I am going to describe my favorite knitr trick: **Using lists to simplify inline reporting**. *Trick* might not do it justice. I consider this a *best practice* for working with knitr.

Plug it in

Inline reporting lets you insert R expressions inside of markdown text. Those expressions are evaluated and their results are plugged in as text. The following example shows a common use case: Reporting descriptive statistics. The `sitka` dataset describes the longitudinal growth of Sitka spruce trees in different ozone conditions, so here are some lines we might report:

````
```{r}
library(magrittr)
data("sitka", package = "gamair")
n_trees <- length(levels(sitka$id.num))
n_conditions <- length(unique(sitka$ozone))
```

The dataset contains `r nrow(sitka)` tree size measurements
from `r n_trees` trees grown in
`r n_conditions` ozone treatment conditions.
````

which produces

```
The dataset contains 1027 tree size measurements
from 0 trees grown in
2 ozone treatment conditions.
```

If we update the dataset, the numbers will update automatically when the document is reknitted. It's just magical. Besides reporting statistics, I routinely use inline reporting for package versions, dates, file provenance, links to package documentation, and emoji. Here is an example of each:

````
```{r}
knitted_when <- format(Sys.Date())
knitted_where <- knitr::current_input()
knitted_with <- packageVersion("knitr")
knitted_doc_url <- downlit::autolink_url("knitr::knit()")
```

Reported prepared on `r knitted_when` from ``r knitted_where``
with knitr version `r knitted_with` `r emo::ji('smile')`.
Read more about [`knitr::knit()`](`r knitted_doc_url`).

::::

Reported prepared on 2021-02-05 from `2021-02-05-lists-knitr-secret-
weapon.Rmd`
with knitr version 1.31 😄.
Read more about [`knitr::knit()`](https://rdrr.io/pkg/knitr/man/knit.html)`.
````

Are your variable names doing a list's job?

In this last example, I used prefixes in variable names to convey that the data were related. `knitted_when`, `knitted_where` and `knitted_with` are all facts about the knitting process. They are all reported in the narrative text pretty close to each other. The prefix informally bundles them together. The prefix also helps with writing our code because we have to remember less. We can type `knitted_` and press `Tab` and let autocompletion remind us which variables are available.

```
# simulate tab completion
"knitted_" %>% tab()
#> knitted_doc_url
#> knitted_when
#> knitted_where
#> knitted_with
```

But—and here is the key insight—what if we change that underscore `_` into a dollar sign `$`, so to speak? That is, let's bundle everything into a list and then report the results by accessing list elements.

````
```{r}
knitted <- list(
  when = format(Sys.Date()),
  where = knitr::current_input(),
  with = packageVersion("knitr"),
  doc_url = downlit::autolink_url("knitr::knit()")
)
```

Reported prepared on `r knitted$when` from ``r knitted$where``
with knitr version `r knitted$with` `r emo::ji('happy')`.
Read more about [`knitr::knit()`](`r knitted$doc_url`).

::::

Reported prepared on 2021-02-05 from `2021-02-05-lists-knitr-secret-
weapon.Rmd`
with knitr version 1.31 😆.
Read more about [`knitr::knit()`](https://rdrr.io/pkg/knitr/man/knit.html)`.
````

We have structured names, and we still retain our `Tab` completion:

```
"knitted$" %>% tab()
#> knitted$when
#> knitted$where
#> knitted$with
#> knitted$doc_url
```

But we can also glance at our list to see everything about the knitting process all at once.

```
knitted
#> $when
#> [1] "2021-02-05"
#>
#> $where
```

```
#> [1] "2021-02-05-lists-knitr-secret-weapon.Rmd"
#>
#> $with
#> [1] '1.31'
#>
#> $doc_url
#> [1] "https://rdrr.io/pkg/knitr/man/knit.html"
```

Basically, using a list formalizes the relationship we had implicitly set out by using our naming convention, but so what? How does this help inline reporting? Lists have all the nice benefits of using a naming convention, plus one important feature: **We can create lists programmatically**.

