Cereal data example

The purpose of this document is to introduce how to use `knitr` with LyX. There is a separate document that explains how to use `knitr`, LyX, and Beamer together. This document is written more so to be read than to be presented.

At the beginning of any document using `knitr` with LyX, you may want to set some global options. For example, within the LyX document only, you will see some options set which control the appearance of the code/output throughout the document. This code will be discussed in Section 2 after I explain a few other items first. I have purposely commented over each line of code (using the `%` LaTeX comment symbol) so that they do not have an effect right now.

1 Basics

A few years ago, I collected information on the nutritional content of dry cereals at a grocery store. This was done by first noting that one side of one aisle in many grocery stores usually contains all the cereals within a store. For example, Figure 1 shows what the cereal aisle used to look like in the HyVee at 5020 N 27th St. before its recent renovation. My research hypothesis was that there were different mean nutritional contents by shelf. For example, lower shelves may have more sugar content cereals than higher shelves.

The data used for this example was collected from a store a few years ago (not the HyVee in the picture). Note that there were only four shelves at this store and my sample size was 10 from each shelf. Below is how I read the data into R.

```
cereal <- read.csv(file = "cereal.csv")
head(cereal)

##   ID Shelf                              Cereal size_g sugar_g fat_g
## 1  1     1 Kellog's Razzle Dazzle Rice Crispies     28      10     0
## 2  2     1            Post Toasties Corn Flakes     28       2     0
## 3  3     1                Kellogg's Corn Flakes     28       2     0
## 4  4     1               Food Club Toasted Oats     32       2     2
## 5  5     1                     Frosted Cheerios     30      13     1
## 6  6     1             Food Club Frosted Flakes     31      11     0
##   sodium_mg
## 1       170
## 2       270
## 3       300
## 4       280
## 5       210
## 6       180
```

The shelves are numbered from lowest (1) to highest (4).

We need to adjust the nutritional content variables (`sugar_g`, `fat_g`, and `sodium_g`) for the serving size because cereal boxes tend to have different serving sizes. Below is how I make the adjustment in R.

Figure 1: Cereal aisle at HyVee.


```
cereal$sugar <- cereal$sugar_g/cereal$size_g
cereal$fat <- cereal$fat_g/cereal$size_g
cereal$sodium <- cereal$sodium_mg/cereal$size_g
head(cereal)
```

```
##   ID Shelf                              Cereal size_g sugar_g fat_g
## 1  1     1 Kellog's Razzle Dazzle Rice Crispies     28      10     0
## 2  2     1            Post Toasties Corn Flakes     28       2     0
## 3  3     1               Kellogg's Corn Flakes     28       2     0
## 4  4     1              Food Club Toasted Oats     32       2     2
## 5  5     1                    Frosted Cheerios     30      13     1
## 6  6     1            Food Club Frosted Flakes     31      11     0
##   sodium_mg      sugar        fat    sodium
## 1       170 0.35714286 0.00000000  6.071429
## 2       270 0.07142857 0.00000000  9.642857
## 3       300 0.07142857 0.00000000 10.714286
## 4       280 0.06250000 0.06250000  8.750000
## 5       210 0.43333333 0.03333333  7.000000
## 6       180 0.35483871 0.00000000  5.806452
```

For example, the `sugar` value for Kellogg's Razzle Dazzle Rice Crispies is

$$\text{(sugar grams per serving)}/\text{(\# of grams per serving size)} = 10/28 = 0.3571.$$

Alternatively, taking advantage of `knitr`, the calculation can be automatically done here using R:

$$\text{(sugar grams per serving)}/\text{(\# of grams per serving size)} = 10/28 = 0.3571429.$$

In this case, I directly used LaTeX to enter the the equation and the "S expression" function `\Sexpr{}` to compute values within R. The S expression function can be used inline as well. For example, the number of observations in the data set is 40. Notice that the values given are in the normal text font rather than the font used for code and output resulting from chunks.

