Working with R

A University of Queensland Advanced Workshop

Session 2: Objects and Their Manipulation

Bill Venables, CSIRO/Data61, Dutton Park

Rhetta Chappell, Griffith University

2–5 February, 2021

Contents

1 Objects and their attributes

"In **S** *everything* is an object and *every* object has a *class*."
John Chambers, c. 1996.

R is a *language* for *manipulating objects*.
(me)

1.1 Atomic and recursive objects

There are five kinds of *atomic* object. In a sense, all other objects are made from these.

`numeric` Vectors of numerical quantities.

`character` Vectors of *character strings*.

`logical` Vectors with components either `TRUE` or `FALSE`.

`complex` Vectors of complex numbers (each component a pair of numerical quantities).

`raw` Vectors of raw bites, mainly used with compiled code.

All have the extra possibility of a *missing value marker* in place of a component. This is shown either as `NA` or `<NA>`.

Other types of objects are known as *recursive*

`list` objects: `data.frame`, fitted model objects, loads of other stuff....

`date` objects and their allies: `POSIXct`, `POSIXlt`, `Date`, ...

`language` objects: `formula`, `call`, `expression`, ...

`environment` objects: the most mysterious of all!

Even *atomic* objects are more than they seem.

- All objects have two *intrinsic attributes*, namely `mode` and `length`.
- Objects can have many *assigned* attributes as well, such as `class`, `names`, `names`, `dimnames`, &c.
- Attributes *self-describe* the object.
- Object attributes can *allow* some operations and *inhibit* others.

Examples

```
set.seed(12345)
x <- runif(16) %>% round(2)
dim(x) <- c(4,4)
dimnames(x) <- list(rows = letters[1:4], columns = LETTERS[1:4]); x[1:2, ]

    columns
rows    A    B    C    D
   a 0.72 0.46 0.73 0.74
   b 0.88 0.17 0.99 0.00

attributes(x)

$dim
[1] 4 4


$dimnames
$dimnames$rows
[1] "a" "b" "c" "d"


$dimnames$columns
[1] "A" "B" "C" "D"
```

```r
c(length(x), mode(x), class(x))
```

```
[1] "16"      "numeric" "matrix"  "array"
```

```r
tx <- as.table(x) %>% as.data.frame(responseName = "values")
tx              ## turn a numeric matrix into a 'long form' data frame
```

```
   rows columns values
1     a       A   0.72
2     b       A   0.88
3     c       A   0.76
....
14    b       D   0.00
15    c       D   0.39
16    d       D   0.46
```

```r
class(tx)
```

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

```r
x <- as.vector(x)  ## clear all applied attributes
attributes(x)
```

```
NULL
```

```
f <- factor(fill0(round(10*x)))  ## see note to follow on fill0(...)
table(f)

f
00 02 03 04 05 07 08 09 10
 2  2  1  1  3  3  1  2  1

attributes(f); mode(f)

$levels
[1] "00" "02" "03" "04" "05" "07" "08" "09" "10"


$class
[1] "factor"
[1] "numeric"

f+1       ## arithmetic with factors is inhibited!

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

Turning a factor back into a numeric vector: two possibilities

```
as.character(f)
```

```
 [1] "07" "09" "08" "09" "05" "02" "03" "05" "07" "10" "00" "02" "07" "00"
[15] "04" "05"
```

```
rbind(A = as.numeric(f),
      B = as.numeric(as.character(f))) %>%
  booktabs(digits=0)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	6	8	7	8	5	2	3	5	6	9	1	2	6	1	4	5
B	7	9	8	9	5	2	3	5	7	10	0	2	7	0	4	5

Filling with zeros

- Filling up numerical strings with initial zeros to a constant number of digits can preserve lexicographic ordering in line with numeric.
- The `WWRCourse` package has two functions that behave identically, but are coded very differently:

```
##
## Method 1: using string substitution
fill0 <- function(x) {
  gsub(" ", "0", format(x, justify = "right"))
}
##
## Method 2: using character size computations and paste
zfill <- function(x) {
  m <- max(n <- nchar(x <- as.character(x)))
  paste0(strrep(0, m-n), x)
}
```

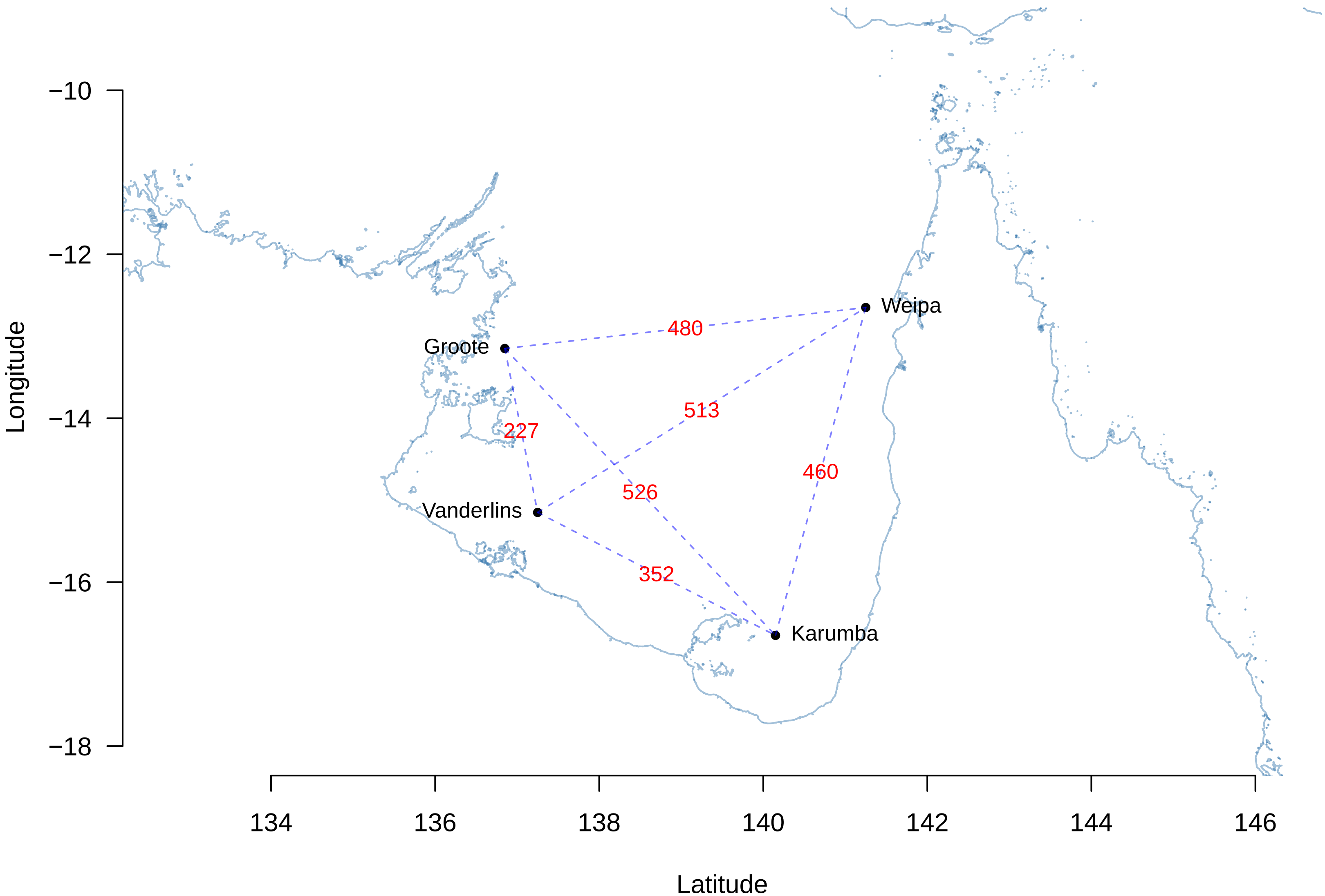
- Make sure you know how both work!

Complex types

The forgotten atomic type?

