6.605  34.641

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.06466    4.27194  -1.420    0.163
education    0.54583    0.09825   5.555 1.73e-06 ***
income       0.59873    0.11967   5.003 1.05e-05 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.37 on 42 degrees of freedom
Multiple R-squared:  0.8282,  Adjusted R-squared:   0.82
F-statistic: 101.2 on 2 and 42 DF,  p-value: < 2.2e-16
```

```
class(Duncan$type)
```

```
[1] "factor"
```

```
class(Duncan$prestige)
```

```
[1] "integer"
```

```
class(Duncan)
```

```
[1] "data.frame"
```

```
duncan.model <- lm(prestige ~ education + income, data=Duncan)
class(duncan.model)
```

```
[1] "lm"
```

9 Basics of Object-Oriented Programming in R

What happens when you use the function 'summary' on an 'lm' model:

```
summary(duncan.model)
```

```
Call:
lm(formula = prestige ~ education + income, data = Duncan)

Residuals:
    Min      1Q  Median      3Q     Max
-29.538  -6.417   0.655   6.605  34.641

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.06466    4.27194  -1.420    0.163
education    0.54583    0.09825   5.555 1.73e-06 ***
```

```
income       0.59873    0.11967   5.003 1.05e-05 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.37 on 42 degrees of freedom
Multiple R-squared:  0.8282,  Adjusted R-squared:   0.82
F-statistic: 101.2 on 2 and 42 DF,  p-value: < 2.2e-16
```

A **generic function**:

```
summary
```

```
function (object, ...)
UseMethod("summary")
<bytecode: 0x55a57f9302c8>
<environment: namespace:base>
```

Class of 'duncan.model':

```
class(duncan.model)
```

```
[1] "lm"
```

A **method** for the 'lm' **class** for the generic function

```
args(summary.lm)
```

```
function (object, correlation = FALSE, symbolic.cor = FALSE,
    ...)
NULL
```

The real work happens in the special function that is the method for the generic function 'summary' for the 'lm' class.

```
summary.lm
```

```
function (object, correlation = FALSE, symbolic.cor = FALSE,
    ...)
{
    z <- object
```

```
p <- z$rank
rdf <- z$df.residual
if (p == 0) {
    r <- z$residuals
    n <- length(r)
    w <- z$weights
    if (is.null(w)) {
        rss <- sum(r^2)
    }
    else {
        rss <- sum(w * r^2)
        r <- sqrt(w) * r
    }
    resvar <- rss/rdf
    ans <- z[c("call", "terms", if (!is.null(z$weights)) "weig
    class(ans) <- "summary.lm"
    ans$aliased <- is.na(coef(object))
    ans$residuals <- r
    ans$df <- c(0L, n, length(ans$aliased))
```

```
    ans$coefficients <- matrix(NA_real_, 0L, 4L, dimnames = li
        c("Estimate", "Std. Error", "t value", "Pr(>|t|)")))
    ans$sigma <- sqrt(resvar)
    ans$r.squared <- ans$adj.r.squared <- 0
    ans$cov.unscaled <- matrix(NA_real_, 0L, 0L)
    if (correlation)
        ans$correlation <- ans$cov.unscaled
    return(ans)
}
if (is.null(z$terms))
    stop("invalid 'lm' object:  no 'terms' component")
if (!inherits(object, "lm"))
    warning("calling summary.lm(<fake-lm-object>) ...")
Qr <- qr.lm(object)
n <- NROW(Qr$qr)
if (is.na(z$df.residual) || n - p != z$df.residual)
    warning("residual degrees of freedom in object suggest thi
r <- z$residuals
f <- z$fitted.values
```

```
    w <- z$weights
    if (is.null(w)) {
        mss <- if (attr(z$terms, "intercept"))
            sum((f - mean(f))^2)
        else sum(f^2)
        rss <- sum(r^2)
    }
    else {
        mss <- if (attr(z$terms, "intercept")) {
            m <- sum(w * f/sum(w))
            sum(w * (f - m)^2)
        }
        else sum(w * f^2)
        rss <- sum(w * r^2)
        r <- sqrt(w) * r
    }
    resvar <- rss/rdf
    if (is.finite(resvar) && resvar < (mean(f)^2 + var(c(f))) *
        1e-30)
```

```
    warning("essentially perfect fit: summary may be unreliabl
p1 <- 1L:p
R <- chol2inv(Qr$qr[p1, p1, drop = FALSE])
se <- sqrt(diag(R) * resvar)
est <- z$coefficients[Qr$pivot[p1]]
tval <- est/se
ans <- z[c("call", "terms", if (!is.null(z$weights)) "weights"
ans$residuals <- r
ans$coefficients <- cbind(Estimate = est, `Std. Error` = se,
    `t value` = tval, `Pr(>|t|)` = 2 * pt(abs(tval), rdf,
        lower.tail = FALSE))
ans$aliased <- is.na(z$coefficients)
ans$sigma <- sqrt(resvar)
ans$df <- c(p, rdf, NCOL(Qr$qr))
if (p != attr(z$terms, "intercept")) {
    df.int <- if (attr(z$terms, "intercept"))
        1L
    else 0L
    ans$r.squared <- mss/(mss + rss)
```

```
        ans$adj.r.squared <- 1 - (1 - ans$r.squared) * ((n -
            df.int)/rdf)
        ans$fstatistic <- c(value = (mss/(p - df.int))/resvar,
            numdf = p - df.int, dendf = rdf)
    }
    else ans$r.squared <- ans$adj.r.squared <- 0
    ans$cov.unscaled <- R
    dimnames(ans$cov.unscaled) <- dimnames(ans$coefficients)[c(1,
        1)]
    if (correlation) {
        ans$correlation <- (R * resvar)/outer(se, se)
        dimnames(ans$correlation) <- dimnames(ans$cov.unscaled)
        ans$symbolic.cor <- symbolic.cor
    }
    if (!is.null(z$na.action))
        ans$na.action <- z$na.action
    class(ans) <- "summary.lm"
    ans
}
```

```
<bytecode: 0x55a580cae348>
<environment: namespace:stats>
```

This is R's basic form of Object-Oriented-Programming (OOP). It's what makes it possible for R to have grown through the work of different contributors working relatively independently.

If you create a new statistical method that produces a special kind of object, you don't have to request that 'summary' be changed to work on your object. You just give your object a class and write a summary method for it.

- **summary** is the **generic** function
- **summary.lm** is a **method** for this generic function
- **lm** is the **class** that this methods works on

All the **methods** for the generic function **summary**, i.e. all the classes that **summary** works on.

```
methods(summary)
```

```
[1] summary.Anova.mlm*
```

```
 [2] summary.aov
 [3] summary.aovlist*
 [4] summary.aspell*
 [5] summary.bcnPowerTransform*
 [6] summary.bcnPowerTransformlmer*
 [7] summary.boot*
 [8] summary.check_packages_in_dir*
 [9] summary.connection
[10] summary.data.frame
[11] summary.Date
[12] summary.default
[13] summary.ecdf*
[14] summary.factor
[15] summary.glm
[16] summary.haven_labelled*
[17] summary.infl*
[18] summary.lm
[19] summary.loess*
[20] summary.loglm*
```

```
[21] summary.manova
[22] summary.matrix
[23] summary.mlm*
[24] summary.negbin*
[25] summary.nls*
[26] summary.packageStatus*
[27] summary.polr*
[28] summary.POSIXct
[29] summary.POSIXlt
[30] summary.powerTransform*
[31] summary.ppr*
[32] summary.prcomp*
[33] summary.princomp*
[34] summary.proc_time
[35] summary.rlang_error*
[36] summary.rlang_trace*
[37] summary.rlm*
[38] summary.srcfile
[39] summary.srcref
```