2 Code/output appearance

The appearance of the code/output can be controlled by arguments after the chunk name. For example, to change the format of the code/output to something closer to what I use in my notes, set the following options:

1. `prompt = TRUE` (default is `FALSE`) – Includes the `>` prompt at the beginning of every line
2. `comment = NA` (default is "##") – Removes the use of ## at the beginning of output lines
3. `background = "white"` (default is `"#F7F7F7"`) – Uses a white background; any R color can be specified here

Below is an example showing the implementation of these different option values, where the purpose is to find the adjusted mean and standard deviation for the sugar content by shelf.

```
> #Mean by shelf
> aggregate(formula = sugar ~ Shelf, data = cereal, FUN = mean)

  Shelf     sugar
1     1 0.2568366
2     2 0.4149686
3     3 0.2303732
4     4 0.2554839

> #Standard deviation by shelf
> aggregate(formula = sugar ~ Shelf, data = cereal, FUN = sd)

  Shelf      sugar
1     1 0.16729566
2     2 0.09001019
3     3 0.15770057
4     4 0.11010226
```

If you want to turn off the syntax highlighting, use `highlight = FALSE` as an option.

When there is a long set of code/output, you may want to change the font size resulting from a chunk. The `size` option can be used for this purpose, with values `normalsize` (default), `tiny`, `scriptsize`, `footnotesize`, `small`, `large`, `Large`, `LARGE`, `huge`, and `Huge`. Larger font sizes are helpful for presentations. Smaller font sizes are helpful when the output is too wide for a page. Below is an example where I use a larger font size for demonstration purposes when estimating a one-factor ANOVA model for the sugar content.

```
mod.fit <- aov(formula = sugar ~ factor(Shelf), data = cereal)
mod.fit$coefficients

##    (Intercept) factor(Shelf)2 factor(Shelf)3 factor(Shelf)4
##    0.256836623    0.158131957   -0.026463461   -0.001352752
```

When output extends into the right margin, the `options(width = <NUMBER>)` function can be used within a chunk to reduce the width. The default for `knitr` is `options(width = 75)`. Unfortunately, this does not control the width of displayed code. Instead, the width of displayed code is controlled by the `tidy` chunk option. Adding `tidy = TRUE` leads to "nice" formatting of the code displayed by a chunk. Specific options of how to keep the code "tidy" can be specified by `tidy.opts`. For example, to control the width of displayed code, the `width.cutoff` option can be specified in a list as `tidy.opts = list(width.cutoff = 60)`. This causes R to attempt a line break after 60 characters. The first chunk of Section 4 provides an example of its use; without this additional option, the code would extend into the right margin.

It is often desirable to keep the same code/output appearance for an entire document. While specifying the appearance options in each chunk can be done, it may be more preferable to make the changes globally; i.e., set the default appearance for the entire document. The `opts_chunk$set()` function is one of many functions that allow you to change global options. Immediately before Section 1, you will see a chunk where I use this function with the same options as we have seen in other chunks of Section 2. I recommend uncommenting the chunk code to see its effect on the document. Note that you can still override these options by specifying new option values in a chunk. For example, if I want to set the result of one chunk to have a red background, I can simply specify `background = "red"` as a option for that particular chunk.

There are other global option functions, and I recommend examining these in Xie's website or in his books. The way that Xie organized these functions is unique. Notice that `$set` is given after `opts_chunk` in the previous paragraph. This is because `opts_chunk` is a list with components:

```
names(opts_chunk)

## [1] "get"     "set"     "merge"   "restore"

opts_chunk$set

## function (...)
## {
##     dots = list(...)
##     if (length(dots) == 0)
##         return()
##     if (is.null(names(dots)) && length(dots) == 1 && is.list(dots[[1]]))
##         if (length(dots <- dots[[1]]) == 0)
##             return()
##     defaults <<- merge(dots)
##     invisible(NULL)
## }
## <environment: 0x000000001409aee0>
```

Each component of the list is a function, like `opts_chunk$set()`.