For four given locations in the Gulf of Carpentaria, find the shortest marine distances between each of them and show on a map.

```r
with(data4, {
  blueish <- alpha("steelblue", 0.5)
  z <- complex(real = Longitude, imaginary = Latitude)
  plot(z, asp = 1, xlim = c(135, 144), ylim = c(-18, -9),
       bty = "n", xlab = "Latitude", ylab = "Longitude")
  lines(Oz, col = blueish)
  text(z, Place, pos = c(2,2,4,4), cex=0.8)
  ij <- utils::combn(4,2)
  i <- ij[1, ]
  j <- ij[2, ]
  segments(z[i], z[j], lty = "dashed", col = alpha("blue", 0.5))
  text((z[i]+z[j])/2, round(gcd_km(z[i], z[j])), col="red", cex=0.8)
})
```



Weipa
Groote
Vanderlins
Karumba
480
513
227
460
526
352
−10
−12
−14
−16
−18
Longitude
134
136
138
140
142
144
146
Latitude

2 A real pain in the neck

- We use some "typical" data[a] on the outcome of a physiotherapy project to study two ways of ameliorating neck pain, mainly in office workers.
- Two groups:
  - Control: Given advice on neck pain management
  - Intervention: Given an exercise programme and advice.
- Five assessment times: 0, 3, 6, 9 and 12 months. A *longitudinal* study.
- Several explanatory variables: `Age`, `Sex`, `Occupation`, &c.
- Data supplied as a `stata` binary, available in the `WWRCourse` package as extra data folder, *extdata/stata*.

[a]Kindly supplied by Dr Xiaoqi Chen, UQ

2.1 Reading in the data

- The modern tool collection, ("*tidyverse*"), supplies a number of packages for specific data input jobs:

  readr For reading mainly text files, especially `.csv` files.

  readxl For reading mainly **Microsoft** Excel files.

  haven For reading a range of foreign system files, *including* *stata* *binaries*.

- All three produce as the result of their main functions a peculiarly enhanced *data.frame* known (annoyingly) as a "`tibble`"[a].

  Fortunately it is easy to turn a `tibble` back into a normal `data.frame`, and for the most part, we shall.

- We will use the "chaining" operator, *%>%*, (often incorrectly called a 'pipe', but we do it too!) throughout this session without further comment, for clarity.

---

[a]No, it's not a cat.

```r
library(haven)
NPStudy <- read_stata(system.file("extdata", "stata", "NeckPainStudy.dta",
                                  package = "WWRCourse"))
sapply(NPStudy, class)
```

```
$idd
[1] "numeric"

$grp
[1] "haven_labelled" "vctrs_vctr"     "double"
....
$pain_num_9m
[1] "numeric"

$pain_cat_9m
[1] "numeric"
```

`labelled` columns will become `character`;
`numeric` columns need to be cleared.

2.2 Key functions

The *dplyr* package supplies five *key* functions, namely

filter For selecting rows (cf. *subset*)

select For selecting columns (cf. *subset(...,select = ...)*)

mutate (sic!) For computations on columns (cf. *within()*)

arrange For ordering rows of a data set

summarise Together with *group_by*, for summarising data sets in various ways. (cf. *tapply()*.)

do While not a key function, often useful. (cf, *by()* in old speak)

```r
NPStudy <- read_stata(system.file("extdata", "stata", "NeckPainStudy.dta",
                                   package = "WWRCourse")) %>%
  lapply(function(x) if(is.labelled(x)) {
    as_factor(x)
  } else {
    as.vector(x)
  }) %>% untibble()
Store(NPStudy)  ## for safe keeping
```

```r
NeckPain <- NPStudy %>%
    select(-contains("_cat_")) %>% ## remove categoricals
  within({
    Ident <-   paste0("S", fill0(idd))      %>% factor()
    Cluster <- paste0("C", fill0(cluster)) %>% factor()
  }) %>%
  rename(Sex = sex, Group = grp, Organisation = organisation,
         Industry = industry, Ergo = ergo, Age = age,
         BMI = bmi, Education = edu, Occupation = occ,
         Comorbidity = cm) %>%
  select(Ident, Cluster, Group, Organisation:Occupation,
                Comorbidity, pain_num_b:pain_num_9m)
```

2.3 Missing values – to plug or not to plug?

```
colSums(is.na(NeckPain))

       Ident      Cluster        Group Organisation     Industry
           0            0            0            0            0
        Ergo          Age          BMI          Sex    Education
          30            6            2            0            0
  Occupation  Comorbidity   pain_num_b  pain_num_3m pain_num_12m
           0            0            0          150          357
 pain_num_6m  pain_num_9m
         394          466
```

These are a problem. In an exploratory analysis it seems useful:

- To plug missing values, if there are not too many, in the *explanatory* variables *only*.
- To do this, use *only* information in the *explanatory* variables, **not** the responses.
- Don't forget this is *only* for exploratory purposes!

We will use an imputation process based on random forest technology. This is provided in the workshop software.

```
set.seed(20200211)
Extract <- NeckPain %>% select(Group:Comorbidity) %>%  ## only these
  rfImputeUnsupervised() %>%                       ## The black box plugger
  within({                                         ## do a bit of cleaning-up
    Ergo <- round(Ergo)                            ## make things look innocent
    Age <- round(Age)
    BMI <- round(BMI, 2)
  })
NeckPain[, names(Extract)] <- Extract   ## plug the gaps
rm(Extract)                             ## destroy the evidence...
colSums(is.na(NeckPain))                ## check all is OK

        Ident       Cluster         Group Organisation      Industry
            0             0             0            0             0
         Ergo           Age           BMI          Sex     Education
            0             0             0            0             0
   Occupation   Comorbidity    pain_num_b  pain_num_3m  pain_num_12m
            0             0             0          150           357
  pain_num_6m   pain_num_9m
          394           466
```

2.4 Wide form to long form

The *tidyr* package is used for re-shaping data sets, most commonly "wide form" to or from "long form".

Two main functions: *pivot_longer* and *pivot_wider*.

```
longNeckPain <- NeckPain %>%
  pivot_longer(pain_num_b:pain_num_9m,
               names_to = "Time", values_to = "NPain") %>%
  within({
    time <- Time                      %>% ## currently a mess
      sub("b$", "0", .)               %>% ## final 'b' -> 0
      gsub("[^[:digit:]]", "", .) %>% ## ditch any non-digit
      as.numeric()                        ## coerce to a number
    Time <- ordered(paste0("T", fill0(time)))
    Treat <- ifelse(time == 0, "Base", paste0(substring(Group, 0, 1),
                                              substring(Time, 2)))
  }) %>% select(Ident:Group, Time, time, Treat,
                Organisation:Comorbidity, NPain) %>%
  arrange(Ident, Time) %>%  na.omit() %>% untibble()
Store(NeckPain, longNeckPain)
```

What have we got? Let's see a few bits:

```
longNeckPain %>%
  select(Ident, Group:Treat, Ergo:Sex, NPain)

     Ident         Group Time time Treat Ergo Age   BMI    Sex NPain
1      S001      Control  T00    0  Base   27  50 26.56 female   2.0
2      S001      Control  T03    3   C03   27  50 26.56 female   0.0
3      S002 Intervention  T00    0  Base   28  33 23.15 female   0.0
4      S002 Intervention  T03    3   I03   28  33 23.15 female   0.0
5      S002 Intervention  T06    6   I06   28  33 23.15 female   0.0
....
2328   S761      Control  T06    6   C06   37  30 29.40 female   2.0
2329   S761      Control  T09    9   C09   37  30 29.40 female   6.0
2330   S761      Control  T12   12   C12   37  30 29.40 female   1.0
2331   S762      Control  T00    0  Base   31  26 21.60   male   7.0
2332   S763      Control  T00    0  Base   34  38 34.05 female   5.0
2333   S763      Control  T03    3   C03   34  38 34.05 female   3.0
```

What is the *Treat* factor?

```
with(longNeckPain,
     tapply(Treat, list(Group, Time), unique)) %>%
  booktabs()                                     ## for a neat result!
```

	T00	T03	T06	T09	T12
Control	Base	C03	C06	C09	C12
Intervention	Base	I03	I06	I09	I12

It reflects the fact that at time 0 no intervention has been applied at all: both groups supply a baseline neck pain level to which the other "treatments" may be compared.