```
   [40] summary.stepfun
   [41] summary.stl*
   [42] summary.table
   [43] summary.tukeysmooth*
   [44] summary.vctrs_sclr*
   [45] summary.vctrs_vctr*
   [46] summary.warnings
   see '?methods' for accessing help and source code
```

```
getAnywhere('summary.ppr')
```

```
   A single object matching 'summary.ppr' was found
   It was found in the following places
     registered S3 method for summary from namespace stats
     namespace:stats
   with value

   function (object, ...)
   {
```

```
    class(object) <- "summary.ppr"
    object
}
<bytecode: 0x55a58091fa00>
<environment: namespace:stats>
```

All the methods that work on class 'lm':

```
methods(class = 'lm')
```

```
 [1] add1                add.scope           alias
 [3] anova               Anova               Boot
 [5] avPlot              Boot                boxCox
 [7] bootCase            boxCox              case.names
 [9] brief               case.names          coerce
[11] ceresPlot           coerce              confint
[13] confidenceEllipse   confint            cooks.distance
[15] Confint             cooks.distance      deltaMethod
[17] crPlot              deltaMethod         dfbeta
[19] deviance            dfbeta              dfbetas
```

```
 [21] dfbetaPlots          dfbetas
 [23] dfbetasPlots         drop1
 [25] dummy.coef           durbinWatsonTest
 [27] effects              extractAIC
 [29] family               formula
 [31] getData              getFix
 [33] hatvalues            hccm
 [35] infIndexPlot         influence
 [37] influencePlot        initialize
 [39] inverseResponsePlot kappa
 [41] labels               leveneTest
 [43] leveragePlot         linearHypothesis
 [45] logLik               mcPlot
 [47] mmp                  model.frame
 [49] model.matrix         ncvTest
 [51] nextBoot             nobs
 [53] outlierTest          plot
 [55] powerTransform       predict
 [57] Predict              print
```

```
 [59] proj                qqPlot
 [61] qr                  residualPlot
 [63] residualPlots       residuals
 [65] rstandard           rstudent
 [67] S                   show
 [69] sigmaHat            simulate
 [71] slotsFromS3         spreadLevelPlot
 [73] summary             symbox
 [75] variable.names      vcov
see '?methods' for accessing help and source code
```

```
getAnywhere('logLik.lm')
```

```
A single object matching 'logLik.lm' was found
It was found in the following places
  registered S3 method for logLik from namespace stats
  namespace:stats
with value
```

```
function (object, REML = FALSE, ...)
{
    if (inherits(object, "mlm"))
        stop("'logLik.lm' does not support multiple responses")
    res <- object$residuals
    p <- object$rank
    N <- length(res)
    if (is.null(w <- object$weights)) {
        w <- rep.int(1, N)
    }
    else {
        excl <- w == 0
        if (any(excl)) {
            res <- res[!excl]
            N <- length(res)
            w <- w[!excl]
        }
    }
    N0 <- N
```

```
    if (REML)
        N <- N - p
    val <- 0.5 * (sum(log(w)) - N * (log(2 * pi) + 1 - log(N) +
        log(sum(w * res^2))))
    if (REML)
        val <- val - sum(log(abs(diag(object$qr$qr)[1L:p])))
    attr(val, "nall") <- N0
    attr(val, "nobs") <- N
    attr(val, "df") <- p + 1
    class(val) <- "logLik"
    val
}
<bytecode: 0x55a581069898>
<environment: namespace:stats>
```

9.1 Creating your classes and methods

Here's latest discovery of a new statistical method: Use the median to 'fit' data!

```
silly <- function(x) {
    ret <- median(x)
    class(ret) <- 'silly'
    ret
}
fit <- silly(x)
fit
```

```
  [1] 0.05293737
  attr(,"class")
  [1] "silly"
```

```
class(fit)
```

```
  [1] "silly"
```

I write a **method** for the **summary** generic function for the **class** 'silly'.

```
summary.silly <- function(fit) {
    cat('This is the fitted value from the silly method: ')
```

```
    cat(fit,'\n')
    invisible(fit)
}
```

See what it does:

```
summary(fit)
```

```
  This is the fitted value from the silly method: 0.05293737
```

When I called the 'summary' function, it checked the class of its first argument, 'fit'.

Finding that the class was 'silly', summary looks for a function called 'summary.silly' and uses that function. This is called **dispatching**. If 'summary.silly' hadn't existed summary would have used 'summary.default'.

For example, when you use 'glm'

```
mod.mroz <- glm(lfp ~ ., family=binomial, data=Mroz)
class(mod.mroz)
```

```
  [1] "glm" "lm"
```

```
summary.glm
```

```
  function (object, dispersion = NULL, correlation = FALSE, symbolic
      ...)
  {
      est.disp <- FALSE
      df.r <- object$df.residual
      if (is.null(dispersion))
          dispersion <- if (object$family$family %in% c("poisson",
              "binomial"))
              1
          else if (df.r > 0) {
              est.disp <- TRUE
              if (any(object$weights == 0))
                  warning("observations with zero weight not used fo
              sum((object$weights * object$residuals^2)[object$weigh
                  0])/df.r
```

```
    }
    else {
        est.disp <- TRUE
        NaN
    }
aliased <- is.na(coef(object))
p <- object$rank
if (p > 0) {
    p1 <- 1L:p
    Qr <- qr.lm(object)
    coef.p <- object$coefficients[Qr$pivot[p1]]
    covmat.unscaled <- chol2inv(Qr$qr[p1, p1, drop = FALSE])
    dimnames(covmat.unscaled) <- list(names(coef.p), names(coe
    covmat <- dispersion * covmat.unscaled
    var.cf <- diag(covmat)
    s.err <- sqrt(var.cf)
    tvalue <- coef.p/s.err
    dn <- c("Estimate", "Std. Error")
    if (!est.disp) {
```

```
            pvalue <- 2 * pnorm(-abs(tvalue))
            coef.table <- cbind(coef.p, s.err, tvalue, pvalue)
            dimnames(coef.table) <- list(names(coef.p), c(dn,
                "z value", "Pr(>|z|)"))
        }
        else if (df.r > 0) {
            pvalue <- 2 * pt(-abs(tvalue), df.r)
            coef.table <- cbind(coef.p, s.err, tvalue, pvalue)
            dimnames(coef.table) <- list(names(coef.p), c(dn,
                "t value", "Pr(>|t|)"))
        }
        else {
            coef.table <- cbind(coef.p, NaN, NaN, NaN)
            dimnames(coef.table) <- list(names(coef.p), c(dn,
                "t value", "Pr(>|t|)"))
        }
        df.f <- NCOL(Qr$qr)
    }
    else {
```

```
    coef.table <- matrix(, 0L, 4L)
    dimnames(coef.table) <- list(NULL, c("Estimate", "Std. Err
        "t value", "Pr(>|t|)"))
    covmat.unscaled <- covmat <- matrix(, 0L, 0L)
    df.f <- length(aliased)
}
keep <- match(c("call", "terms", "family", "deviance", "aic",
    "contrasts", "df.residual", "null.deviance", "df.null",
    "iter", "na.action"), names(object), 0L)
ans <- c(object[keep], list(deviance.resid = residuals(object,
    type = "deviance"), coefficients = coef.table, aliased = a
    dispersion = dispersion, df = c(object$rank, df.r, df.f),
    cov.unscaled = covmat.unscaled, cov.scaled = covmat))
if (correlation && p > 0) {
    dd <- sqrt(diag(covmat.unscaled))
    ans$correlation <- covmat.unscaled/outer(dd, dd)
    ans$symbolic.cor <- symbolic.cor
}
class(ans) <- "summary.glm"
```

```
    return(ans)
}
<bytecode: 0x55a5805f9768>
<environment: namespace:stats>
```

10 Loops in R

You almost never need to use a for loop in R.

Let's see why:

Most functions and operators are already vectorized.

In C, if you want to add two vectors, you need to do this:

```
(x <- 1:5)
```

```
[1] 1 2 3 4 5
```

```
(y <- 11:15)
```

```
[1] 11 12 13 14 15
```

```
# Adding 2 vectors:

z <- rep(0, length(x))   # create an object to hold the result
i <- 1 # initialize the index
while(i <= 5) {
    z[i] <- x[i] + y[i]
    i <- i + 1   # increment the index
}
z # et voila
```

```
[1] 12 14 16 18 20
```

What does it mean to say that the '+' operator is vectorized?

