Simple Markdown Example

PDF/LaTex version

Jenny Rieck
Derek Beaton

May 16, 2019

Introduction

For reference, see RMarkdown Driven Development by Emily Riederer for a comprehensive overview of how to best structure RMarkdown for projects, packages, and other development-driven tasks.

In general for this RMarkdown file we follow Emily's fourth example with a directory structure, minimized redundancies, and heavy-duty code elsewhere. This RMarkdown, generally, serves as place to describe, analyze, and visualize our data. Thus, this text is shown at the top of the output, but actually appears after several `R` code chunks, which exist between the `Introduction` heading and YAML header information (which you see as title, authors, dates, etc. . .).

Most of the RMarkdown files in this directory will show generally the same content, but help highlight the different ways you can use RMarkdown, `knitr`, `pandoc`, `LateX`, and various package built for those, such as `beamer` (`LateX`) for presentations, and `papaja` & `rticles` (R/RMarkdown) for writing manuscripts that export to `LaTeX` or MS Word. If you decide to write MS Word documents through RMarkdown, you should also use the `redoc` package.

R chunks: A word of caution

It is good practice to name your `R` chunks. If you do not, then the `R` chunks will still produce the intended material. However, when you do name them, you should ensure they have unique names (else, you will likely see some cryptic and not always informative error messages).

Tables

There are multiple approaches and packages to help visualize tables or tabular information. Let's start by looking at a simple summary of all the continuous variables. First, we will visualize the summary table through two methods within R: `knitr::kable` and the `kablExtra` package, followed by `grid` and `gridExtra`. Next, we will use the same data and illustrate what happens when we pass it to Python through `reticulate`.

In this section we wil also show the code chunks that generate these tables and visuals, which are embedded in the RMarkdown document.

knitr and kableExtra

To make HTML and LaTeX tables in RMarkdown, one of the easiest and most common options is through `knitr`. The `knitr` package is, effectively, the tool to make RMarkdown documents go from R & RMarkdown (plus other code and LaTeX) into PDFs or HTML pages. We'll start with the `knitr::kable()`.

```
example_table <- amerge_subset[, c("AGE", "MOCA", "CDRSB", "WholeBrain",
    "Hippocampus", "MidTemp")]
example_table_Dx <- amerge_subset[, c("DX", "AGE", "MOCA", "CDRSB",
    "WholeBrain", "Hippocampus", "MidTemp")]
```

```
kable(summary(example_table))
```

	AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
	Min. :55.00	Min. :16.00	Min. :0.000	Min. : 817421	Min. : 3731	Min. :12213
	1st Qu.:67.20	1st Qu.:22.00	1st Qu.:0.000	1st Qu.: 984410	1st Qu.: 6510	1st Qu.:18535
	Median :71.90	Median :24.00	Median :1.000	Median :1051621	Median : 7223	Median :20186
	Mean :71.92	Mean :23.89	Mean :1.202	Mean :1057026	Mean : 7150	Mean :20302
	3rd Qu.:76.60	3rd Qu.:26.00	3rd Qu.:2.000	3rd Qu.:1120570	3rd Qu.: 7834	3rd Qu.:22088
	Max. :89.60	Max. :30.00	Max. :5.500	Max. :1486036	Max. :10602	Max. :32189

But that is not particularly nice looking. So we can use some parameters to make this table look better (which depend on having LaTeX).

```
kable(summary(example_table), format = "latex", booktabs = T)
```

AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
Min. :55.00	Min. :16.00	Min. :0.000	Min. : 817421	Min. : 3731	Min. :12213
1st Qu.:67.20	1st Qu.:22.00	1st Qu.:0.000	1st Qu.: 984410	1st Qu.: 6510	1st Qu.:18535
Median :71.90	Median :24.00	Median :1.000	Median :1051621	Median : 7223	Median :20186
Mean :71.92	Mean :23.89	Mean :1.202	Mean :1057026	Mean : 7150	Mean :20302
3rd Qu.:76.60	3rd Qu.:26.00	3rd Qu.:2.000	3rd Qu.:1120570	3rd Qu.: 7834	3rd Qu.:22088
Max. :89.60	Max. :30.00	Max. :5.500	Max. :1486036	Max. :10602	Max. :32189

With `booktabs` and `latex` format, we've made the table look a little better. But can we make it look even better than that? We can with `kableExtra`.

```
kable(summary(example_table), format = "latex", booktabs = T) %>%
    kable_styling(font_size = 10, position = "center")
```

AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
Min. :55.00	Min. :16.00	Min. :0.000	Min. : 817421	Min. : 3731	Min. :12213
1st Qu.:67.20	1st Qu.:22.00	1st Qu.:0.000	1st Qu.: 984410	1st Qu.: 6510	1st Qu.:18535
Median :71.90	Median :24.00	Median :1.000	Median :1051621	Median : 7223	Median :20186
Mean :71.92	Mean :23.89	Mean :1.202	Mean :1057026	Mean : 7150	Mean :20302
3rd Qu.:76.60	3rd Qu.:26.00	3rd Qu.:2.000	3rd Qu.:1120570	3rd Qu.: 7834	3rd Qu.:22088
Max. :89.60	Max. :30.00	Max. :5.500	Max. :1486036	Max. :10602	Max. :32189

We can take the table look even further with additional options, like "stripes".

```
kable(summary(example_table), format = "latex", booktabs = T) %>%
    kable_styling(font_size = 10, position = "center", latex_options = "striped")
```

AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
Min. :55.00	Min. :16.00	Min. :0.000	Min. : 817421	Min. : 3731	Min. :12213
1st Qu.:67.20	1st Qu.:22.00	1st Qu.:0.000	1st Qu.: 984410	1st Qu.: 6510	1st Qu.:18535
Median :71.90	Median :24.00	Median :1.000	Median :1051621	Median : 7223	Median :20186
Mean :71.92	Mean :23.89	Mean :1.202	Mean :1057026	Mean : 7150	Mean :20302
3rd Qu.:76.60	3rd Qu.:26.00	3rd Qu.:2.000	3rd Qu.:1120570	3rd Qu.: 7834	3rd Qu.:22088
Max. :89.60	Max. :30.00	Max. :5.500	Max. :1486036	Max. :10602	Max. :32189

Given that we have redundant information in the table (min/max, etc...) we can do a better job and make an even nicer table with an `apply()`, and then use multiple `kable` and `kableExtra` features to make a really nice table.

```
better_example_table <- apply(example_table, 2, summary)

kable(better_example_table, format = "latex", booktabs = T, digits = 2) %>%
    kableExtra::add_header_above(c(Statistic = 1, Demographic = 1,
        Clinical = 2, Brain = 3)) %>% kable_styling(font_size = 10,
    position = "center", latex_options = "striped") %>% row_spec(0,
    angle = 15, bold = T)
```

Statistic	Demographic	Clinical		Brain		
	AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
Min.	55.00	16.00	0.0	817421.2	3731.00	12213.00
1st Qu.	67.20	22.00	0.0	984409.9	6510.00	18535.00
Median	71.90	24.00	1.0	1051621.3	7223.00	20186.00
Mean	71.92	23.89	1.2	1057025.6	7149.61	20301.93
3rd Qu.	76.60	26.00	2.0	1120569.5	7834.00	22088.00
Max.	89.60	30.00	5.5	1486035.6	10602.00	32189.00

grid and gridExtra

Sometimes we need tables to be graphics. This is where the `grid` and `gridExtra` packages come in. The `grid` package uses `grobs` to turn items—tables, figures, all sorts of things—into configurable parts of a figure.

```
grid.table(better_example_table)
```

	MOCA	CDRSB	WholeBrain	Hippocampus
	16	0	817421.228648	3731
	22	0	984409.906002	6510
	24	1	1051621.329331	7223
9774	23.8902255639098	1.20225563909774	1057025.5509328	7149.613533834
	26	2	1120569.496532	7834
	30	5.5	1486035.644341	10602

But it's clear we have a few extra things we should do to make this figure of a table look better.

```
better_example_table_round <- apply(better_example_table, 2,
    format, digits = 2, nsmall = 2)

grid.table(better_example_table_round)
```

	AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
Min.	55.00	16.00	0.00	817421.23	3731.00	12213.00
1st Qu.	67.20	22.00	0.00	984409.91	6510.00	18535.00
Median	71.90	24.00	1.00	1051621.33	7223.00	20186.00
Mean	71.92	23.89	1.20	1057025.55	7149.61	20301.93
3rd Qu.	76.60	26.00	2.00	1120569.50	7834.00	22088.00
Max.	89.60	30.00	5.50	1486035.64	10602.00	32189.00

However, the `grid` and `gridExtra` packages can be difficult to customize many of the pieces. Therefore it might be easier to stick to the `LaTeX` approaches with `kable` or it is well worth checking out the `gt` package.

Python via reticulate

What if you now love RMarkdown but are still really into Python? Not a problem. The `reticulate` package has you covered. It's an R package to connect to your `Python` installation and bring the data or results back into `R`, but with some of the same features as you're used to in either `Python` or `R`. Let's start out with a *head to head* of `R`'s `head()` vs. `Python`'s `.head()`.