2.5 Looking at the data

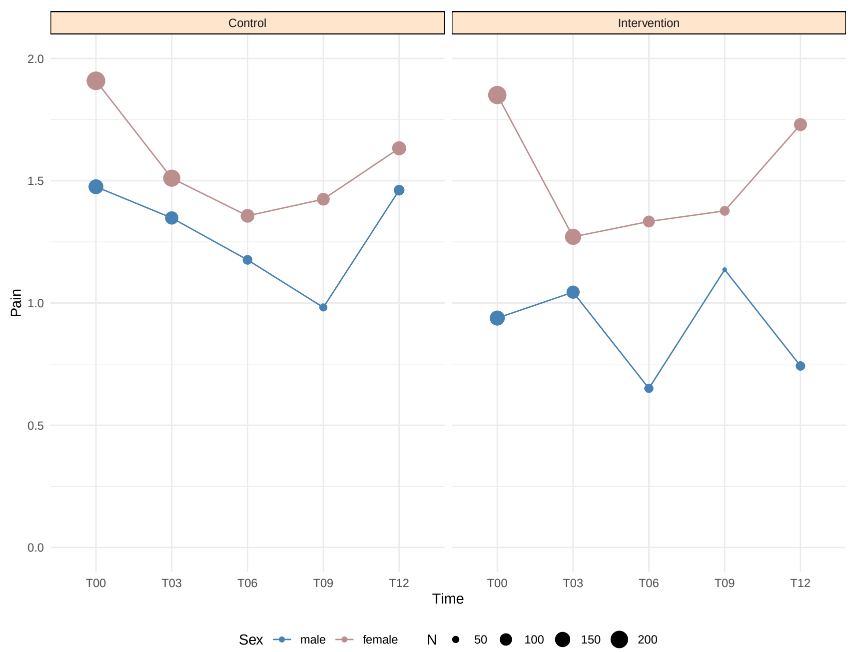
Ignore most explanatory variables and look and numbers and means, first.

```
meanNPain <- longNeckPain                                              %>%
  group_by(Group, Time, Treat, Sex)                                    %>%
  summarise(N = n(), Pain = mean(NPain), .groups = "drop") %>%
  ungroup()                                                            %>%
  untibble()                                                           %>%
  arrange(Treat, Sex)
meanNPain
```

```
          Group Time Treat    Sex   N      Pain
1       Control  T00  Base   male 143 1.4755245
2  Intervention  T00  Base   male 146 0.9383562
3       Control  T00  Base female 230 1.9086957
4  Intervention  T00  Base female 221 1.8506787
....
17 Intervention  T09   I09   male  44 1.1363636
18 Intervention  T09   I09 female  69 1.3768116
19 Intervention  T12   I12   male  66 0.7424242
20 Intervention  T12   I12 female 111 1.7297297
```

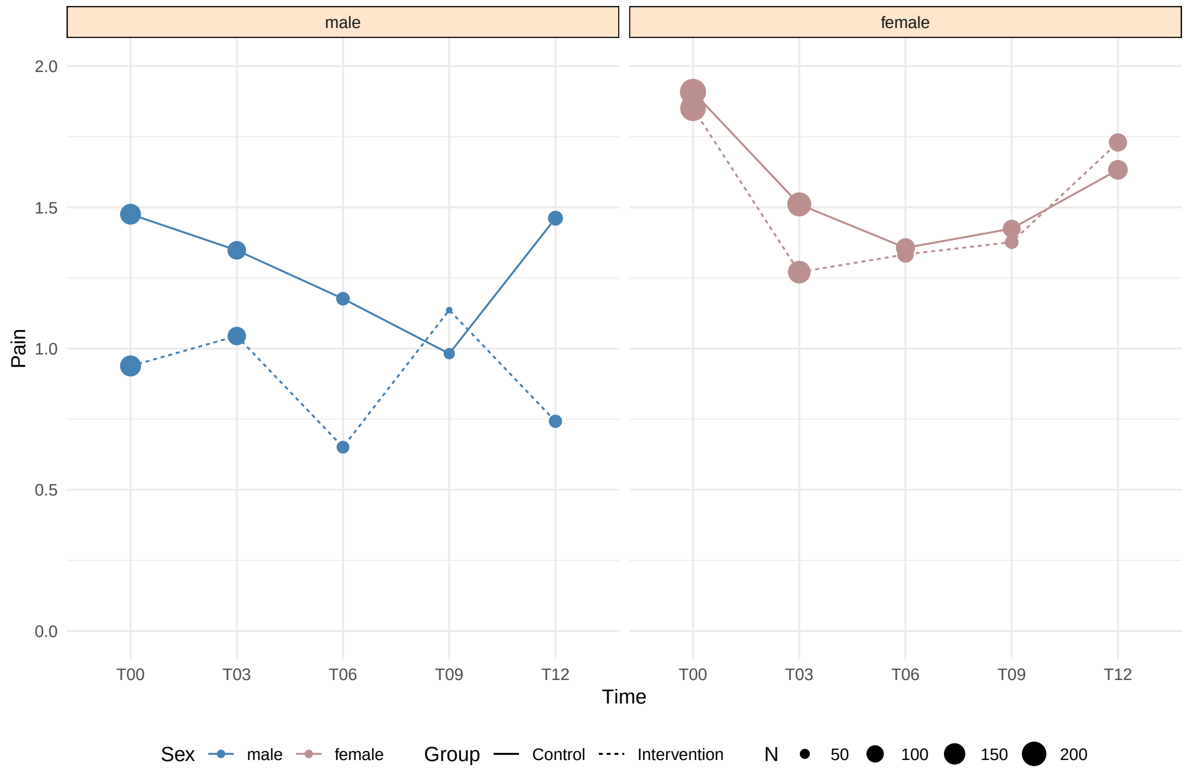
The mean neck pain score is low, (fortunately!), but how does it vary?

```
sex_cols <- c(male = "steelblue", female = "rosybrown")
ggplot(meanNPain)                                         +
  aes(x = Time, y = Pain, colour = Sex, group = Sex) +
  geom_point(aes(size = N))                               +
  geom_line()                                             +
  facet_wrap(~Group, ncol=2)                              +
  theme(legend.position = "bottom")                       +
  ylim(0, 2)                                              +
  scale_colour_manual(values = sex_cols)
```



To put it the other way:

```
ggplot(meanNPain) + aes(x = Time, y = Pain, colour = Sex,
                        linetype = Group, group = Group) +
  geom_point(aes(size = N)) + geom_line() + facet_wrap(~Sex, ncol=2) +
  theme(legend.position = "bottom") + ylim(0, 2) +
  scale_colour_manual(values = sex_cols)
```