```
z <- x + y   # much easier
z
```

```
[1] 12 14 16 18 20
```

Here's a simple problem:

From a matrix, for each row, count the number of entries greater than 4

Create a matrix:

```
set.seed(123345)
mat <- matrix(sample(0:10, 30, replace = TRUE), nrow = 5)
mat
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    6    3    5    4    0    2
[2,]    3    0    6   10    2    1
[3,]    1    5    3    8    6    4
[4,]    6    3    5    6    0   10
[5,]    1    0    3    8    2    8
```

Using for loops:

```
count <- rep(0, 5)       # initialize count
for(i in 1:5) {          # index over rows
    for(j in 1:6) {       # index over columns
        if(mat[i,j] >4) count[i] <- count[i] + 1    # increment count
```

```
    }
}
count
```

```
[1] 2 2 3 4 2
```

```
# check
mat
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    6    3    5    4    0    2
[2,]    3    0    6   10    2    1
[3,]    1    5    3    8    6    4
[4,]    6    3    5    6    0   10
[5,]    1    0    3    8    2    8
```

10.1 Using 'apply' functions in R

R is OO so we can work directly with things like vectors and matrices

Suppose we had a function that can count how many elements of a vector are greater than 4

```
g4 <- function(x) {
    sum(x > 4)
}
g4(1:10)
```

```
[1] 6
```

Use **apply:** works on dimensions of an array, e.g. rows or columns of a matrix

```
apply(mat, 1, g4) # the '1' says apply the function g4 to each **ro
```

```
[1] 2 2 3 4 2
```

If you wanted to apply it to each column:

```
apply(mat, 2, g4)
```

```
[1] 2 1 3 4 1 2
```

Another way is to use the fact that '<' is vectorized:

```
mat > 4
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,]  TRUE FALSE  TRUE FALSE FALSE FALSE
[2,] FALSE FALSE  TRUE  TRUE FALSE FALSE
[3,] FALSE  TRUE FALSE  TRUE  TRUE FALSE
[4,]  TRUE FALSE  TRUE  TRUE FALSE  TRUE
[5,] FALSE FALSE FALSE  TRUE FALSE  TRUE
```

```
apply(mat > 4, 1, sum)
```

```
[1] 2 2 3 4 2
```

Note that 'apply' takes 3 or more arguments:

1. matrix or array
2. dimension(s) to define dimensions (facets, rows, columns, etc.) of the array to work on
3. a function to apply to each object

The fact that a function is an ordinary argument of a

There are many other 'apply' functions that work on difference kinds of objects:

- lapply on lists or vectors: returns a list
- sapply on lists of vectors: returns a vector if it can
- mapply applies a function to lists of arguments
- Map does something similar
- do.call applies a function to a list of arguments
- capply in spida2 applies a function to chunks of values of an argument, the chunks are defined by the values of another argument.

Experienced R programmers rarely use for loops (sometimes they are useful) and tend to use 'apply' functions instead.

10.2 Examples of lapply and sapply

A data frame is like a list of its variables

Suppose you would like to make sure that every factor is turned into a character variable

```r
dd <- carData::Prestige
dim(dd)
```

```
[1] 102   6
```

```r
head(dd)
```

```
                    education income women prestige census
gov.administrators      13.11  12351 11.16     68.8   1113
general.managers        12.26  25879  4.02     69.1   1130
accountants             12.77   9271 15.70     63.4   1171
purchasing.officers     11.42   8865  9.11     56.8   1175
chemists                14.62   8403 11.68     73.5   2111
physicists              15.64  11030  5.13     77.6   2113
                    type
gov.administrators  prof
general.managers    prof
accountants         prof
purchasing.officers prof
```

```
  chemists            prof
  physicists          prof
```

```
class(dd)
```

```
  [1] "data.frame"
```

```
# how to get the class of each variable?
sapply(dd, class)
```

```
  education    income     women  prestige    census      type
  "numeric" "integer" "numeric" "numeric" "integer"  "factor"
```

```
sapply(dd, typeof)  # works variable by variable and returns a vect
```

```
  education    income     women  prestige    census      type
   "double" "integer"  "double"  "double" "integer" "integer"
```

```
lapply(dd, typeof)  # works variable by variable and returns a list
```

```
  $education
  [1] "double"
```

```
$income
[1] "integer"

$women
[1] "double"

$prestige
[1] "double"

$census
[1] "integer"

$type
[1] "integer"
```

```
lapply(dd, function(x) if(is.factor(x)) as.character(x) else x)
```

```
$education
```

```
  [1] 13.11 12.26 12.77 11.42 14.62 15.64 15.09 15.44 14.52
 [10] 14.64 12.39 12.30 13.83 14.44 14.36 14.21 15.77 14.15
 [19] 15.22 14.50 15.97 13.62 15.08 15.96 15.94 14.71 12.46
 [28]  9.45 13.62 15.21 12.79 11.09 12.71 11.44 11.59 11.49
 [37] 11.32 10.64 11.36  9.17 12.09 11.04  9.22 10.07 10.51
 [46] 11.20 11.13 11.43 11.00  9.84 11.13 10.05  9.62  9.93
 [55] 11.60 11.09 11.03  9.47 10.93  7.74  8.50 10.57  9.46
 [64]  7.33  7.11  7.58  6.84  8.60  8.88  7.54  7.64  7.64
 [73]  7.42  6.69  6.74 10.09  8.81  8.40  7.92  8.43  8.78
 [82]  8.76 10.29  6.38  8.10 10.10  6.67  9.05  9.93  8.24
 [91]  6.92  6.60  7.81  8.33  7.52 12.27  8.49  7.58  7.93
[100]  8.37 10.00  8.55

$income
  [1] 12351 25879  9271  8865  8403 11030  8258 14163 11377
 [10] 11023  5902  7059  8425  8049  7405  6336 19263  6112
 [19]  9593  4686 12480  5648  8034 25308 14558 17498  4614
 [28]  3485  5092 10432  5180  6197  7562  8206  4036  3148
 [37]  4348  2448  4330  4761  3016  2901  5511  3739  3161
```

```
 [46]  4741  5052  6259  4075  7482  8780  2594   918  2370
 [55]  8131  6992  7956  8895  8891  3116  3930  7869   611
 [64]  3000  3472  3582  3643  1656  6860  4199  5134  5134
 [73]  1890  4443  3485  8043  6686  6565  6477  5811  6573
 [82]  3942  5449  2847  5795  7716  4696  8316  7147  8880
 [91]  5299  5959  4549  6928  3910 14032  8845  5562  4224
[100]  4753  6462  3617