In `R` we write the call in RMarkdown as if we would normally in `R`:

```
head(example_table)
```

```
##       AGE MOCA CDRSB WholeBrain Hippocampus MidTemp
## 2002 64.8   28   2.5  1135556.6        7960   21867
## 2003 63.6   24   2.0  1070369.5        7611   21580
## 2007 83.4   23   2.5   920710.1        5614   20567
## 2010 62.9   27   0.5   986402.9        8004   20358
## 2011 69.9   25   1.5   987822.5        6686   20366
## 2018 76.4   26   1.5  1004817.0        7774   19531
```

Unlike the previous code chunks, we have to tell RMarkdown that the language it should expect is `python` instead of `R`.

```
print(r.example_table.head())
```

```
##        AGE  MOCA  CDRSB    WholeBrain  Hippocampus  MidTemp
## 2002  64.8  28.0    2.5  1.135557e+06       7960.0  21867.0
## 2003  63.6  24.0    2.0  1.070369e+06       7611.0  21580.0
## 2007  83.4  23.0    2.5  9.207101e+05       5614.0  20567.0
## 2010  62.9  27.0    0.5  9.864029e+05       8004.0  20358.0
## 2011  69.9  25.0    1.5  9.878225e+05       6686.0  20366.0
```

For `Python` via `R` we also need to use the `.` (dot notation) because of `Python`'s object oriented approach. From the `r` object, we get the `example_table` attribute and then perform the `head` method. That's because `Python` needs to know about the object coming from `R`.

Here be .dragons

For those more familiar with `R` style that stems from the `S` language origin or Google's style guide, the `.` can be a substantial source of confusion both here and in the `tidyverse`. In base `R`, it is a valid character for user defined items. But in base `R` it is *also* used for objects and classes (see, e.g., `.print()`). In the `tidyverse` the `.` has a special purpose when alone (not amongst other characters), often as a placeholder for where to pass a variable as an argument into a function.

In `R`: use `.` either with caution or reckless abandon.

Passssing back and foRth

That subtitle is a stretch! With the `reticulate` package and `R` we can pass items between the the two languages/environments. And, depending on which language, we use that language's preferred/standard approach of referring to attributes or objects. In this next example we call into the `.describe()` method in `Python`, which is similar to `R`'s `summary()`, but allows/requires us to define certain parameters. We use the `r` object to pass `example_table` through to `Python` and call the `describe()` method. We can visualize the results directly using the `print()` method.

```
perc = [.25, .50, .75]
desc = r.example_table.describe(percentiles = perc)
print(desc)
```

```
##               AGE        MOCA       CDRSB    WholeBrain   Hippocampus  \
## count  665.000000  665.000000  665.000000  6.650000e+02    665.000000
## mean    71.922556   23.890226    1.202256  1.057026e+06   7149.613534
## std      6.868621    3.279405    1.343238  1.036727e+05   1086.040463
## min     55.000000   16.000000    0.000000  8.174212e+05   3731.000000
## 25%     67.200000   22.000000    0.000000  9.844099e+05   6510.000000
## 50%     71.900000   24.000000    1.000000  1.051621e+06   7223.000000
## 75%     76.600000   26.000000    2.000000  1.120569e+06   7834.000000
## max     89.600000   30.000000    5.500000  1.486036e+06  10602.000000
##
##             MidTemp
## count    665.000000
## mean   20301.933835
## std     2675.571327
## min    12213.000000
## 25%    18535.000000
## 50%    20186.000000
## 75%    22088.000000
```

```
## max    32189.000000
```

But like previous results shown in R, we can do a lot better with how we display the table. We can pass the desc object from Python back to R with the $ notation (generally for lists or certain classes in R). When we retrieve an object from python (via py$), we can then do what we would usually do in R. Here, we go back to kable() and kable_styling() with some of the parameters to make very nice looking LaTeX tables.

```
kable(py$desc, digits = 2, format = "latex", booktabs = T) %>%
    kable_styling(font_size = 10, position = "center", latex_options = "striped")
```

	AGE	MOCA	CDRSB	WholeBrain	Hippocampus	MidTemp
count	665.00	665.00	665.00	665.0	665.00	665.00
mean	71.92	23.89	1.20	1057025.6	7149.61	20301.93
std	6.87	3.28	1.34	103672.7	1086.04	2675.57
min	55.00	16.00	0.00	817421.2	3731.00	12213.00
25%	67.20	22.00	0.00	984409.9	6510.00	18535.00
50%	71.90	24.00	1.00	1051621.3	7223.00	20186.00
75%	76.60	26.00	2.00	1120569.5	7834.00	22088.00
max	89.60	30.00	5.50	1486035.6	10602.00	32189.00

Et voila!