Set up model results with `tidy()`

Let's say we model the growth of each tree in each ozone condition and want to know how much steeper the growth rate is for the ozone treatment condition. We fit a linear mixed model where we estimate the population averages for the intercept and slope for each ozone condition, and we use random effects to accord each tree its own intercept and slope.

```
library(lme4)
#> Loading required package: Matrix
#>
#> Attaching package: 'Matrix'
#> The following objects are masked from 'package:tidyr':
#>
#>     expand, pack, unpack
# Rescale to get improve convergence
sitka$hund_days <- sitka$days / 100
m <- lmer(
  log.size ~ hund_days * ozone + (hund_days | id.num),
  sitka
)
summary(m)
#> Linear mixed model fit by REML ['lmerMod']
#> Formula: log.size ~ hund_days * ozone + (hund_days | id.num)
#>    Data: sitka
#>
#> REML criterion at convergence: 767
#>
#> Scaled residuals:
#>     Min      1Q  Median      3Q     Max
#> -3.6836 -0.4723  0.0078  0.4237  2.5919
#>
#> Random effects:
#>  Groups   Name        Variance Std.Dev. Corr
#>  id.num   (Intercept) 0.390928 0.62524
#>           hund_days   0.002459 0.04959  -0.23
#>  Residual             0.082770 0.28770
#> Number of obs: 1027, groups:  id.num, 79
#>
#> Fixed effects:
#>                 Estimate Std. Error t value
```

```
#> (Intercept)      4.25491    0.13100  32.481
#> hund_days        0.33903    0.01278  26.528
#> ozone           -0.14101    0.15844  -0.890
#> hund_days:ozone -0.03611    0.01546  -2.336
#>
#> Correlation of Fixed Effects:
#>             (Intr) hnd_dy ozone
#> hund_days   -0.345
#> ozone       -0.827  0.286
#> hund_dys:zn  0.286 -0.827 -0.345
```

Our job is get to numbers from this summary view into prose. For this example, we want to report that the two groups don't have a statistically clear difference in their intercepts, as given by the `ozone` line in the model summary. We also want to report that growth per 100 days is statistically significantly slower in the ozone group `hund_days:ozone`.

First, we `tidy()` the model summary using broom.mixed.

```
library(tidyverse)
library(broom.mixed)
#> Registered S3 method overwritten by 'broom.mixed':
#>   method      from
#>   tidy.gamlss broom
tidy(m, conf.int = TRUE) %>%
  filter(effect == "fixed")
#> # A tibble: 4 x 8
#>   effect group term            estimate std.error statistic conf.low
conf.high
#>
#> 1 fixed    (Intercept)       4.25      0.131     32.5     4.00
4.51
#> 2 fixed    hund_days         0.339     0.0128    26.5     0.314
0.364
#> 3 fixed    ozone            -0.141     0.158     -0.890  -0.452
0.170
#> 4 fixed    hund_days:ozone  -0.0361    0.0155    -2.34   -0.0664
-0.00581
```

We are also going to format the numbers. I have my own R package for this job called printy. Below I use it to round numbers—`round()` drops 0s off the ends of rounded numbers whereas `printy::fmt_fixed_digits()` keeps them. I also use it for formatting minus signs.

```
text_ready <- tidy(m, conf.int = TRUE) %>%
  filter(effect == "fixed") %>%
  mutate(
    # round the numbers
    across(
      c(estimate, conf.low, conf.high),
      printy::fmt_fix_digits,
      2
    ),
    se = round(std.error, 3),
    # use a minus sign instead of a hyphen for negative numbers
```

```
    across(
      c(estimate, conf.low, conf.high),
      printy::fmt_minus_sign
    ),
    ci = glue::glue("[{conf.low}, {conf.high}]")
  ) %>%
  select(term, estimate, se, ci)
text_ready
#> # A tibble: 4 x 4
#>   term            estimate       se ci
#>
#> 1 (Intercept)     4.25        0.131 [4.00, 4.51]
#> 2 hund_days       0.34        0.013 [0.31, 0.36]
#> 3 ozone           -0.14 0.158 [-0.45, 0.17]
#> 4 hund_days:ozone -0.04 0.015 [-0.07, -0.01]
```