In addition to setting the global options yourself, there are other ways to control the code/output appearance all at once rather than within each chunk. One way is to use a theme. Page 56 of Xie (2015) discusses themes, and a list of them can be obtained by executing `knit_theme$get()`. Another way to control code/output appearance is to use `render_sweave()` or `render_listings()`. The former function changes the appearance to be the Sweave default type, and the latter function changes the appearance to be of the same form as given by the LaTeX listings package. Page 41 of Xie (2015) and `http://yihui.name/knitr/hooks` provides examples resulting from both of these functions.

3 Code/output content

The actual content in a compiled LyX document which results from a chunk is controlled by the following options:

1. `eval = <LINE NUMBER>` (default is `TRUE`) – specifies which lines of code to evaluate; for example, `eval = c(1, 3:4)` evaluates only lines 1, 3, and 4, while `eval = -2` evaluates all lines within a chunk excluding line 2.

2. `include = FALSE` (default is `TRUE`) – prevents the code/output from showing in the compiled document; this is helpful when you do not want to show all the code/output leading up to a particular point. Also, this is helpful when you are setting global options at a beginning of a document, like what I did before Section 1.

3. `echo = FALSE` (default is `TRUE`) – prevents the code from being printed, but allows the output; line number echoing can be done too (see "Advanced usage of the `echo` option" at `http://yihui.name/knitr/demo/output`)

For example, suppose that you only want the ANOVA table to appear in the document.

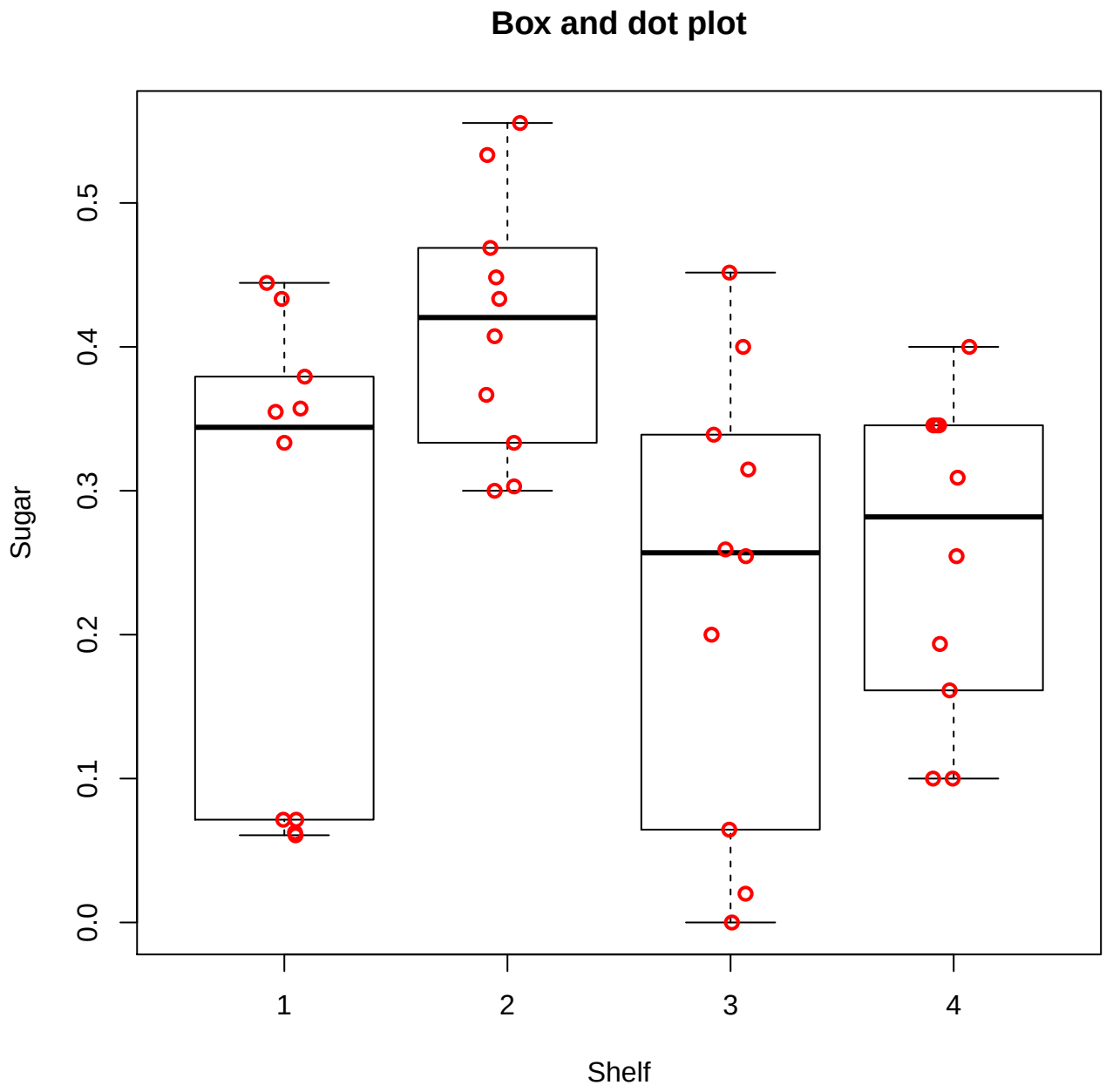
```
              Df Sum Sq Mean Sq F value Pr(>F)
factor(Shelf)  3 0.2146 0.07154   3.916 0.0162 *
Residuals     36 0.6577 0.01827
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4 Plots

When a plot is created by a chunk, it is put into a PDF file and stored in a folder named "figure" by default. This folder is automatically created by `knitr`, and it is located in the same folder that the .lyx file is found. Note that this folder is not removed when LyX is exited. File names inside the folder correspond to chunk names.

Within the compiled document, a plot will be displayed immediately after the final line of code used to create the plot. Below is a side-by-side box plot with a dot plot overlayed to demonstrate the process.