Analyses

In order to make work more reproducible, most analyses can—and should—be performed each time we knit an RMarkdown document. Here we show two examples of how to do this: (1) directly in an R chunk and (2) calling a separate file from an R chunk. To note, this latter option is best done early in the RMarkdown file but is located here for illustrative purposes.

Mixed data MCA

Here we perform a small analyses of mixed data types (continuous, categorical, ordinal) through multiple correspondence analysis. For more information on this, see this workshop.

```
continuous_data <- amerge_subset[, which(variable_type_map[,
    "Continuous"] == 1)]
ordinal_data <- amerge_subset[, which(variable_type_map[, "Ordinal"] ==
    1 & variable_type_map[, "Categorical"] != 1)]
categorical_data <- amerge_subset[, which(variable_type_map[,
    "Categorical"] == 1)]
categorical_data <- categorical_data[, -c(which("DX" %in% colnames(categorical_data)))]

continuous_escofier_data <- escofier.coding(continuous_data,
    scale = T)
ordinal_thermometer_data <- thermometer.coding(ordinal_data)
categorical_disjunctive_data <- make.data.nominal(categorical_data)
mixed_data <- cbind(continuous_escofier_data, ordinal_thermometer_data,
    categorical_disjunctive_data)

ca_results <- epCA(mixed_data, DESIGN = amerge_subset$DX, make_design_nominal = T,
    graphs = F)
```

This code is shown, but is directly show in the RMarkdown besides the code within the chunk. This is because the values are stored in their respective variables, and none of these are visuals (e.g., tables, figures).

CovSTATIS

Finally, we will also perform an analysis called 'covSTATIS', which is a multi-table PCA technique that allows us to peform the analysis of multiple correlation or covariance matrices. Here we have one correlation matrix per group in the data (control, MCI, AD). For this analysis, we will actually use the `R` chunk to call off to a script in the correct folder. We introduce another package to help with reproducibility and sharing work across multiple places or individuals: the `here` package. However, for this example, we will actually *hide* the code chunk with parameters in the chunk header: `echo=FALSE`. thus, in order to see this, you'll have to look at the .Rmd file, as it will not appear in the .pdf file.