We could use dataframe functions to `filter()` down to the down the terms and `pull()` the values and use a list:

```
```{r}
stats <- list()
stats$b_intercept <- text_ready %>%
  filter(term == "(Intercept)") %>%
  pull(estimate)
```

The average log-size in the control condition was `r stats$b_intercept` units.

::::

The average log-size in the control condition was 4.25 units.
```

(The documentation for `sitka$log.size` doesn't say what units the data are in, so I'm sticking with "units" 🙃.)

`split()` makes lists

A much better approach is to use `split()` to create a list using the values in a dataframe column. To make the list easier for typing, I use `janitor::make_clean_names()` to clean up the `term` value.

```
stats <- text_ready %>%
  mutate(term = janitor::make_clean_names(term)) %>%
  split(.$term)
```

Now we have a list of one-row dataframes:

```
str(stats)
#> List of 4
#>  $ hund_days      : tibble [1 x 4] (S3: tbl_df/tbl/data.frame)
#>   ..$ term    : chr "hund_days"
#>   ..$ estimate: chr "0.34"
```

```
#>   ..$ se      : num 0.013
#>   ..$ ci      : 'glue' chr "[0.31, 0.36]"
#>  $ hund_days_ozone: tibble [1 x 4] (S3: tbl_df/tbl/data.frame)
#>   ..$ term    : chr "hund_days_ozone"
#>   ..$ estimate: chr "-0.04"
#>   ..$ se      : num 0.015
#>   ..$ ci      : 'glue' chr "[-0.07, -0.01]"
#>  $ intercept      : tibble [1 x 4] (S3: tbl_df/tbl/data.frame)
#>   ..$ term    : chr "intercept"
#>   ..$ estimate: chr "4.25"
#>   ..$ se      : num 0.131
#>   ..$ ci      : 'glue' chr "[4.00, 4.51]"
#>  $ ozone          : tibble [1 x 4] (S3: tbl_df/tbl/data.frame)
#>   ..$ term    : chr "ozone"
#>   ..$ estimate: chr "-0.14"
#>   ..$ se      : num 0.158
#>   ..$ ci      : 'glue' chr "[-0.45, 0.17]"
```

And we have structured, autocomplete-friendly names too:

```
"stats$" %>% tab()
#> stats$hund_days
#> stats$hund_days_ozone
#> stats$intercept
#> stats$ozone

"stats$ozone$" %>% tab()
#> stats$ozone$term
#> stats$ozone$estimate
#> stats$ozone$se
#> stats$ozone$ci
```

Now, we can write up our results with inline reporting:

````
```{r}
stats <- text_ready %>%
  mutate(term = janitor::make_clean_names(term)) %>%
  split(.$term)
```

The average log-size in the control condition was
`r stats$intercept$estimate` units, 95% Wald CI `r stats$intercept$ci`.
There was not a statistically clear difference between the
ozone conditions for their intercepts (day-0 values),
*B* = `r stats$ozone$estimate`, `r stats$ozone$ci`.
For the control group, the average growth rate was
`r stats$hund_days$estimate` log-size units per 100 days,
`r stats$hund_days$ci`. The growth rate for
the ozone treatment group was significantly slower,
*diff* = `r stats$hund_days_ozone$estimate`,
`r stats$hund_days_ozone$ci`.

::::
````

```
The average log-size in the control condition was
4.25 units, 95% Wald CI [4.00, 4.51].
There was not a statistically clear difference between the
ozone conditions for their intercepts (day-0 values),
*B* = -0.14, [-0.45, 0.17].
For the control group, the average growth rate was
0.34 log-size units per 100 days,
[0.31, 0.36]. The growth rate for the ozone treatment group was
significantly slower, *diff* = -0.04,
[-0.07, -0.01].
```

Isn't that RMarkdown text just *a joy* to read? Everything so neatly named and organized, and we got all of that for free by using `tidy()` and `split()` to make a list.