$women
  [1] 11.16  4.02 15.70  9.11 11.68  5.13 25.65  2.69  1.03
 [10]  0.94  1.91  7.83 15.33 57.31 48.28 54.77  5.13 77.10
 [19] 34.89  4.14 19.59 83.78 46.80 10.56  4.32  6.91 96.12
 [28] 76.14 82.66 24.71 76.04 21.03 11.15  8.13 97.51 95.97
 [37] 68.24 91.76 75.92 11.37 83.19 92.86  7.62 52.27 96.14
 [46] 47.06 56.10 39.17 63.23 17.04  3.16 67.82  7.00  3.69
 [55] 13.09 24.44 23.88  0.00  1.65 52.00 15.51  6.01 96.53
 [64] 69.31 33.57 30.08  3.60 27.75  0.00 33.30 17.26 17.26
 [73] 72.24 31.36 39.48  1.50  4.28  2.30  5.17 13.62  5.78
 [82] 74.54  2.92 90.67  0.81  0.78  0.00  1.34  0.99  0.65
```

```
 [91]  0.56  0.52  2.46  0.61  1.09  0.58  0.00  9.47  3.59
[100]  0.00 13.58 70.87

$prestige
  [1] 68.8 69.1 63.4 56.8 73.5 77.6 72.6 78.1 73.1 68.8 62.0
 [12] 60.0 53.8 62.2 74.9 55.1 82.3 58.1 58.3 72.8 84.6 59.6
 [23] 66.1 87.2 66.7 68.4 64.7 34.9 72.1 69.3 67.5 57.2 57.6
 [34] 54.1 46.0 41.9 49.4 42.3 47.7 30.9 32.7 38.7 36.1 37.2
 [45] 38.1 29.4 51.1 35.7 35.6 41.5 40.2 26.5 14.8 23.3 47.3
 [56] 47.1 51.1 43.5 51.6 29.7 20.2 54.9 25.9 20.8 17.3 20.1
 [67] 44.1 21.5 35.3 38.9 25.2 34.8 23.2 33.3 28.8 42.5 44.2
 [78] 35.9 41.8 35.9 43.7 50.8 37.2 28.2 38.1 50.3 27.3 40.9
 [89] 50.2 51.1 38.9 36.2 29.9 42.9 26.5 66.1 48.9 35.9 25.1
[100] 26.1 42.2 35.2

$census
  [1] 1113 1130 1171 1175 2111 2113 2133 2141 2143 2153 2161
 [12] 2163 2183 2311 2315 2331 2343 2351 2391 2511 2711 2731
 [23] 2733 3111 3115 3117 3131 3135 3137 3151 3156 3314 3337
```

```
 [34] 3373 4111 4113 4131 4133 4143 4153 4161 4171 4172 4173
 [45] 4175 4191 4192 4193 4197 5130 5133 5137 5143 5145 5171
 [56] 5172 5191 6111 6112 6121 6123 6141 6147 6162 6191 6193
 [67] 7112 7182 7711 8213 8215 8215 8221 8267 8278 8311 8313
 [78] 8333 8335 8513 8515 8534 8537 8563 8581 8582 8715 8731
 [89] 8733 8780 8781 8782 8785 8791 8798 9111 9131 9171 9173
[100] 9313 9511 9517

$type
  [1] "prof" "prof" "prof" "prof" "prof" "prof" "prof"
  [8] "prof" "prof" "prof" "prof" "prof" "prof" "prof"
 [15] "prof" "prof" "prof" "prof" "prof" "prof" "prof"
 [22] "prof" "prof" "prof" "prof" "prof" "prof" "bc"
 [29] "prof" "prof" "wc"   "prof" "wc"   NA     "wc"
 [36] "wc"   "wc"   "wc"   "wc"   "wc"   "wc"   "wc"
 [43] "wc"   "wc"   "wc"   "wc"   "wc"   "wc"   "wc"
 [50] "wc"   "wc"   "wc"   NA     "bc"   "wc"   "wc"
 [57] "wc"   "bc"   "bc"   "bc"   "bc"   "bc"   NA
 [64] "bc"   "bc"   "bc"   NA     "bc"   "bc"   "bc"
```

```
 [71] "bc"   "bc"   "bc"   "bc"   "bc"   "bc"   "bc"
 [78] "bc"   "bc"   "bc"   "bc"   "bc"   "bc"   "bc"
 [85] "bc"   "bc"   "bc"   "bc"   "bc"   "bc"   "bc"
 [92] "bc"   "bc"   "bc"   "bc"   "prof" "bc"   "bc"
 [99] "bc"   "bc"   "bc"   "bc"
```

```
as.data.frame(lapply(dd, function(x) if(is.factor(x)) as.character(x
```

```
   education income women prestige census type
1      13.11  12351 11.16     68.8   1113 prof
2      12.26  25879  4.02     69.1   1130 prof
3      12.77   9271 15.70     63.4   1171 prof
4      11.42   8865  9.11     56.8   1175 prof
5      14.62   8403 11.68     73.5   2111 prof
6      15.64  11030  5.13     77.6   2113 prof
7      15.09   8258 25.65     72.6   2133 prof
8      15.44  14163  2.69     78.1   2141 prof
9      14.52  11377  1.03     73.1   2143 prof
10     14.64  11023  0.94     68.8   2153 prof
```

11	12.39	5902	1.91	62.0	2161	prof
12	12.30	7059	7.83	60.0	2163	prof
13	13.83	8425	15.33	53.8	2183	prof
14	14.44	8049	57.31	62.2	2311	prof
15	14.36	7405	48.28	74.9	2315	prof
16	14.21	6336	54.77	55.1	2331	prof
17	15.77	19263	5.13	82.3	2343	prof
18	14.15	6112	77.10	58.1	2351	prof
19	15.22	9593	34.89	58.3	2391	prof
20	14.50	4686	4.14	72.8	2511	prof
21	15.97	12480	19.59	84.6	2711	prof
22	13.62	5648	83.78	59.6	2731	prof
23	15.08	8034	46.80	66.1	2733	prof
24	15.96	25308	10.56	87.2	3111	prof
25	15.94	14558	4.32	66.7	3115	prof
26	14.71	17498	6.91	68.4	3117	prof
27	12.46	4614	96.12	64.7	3131	prof
28	9.45	3485	76.14	34.9	3135	bc
29	13.62	5092	82.66	72.1	3137	prof

30	15.21	10432	24.71	69.3	3151	prof
31	12.79	5180	76.04	67.5	3156	wc
32	11.09	6197	21.03	57.2	3314	prof
33	12.71	7562	11.15	57.6	3337	wc
34	11.44	8206	8.13	54.1	3373	<NA>
35	11.59	4036	97.51	46.0	4111	wc
36	11.49	3148	95.97	41.9	4113	wc
37	11.32	4348	68.24	49.4	4131	wc
38	10.64	2448	91.76	42.3	4133	wc
39	11.36	4330	75.92	47.7	4143	wc
40	9.17	4761	11.37	30.9	4153	wc
41	12.09	3016	83.19	32.7	4161	wc
42	11.04	2901	92.86	38.7	4171	wc
43	9.22	5511	7.62	36.1	4172	wc
44	10.07	3739	52.27	37.2	4173	wc
45	10.51	3161	96.14	38.1	4175	wc
46	11.20	4741	47.06	29.4	4191	wc
47	11.13	5052	56.10	51.1	4192	wc
48	11.43	6259	39.17	35.7	4193	wc

49	11.00	4075	63.23	35.6	4197	wc
50	9.84	7482	17.04	41.5	5130	wc
51	11.13	8780	3.16	40.2	5133	wc
52	10.05	2594	67.82	26.5	5137	wc
53	9.62	918	7.00	14.8	5143	<NA>
54	9.93	2370	3.69	23.3	5145	bc
55	11.60	8131	13.09	47.3	5171	wc
56	11.09	6992	24.44	47.1	5172	wc
57	11.03	7956	23.88	51.1	5191	wc
58	9.47	8895	0.00	43.5	6111	bc
59	10.93	8891	1.65	51.6	6112	bc
60	7.74	3116	52.00	29.7	6121	bc
61	8.50	3930	15.51	20.2	6123	bc
62	10.57	7869	6.01	54.9	6141	bc
63	9.46	611	96.53	25.9	6147	<NA>
64	7.33	3000	69.31	20.8	6162	bc
65	7.11	3472	33.57	17.3	6191	bc
66	7.58	3582	30.08	20.1	6193	bc
67	6.84	3643	3.60	44.1	7112	<NA>

68	8.60	1656	27.75	21.5	7182	bc
69	8.88	6860	0.00	35.3	7711	bc
70	7.54	4199	33.30	38.9	8213	bc
71	7.64	5134	17.26	25.2	8215	bc
72	7.64	5134	17.26	34.8	8215	bc
73	7.42	1890	72.24	23.2	8221	bc
74	6.69	4443	31.36	33.3	8267	bc
75	6.74	3485	39.48	28.8	8278	bc
76	10.09	8043	1.50	42.5	8311	bc
77	8.81	6686	4.28	44.2	8313	bc
78	8.40	6565	2.30	35.9	8333	bc
79	7.92	6477	5.17	41.8	8335	bc
80	8.43	5811	13.62	35.9	8513	bc
81	8.78	6573	5.78	43.7	8515	bc
82	8.76	3942	74.54	50.8	8534	bc
83	10.29	5449	2.92	37.2	8537	bc
84	6.38	2847	90.67	28.2	8563	bc
85	8.10	5795	0.81	38.1	8581	bc
86	10.10	7716	0.78	50.3	8582	bc