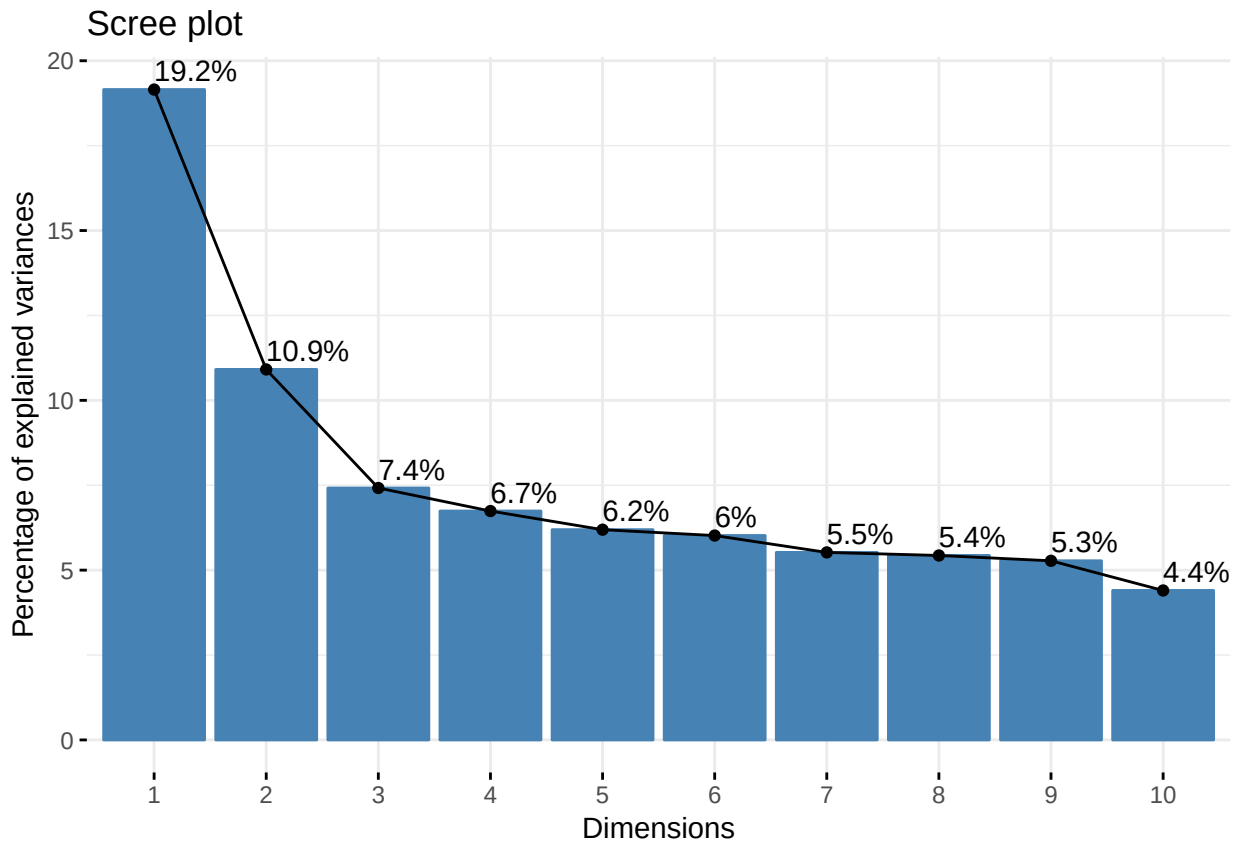
Visualizations & Graphics

RMarkdown also allows us to include graphics directly from the code. In the global settings for this document (see the top), we have to set a specific `knitr` flag (`fig.pos='H'`) which passes through to `LaTeX` to enforce that images appear in the document at their respective locations. Else, because of `LaTeX`'s magic, figures end up in slightly different places than expected (but still in order).

In our first example, we performed MCA through the `ExPosition` package. We'd like to visualize these results, so, we will use a `ggplot2`-based package called `factoextra`, which provides standard visualizations for PCA-like techniques across multiple packages (e.g., `ExPosition`, `ade4`, `ca`, `FactoMineR`).

First we show a scree plot (explained variance per component) completely as is.

```
fviz_screeplot(ca_results, addlabels = TRUE)
```



```
plus_minus <- ifelse(grepl("\\-", rownames(ca_results$ExPosition.Data$fj)),
    "-", "+")
ca_col <- fviz_ca_col(ca_results, alpha = 0, labels = F, col.col = "white") +
    geom_text(aes(label = rownames(ca_results$ExPosition.Data$fj),
        color = plus_minus)) + theme(panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(), axis.text.x = element_blank(),
    axis.ticks.x = element_blank(), axis.text.y = element_blank(),
    axis.ticks.y = element_blank(), legend.position = "none") +
    xlim(c(ca_results$Plotting.Data$constraints$minx, ca_results$Plotting.Data$constraints$maxx)) +
    ylim(c(ca_results$Plotting.Data$constraints$miny, ca_results$Plotting.Data$constraints$maxy)) +
    xlab(paste0("Component 1. Explained variance: ", round(ca_results$ExPosition.Data$t[1],
        digits = 2), "%")) + ylab(paste0("Component 2. Explained variance: ",
    round(ca_results$ExPosition.Data$t[2], digits = 2), "%")) +
    ggtitle("CA:\nVariable Component Scores")
```

This image is a bit big. So, we can resize and align this figure, as well as provide a caption for it directly from the header of the RMarkdown chunk. For this, we need to use and set the following parameters: `fig.height=4`, `fig.width=4`, `fig.cap="Scree plot that shows the explained variance per component."`, and `fig.align="center"` parameters in the chunk header.

```
fviz_screeplot(ca_results, addlabels = TRUE)
```

