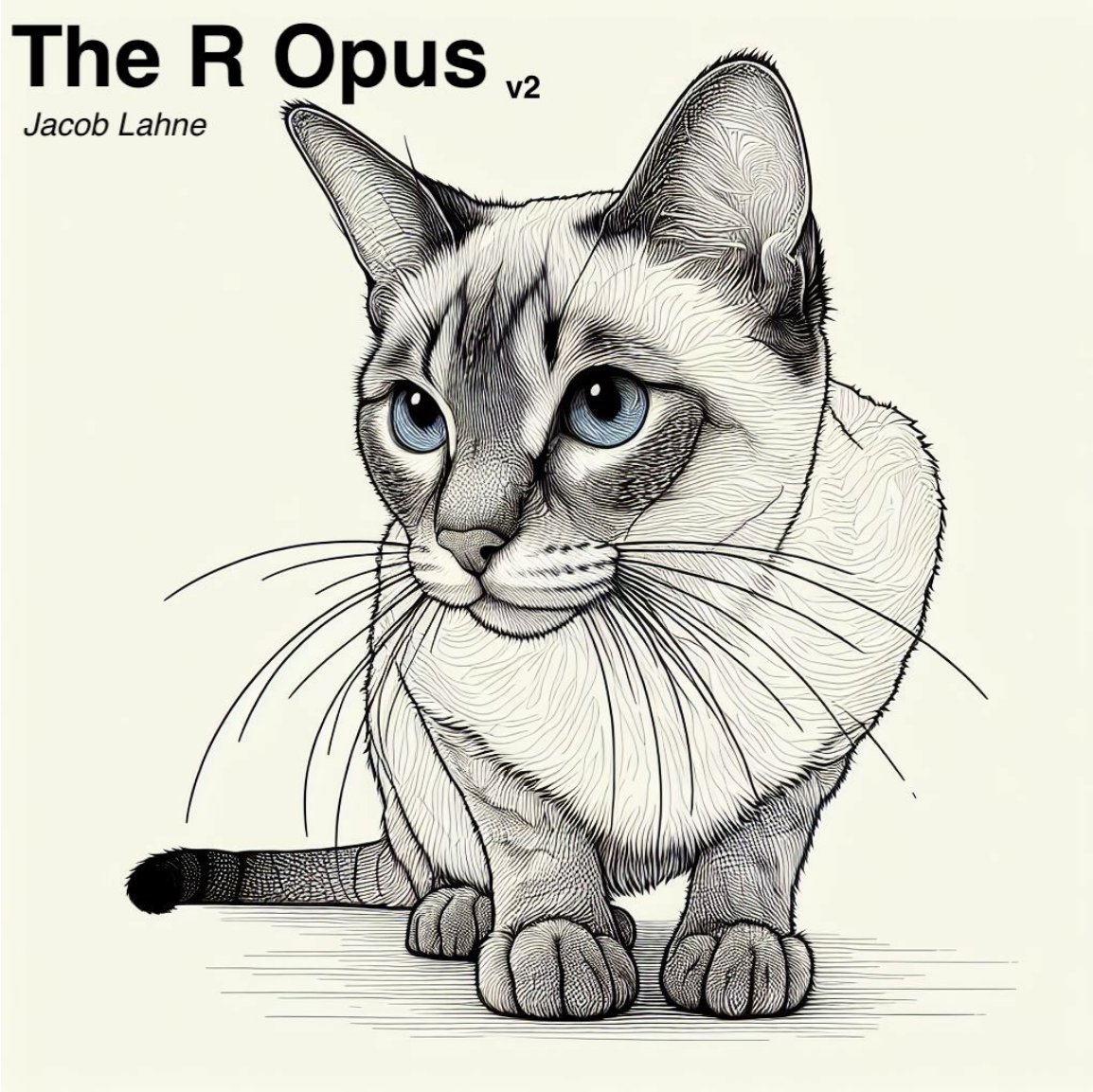


"The R Opus v2"

Jacob Lahne

2024-01-10

Contents

About

To paraphrase one of my inspirations for this project [Kurz, 2023]: This is a labor of love. In 2015, Hildegarde Heymann (from here on: HGH), Distinguished Professor of Viticulture & Enology at UC-Davis, was kind enough to host me as a visiting scholar at her laboratory. Among many other formative experiences I had as a sensory scientist at UC-Davis, HGH shared with me the **R Opus**, a handbook she had written and compiled of common analytical approaches to multivariate (food-sensory) data in `R`. Like many of her other mentees, students, and postdocs, I benefited immensely from HGH's practical insight into how to apply abstruse multivariate analyses to real problems in research, and the **R Opus** manifested that knowledge into a hands-on guide for how to implement those tools.

