Data Science with R

Part IV: Non-primitive Data Structures

Raphael Schleutker

March 13, 2020

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.
— Rick Cook

Table of contents

A basic data model

A basic data model

Let us create a very simple example. We want to know whether tSJ in a wildtypic background have more Anakonda than tSJ in an M6 mutant background. For this, we want to measure the fluorescence intensity of endogenously tagged Anakonda in embryos of two different genotypes. To make the result a little bit more robust, we also measure at different developmental stages.

A basic data model

For the beginning, let us look at only one single embryo. This embryo has many properties.

- The time it was laid (a date and time value).
- The temperature, at which it was laid (numerical).
- Its size (numerical).
- The developmental stage (ordinal).
- The color (nominal).
- The number of cells the embryo consist of (numerical).
- etc.

Basically, we could measure thousands of different aspects.

A basic data model

Usually, we are only interested in a few aspects. For example, we are interested in the intensity of GFP tagged Anakonda at tSJ in different genotypes and at different stages. Then, we measure the following things:

- The genotype of the embryo.
- The developmental stage of the embryo.
- The fluorescence intensity at tSJ.

One might be tempted to view the first two properties as qualitatively different from the third one as they are only needed to put the fluorescence intensity into a context and as we do not really measure them but actually know them in advance. However, for our data model they are not qualitatively different. They are all properties of the embryo.

A basic data model

We could put all these values together like the following:

$$(\text{Genotype},\ \text{Stage},\ \text{Intensity})$$

We call this a tuple. A tuple has two characteristics that are important for us

- A tuple is ordered. This means that the genotype is always at the first position, the stage is always at the second position and the fluorescence intensity is always at the third position.
- The elements of a tuple can have different scales (types).

One tuple makes up one observation, i.e. an observation always consists of all properties we have measured at one experimental unit (the embryo).

A basic data model

Usually, we are measuring more than one experimental unit. We measure in total n embryos. In this case we get n tuples:

$$(\text{Genotype}_1,\ \text{Stage}_1,\ \text{Intensity}_1)$$
$$(\text{Genotype}_2,\ \text{Stage}_2,\ \text{Intensity}_2)$$
$$\vdots$$
$$(\text{Genotype}_n,\ \text{Stage}_n,\ \text{Intensity}_n)$$

The index refers to the experimental unit, not the value. So Stage_1 and Stage_2 might have the same value if the two corresponding embryos have the same stage.

A basic data model

Since tuples are ordered, we can state that the i-th element of each tuple has the same scale (numerical, categorical, ...). We know that a vector always has one type only. This could bring us to the following idea:

For each measured property we create a vector of length n. The i-th element of this vector is than the value we measured from the i-th experimental unit.

A basic data model

$$
\begin{aligned}
&(\text{Genotype}_1\ ,\ \text{Stage}_1\ ,\ \text{Intensity}_1)\\
&(\text{Genotype}_2\ ,\ \text{Stage}_2\ ,\ \text{Intensity}_2)\\
&\qquad\vdots\\
&(\text{Genotype}_n\ ,\ \text{Stage}_n\ ,\ \text{Intensity}_n)
\end{aligned}
$$

This becomes then

$$
\begin{aligned}
\text{Genotype} &= c(\text{Genotype}_1\ ,\ \text{Genotype}_2\ ,\ \ldots\ ,\ \text{Genotype}_n)\\
\text{Stage} &= c(\text{Stage}_1\ ,\ \text{Stage}_2\ ,\ \ldots\ ,\ \text{Stage}_n)\\
\text{Intensity} &= c(\text{Intensity}_1\ ,\ \text{Intensity}_2\ ,\ \ldots\ ,\ \text{Intensity}_n)
\end{aligned}
$$

We can view these now as vectors and we can reconstruct one observation by taking the i-th element from each vector.

A basic data model

From the previous thoughts we can extract some characteristics.