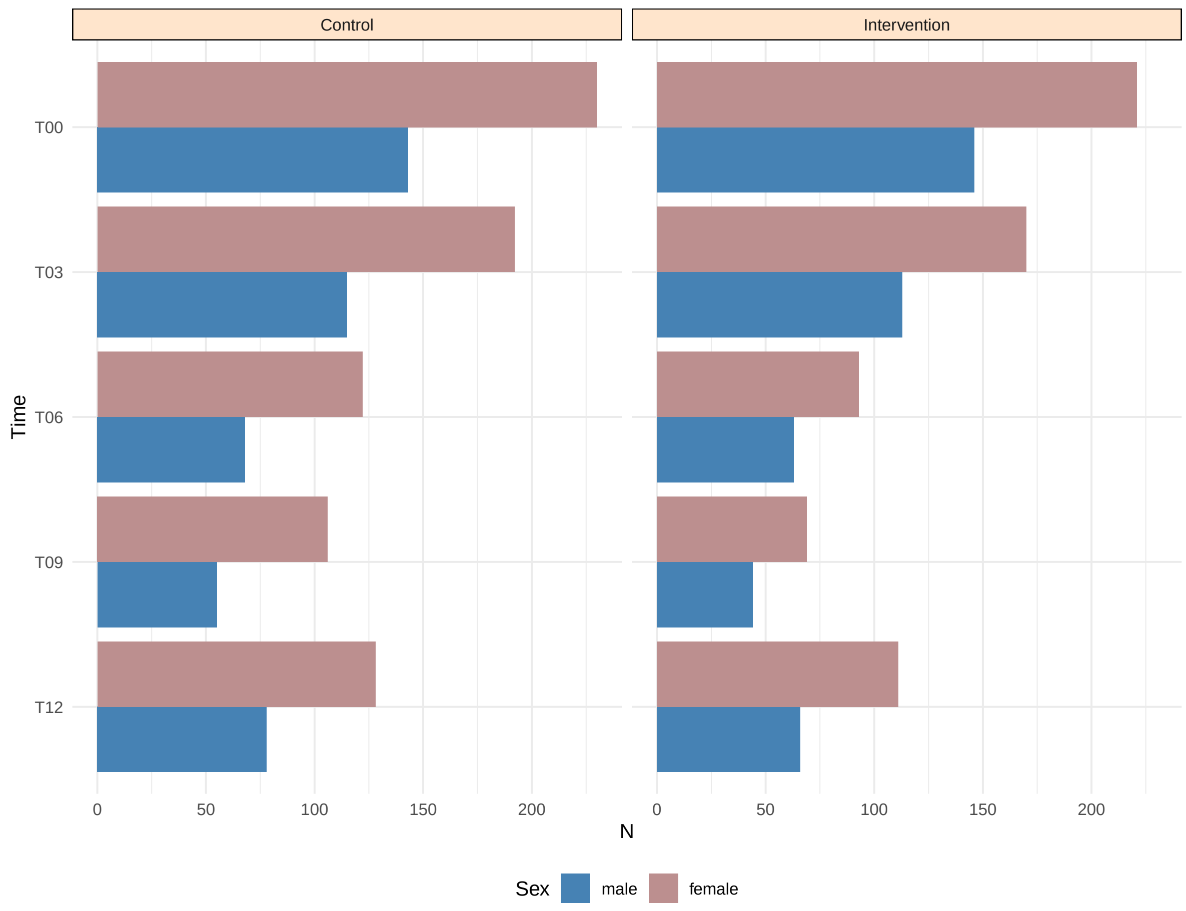
How many people stuck with the programme?

```
meanNPain                              %>%
  pivot_wider(id_cols = c(Group, Sex),  ## These columns identify rows
              names_from = Time,
              values_from = N) %>% booktabs
```

Group	Sex	T00	T03	T06	T09	T12
Control	male	143	115	68	55	78
Intervention	male	146	113	63	44	66
Control	female	230	192	122	106	128
Intervention	female	221	170	93	69	111

Some drop-off mid-programme, but some return at the end of the year. A picture makes this much easier to notice:

```
mNPain <- meanNPain %>% within({
  Time <- factor(as.character(Time), levels = rev(levels(Time)))
})
ggplot(mNPain) + aes(x = Time, y = N,  fill = Sex) +
  geom_bar(stat = "identity", position = "dodge") + coord_flip() +
  facet_wrap(~ Group) + theme(legend.position = "bottom") +
  scale_fill_manual(values = sex_cols)
```

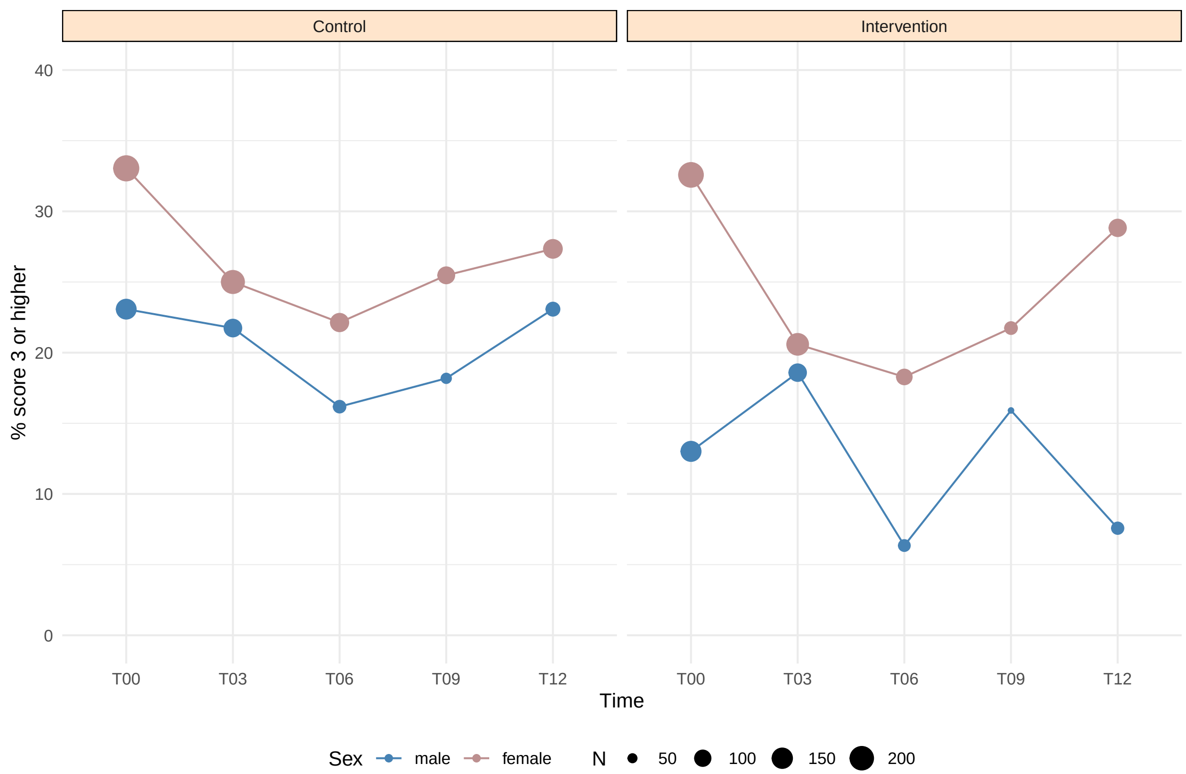

Control
Intervention
T00
T03
T06
T09
T12
Time
0
50
100
150
200
N
Sex
male
female

Rather than mean pain level, the researchers were more interested in whether or nor participants exceeded a pain threshold score of 3 or more.

```
percentNPain <- longNeckPain                                   %>%
  group_by(Group, Time, Treat, Sex)                            %>%
  summarise(N = n(), Severe = mean(NPain >= 3) * 100,
            .groups = "drop")                                  %>%
  ungroup()                                                    %>%
  arrange(Treat, Sex)                                          %>%
  untibble()
percentNPain                                                   %>%
  pivot_wider(id_cols = Group:Treat, names_from = Sex,
              values_from = Severe)                            %>% booktabs()
```