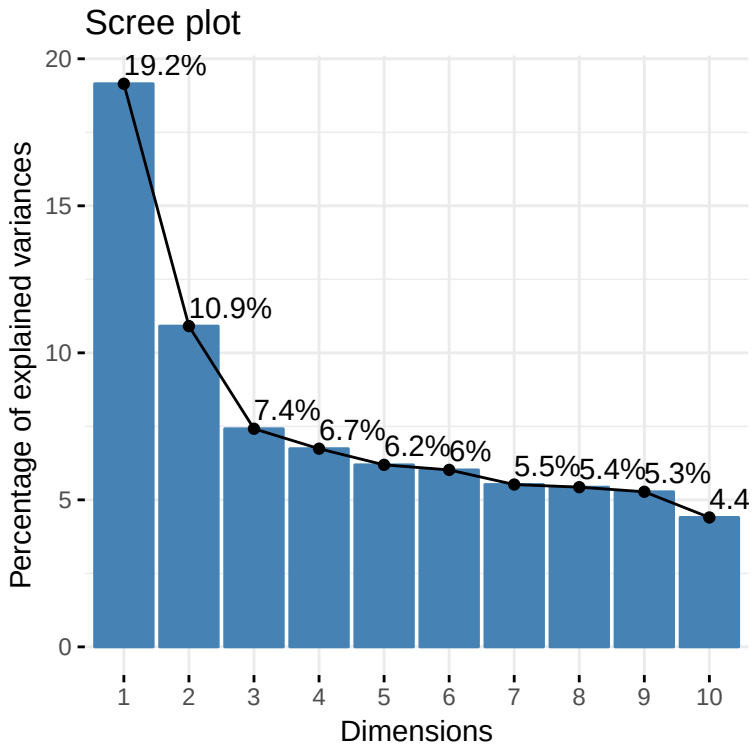


Figure 1: Scree plot that shows the explained variance per component.

That's much nicer! Next we have another hidden chunk (i.e., `echo=FALSE`) that performs required set up for the subsequent visuals.

The hidden code chunk makes graphics based on `factoextra` and `ggplot2`. To see this code, open this .Rmd file. From the chunks we can visualize multiple items, as opposed to just one at a time like we did with the

tables and the screeplot. One way is to just directly call for the graphic in the chunk.

```
ca_col
```