```
 87       6.67   4696  0.00     27.3   8715   bc
 88       9.05   8316  1.34     40.9   8731   bc
 89       9.93   7147  0.99     50.2   8733   bc
 90       8.24   8880  0.65     51.1   8780   bc
 91       6.92   5299  0.56     38.9   8781   bc
 92       6.60   5959  0.52     36.2   8782   bc
 93       7.81   4549  2.46     29.9   8785   bc
 94       8.33   6928  0.61     42.9   8791   bc
 95       7.52   3910  1.09     26.5   8798   bc
 96      12.27  14032  0.58     66.1   9111 prof
 97       8.49   8845  0.00     48.9   9131   bc
 98       7.58   5562  9.47     35.9   9171   bc
 99       7.93   4224  3.59     25.1   9173   bc
100       8.37   4753  0.00     26.1   9313   bc
101      10.00   6462 13.58     42.2   9511   bc
102       8.55   3617 70.87     35.2   9517   bc
```

Pipes %>%

Sends the output of one command as the first argument of the next command

```
library(spida2)
lapply(dd, function(x) if(is.factor(x)) as.character(x) else x)
```

```
$education
  [1] 13.11 12.26 12.77 11.42 14.62 15.64 15.09 15.44 14.52
 [10] 14.64 12.39 12.30 13.83 14.44 14.36 14.21 15.77 14.15
 [19] 15.22 14.50 15.97 13.62 15.08 15.96 15.94 14.71 12.46
 [28]  9.45 13.62 15.21 12.79 11.09 12.71 11.44 11.59 11.49
 [37] 11.32 10.64 11.36  9.17 12.09 11.04  9.22 10.07 10.51
 [46] 11.20 11.13 11.43 11.00  9.84 11.13 10.05  9.62  9.93
 [55] 11.60 11.09 11.03  9.47 10.93  7.74  8.50 10.57  9.46
 [64]  7.33  7.11  7.58  6.84  8.60  8.88  7.54  7.64  7.64
 [73]  7.42  6.69  6.74 10.09  8.81  8.40  7.92  8.43  8.78
 [82]  8.76 10.29  6.38  8.10 10.10  6.67  9.05  9.93  8.24
 [91]  6.92  6.60  7.81  8.33  7.52 12.27  8.49  7.58  7.93
[100]  8.37 10.00  8.55

$income
```

```
  [1] 12351 25879  9271  8865  8403 11030  8258 14163 11377
 [10] 11023  5902  7059  8425  8049  7405  6336 19263  6112
 [19]  9593  4686 12480  5648  8034 25308 14558 17498  4614
 [28]  3485  5092 10432  5180  6197  7562  8206  4036  3148
 [37]  4348  2448  4330  4761  3016  2901  5511  3739  3161
 [46]  4741  5052  6259  4075  7482  8780  2594   918  2370
 [55]  8131  6992  7956  8895  8891  3116  3930  7869   611
 [64]  3000  3472  3582  3643  1656  6860  4199  5134  5134
 [73]  1890  4443  3485  8043  6686  6565  6477  5811  6573
 [82]  3942  5449  2847  5795  7716  4696  8316  7147  8880
 [91]  5299  5959  4549  6928  3910 14032  8845  5562  4224
[100]  4753  6462  3617

$women
  [1] 11.16  4.02 15.70  9.11 11.68  5.13 25.65  2.69  1.03
 [10]  0.94  1.91  7.83 15.33 57.31 48.28 54.77  5.13 77.10
 [19] 34.89  4.14 19.59 83.78 46.80 10.56  4.32  6.91 96.12
 [28] 76.14 82.66 24.71 76.04 21.03 11.15  8.13 97.51 95.97
 [37] 68.24 91.76 75.92 11.37 83.19 92.86  7.62 52.27 96.14
```

```
 [46] 47.06 56.10 39.17 63.23 17.04  3.16 67.82  7.00  3.69
 [55] 13.09 24.44 23.88  0.00  1.65 52.00 15.51  6.01 96.53
 [64] 69.31 33.57 30.08  3.60 27.75  0.00 33.30 17.26 17.26
 [73] 72.24 31.36 39.48  1.50  4.28  2.30  5.17 13.62  5.78
 [82] 74.54  2.92 90.67  0.81  0.78  0.00  1.34  0.99  0.65
 [91]  0.56  0.52  2.46  0.61  1.09  0.58  0.00  9.47  3.59
[100]  0.00 13.58 70.87

$prestige
  [1] 68.8 69.1 63.4 56.8 73.5 77.6 72.6 78.1 73.1 68.8 62.0
 [12] 60.0 53.8 62.2 74.9 55.1 82.3 58.1 58.3 72.8 84.6 59.6
 [23] 66.1 87.2 66.7 68.4 64.7 34.9 72.1 69.3 67.5 57.2 57.6
 [34] 54.1 46.0 41.9 49.4 42.3 47.7 30.9 32.7 38.7 36.1 37.2
 [45] 38.1 29.4 51.1 35.7 35.6 41.5 40.2 26.5 14.8 23.3 47.3
 [56] 47.1 51.1 43.5 51.6 29.7 20.2 54.9 25.9 20.8 17.3 20.1
 [67] 44.1 21.5 35.3 38.9 25.2 34.8 23.2 33.3 28.8 42.5 44.2
 [78] 35.9 41.8 35.9 43.7 50.8 37.2 28.2 38.1 50.3 27.3 40.9
 [89] 50.2 51.1 38.9 36.2 29.9 42.9 26.5 66.1 48.9 35.9 25.1
[100] 26.1 42.2 35.2
```

```
$census
  [1] 1113 1130 1171 1175 2111 2113 2133 2141 2143 2153 2161
 [12] 2163 2183 2311 2315 2331 2343 2351 2391 2511 2711 2731
 [23] 2733 3111 3115 3117 3131 3135 3137 3151 3156 3314 3337
 [34] 3373 4111 4113 4131 4133 4143 4153 4161 4171 4172 4173
 [45] 4175 4191 4192 4193 4197 5130 5133 5137 5143 5145 5171
 [56] 5172 5191 6111 6112 6121 6123 6141 6147 6162 6191 6193
 [67] 7112 7182 7711 8213 8215 8215 8221 8267 8278 8311 8313
 [78] 8333 8335 8513 8515 8534 8537 8563 8581 8582 8715 8731
 [89] 8733 8780 8781 8782 8785 8791 8798 9111 9131 9171 9173
[100] 9313 9511 9517