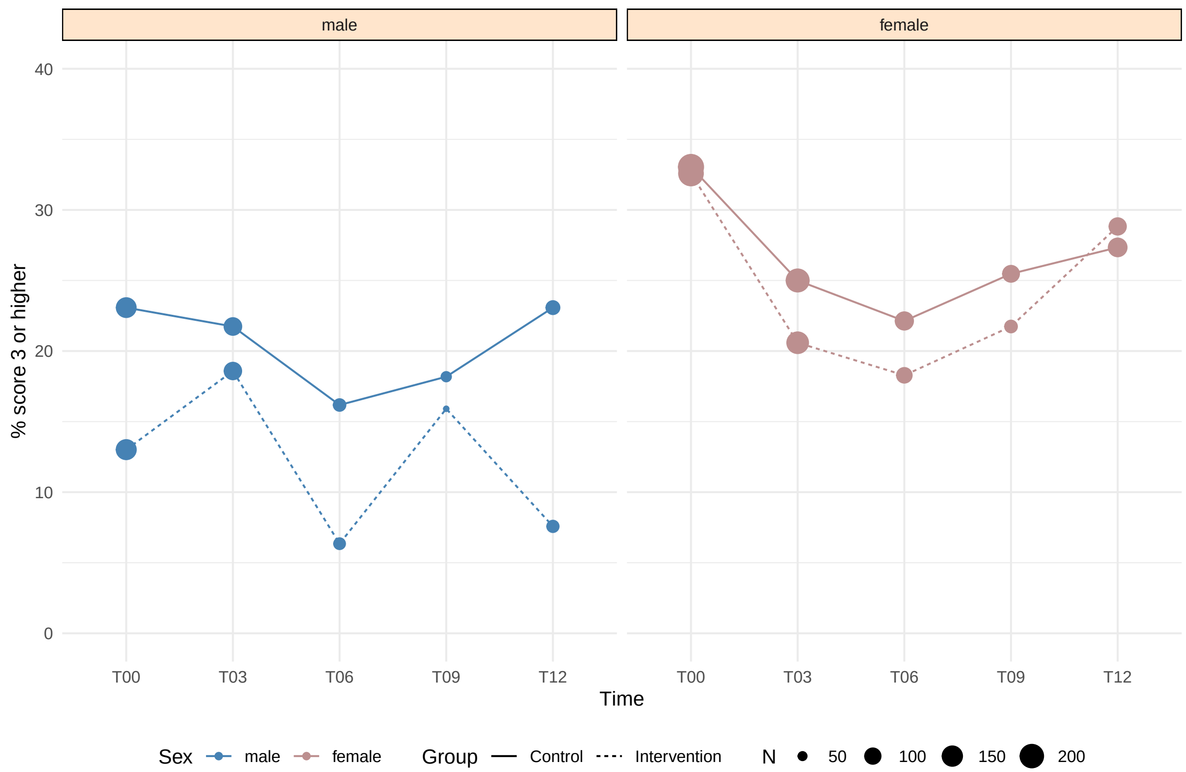
Group	Time	Treat	male	female
Control	T00	Base	23.08	33.04
Intervention	T00	Base	13.01	32.58
Control	T03	C03	21.74	25.00
Control	T06	C06	16.18	22.13
Control	T09	C09	18.18	25.47
Control	T12	C12	23.08	27.34
Intervention	T03	I03	18.58	20.59
Intervention	T06	I06	6.35	18.28
Intervention	T09	I09	15.91	21.74
Intervention	T12	I12	7.58	28.83

```r
ggplot(percentNPain) + aes(x = Time, y = Severe, colour = Sex, group = Sex) +
  geom_point(aes(size = N)) + geom_line() + facet_wrap(~Group, ncol=2) +
  theme(legend.position="bottom") + ylab("% score 3 or higher") +
  ylim(0, 40) + scale_colour_manual(values = c(male = "steelblue",
                                                female = "rosybrown"))
```



Or, to put it another way:

```
ggplot(percentNPain) + aes(x = Time, y = Severe, colour = Sex,
                           linetype = Group, group = Group) +
  geom_point(aes(size = N)) + geom_line() + facet_wrap(~Sex, ncol=2) +
  theme(legend.position="bottom") + ylab("% score 3 or higher") +
  ylim(0, 40) + scale_colour_manual(values = sex_cols)
```



3 Another example: How fast do you speak?

We use a data set described and provided in the package `hqmisc`.[a] Our version is called *talkers*.

From the help information on the package:

```
talkers                  package:hqmisc                 R Documentation


     A data frame with 80 observations on the following 6 variables.
     'id' identifier code (from data source, see Source)
     'sex' sex (0=female, 1=male)
     'age' age (in years)
     'region' region of origin (a factor with levels 'M'=Mid,
          'N'=North, 'S'=South, or 'W'=West)
     'syldur' average duration of syllables, or seconds per syllable
          (in seconds, excluding pause time, 1/(articulation rate) )
     'nsyl' average number of syllables per phrase, or average phrase
          length in syllables
```

[a] Which is not included in the set of packages needed for this course.

We begin by setting up a more convenient version of the data.

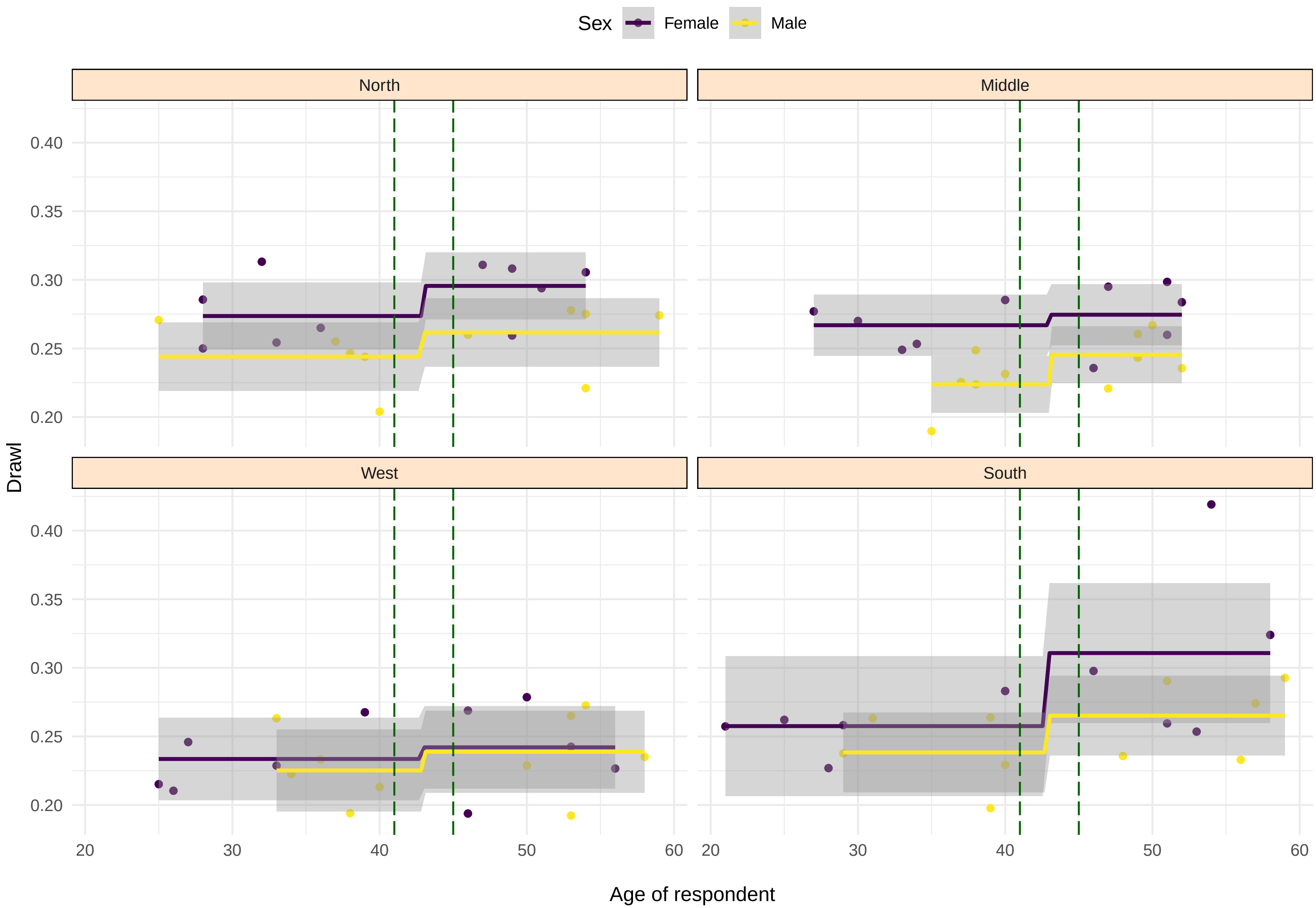
```
dutchSpeakers <- talkers %>% within({
  id <- paste0("S", zfill(id))   ## function in WWRSoftware
  sex <- recode_factor(sex, `0` = "Female", `1` = "Male")
  AgeGroup <- factor(ifelse(age > 42, "45+", "40-"),
                     levels = c("40-", "45+"))
  region <- factor(as.character(region),
                   levels = c("N",     "M",      "W",    "S"),
                   labels = c("North", "Middle", "West", "South"))
}) %>%
  rename(Ident = id, Region = region, Sex = sex, ## New = old
         Drawl = syldur, Verbose = nsyl) %>%
  arrange(Region, Sex, age)
```