```
boxplot(formula = sugar ~ Shelf, data = cereal, main = "Box and dot plot",
    ylab = "Sugar", xlab = "Shelf", pars = list(outpch = NA))
stripchart(x = cereal$sugar ~ cereal$Shelf, lwd = 2, col = "red",
    method = "jitter", vertical = TRUE, pch = 1, add = TRUE)
```



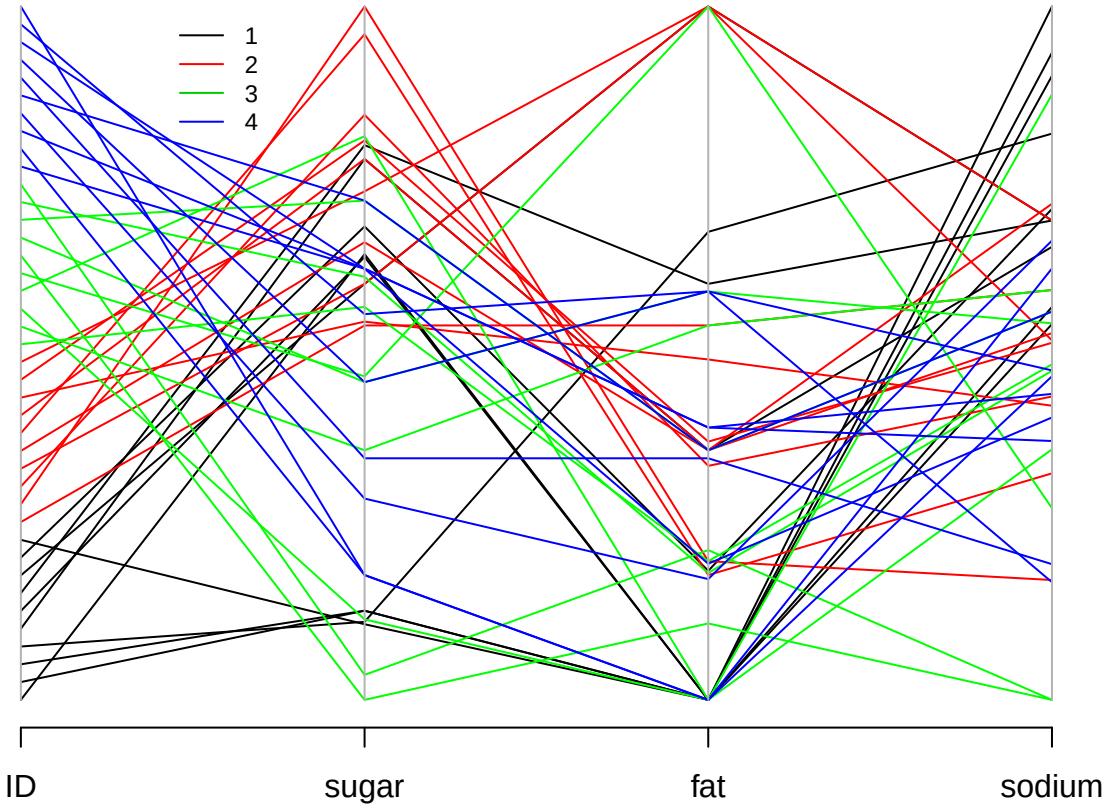