$type
  [1] "prof" "prof" "prof" "prof" "prof" "prof" "prof"
  [8] "prof" "prof" "prof" "prof" "prof" "prof" "prof"
 [15] "prof" "prof" "prof" "prof" "prof" "prof" "prof"
 [22] "prof" "prof" "prof" "prof" "prof" "prof" "bc"
 [29] "prof" "prof" "wc"   "prof" "wc"   NA     "wc"
```

```
 [36] "wc"   "wc"   "wc"   "wc"   "wc"   "wc"   "wc"
 [43] "wc"   "wc"   "wc"   "wc"   "wc"   "wc"   "wc"
 [50] "wc"   "wc"   "wc"   NA     "bc"   "wc"   "wc"
 [57] "wc"   "bc"   "bc"   "bc"   "bc"   "bc"   NA
 [64] "bc"   "bc"   "bc"   NA     "bc"   "bc"   "bc"
 [71] "bc"   "bc"   "bc"   "bc"   "bc"   "bc"   "bc"
 [78] "bc"   "bc"   "bc"   "bc"   "bc"   "bc"   "bc"
 [85] "bc"   "bc"   "bc"   "bc"   "bc"   "bc"   "bc"
 [92] "bc"   "bc"   "bc"   "bc"   "prof" "bc"   "bc"
 [99] "bc"   "bc"   "bc"   "bc"
```

```
as.data.frame(lapply(dd, function(x) if(is.factor(x)) as.character(x
```

```
    education income women prestige census type
1       13.11  12351 11.16     68.8   1113 prof
2       12.26  25879  4.02     69.1   1130 prof
3       12.77   9271 15.70     63.4   1171 prof
4       11.42   8865  9.11     56.8   1175 prof
5       14.62   8403 11.68     73.5   2111 prof
```

6	15.64	11030	5.13	77.6	2113	prof
7	15.09	8258	25.65	72.6	2133	prof
8	15.44	14163	2.69	78.1	2141	prof
9	14.52	11377	1.03	73.1	2143	prof
10	14.64	11023	0.94	68.8	2153	prof
11	12.39	5902	1.91	62.0	2161	prof
12	12.30	7059	7.83	60.0	2163	prof
13	13.83	8425	15.33	53.8	2183	prof
14	14.44	8049	57.31	62.2	2311	prof
15	14.36	7405	48.28	74.9	2315	prof
16	14.21	6336	54.77	55.1	2331	prof
17	15.77	19263	5.13	82.3	2343	prof
18	14.15	6112	77.10	58.1	2351	prof
19	15.22	9593	34.89	58.3	2391	prof
20	14.50	4686	4.14	72.8	2511	prof
21	15.97	12480	19.59	84.6	2711	prof
22	13.62	5648	83.78	59.6	2731	prof
23	15.08	8034	46.80	66.1	2733	prof
24	15.96	25308	10.56	87.2	3111	prof

25	15.94	14558	4.32	66.7	3115	prof
26	14.71	17498	6.91	68.4	3117	prof
27	12.46	4614	96.12	64.7	3131	prof
28	9.45	3485	76.14	34.9	3135	bc
29	13.62	5092	82.66	72.1	3137	prof
30	15.21	10432	24.71	69.3	3151	prof
31	12.79	5180	76.04	67.5	3156	wc
32	11.09	6197	21.03	57.2	3314	prof
33	12.71	7562	11.15	57.6	3337	wc
34	11.44	8206	8.13	54.1	3373	<NA>
35	11.59	4036	97.51	46.0	4111	wc
36	11.49	3148	95.97	41.9	4113	wc
37	11.32	4348	68.24	49.4	4131	wc
38	10.64	2448	91.76	42.3	4133	wc
39	11.36	4330	75.92	47.7	4143	wc
40	9.17	4761	11.37	30.9	4153	wc
41	12.09	3016	83.19	32.7	4161	wc
42	11.04	2901	92.86	38.7	4171	wc
43	9.22	5511	7.62	36.1	4172	wc

44	10.07	3739	52.27	37.2	4173	wc
45	10.51	3161	96.14	38.1	4175	wc
46	11.20	4741	47.06	29.4	4191	wc
47	11.13	5052	56.10	51.1	4192	wc
48	11.43	6259	39.17	35.7	4193	wc
49	11.00	4075	63.23	35.6	4197	wc
50	9.84	7482	17.04	41.5	5130	wc
51	11.13	8780	3.16	40.2	5133	wc
52	10.05	2594	67.82	26.5	5137	wc
53	9.62	918	7.00	14.8	5143	<NA>
54	9.93	2370	3.69	23.3	5145	bc
55	11.60	8131	13.09	47.3	5171	wc
56	11.09	6992	24.44	47.1	5172	wc
57	11.03	7956	23.88	51.1	5191	wc
58	9.47	8895	0.00	43.5	6111	bc
59	10.93	8891	1.65	51.6	6112	bc
60	7.74	3116	52.00	29.7	6121	bc
61	8.50	3930	15.51	20.2	6123	bc
62	10.57	7869	6.01	54.9	6141	bc

63	9.46	611	96.53	25.9	6147	<NA>
64	7.33	3000	69.31	20.8	6162	bc
65	7.11	3472	33.57	17.3	6191	bc
66	7.58	3582	30.08	20.1	6193	bc
67	6.84	3643	3.60	44.1	7112	<NA>
68	8.60	1656	27.75	21.5	7182	bc
69	8.88	6860	0.00	35.3	7711	bc
70	7.54	4199	33.30	38.9	8213	bc
71	7.64	5134	17.26	25.2	8215	bc
72	7.64	5134	17.26	34.8	8215	bc
73	7.42	1890	72.24	23.2	8221	bc
74	6.69	4443	31.36	33.3	8267	bc
75	6.74	3485	39.48	28.8	8278	bc
76	10.09	8043	1.50	42.5	8311	bc
77	8.81	6686	4.28	44.2	8313	bc
78	8.40	6565	2.30	35.9	8333	bc
79	7.92	6477	5.17	41.8	8335	bc
80	8.43	5811	13.62	35.9	8513	bc
81	8.78	6573	5.78	43.7	8515	bc

82	8.76	3942	74.54	50.8	8534	bc
83	10.29	5449	2.92	37.2	8537	bc
84	6.38	2847	90.67	28.2	8563	bc
85	8.10	5795	0.81	38.1	8581	bc
86	10.10	7716	0.78	50.3	8582	bc
87	6.67	4696	0.00	27.3	8715	bc
88	9.05	8316	1.34	40.9	8731	bc
89	9.93	7147	0.99	50.2	8733	bc
90	8.24	8880	0.65	51.1	8780	bc
91	6.92	5299	0.56	38.9	8781	bc
92	6.60	5959	0.52	36.2	8782	bc
93	7.81	4549	2.46	29.9	8785	bc
94	8.33	6928	0.61	42.9	8791	bc
95	7.52	3910	1.09	26.5	8798	bc
96	12.27	14032	0.58	66.1	9111	prof
97	8.49	8845	0.00	48.9	9131	bc
98	7.58	5562	9.47	35.9	9171	bc
99	7.93	4224	3.59	25.1	9173	bc
100	8.37	4753	0.00	26.1	9313	bc

```
  101      10.00    6462 13.58      42.2    9511    bc
  102       8.55    3617 70.87      35.2    9517    bc
```