It is important to check this all went according to plan.

```
dutchSpeakers  ### look at it

   Ident    Sex age Region  Drawl Verbose AgeGroup
1   S119 Female  28  North 0.2856    7.85      40-
2   S124 Female  28  North 0.2500   12.47      40-
3   S125 Female  32  North 0.3132    7.63      40-
4   S134 Female  33  North 0.2543    6.96      40-
5   S123 Female  36  North 0.2650   10.79      40-
6   S120 Female  47  North 0.3109    7.55      45+
....
75  S114   Male  40  South 0.2293   12.35      40-
76  S148   Male  48  South 0.2357    6.17      45+
77  S147   Male  51  South 0.2904    6.13      45+
78  S144   Male  56  South 0.2330    6.19      45+
79  S149   Male  57  South 0.2741    6.71      45+
80  S142   Male  59  South 0.2927    5.45      45+
```

Look at the data graphically. How *Drawl* vary with *Sex*, *Region* and *AgeGroup*?

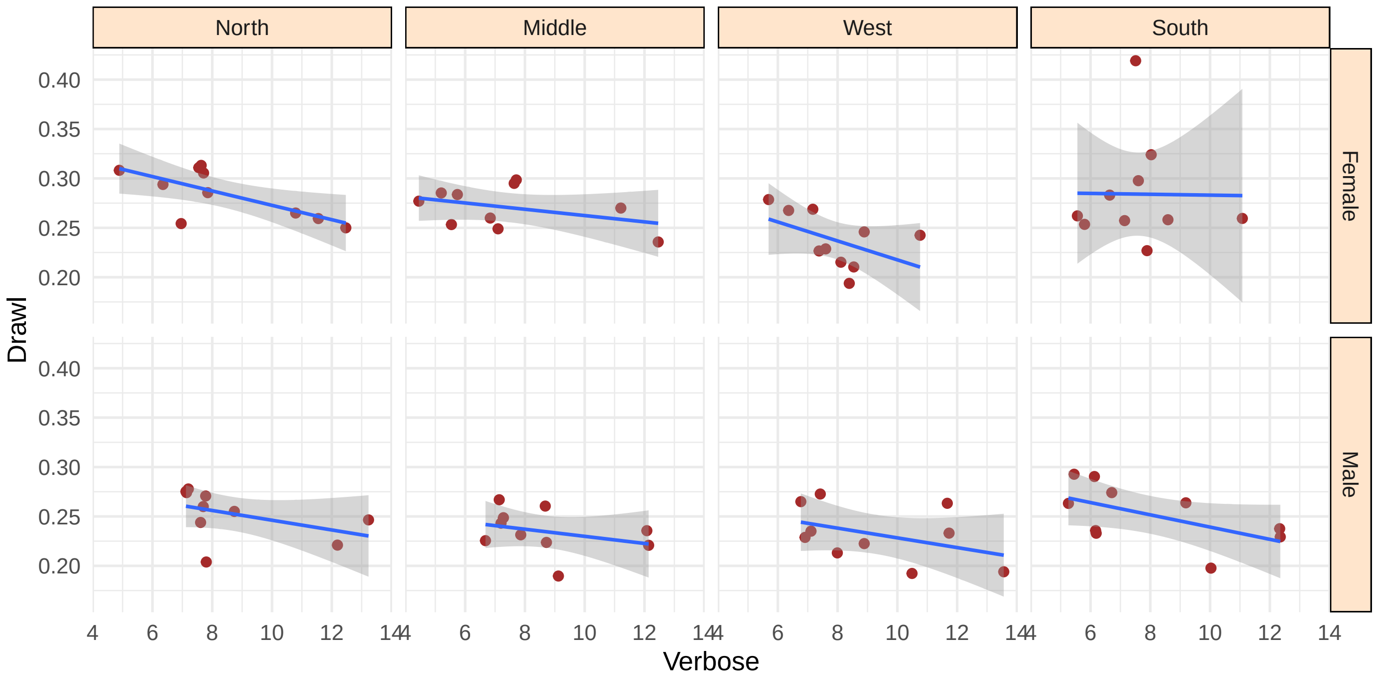
```
ggplot(dutchSpeakers) + aes(x = age, y = Drawl, colour = Sex) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ (x < 43)) +
  theme(legend.position = "top") +
  geom_vline(xintercept = c(41, 45), linetype = "longdash",
             colour = "dark green") + xlab("\nAge of respondent") +
  facet_wrap( ~ Region) +
  scale_colour_viridis_d()
```



Sex
Female
Male
North
Middle
West
South
Drawl
0.40
0.35
0.30
0.25
0.20
20
30
40
50
60
Age of respondent

Does the drawl depend on the prolixity?

```
p0 <- ggplot(dutchSpeakers) + aes(Verbose, Drawl) +
  geom_point(colour="brown") +
  geom_smooth(method="lm",formula=y~x, size=0.7)
p0 + facet_grid(Sex ~ Region)
```



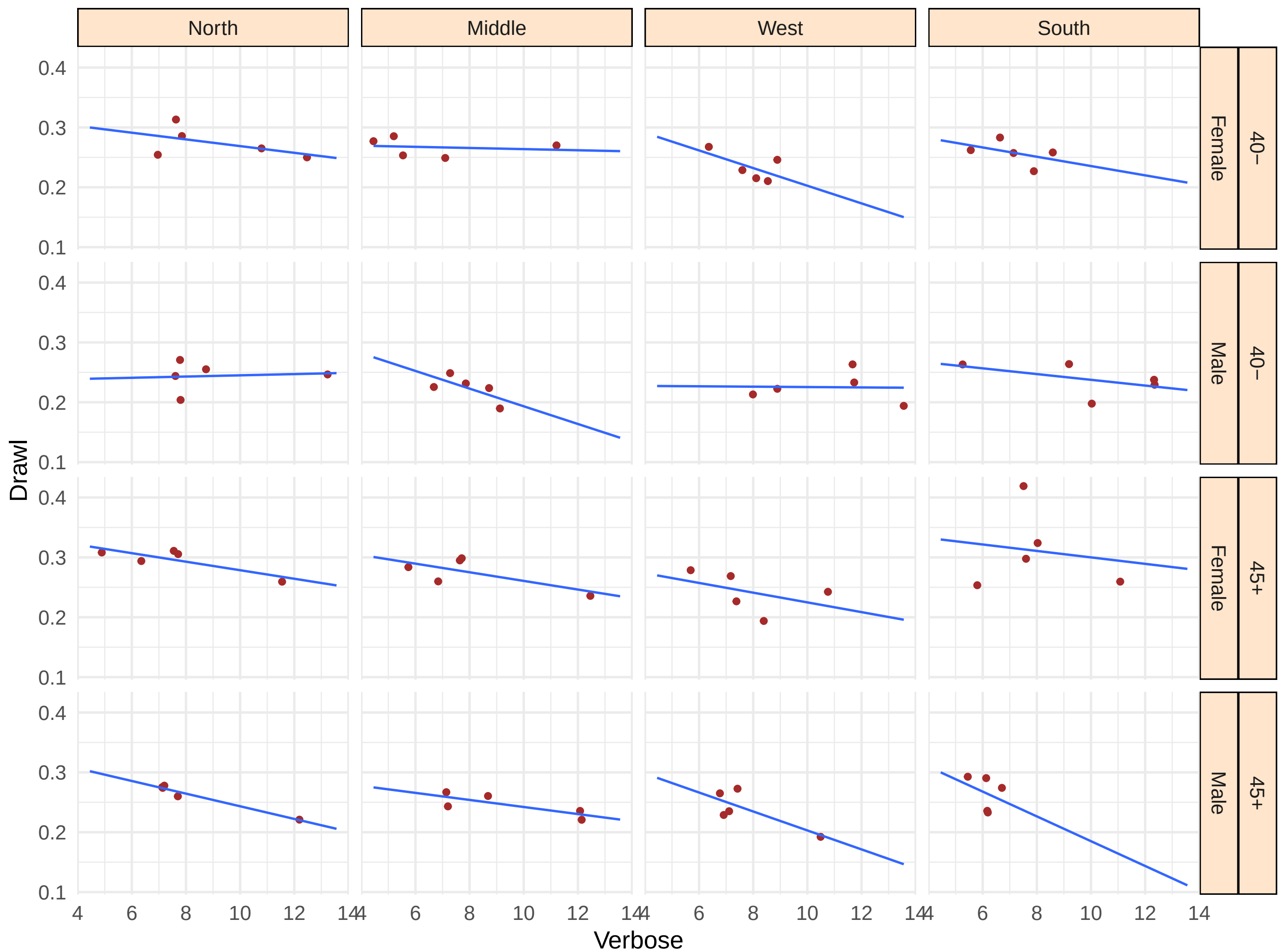
Generally, drawl falls with prolixity: people using longer phrases tend to utter their individual syllables more quickly.