- An observation tuple always has the same length. This length is the number of properties we are measuring at one experimental unit. An observation with a different length is not a valid observation for our experiment.
- Each vector has the same length. If we measure n embryos, all vectors have to have length n, even if we have missing values (in this case we insert `NA`).
- All values in one vector have the same scale. The genotype is always categorical (nominal to be more precise), the stage is always ordinal.
- Each element in each vector is atomic. It can not be further disassembled (we treat dates and times as atomic even though we could disassemble a date into day, month and year).

A basic data model

This can be transferred to tables.

- Each row of a table is one observation.
- Each column is one measured property / variable.
- One table only holds data for one experiment (including repeats, which have to be indicated by a further column.

A basic data model – Bad examples

R knows vectors which are perfectly suited for holding the measurements of one property, e.g. the fluorescence intensity. We can create several vectors to hold all the data from our experiment. This works theoretically but it would be more convenient to have a data structure that holds all these information and to which we can refer with a single variable.

Lists

Lists

Lists are the second data structure in R. We can think of lists as containers, in which we store all the information we need. Lists can hold all kinds of objects.

- Vectors of different types.
- Functions.
- Other lists.

We can even mix all these things in one list.

```
a <- 1:10

l1 <- list(a)

l2 <- list(a, l1, mean)
```

Lists

We can use the `str` function to get an idea about the structure of a list.

```
str(l1)


# List of 1
#  $ : int [1:10] 1 2 3 4 5 6 7 8 9 10

str(l2)


# List of 3
#  $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
#  $ :List of 1
#   ..$ : int [1:10] 1 2 3 4 5 6 7 8 9 10
#  $ :function (x, ...)
```

Lists

Since lists can hold all kind of data they are often used as the return values for functions with complicated output. For instance the return value of a t-test in R is a list that holds the test statistic, the degrees of freedom of the associated t distribution, the p value, the confidence interval, etc.

```
l3 <- t.test(
        x = rnorm(100, mean = 1),
        mu = 0
)
```

Lists

```
str(l3)


# List of 10
#  $ statistic  : Named num 8.3
#   ..- attr(*, "names")= chr "t"
#  $ parameter  : Named num 99
#   ..- attr(*, "names")= chr "df"
#  $ p.value    : num 0.000000000000546
#  $ conf.int   : num [1:2] 0.605 0.985
#   ..- attr(*, "conf.level")= num 0.95
#  $ estimate   : Named num 0.795
#   ..- attr(*, "names")= chr "mean of x"
#  $ null.value : Named num 0
#   ..- attr(*, "names")= chr "mean"
#  $ stderr     : num 0.0957
#  $ alternative: chr "two.sided"
#  $ method     : chr "One Sample t-test"
#  $ data.name  : chr "rnorm(100, mean = 1)"
#  - attr(*, "class")= chr "htest"
```

Lists

We can give elements of lists names to make more obvious what that element is.

```
data <- list(
        Stage = c(14, 14, 14, 15, 15, 14, 15, 15),
        Genotype = c("yw", "yw", "M6[W186*]", "M6[W186*]", "yw", "yw", "M6[W186
        Intensity = c(1.35, 0.36, 2.46, 0.03, 0.74, 1.49, 2.01, 3.21)
)

str(data)


# List of 3
#  $ Stage    : num [1:8] 14 14 14 15 15 14 15 15
#  $ Genotype : chr [1:8] "yw" "yw" "M6[W186*]" "M6[W186*]" ...
#  $ Intensity: num [1:8] 1.35 0.36 2.46 0.03 0.74 1.49 2.01 3.21
```

Lists

Such a list is already quite nice. Now we can refer to our data with a single variable name instead of having a single variable for each property. We also make sure that measurements that belong together are logically separated from data of other experiments.

However, since lists give us so much freedom for its content it is easy to mess it up. We could store vectors of different lengths in a list or also stuff that doesn't belong to the data of our experiment.