Splitting splits of splits

I am such a champion of this approach that I wrote my own split function for splitting by multiple variables. In the `mtcars` dataset, suppose we want to report the mean mpg of 6- and 8-cylinder (`cyl`) vehicles split by automatic versus manual (`am`) vehicles. We compute the stats with some basic dplyring and we prepare names that work better with `split()`.

```
car_means <- mtcars %>%
  group_by(cyl, am) %>%
  summarise(
    n = n(),
    mean_mpg = mean(mpg),
    .groups = "drop"
  ) %>%
  # make names for spliting
  mutate(
    a = paste0("am_", am),
    c = paste0("cyl_", cyl),
  )
car_means
#> # A tibble: 6 x 6
#>     cyl    am     n mean_mpg a     c
#> *
#> 1     4     0     3     22.9 am_0  cyl_4
#> 2     4     1     8     28.1 am_1  cyl_4
#> 3     6     0     4     19.1 am_0  cyl_6
#> 4     6     1     3     20.6 am_1  cyl_6
#> 5     8     0    12     15.0 am_0  cyl_8
#> 6     8     1     2     15.4 am_1  cyl_8
```

Now enter `super_split()`:

```
car_stats <- car_means %>%
  printy::super_split(a, c)

# set `max.level` to not print individual tibble structures
str(car_stats, max.level = 3)
#> List of 2
```

```
#>  $ am_0:List of 3
#>   ..$ cyl_4: tibble [1 x 6] (S3: tbl_df/tbl/data.frame)
#>   ..$ cyl_6: tibble [1 x 6] (S3: tbl_df/tbl/data.frame)
#>   ..$ cyl_8: tibble [1 x 6] (S3: tbl_df/tbl/data.frame)
#>  $ am_1:List of 3
#>   ..$ cyl_4: tibble [1 x 6] (S3: tbl_df/tbl/data.frame)
#>   ..$ cyl_6: tibble [1 x 6] (S3: tbl_df/tbl/data.frame)
#>   ..$ cyl_8: tibble [1 x 6] (S3: tbl_df/tbl/data.frame)
```

Here, we have a list of lists of 1-row dataframes, and we can just use `$` to drill down the lists during inline reporting.

```
The average mpg of the `r car_stats$am_0$cyl_4$n` automatic,
four-cylinder cars was `r car_stats$am_0$cyl_4$mean_mpg`.

::::

The average mpg of the 3 automatic,
four-cylinder cars was 22.9.
```

For the curious, `super_split()` works behind the scenes by exploiting two functions:

- `split()` adds a level of depth to a list by splitting a list into sublists using a variable.
- `purrr::map_depth(.x, .depth, .f)` applies a function `.f` on the lists at a given `.depth`.

So the function walks through each variable and applies `split()` at successively deeper depths.

```
super_split <- function(.data, ...) {
  dots <- rlang::enquos(...)
  for (var in seq_along(dots)) {
    var_name <- rlang::as_name(dots[[var]])
    .data <- purrr::map_depth(
      .x = .data,
      .depth = var - 1,
      .f = function(xs) split(xs, xs[var_name])
    )
  }
  .data
}
```

The first variable splits the list at depth 0, the second variable splits the sublists at depth 1 (which were created in the prior split), and so on. The business with `enquos(...)` is there to let me refer to the variable names directly.