There is a very slow-speaking female from the South region.

Further break-up of the sample by age group will result in groups of size 5, which may be rather small to see patterns.

Or will it?

```
ggplot(dutchSpeakers) + aes(x = Verbose, y = Drawl) +
    geom_point(colour = "brown", size = 1) +
    geom_smooth(method = "lm", se = FALSE, fullrange = TRUE, size=0.5,
                formula = y~x) +
    facet_grid(AgeGroup + Sex ~ Region)
```



Young people talk faster than old?

Males speak more quickly than females?

Regional differences are present, but hard to categorize?

3.1 A modelling investigation

```
dsModel <- aov(Drawl ~ Verbose+Sex+AgeGroup+Region, dutchSpeakers)
anova(dsModel) %>%
  booktabs(digits = c(0,0,4,4,2,4)) ## for display only.  Cut these bits
```

Df	Sum Sq	Mean Sq	F value	Pr(>F)
1	0.0144	0.0144	18.57	0.0001
1	0.0095	0.0095	12.32	0.0008
1	0.0066	0.0066	8.54	0.0046
3	0.0139	0.0046	6.01	0.0010
73	0.0564	0.0008		

Because the design is very close to orthogonal, the order in which the terms enter the model is not important.

```
dropterm(dsModel)  ## WWRUtilities enhanced version

Single term deletions


Model:
Drawl ~ Verbose + Sex + AgeGroup + Region
         Df Sum of Sq       RSS     AIC F Value     Pr(F)
<none>                 0.056402 -566.58
Verbose   1 0.0059815 0.062384 -560.52  7.7418 0.0068635
AgeGroup  1 0.0067715 0.063174 -559.51  8.7642 0.0041414
Sex       1 0.0102496 0.066652 -555.22 13.2658 0.0005023
Region    3 0.0139359 0.070338 -554.92  6.0123 0.0010189
```

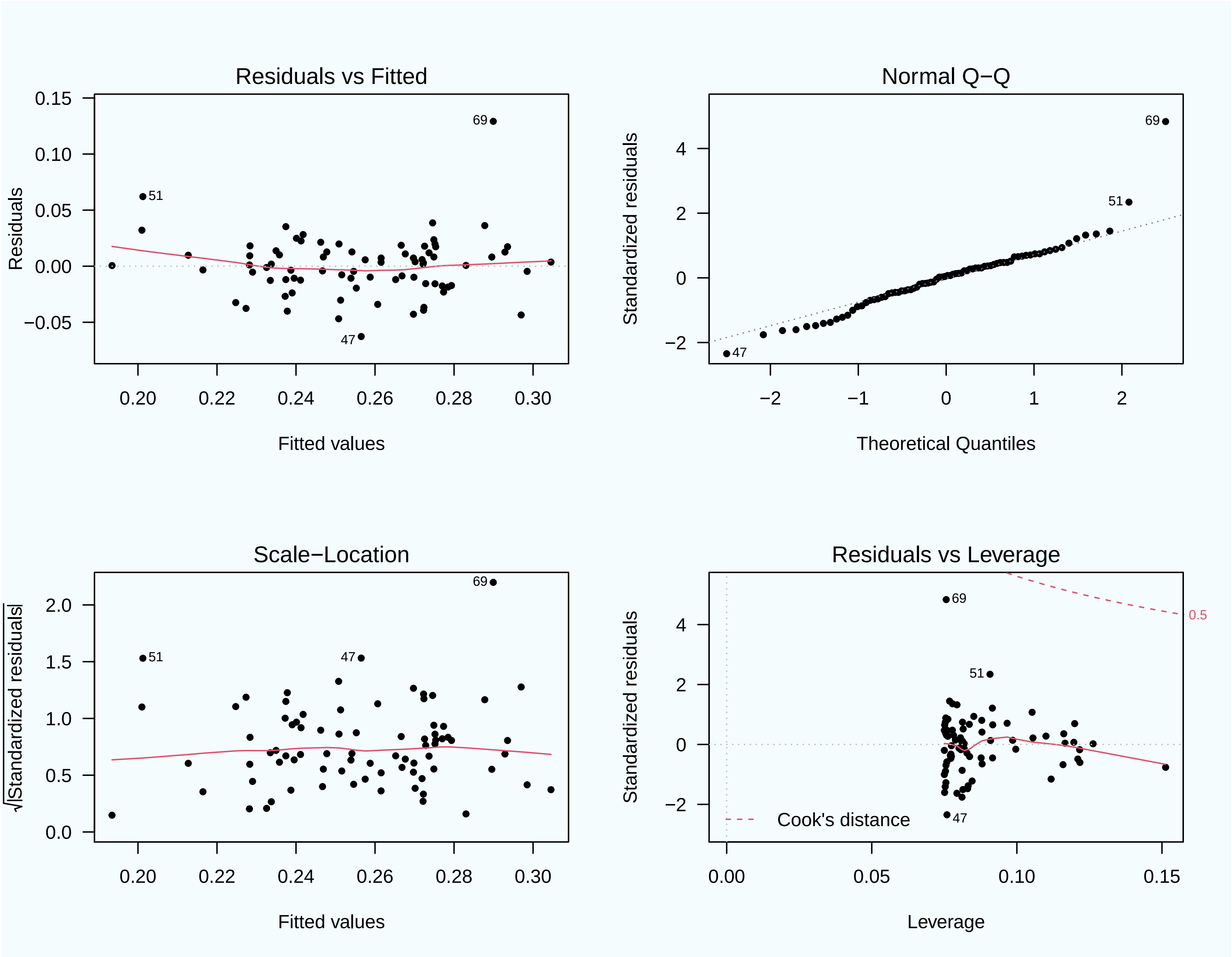
Are the coefficients consistent with what we saw in the graphics?

```
round(summary.lm(dsModel)$coeff, 4) ## check on signs and sizes

             Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.3061     0.0146 20.8989   0.0000
Verbose       -0.0041     0.0015 -2.7824   0.0069
SexMale       -0.0231     0.0063 -3.6422   0.0005
AgeGroup45+    0.0186     0.0063  2.9604   0.0041
RegionMiddle  -0.0180     0.0088 -2.0408   0.0449
RegionWest    -0.0336     0.0088 -3.8188   0.0003
RegionSouth   -0.0038     0.0089 -0.4271   0.6705

#### manual residual analysis
diagDutchSpeakers <- data.frame(rs = scale(resid(dsModel)),
                                fv = fitted(dsModel))
```

3.2 Diagnostics

For diagnostic checks, traditional graphics are simpler and usually adequate. The simplest way is to "plot" the fitted model object:

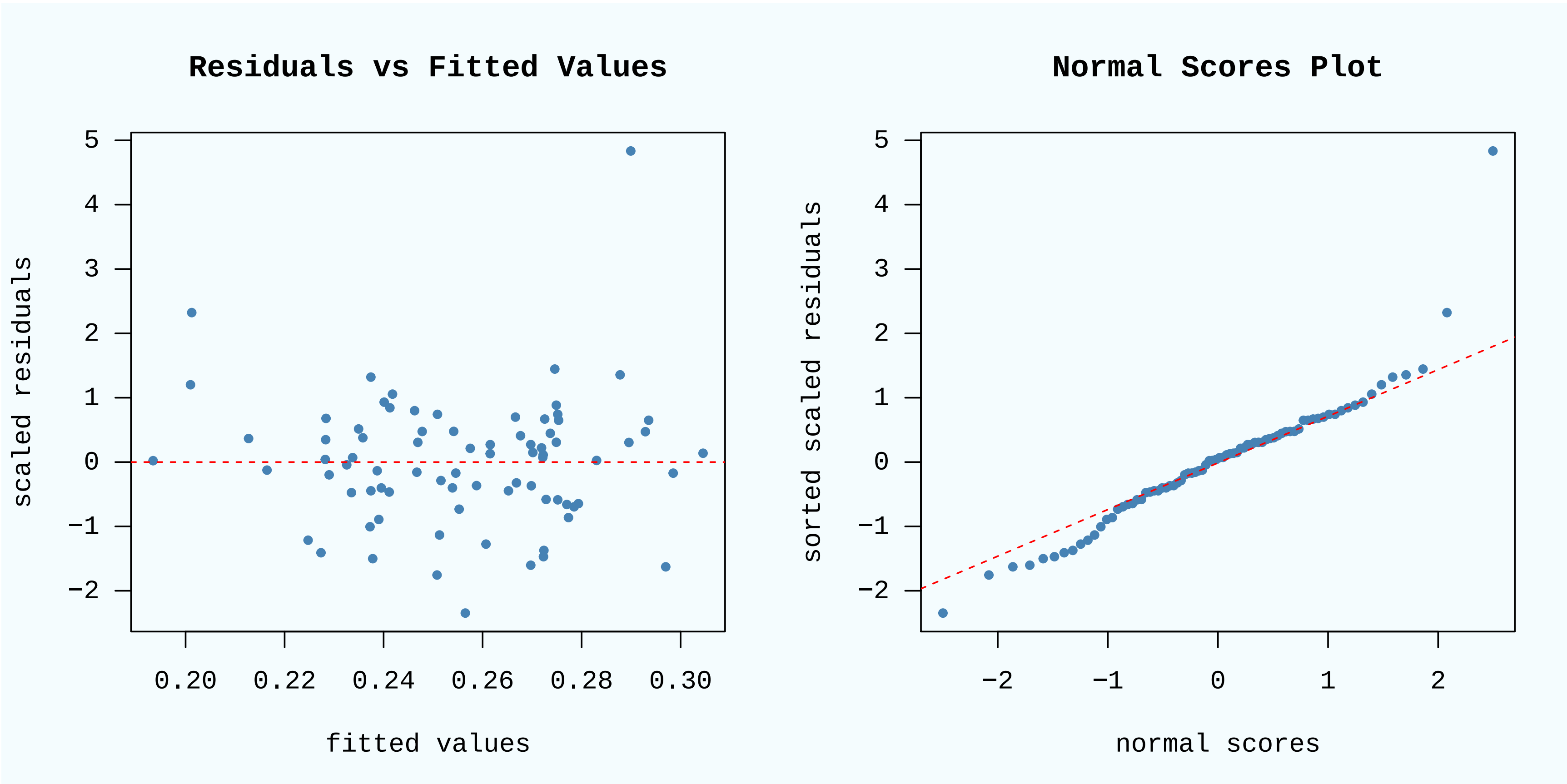
```
layout(rbind(1:2, 3:4))  ## 2 x 2 array of plots, filled by rows
plot(dsModel)
```



Residuals vs Fitted
Residuals
Fitted values
69
51
47
0.15
0.10
0.05
0.00
−0.05
0.20
0.22
0.24
0.26
0.28
0.30
Normal Q−Q
Standardized residuals
Theoretical Quantiles
69
51
47
4
2
0
−2
−1
1
Scale−Location
√|Standardized residuals|
Fitted values
69
51
47
2.0
1.5
1.0
0.5
0.0
Residuals vs Leverage
Standardized residuals
Leverage
Cook's distance
0.5
69
51
47
0.00
0.05
0.10
0.15

We see what's goint on a bit better if we do it by hand:

```r
layout(rbind(1:2))
par(bg = "#F4FCFE", family="mono")  ## mundane, old-fashioned look...
plot(rs ~ fv, diagDutchSpeakers,  col = "steel blue",
     xlab = "fitted values", ylab = "scaled residuals",
     main = "Residuals vs Fitted Values")
abline(h = 0, col = "red", lty = "dashed")
with(diagDutchSpeakers, {
  qqnorm(rs, xlab = "normal scores", col = "steel blue",
         ylab = "sorted scaled residuals",
         main = "Normal Scores Plot")
  qqline(rs, col = "red", lty = "dashed")
})
```



The slow speaking person shows up as a clear outlier.

We should check the extent to which our results are influenced by it.

We could omit it an see if anything changes.

A bette approach is to use a robust method and see what it suggests.

3.3 The robust alternative

With statistical modelling:

- You usually believe the data, and the challenge is to construct a model which will separate the true signals from the noise as well as possible.
- With robust methods, you usually have a good idea of what an appropriate model should be, but you have grave doubts about some of the data. The challenge is to identify the *consistent core of good data* so that your model estimates, and hence your conclusions, are not misleading.
- Using both methods in tandem involves a philosophical inconsistency and needs much caution!
- There is no fully reliable way to identify outliers from the data itself alone: *context is always important*.

Session information

Date: 2021-01-29

- R version 4.0.3 (2020-10-10), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 20.04.1 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: dplyr 1.0.3, english 1.2-5, forcats 0.5.1, ggplot2 3.3.3, ggthemes 4.2.4, gridExtra 2.3, haven 2.3.1, knitr 1.31, lattice 0.20-41, patchwork 1.1.1, purrr 0.3.4, readr 1.4.0, scales 1.1.1, stringr 1.4.0, tibble 3.0.5, tidyr 1.1.2, tidyverse 1.3.0, WWRCourse 0.2.3, WWRData 0.1.0, WWRGraphics 0.1.2, WWRUtilities 0.1.2, xtable 1.8-4
- Loaded via a namespace (and not attached): assertthat 0.2.1, backports 1.2.1, broom 0.7.3, cellranger 1.1.0, cli 2.2.0, colorspace 2.0-0, compiler 4.0.3, crayon 1.3.4, DBI 1.1.1, dbplyr 2.0.0, digest 0.6.27, ellipsis 0.3.1, evaluate 0.14, fansi 0.4.2, farver 2.0.3, fractional 0.1.3, fs 1.5.0, generics 0.1.0, glue 1.4.2, grid 4.0.3, gtable 0.3.0, highr 0.8, hms 1.0.0, httr 1.4.2, iterators 1.0.13, jsonlite 1.7.2, labeling 0.4.2, lazyData 1.1.0, lifecycle 0.2.0, lubridate 1.7.9.2, magrittr 2.0.1, MASS 7.3-53, Matrix 1.3-2, mgcv 1.8-33, modelr 0.1.8, munsell 0.5.0, nlme 3.1-151, parallel 4.0.3, PBSmapping 2.73.0, pillar 1.4.7, pkgconfig 2.0.3, R6 2.5.0, randomForest 4.6-14, Rcpp 1.0.6, readxl 1.3.1, reprex 1.0.0, rlang 0.4.10, rpart 4.1-15, rstudioapi 0.13, rvest 0.3.6, SOAR 0.99-11, splines 4.0.3, stringi 1.5.3, tidyselect 1.1.0, tools 4.0.3, vctrs 0.3.6, viridisLite 0.3.0, withr 2.4.1, xfun 0.20, xml2 1.3.2