Data frames

Data frames

To circumvent the previous problems R has a data structure called data frame. Each data frame is a list (but not every list is a data frame). Indeed, data frames are a special kind of list that restricts what we can store in it.

- Each element of of the data frame is a list.
- Each element has the same length.

Data frames

```r
data <- data.frame(
        Stage = c(14, 14, 14, 15, 15, 14, 15, 15),
        Genotype = c("yw", "yw", "M6[W186*]", "M6[W186*]", "yw", "yw", "M6[W186
        Intensity = c(1.35, 0.36, 2.46, 0.03, 0.74, 1.49, 2.01, 3.21)
)

typeof(data)

# [1] "list"

str(data)

# 'data.frame': 8 obs. of  3 variables:
#  $ Stage    : num  14 14 14 15 15 14 15 15
#  $ Genotype : Factor w/ 2 levels "M6[W186*]","yw": 2 2 1 1 2 2 1 1
#  $ Intensity: num  1.35 0.36 2.46 0.03 0.74 1.49 2.01 3.21
```

Data frames

Using data frames changes the way how functions work on the list. For instance, `print` outputs the data frame in a table like fashion.

```
data


#   Stage  Genotype Intensity
# 1    14         yw      1.35
# 2    14         yw      0.36
# 3    14 M6[W186*]      2.46
# 4    15 M6[W186*]      0.03
# 5    15         yw      0.74
# 6    14         yw      1.49
# 7    15 M6[W186*]      2.01
# 8    15 M6[W186*]      3.21
```

Data frames

Usually, data frames hold many more values and we are only interested in a rough idea. In this case, we can use `head` and `tail`

```
head(data)
```

```
#   Stage  Genotype Intensity
# 1    14        yw      1.35
# 2    14        yw      0.36
# 3    14 M6[W186*]      2.46
# 4    15 M6[W186*]      0.03
# 5    15        yw      0.74
# 6    14        yw      1.49
```

Data frames

A really useful function for data frames is `summary`

```
summary(data)
```

```
#      Stage            Genotype   Intensity
#  Min.   :14.0   M6[W186*]:4   Min.   :0.030
#  1st Qu.:14.0   yw        :4   1st Qu.:0.645
#  Median :14.5                  Median :1.420
#  Mean   :14.5                  Mean   :1.456
#  3rd Qu.:15.0                  3rd Qu.:2.123
#  Max.   :15.0                  Max.   :3.210
```

Depending on the type of the column, it creates useful summary statistics.

Data frames

Sometimes, we are only interested in subsets of data. We can filter data using subset

```
subset(data, Stage == 14 & Intensity > 1)
```

```
#   Stage  Genotype Intensity
# 1    14        yw      1.35
# 3    14 M6[W186*]      2.46
# 6    14        yw      1.49
```

Data frames

We can split a data frame using `split`. This yields a list of data frames.

```
splitted_data <- split(data, data$Genotype)
splitted_data


# $`M6[W186*]`
#   Stage  Genotype Intensity
# 3    14 M6[W186*]      2.46
# 4    15 M6[W186*]      0.03
# 7    15 M6[W186*]      2.01
# 8    15 M6[W186*]      3.21
#
# $yw
#   Stage Genotype Intensity
# 1    14       yw      1.35
# 2    14       yw      0.36
# 5    15       yw      0.74
# 6    14       yw      1.49
```

Data frames

`unsplit` does the reverse.

```
unsplit_data <- unsplit(splitted_data, data$Genotype)
unsplit_data
```

```
#   Stage  Genotype Intensity
# 1    14         yw      1.35
# 2    14         yw      0.36
# 3    14 M6[W186*]      2.46
# 4    15 M6[W186*]      0.03
# 5    15         yw      0.74
# 6    14         yw      1.49
# 7    15 M6[W186*]      2.01
# 8    15 M6[W186*]      3.21
```

Data frames

We can do lot of things with data frames and we will spend another lesson to see what and how.