```
# Example showing that the plot is shown before this code
cereal[cereal$Cereal == "Kellog's Razzle Dazzle Rice Crispies",
    ]$sugar

## [1] 0.3571429
```

Notice that the plot was NOT displayed until after all functions that would change the plot (`stripchart()` used the `add = TRUE` argument value) were completed. This is the default, and it would be specified by `fig.show = "asis"`. To display the plot at the end of the chunk code, use `fig.show = "hold"`.

There are a number of other plotting options that can be set in the chunk. For example, the size of the plot can be changed using `fig.width` and `fig.height`, where both have default values of 7 inches. Also, the `fig.align` argument can be used to align the figure `"right"`, `"center"`, or `"left"` (this can be done the usual way in LyX too).

Figure 2: Parallel coordinate plot.



Plots typically need to be within figures for journal articles rather than in the main text. This can be done by putting the chunk in a figure box. Figure 2 demonstrates this process with a parallel coordinate plot and a floating figure. Note that the code is not shown in the compiled document because `echo = FALSE`.

Global options can be set if you want to use the same options for all plots. These options are set in the same way as we saw in Section 2 through using `opts_chunk$set()`.

5 Referencing other code

Previous code can be re-used again by referencing a chunk name. For example, we can show the sample means and standard deviations again by referencing the `MeanSD` chunk:

```
#Mean by shelf
aggregate(formula = sugar ~ Shelf, data = cereal, FUN = mean)

##   Shelf     sugar
## 1     1 0.2568366
## 2     2 0.4149686
## 3     3 0.2303732
## 4     4 0.2554839

#Standard deviation by shelf
aggregate(formula = sugar ~ Shelf, data = cereal, FUN = sd)
```

```
##   Shelf      sugar
## 1     1 0.16729566
## 2     2 0.09001019
## 3     3 0.15770057
## 4     4 0.11010226
```

Notice that the same options that were originally used with the `MeanSD` chunk are not used here. Also, in general, more than one chunk can be referenced by using the option `ref.label = c(<Chunk1>, <Chunk2>, ..., <Chunkn>)`.

External R code can be read into a chunk by the `read_chunk()` function. For example, suppose we want to read in ExternalCode.r that contains code to perform pairwise comparisons of the mean sugar contents. Below is the code, where I used `\lstset{breaklines = true, breakatwhitespace = true, xrightmargin = -32pt}` in the preamble and added `basicstyle = {\ttfamily}` as a parameter to DOCUMENT > SETTINGS > LISTINGS to format it.