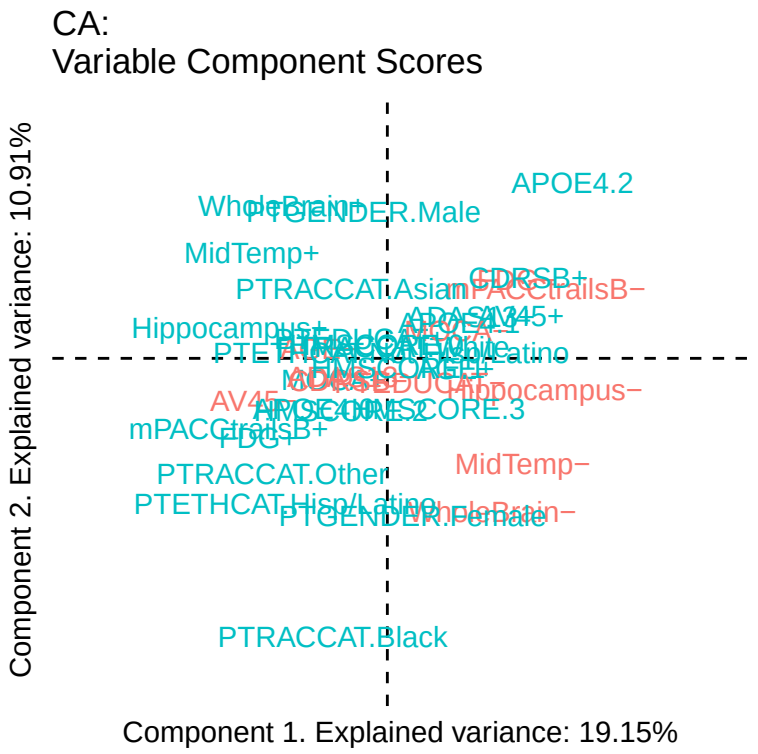


Figure 2: Multiple ggplot2 graphics

```
ca_row
```

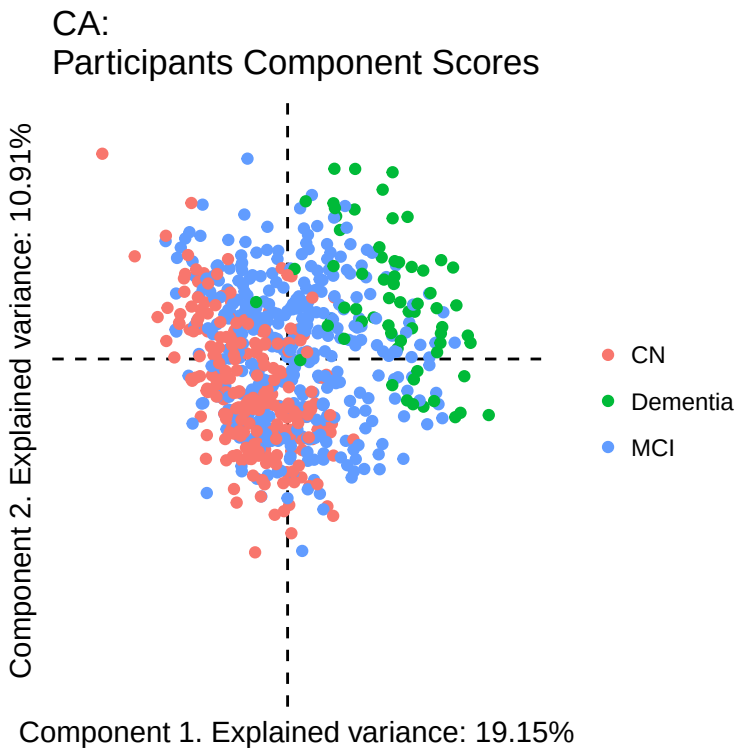


Figure 3: Multiple ggplot2 graphics

But the above figure shows the code chunks and figures in order. To prevent this, we need to hide the chunk with `echo=FALSE`.

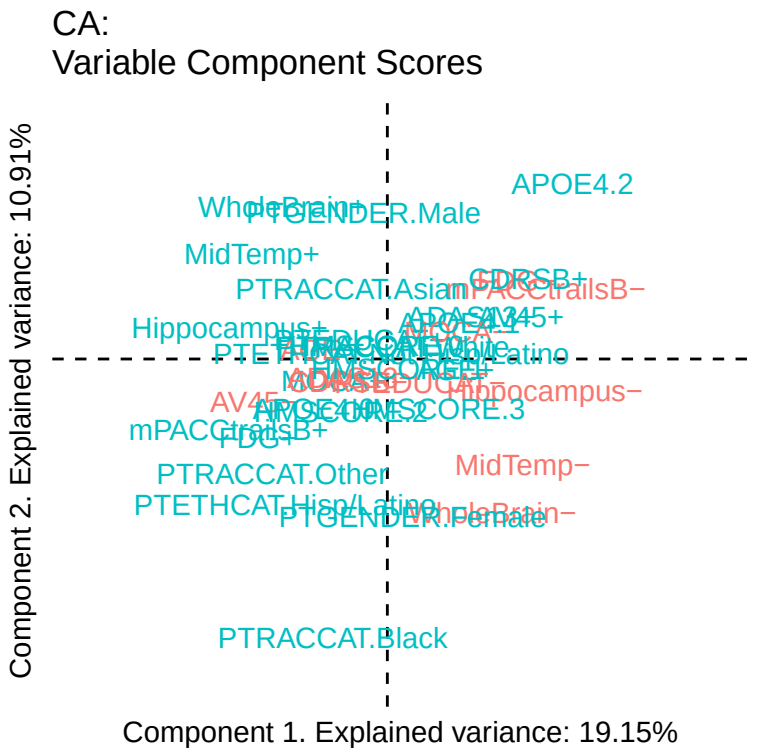


Figure 4: Multiple ggplot2 graphics

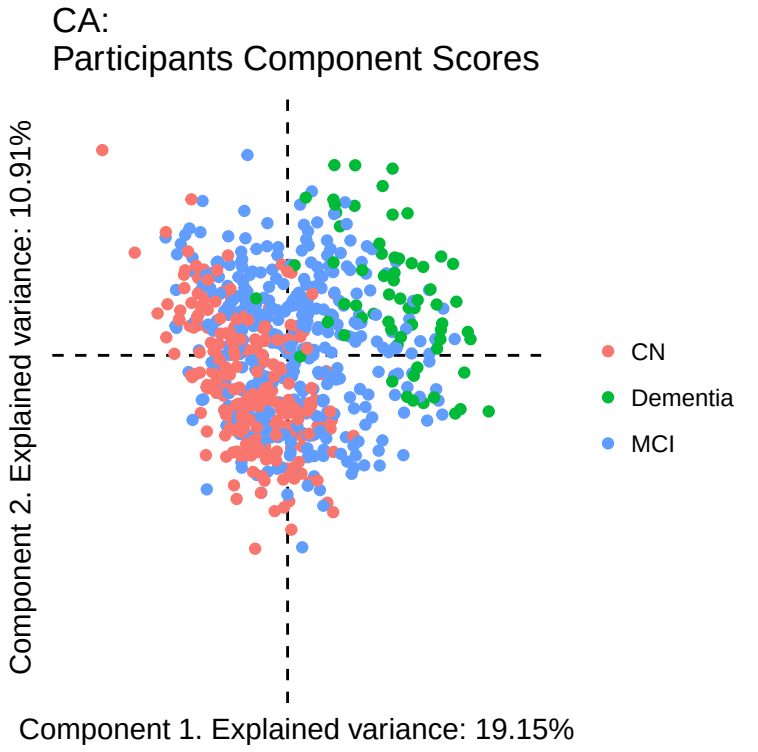


Figure 5: Multiple ggplot2 graphics

However, sometimes we may use "base" R or need much finer grained control over figure placements within the chunk. The subsequent chunk shows the results from the covSTATIS analysis, with the "compromise component scores" in each plot, but with each group in isolation with respect to the "compromise component scores", and then finally all items plotted at once.