Last knitted on 2021-02-05. Source code on GitHub.[1]

1.
```
sessioninfo::session_info()
#> - Session info ---------------------------
---------------------------------
#>  setting  value
#>  version  R version 4.0.3 (2020-10-10)
```

```
#>  os       Windows 10 x64
#>  system   x86_64, mingw32
#>  ui       RTerm
#>  language (EN)
#>  collate  English_United States.1252
#>  ctype    English_United States.1252
#>  tz       America/Chicago
#>  date     2021-02-05
#>
#> - Packages ------------------------------
-------------------------------------
#>  ! package     * version    date       lib source
#>    assertthat    0.2.1      2019-03-21 [1] CRAN (R 4.0.2)
#>    backports     1.2.0      2020-11-02 [1] CRAN (R 4.0.3)
#>    boot          1.3-26     2021-01-25 [1] CRAN (R 4.0.3)
#>    broom         0.7.3      2020-12-16 [1] CRAN (R 4.0.3)
#>    broom.mixed * 0.2.6      2020-05-17 [1] CRAN (R 4.0.2)
#>    cellranger    1.1.0      2016-07-27 [1] CRAN (R 4.0.2)
#>    cli           2.2.0      2020-11-20 [1] CRAN (R 4.0.3)
#>    coda          0.19-4     2020-09-30 [1] CRAN (R 4.0.2)
#>    colorspace    2.0-0      2020-11-11 [1] CRAN (R 4.0.3)
#>    crayon        1.4.0      2021-01-30 [1] CRAN (R 4.0.3)
#>    DBI           1.1.1      2021-01-15 [1] CRAN (R 4.0.3)
#>    dbplyr        2.0.0      2020-11-03 [1] CRAN (R 4.0.2)
#>    downlit       0.2.1      2020-11-04 [1] CRAN (R 4.0.2)
#>    dplyr       * 1.0.3      2021-01-15 [1] CRAN (R 4.0.3)
#>    ellipsis      0.3.1      2020-05-15 [1] CRAN (R 4.0.2)
#>    emo           0.0.0.9000 2020-07-06 [1] Github
(hadley/emo@3f03b11)
#>    evaluate      0.14       2019-05-28 [1] CRAN (R 4.0.2)
#>    fansi         0.4.2      2021-01-15 [1] CRAN (R 4.0.3)
#>    forcats     * 0.5.1      2021-01-27 [1] CRAN (R 4.0.3)
#>    fs            1.5.0      2020-07-31 [1] CRAN (R 4.0.2)
#>    generics      0.1.0      2020-10-31 [1] CRAN (R 4.0.3)
#>    ggplot2     * 3.3.3      2020-12-30 [1] CRAN (R 4.0.3)
#>    git2r         0.28.0     2021-01-10 [1] CRAN (R 4.0.3)
#>    glue          1.4.2      2020-08-27 [1] CRAN (R 4.0.2)
#>    gtable        0.3.0      2019-03-25 [1] CRAN (R 4.0.2)
#>    haven         2.3.1      2020-06-01 [1] CRAN (R 4.0.2)
#>    here          1.0.1      2020-12-13 [1] CRAN (R 4.0.3)
#>    hms           1.0.0      2021-01-13 [1] CRAN (R 4.0.3)
#>    httr          1.4.2      2020-07-20 [1] CRAN (R 4.0.2)
#>    janitor       2.1.0      2021-01-05 [1] CRAN (R 4.0.3)
#>    jsonlite      1.7.2      2020-12-09 [1] CRAN (R 4.0.3)
#>    knitr       * 1.31       2021-01-27 [1] CRAN (R 4.0.3)
#>    lattice       0.20-41    2020-04-02 [1] CRAN (R 4.0.2)
#>    lifecycle     0.2.0      2020-03-06 [1] CRAN (R 4.0.2)
#>    lme4        * 1.1-26     2020-12-01 [1] CRAN (R 4.0.3)