```
1+1 #Example additional calculation

## @knitr LSD
pairwise.t.test(x = cereal$sugar, g = cereal$Shelf, p.adjust.method =
   "none", alternative = "two.sided") #LSD

1+2 #Example additional calculation

## @knitr Bonferroni
pairwise.t.test(x = cereal$sugar, g = cereal$Shelf, p.adjust.method =
   "bonferroni", alternative = "two.sided")  #Bonferroni
```

The `## knitr <Chunk name>` part of the previous code designates separate chunks within the file. Below is a chunk that reads in this file.

```
read_chunk(path = "ExternalCode.r")
```

Next, I reference only the first chunk within the external program.

```
pairwise.t.test(x = cereal$sugar, g = cereal$Shelf, p.adjust.method = "none",
    alternative = "two.sided") #LSD

##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  cereal$sugar and cereal$Shelf
##
##   1      2      3
## 2 0.0129 -      -
## 3 0.6642 0.0042 -
## 4 0.9823 0.0122 0.6803
##
## P value adjustment method: none
```

```
1+2  #Example additional calculation

## [1] 3
```

It is not clear to me how `knitr` knows where a chunk ends when reading an external file. In the case above, the LSD chunk obviously ended before the Bonferroni chunk started, so you can start a new chunk to end a chunk.

6 Cache

Depending on how long it takes for the R code to run, you may not want to run every chunk when a document is compiled. Instead, you may want to run chunks only where code has changed since the last compilation. This can be done by using a cache. Chunks where code will not change, such as reading and modifying the cereal data, can be run just once and its objects saved for subsequent compilations. Caching is turned on by using the `cache = TRUE` option in a chunk or by setting it globally.

When caching is turned on, the `knitr` package automatically creates a folder named cache. This folder is located in the same folder that the .lyx file is found. For example, suppose I want to perform the mean pairwise comparisons now using Tukey's honest significant differences method. Below is how I implement this code using the cache.

```
save.Tukey <- TukeyHSD(x = mod.fit, conf.level = 0.90)
save.Tukey

##   Tukey multiple comparisons of means
##     90% family-wise confidence level
##
## Fit: aov(formula = sugar ~ factor(Shelf), data = cereal)
##
## $`factor(Shelf)`
##             diff         lwr         upr     p adj
## 2-1  0.158131957  0.01447963  0.30178429 0.0595765
## 3-1 -0.026463461 -0.17011579  0.11718887 0.9715220
## 4-1 -0.001352752 -0.14500508  0.14229958 0.9999959
## 3-2 -0.184595418 -0.32824775 -0.04094309 0.0210678
## 4-2 -0.159484709 -0.30313704 -0.01583238 0.0566506
## 4-3  0.025110710 -0.11854162  0.16876304 0.9754841
```

Once the file is compiled, the cache folder appears with new files associated with the chunk so that they can be accessed by other chunks as needed. Note that this folder is not removed when LyX is exited.

Xie discusses the pitfalls to using a cache in his book. In particular, if one chunk depends on what happens in another chunk, this could cause a problem when the non-cached chunk is changed. The `dependson` option can be used in the cached chunk to make sure it is re-run if the chunk it depends on changes. Xie further gives the example of three chunks – `chunkA`, `chunkB`, and `chunkC` – where `chunkB` depends on what happens in `chunkA` and `chunkC` depends on what happens in `chunkB`. By using the option `dependson = "chunkA"` with `chunkB`, `chunkB` will always be re-run whenever `chunkA` changes. Similarly, by using the option `dependson = "chunkB"` with `chunkC`, `chunkC` would also be re-run whenever `chunkB` or even `chunkA` changes due to the chained dependencies.

Xie (2015) also discusses an "experimental" way to deal with dependencies in an automatic manner. A global option can be set by `opts_chunk$set(autodep = TRUE)` and then executing `dep_auto()` to figure

Table 1: Table created by printing a data frame as a table without row labels.