```
sapply(lapply(dd, function(x) if(is.factor(x)) as.character(x) else
```

```
    education      income       women    prestige      census
    "numeric"   "integer"   "numeric"   "numeric"   "integer"
         type
  "character"
```

Using pipes

```
lapply(dd, function(x) if(is.factor(x)) as.character(x) else x) %>%
    as.data.frame
```

```
      education income women prestige census type
  1       13.11  12351 11.16     68.8   1113 prof
  2       12.26  25879  4.02     69.1   1130 prof
  3       12.77   9271 15.70     63.4   1171 prof
  4       11.42   8865  9.11     56.8   1175 prof
  5       14.62   8403 11.68     73.5   2111 prof
```

6	15.64	11030	5.13	77.6	2113	prof
7	15.09	8258	25.65	72.6	2133	prof
8	15.44	14163	2.69	78.1	2141	prof
9	14.52	11377	1.03	73.1	2143	prof
10	14.64	11023	0.94	68.8	2153	prof
11	12.39	5902	1.91	62.0	2161	prof
12	12.30	7059	7.83	60.0	2163	prof
13	13.83	8425	15.33	53.8	2183	prof
14	14.44	8049	57.31	62.2	2311	prof
15	14.36	7405	48.28	74.9	2315	prof
16	14.21	6336	54.77	55.1	2331	prof
17	15.77	19263	5.13	82.3	2343	prof
18	14.15	6112	77.10	58.1	2351	prof
19	15.22	9593	34.89	58.3	2391	prof
20	14.50	4686	4.14	72.8	2511	prof
21	15.97	12480	19.59	84.6	2711	prof
22	13.62	5648	83.78	59.6	2731	prof
23	15.08	8034	46.80	66.1	2733	prof
24	15.96	25308	10.56	87.2	3111	prof

25	15.94	14558	4.32	66.7	3115	prof
26	14.71	17498	6.91	68.4	3117	prof
27	12.46	4614	96.12	64.7	3131	prof
28	9.45	3485	76.14	34.9	3135	bc
29	13.62	5092	82.66	72.1	3137	prof
30	15.21	10432	24.71	69.3	3151	prof
31	12.79	5180	76.04	67.5	3156	wc
32	11.09	6197	21.03	57.2	3314	prof
33	12.71	7562	11.15	57.6	3337	wc
34	11.44	8206	8.13	54.1	3373	<NA>
35	11.59	4036	97.51	46.0	4111	wc
36	11.49	3148	95.97	41.9	4113	wc
37	11.32	4348	68.24	49.4	4131	wc
38	10.64	2448	91.76	42.3	4133	wc
39	11.36	4330	75.92	47.7	4143	wc
40	9.17	4761	11.37	30.9	4153	wc
41	12.09	3016	83.19	32.7	4161	wc
42	11.04	2901	92.86	38.7	4171	wc
43	9.22	5511	7.62	36.1	4172	wc

44	10.07	3739	52.27	37.2	4173	wc
45	10.51	3161	96.14	38.1	4175	wc
46	11.20	4741	47.06	29.4	4191	wc
47	11.13	5052	56.10	51.1	4192	wc
48	11.43	6259	39.17	35.7	4193	wc
49	11.00	4075	63.23	35.6	4197	wc
50	9.84	7482	17.04	41.5	5130	wc
51	11.13	8780	3.16	40.2	5133	wc
52	10.05	2594	67.82	26.5	5137	wc
53	9.62	918	7.00	14.8	5143	<NA>
54	9.93	2370	3.69	23.3	5145	bc
55	11.60	8131	13.09	47.3	5171	wc
56	11.09	6992	24.44	47.1	5172	wc
57	11.03	7956	23.88	51.1	5191	wc
58	9.47	8895	0.00	43.5	6111	bc
59	10.93	8891	1.65	51.6	6112	bc
60	7.74	3116	52.00	29.7	6121	bc
61	8.50	3930	15.51	20.2	6123	bc
62	10.57	7869	6.01	54.9	6141	bc

63	9.46	611	96.53	25.9	6147	<NA>
64	7.33	3000	69.31	20.8	6162	bc
65	7.11	3472	33.57	17.3	6191	bc
66	7.58	3582	30.08	20.1	6193	bc
67	6.84	3643	3.60	44.1	7112	<NA>
68	8.60	1656	27.75	21.5	7182	bc
69	8.88	6860	0.00	35.3	7711	bc
70	7.54	4199	33.30	38.9	8213	bc
71	7.64	5134	17.26	25.2	8215	bc
72	7.64	5134	17.26	34.8	8215	bc
73	7.42	1890	72.24	23.2	8221	bc
74	6.69	4443	31.36	33.3	8267	bc
75	6.74	3485	39.48	28.8	8278	bc
76	10.09	8043	1.50	42.5	8311	bc
77	8.81	6686	4.28	44.2	8313	bc
78	8.40	6565	2.30	35.9	8333	bc
79	7.92	6477	5.17	41.8	8335	bc
80	8.43	5811	13.62	35.9	8513	bc
81	8.78	6573	5.78	43.7	8515	bc

82	8.76	3942	74.54	50.8	8534	bc
83	10.29	5449	2.92	37.2	8537	bc
84	6.38	2847	90.67	28.2	8563	bc
85	8.10	5795	0.81	38.1	8581	bc
86	10.10	7716	0.78	50.3	8582	bc
87	6.67	4696	0.00	27.3	8715	bc
88	9.05	8316	1.34	40.9	8731	bc
89	9.93	7147	0.99	50.2	8733	bc
90	8.24	8880	0.65	51.1	8780	bc
91	6.92	5299	0.56	38.9	8781	bc
92	6.60	5959	0.52	36.2	8782	bc
93	7.81	4549	2.46	29.9	8785	bc
94	8.33	6928	0.61	42.9	8791	bc
95	7.52	3910	1.09	26.5	8798	bc
96	12.27	14032	0.58	66.1	9111	prof
97	8.49	8845	0.00	48.9	9131	bc
98	7.58	5562	9.47	35.9	9171	bc
99	7.93	4224	3.59	25.1	9173	bc
100	8.37	4753	0.00	26.1	9313	bc

```
  101      10.00    6462 13.58      42.2    9511    bc
  102       8.55    3617 70.87      35.2    9517    bc
```

```
lapply(dd, function(x) if(is.factor(x)) as.character(x) else x) %>%
    as.data.frame %>%
    sapply(class)
```

```
    education      income       women    prestige      census
    "numeric"   "integer"   "numeric"   "numeric"   "integer"
         type
  "character"
```

Other trick to replace dd

```
dd[] <- lapply(dd, function(x) if(is.factor(x)) as.character(x) else
dd # note that dd is still a data.frame and not a list
```