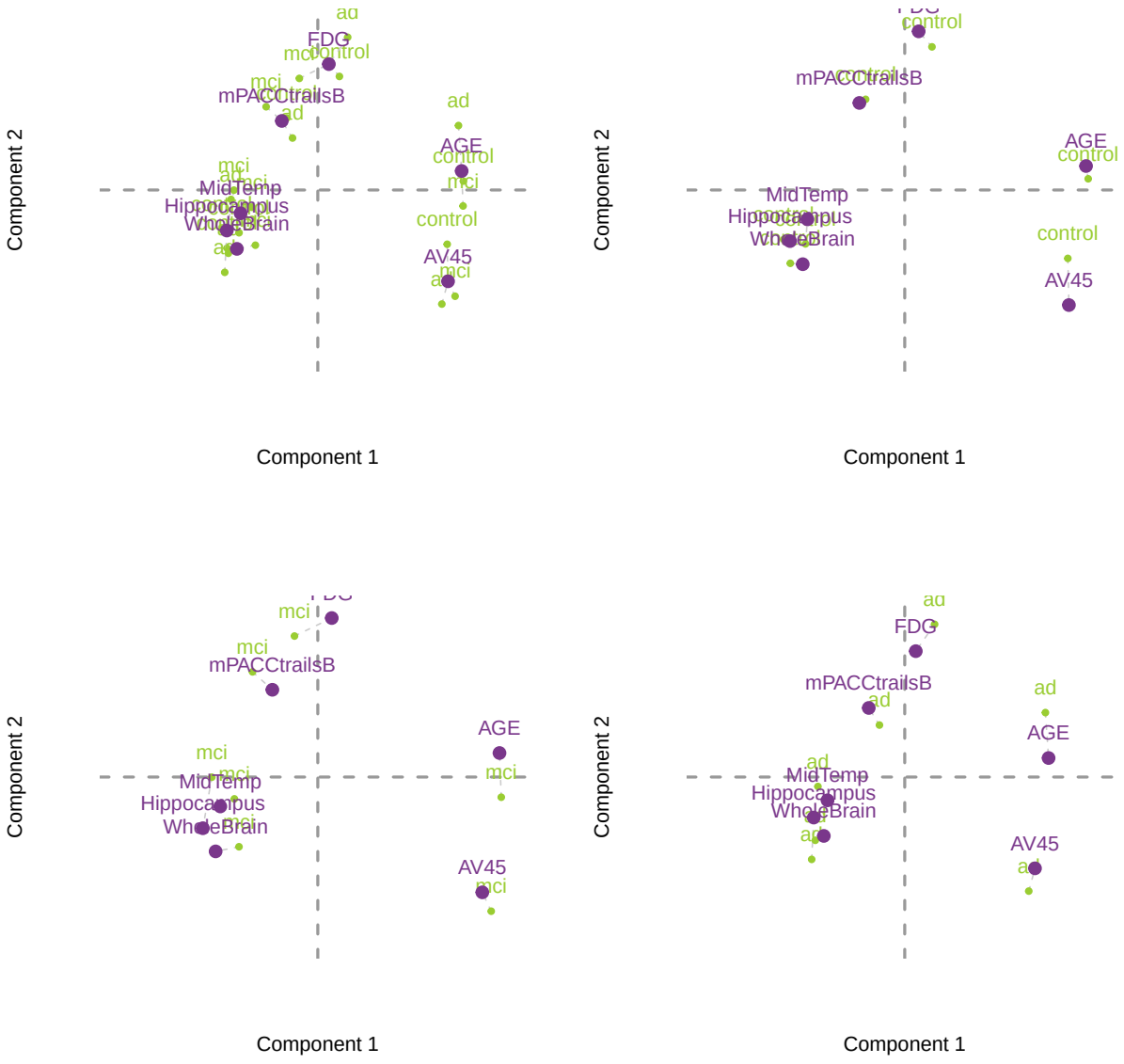


Figure 6: Using par() to make a layout of multiple base plots

Conclusions

RMarkdown—and a number of other tools provided through RStudio—allow you to make relatively simple, integrated, and reproducible documents. This particular example includes a wide variety of tools (R, Python, multiple packages, in-chunk code, and external scripts), but relies only on relatively standard and simple RMarkdown options.