```
 Shelf Mean   SD
     1 0.26 0.17
     2 0.41 0.09
     3 0.23 0.16
     4 0.26 0.11
```

Table 2: Table created by using the `gridExtra` and `grid` packages to produce a plot in a table format. Additional control of the table coloring is described in the help for `tableGrob()`.

Shelf	Mean	SD
1	0.26	0.17
2	0.41	0.09
3	0.23	0.16
4	0.26	0.11

out the dependencies. Cached chunks should be re-run when the chunks they depend on change, but note that Xie says that an "approximation" is made when deciding on dependencies. Xie (2015) briefly mentions another way to automatically deal with dependencies by using the `dep_prev()` function. Note that I have not tried either of these approaches.

7 Tables

Tables are often essential for papers and reports. By creating a table completely using R, potential data entry errors can be avoided that may occur during manual construction. The use of `knitr` to run R code and create output then helps in this process. For example, data frames can simply be printed. However, you may want to make your tables look a little more "fancy" than what is given by a printed data frame. Xie suggests the use of the `xtable` and `tables` packages. These packages simply create the LaTeX code that would normally be used to display a table with LaTeX. Xie (2013) also provides an example on page 129 using the `gridExtra` package. Xie recently added the `kable()` function in `knitr` to help create tables as well. Tables 1, 2, 3, 4, and 5 show a few examples of constructing tables. In particular, Tables 3 and 4 are in the format commonly found in statistical research papers.

Table 3: Table created using the xtable package.

Shelf	Mean	SD
1	0.26	0.17
2	0.41	0.09
3	0.23	0.16
4	0.26	0.11

Table 4: Table created using the `xtable` package within a LyX float box.

Shelf	Mean	SD
1	0.26	0.17
2	0.41	0.09
3	0.23	0.16
4	0.26	0.11

Table 5: Table created using the `kable()` function within a LyX float box.

Shelf	Mean	SD
1	0.26	0.17
2	0.41	0.09
3	0.23	0.16
4	0.26	0.11

8 Errors

How do you know if there was a `knitr` code problem? Indications of a problem include:

- Chunk code in the compiled document is formatted in the same way as regular text, and there is no output corresponding to the code.
- A window similar to what is shown in Figure 3 will appear during compilation. The error for this particular figure was generated by using two chunks with the same name.

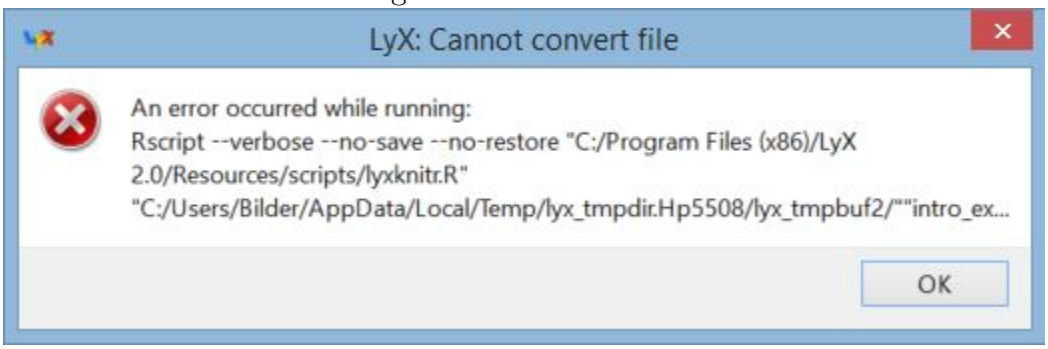
Note that `knitr` does not stop when there are R coding errors. For example, if R code included `1 + "a"`, R would indicate this was a syntax R, but the LyX document would continue to compile.

9 Additional notes

Below are some additional notes:

- `library(package = "knitr")` does not need to be specified anywhere.
- The `all_labels()` function shows a list of all chunks within a document.

Figure 3: Error window