```
                          education income women prestige
  gov.administrators          13.11  12351 11.16     68.8
  general.managers            12.26  25879  4.02     69.1
  accountants                 12.77   9271 15.70     63.4
```

purchasing.officers	11.42	8865	9.11	56.8
chemists	14.62	8403	11.68	73.5
physicists	15.64	11030	5.13	77.6
biologists	15.09	8258	25.65	72.6
architects	15.44	14163	2.69	78.1
civil.engineers	14.52	11377	1.03	73.1
mining.engineers	14.64	11023	0.94	68.8
surveyors	12.39	5902	1.91	62.0
draughtsmen	12.30	7059	7.83	60.0
computer.programers	13.83	8425	15.33	53.8
economists	14.44	8049	57.31	62.2
psychologists	14.36	7405	48.28	74.9
social.workers	14.21	6336	54.77	55.1
lawyers	15.77	19263	5.13	82.3
librarians	14.15	6112	77.10	58.1
vocational.counsellors	15.22	9593	34.89	58.3
ministers	14.50	4686	4.14	72.8
university.teachers	15.97	12480	19.59	84.6
primary.school.teachers	13.62	5648	83.78	59.6

secondary.school.teachers	15.08	8034	46.80	66.1
physicians	15.96	25308	10.56	87.2
veterinarians	15.94	14558	4.32	66.7
osteopaths.chiropractors	14.71	17498	6.91	68.4
nurses	12.46	4614	96.12	64.7
nursing.aides	9.45	3485	76.14	34.9
physio.therapsts	13.62	5092	82.66	72.1
pharmacists	15.21	10432	24.71	69.3
medical.technicians	12.79	5180	76.04	67.5
commercial.artists	11.09	6197	21.03	57.2
radio.tv.announcers	12.71	7562	11.15	57.6
athletes	11.44	8206	8.13	54.1
secretaries	11.59	4036	97.51	46.0
typists	11.49	3148	95.97	41.9
bookkeepers	11.32	4348	68.24	49.4
tellers.cashiers	10.64	2448	91.76	42.3
computer.operators	11.36	4330	75.92	47.7
shipping.clerks	9.17	4761	11.37	30.9
file.clerks	12.09	3016	83.19	32.7

receptionsts	11.04	2901	92.86	38.7
mail.carriers	9.22	5511	7.62	36.1
postal.clerks	10.07	3739	52.27	37.2
telephone.operators	10.51	3161	96.14	38.1
collectors	11.20	4741	47.06	29.4
claim.adjustors	11.13	5052	56.10	51.1
travel.clerks	11.43	6259	39.17	35.7
office.clerks	11.00	4075	63.23	35.6
sales.supervisors	9.84	7482	17.04	41.5
commercial.travellers	11.13	8780	3.16	40.2
sales.clerks	10.05	2594	67.82	26.5
newsboys	9.62	918	7.00	14.8
service.station.attendant	9.93	2370	3.69	23.3
insurance.agents	11.60	8131	13.09	47.3
real.estate.salesmen	11.09	6992	24.44	47.1
buyers	11.03	7956	23.88	51.1
firefighters	9.47	8895	0.00	43.5
policemen	10.93	8891	1.65	51.6
cooks	7.74	3116	52.00	29.7

bartenders	8.50	3930	15.51	20.2
funeral.directors	10.57	7869	6.01	54.9
babysitters	9.46	611	96.53	25.9
launderers	7.33	3000	69.31	20.8
janitors	7.11	3472	33.57	17.3
elevator.operators	7.58	3582	30.08	20.1
farmers	6.84	3643	3.60	44.1
farm.workers	8.60	1656	27.75	21.5
rotary.well.drillers	8.88	6860	0.00	35.3
bakers	7.54	4199	33.30	38.9
slaughterers.1	7.64	5134	17.26	25.2
slaughterers.2	7.64	5134	17.26	34.8
canners	7.42	1890	72.24	23.2
textile.weavers	6.69	4443	31.36	33.3
textile.labourers	6.74	3485	39.48	28.8
tool.die.makers	10.09	8043	1.50	42.5
machinists	8.81	6686	4.28	44.2
sheet.metal.workers	8.40	6565	2.30	35.9
welders	7.92	6477	5.17	41.8

auto.workers	8.43	5811	13.62	35.9
aircraft.workers	8.78	6573	5.78	43.7
electronic.workers	8.76	3942	74.54	50.8
radio.tv.repairmen	10.29	5449	2.92	37.2
sewing.mach.operators	6.38	2847	90.67	28.2
auto.repairmen	8.10	5795	0.81	38.1
aircraft.repairmen	10.10	7716	0.78	50.3
railway.sectionmen	6.67	4696	0.00	27.3
electrical.linemen	9.05	8316	1.34	40.9
electricians	9.93	7147	0.99	50.2
construction.foremen	8.24	8880	0.65	51.1
carpenters	6.92	5299	0.56	38.9
masons	6.60	5959	0.52	36.2
house.painters	7.81	4549	2.46	29.9
plumbers	8.33	6928	0.61	42.9
construction.labourers	7.52	3910	1.09	26.5
pilots	12.27	14032	0.58	66.1
train.engineers	8.49	8845	0.00	48.9
bus.drivers	7.58	5562	9.47	35.9

taxi.drivers	7.93	4224	3.59	25.1
longshoremen	8.37	4753	0.00	26.1
typesetters	10.00	6462	13.58	42.2
bookbinders	8.55	3617	70.87	35.2

	census	type
gov.administrators	1113	prof
general.managers	1130	prof
accountants	1171	prof
purchasing.officers	1175	prof
chemists	2111	prof
physicists	2113	prof
biologists	2133	prof
architects	2141	prof
civil.engineers	2143	prof
mining.engineers	2153	prof
surveyors	2161	prof
draughtsmen	2163	prof
computer.programers	2183	prof
economists	2311	prof

```
psychologists                2315 prof
social.workers               2331 prof
lawyers                      2343 prof
librarians                   2351 prof
vocational.counsellors       2391 prof
ministers                    2511 prof
university.teachers          2711 prof
primary.school.teachers      2731 prof
secondary.school.teachers    2733 prof
physicians                   3111 prof
veterinarians                3115 prof
osteopaths.chiropractors     3117 prof
nurses                       3131 prof
nursing.aides                3135   bc
physio.therapsts             3137 prof
pharmacists                  3151 prof
medical.technicians          3156   wc
commercial.artists           3314 prof
radio.tv.announcers          3337   wc
```

athletes	3373	<NA>
secretaries	4111	wc
typists	4113	wc
bookkeepers	4131	wc
tellers.cashiers	4133	wc
computer.operators	4143	wc
shipping.clerks	4153	wc
file.clerks	4161	wc
receptionsts	4171	wc
mail.carriers	4172	wc
postal.clerks	4173	wc
telephone.operators	4175	wc
collectors	4191	wc
claim.adjustors	4192	wc
travel.clerks	4193	wc
office.clerks	4197	wc
sales.supervisors	5130	wc
commercial.travellers	5133	wc
sales.clerks	5137	wc

newsboys	5143	<NA>
service.station.attendant	5145	bc
insurance.agents	5171	wc
real.estate.salesmen	5172	wc
buyers	5191	wc
firefighters	6111	bc
policemen	6112	bc
cooks	6121	bc
bartenders	6123	bc
funeral.directors	6141	bc
babysitters	6147	<NA>
launderers	6162	bc
janitors	6191	bc
elevator.operators	6193	bc
farmers	7112	<NA>
farm.workers	7182	bc
rotary.well.drillers	7711	bc
bakers	8213	bc
slaughterers.1	8215	bc

slaughterers.2	8215	bc
canners	8221	bc
textile.weavers	8267	bc
textile.labourers	8278	bc
tool.die.makers	8311	bc
machinists	8313	bc
sheet.metal.workers	8333	bc
welders	8335	bc
auto.workers	8513	bc
aircraft.workers	8515	bc
electronic.workers	8534	bc
radio.tv.repairmen	8537	bc
sewing.mach.operators	8563	bc
auto.repairmen	8581	bc
aircraft.repairmen	8582	bc
railway.sectionmen	8715	bc
electrical.linemen	8731	bc
electricians	8733	bc
construction.foremen	8780	bc

```
carpenters                8781   bc
masons                    8782   bc
house.painters            8785   bc
plumbers                  8791   bc
construction.labourers    8798   bc
pilots                    9111 prof
train.engineers           9131   bc
bus.drivers               9171   bc
taxi.drivers              9173   bc
longshoremen              9313   bc
typesetters               9511   bc
bookbinders               9517   bc
```

```
sapply(dd, class)
```

```
    education        income         women      prestige        census
    "numeric"     "integer"     "numeric"     "numeric"     "integer"
         type
  "character"
```