#>    lubridate     1.7.9.2    2020-11-13 [1] CRAN (R 4.0.3)
#>    magrittr      2.0.1      2020-11-17 [1] CRAN (R 4.0.3)
#>    MASS          7.3-53     2020-09-09 [1] CRAN (R 4.0.2)
#>    Matrix      * 1.2-18     2019-11-27 [1] CRAN (R 4.0.3)
```

```
#>    minqa         1.2.4      2014-10-09 [1] CRAN (R 4.0.2)
#>    modelr        0.1.8      2020-05-19 [1] CRAN (R 4.0.2)
#>    munsell       0.5.0      2018-06-12 [1] CRAN (R 4.0.2)
#>    nlme          3.1-151    2020-12-10 [1] CRAN (R 4.0.3)
#>    nloptr        1.2.2.2    2020-07-02 [1] CRAN (R 4.0.2)
#>    pillar        1.4.7      2020-11-20 [1] CRAN (R 4.0.3)
#>    pkgconfig     2.0.3      2019-09-22 [1] CRAN (R 4.0.2)
#>    plyr          1.8.6      2020-03-03 [1] CRAN (R 4.0.2)
#>    printy        0.0.0.9003 2020-07-08 [1] Github
(tjmahr/printy@61ad449)
#>    ps            1.5.0      2020-12-05 [1] CRAN (R 4.0.3)
#>    purrr       * 0.3.4      2020-04-17 [1] CRAN (R 4.0.2)
#>    R6            2.5.0      2020-10-28 [1] CRAN (R 4.0.2)
#>    Rcpp          1.0.6      2021-01-15 [1] CRAN (R 4.0.3)
#>    readr       * 1.4.0      2020-10-05 [1] CRAN (R 4.0.2)
#>    readxl        1.3.1      2019-03-13 [1] CRAN (R 4.0.2)
#>    reprex        1.0.0      2021-01-27 [1] CRAN (R 4.0.3)
#>    reshape2      1.4.4      2020-04-09 [1] CRAN (R 4.0.2)
#>    rlang         0.4.10     2020-12-30 [1] CRAN (R 4.0.3)
#>    rprojroot     2.0.2      2020-11-15 [1] CRAN (R 4.0.3)
#>    rstudioapi    0.13       2020-11-12 [1] CRAN (R 4.0.3)
#>    rvest         0.3.6      2020-07-25 [1] CRAN (R 4.0.2)
#>    scales        1.1.1      2020-05-11 [1] CRAN (R 4.0.2)
#>    sessioninfo   1.1.1      2018-11-05 [1] CRAN (R 4.0.2)
#>    snakecase     0.11.0     2019-05-25 [1] CRAN (R 4.0.2)
#>    statmod       1.4.35     2020-10-19 [1] CRAN (R 4.0.3)
#>    stringi       1.5.3      2020-09-09 [1] CRAN (R 4.0.2)
#>    stringr     * 1.4.0      2019-02-10 [1] CRAN (R 4.0.2)
#>    tibble      * 3.0.6      2021-01-29 [1] CRAN (R 4.0.3)
#>    tidyr       * 1.1.2      2020-08-27 [1] CRAN (R 4.0.2)
#>    tidyselect    1.1.0      2020-05-11 [1] CRAN (R 4.0.2)
#>    tidyverse   * 1.3.0      2019-11-21 [1] CRAN (R 4.0.2)
#>  D TMB           1.7.18     2020-07-27 [1] CRAN (R 4.0.2)
#>    utf8          1.1.4      2018-05-24 [1] CRAN (R 4.0.2)
#>    vctrs         0.3.6      2020-12-17 [1] CRAN (R 4.0.3)
#>    withr         2.4.1      2021-01-26 [1] CRAN (R 4.0.3)
#>    xfun          0.20       2021-01-06 [1] CRAN (R 4.0.3)
#>    xml2          1.3.2      2020-04-23 [1] CRAN (R 4.0.2)
#>    yaml          2.2.1      2020-02-01 [1] CRAN (R 4.0.0)
#>
#> [1] C:/Users/Tristan/Documents/R/win-library/4.0
#> [2] C:/Program Files/R/R-4.0.3/library
#>
#>  D -- DLL MD5 mismatch, broken installation…
```