```
all_labels()
```

```
##  [1] "ReadInData"      "ChangeVar"       "MeanSD"
##  [4] "ANOVAmodel1"     "GlobOpts"        "ANOVAmodel2"
##  [7] "BoxDotplot1"     "ParCoord"        "ExampleCR"
## [10] "ReadChunk"       "ShowLSD"         "Tukey"
## [13] "Table1"          "Table2"          "Table3"
## [16] "Table4"          "Table5"          "labels"
## [19] "version"         "options"         "unnamed-chunk-1"
## [22] "LSD"             "Bonferroni"
```

- Page 137 of Xie (2015) gives an example of how to suppress part of a long set of output from being printed.
- Hooks are detailed in Chapter 10 of Xie (2015) and at `http://yihui.name/knitr/hooks`. These are user written functions that provide extensions to the default capabilities of `knitr`. For example, page 103 describes a hook to crop plots.
- `knitR` uses Rscript.exe to run the R code. This is a way to run one line at a time in what is known as "batch mode" for running R. Notice that "Rscript" with some options is mentioned in the window of Figure 3 indicating there was a problem with running `knitr` and R.
- Please see my R lecture notes for this course to see how I use `knitr` and LyX extensively!
- What version of R is LyX using?

```
R.Version() # Look at major and minor
```

```
## $platform
## [1] "x86_64-w64-mingw32"
##
## $arch
## [1] "x86_64"
##
## $os
## [1] "mingw32"
##
## $system
## [1] "x86_64, mingw32"
##
## $status
## [1] ""
##
## $major
## [1] "3"
##
## $minor
## [1] "2.2"
```

```
##
## $year
## [1] "2015"
##
## $month
## [1] "08"
##
## $day
## [1] "14"
##
## $`svn rev`
## [1] "69053"
##
## $language
## [1] "R"
##
## $version.string
## [1] "R version 3.2.2 (2015-08-14)"
##
## $nickname
## [1] "Fire Safety"
```

- The `opts_chunk$get()` function shows a list of all options and their values.

```
opts_chunk$get()

## $eval
## [1] TRUE
##
## $echo
## [1] TRUE
##
## $results
## [1] "markup"
##
## $tidy
## [1] FALSE
##
## $tidy.opts
## NULL
##
## $collapse
## [1] FALSE
##
## $prompt
## [1] FALSE
##
```

```
## $comment
## [1] "##"
##
## $highlight
## [1] TRUE
##
## $strip.white
## [1] TRUE
##
## $size
## [1] "normalsize"
##
## $background
## [1] "#F7F7F7"
##
## $cache
## [1] FALSE
##
## $cache.path
## [1] "cache/"
##
## $cache.vars
## NULL
##
## $cache.lazy
## [1] TRUE
##
## $dependson
## NULL
##
## $autodep
## [1] FALSE
##
## $cache.rebuild
## [1] FALSE
##
## $fig.keep
## [1] "high"
##
## $fig.show
## [1] "asis"
##
## $fig.align
## [1] "default"
##
## $fig.path
## [1] "figure/"
##
```

```
## $dev
## [1] "pdf"
##
## $dev.args
## NULL
##
## $dpi
## [1] 72
##
## $fig.ext
## NULL
##
## $fig.width
## [1] 7
##
## $fig.height
## [1] 7
##
## $fig.env
## [1] "figure"
##
## $fig.cap
## NULL
##
## $fig.scap
## NULL
##
## $fig.lp
## [1] "fig:"
##
## $fig.subcap
## NULL
##
## $fig.pos
## [1] ""
##
## $out.width
## [1] "\\maxwidth"
##
## $out.height
## NULL
##
## $out.extra
## NULL
##
## $fig.retina
## [1] 1
##
```

```
## $external
## [1] TRUE
##
## $sanitize
## [1] FALSE
##
## $interval
## [1] 1
##
## $aniopts
## [1] "controls,loop"
##
## $warning
## [1] TRUE
##
## $error
## [1] TRUE
##
## $message
## [1] TRUE
##
## $render
## NULL
##
## $ref.label
## NULL
##
## $child
## NULL
##
## $engine
## [1] "R"
##
## $split
## [1] FALSE
##
## $include
## [1] TRUE
##
## $purl
## [1] TRUE
```