In the time since, I have passed on the **R Opus** to my own mentees, students, and postdocs. As `R` has continued to mature and become more accessible as a scripting language for data analysis–in particular, as "tidy" programming principles have become more dominant–I have found myself also passing on my own set of tips and tricks for how to transform the tools found in the original **R Opus** into the current vernacular. I began teaching data analytics and coding for researchers using `R`, and after learning how to (clumsily) transform my course notes into accessible **bookdowns**, I thought: why not the **R Opus**?

This is that thought put into some sort of action. I hope it is useful for you.

Usage

This **bookdown** is constructed around typical workflows and data analysis needs for sensory scientists. You know who you are.

For all others, this bookdown is a structured introduction to the analysis of multivariate data from a practical and applied perspective. Specifically, we investigate how to apply a series of common multivariate statistical analyses to a set of data derived from the tasting and rating of wines by both trained and untrained human subjects. I place almost all of the emphasis on "how to", and much less on the statistical theory behind these approaches. As I have spent

longer and longer using the statistical tools sensory scientists commonly apply (and occasionally developing and adapting new ones), I have come to believe that it is much more important to think of statistical analyses as tools that we apply, rather than worrying about our complete abstract understanding. The latter, of course, is also extremely important, but cannot be built without any possibility of first understanding why we might choose a particular analysis, and what it will look like applied to a particular dataset.

Even if you do not work in the field of sensory science, I hope that these examples will prove useful and easily understandable. Plus: thinking about how wine tastes is interesting, especially when we combine it with complicated statistics!

`R` Setup

You can read this bookdown entirely online using the navigation panels. However, if you want to learn to use `R` to conduct analyses like these, I strongly suggest you follow along. You'll need to install `R` to do so. To install `R`, go to https://cran.r-project.org/ and follow the appropriate instructions for your operating system.

While it is not strictly necessary, you will almost certainly find it more pleasant to use the RStudio IDE (Interactive Development Environment) for `R`, which can be downloaded and installed from https://posit.co/products/open-source/rstudio/.

Here's the list of packages I used in this bookdown:

package
tidyverse
FactoMineR
patchwork
ggrepel
ggedit
ggforce
DistatisR
SensoMineR
paletteer
here
broom
skimr
factoextra
naniar
agricolae
tidytext
brms
tidybayes
simputation
missMDA
corrr
widyr
rgl
candisc
MASS
ca
pls

Once you have set up `R` (and RStudio), you can run the following lines of code to install the packages I use in this bookdown. This might take a minute–and you might have to restart `R` to do it. Go get a snack!

```
packages <- c("tidyverse", "FactoMineR", "patchwork", "ggrepel", "ggedit", "ggforce", "DistatisR"

install.packages(packages, dependencies = TRUE)
```

Some of these tools install some additional (extra-`R`) tools. The Stan-based tools are most likely to cause trouble. If you cannot install them, consider the walkthroughs on the main Stan website. Not installing them means you just won't be able to replicate my extended experiments with Bayesian modeling (probably not the greatest loss in the world for you).

What this is not

I will not be going over the basics of R coding and programming. You can pick up a a fair amount by following along, or if you are truly new to R I recommend checking out Wickham et al's now-classic introduction, the Stat545 website, or any of the Carpentries workshops.

About me

I'm an associate professor of Food Science & Technology at Virginia Tech. I teach about sensory evaluation and about applied data analysis for food and ag scientists. I am not a statistician, but I interact with and consume a lot of statistics.

As you work, you may start a local server to live preview this HTML book. This preview will update as you edit the book when you save individual .Rmd files. You can start the server in a work session by using the RStudio add-in "Preview book", or from the R console:

Session Info

At the end of chapter, I will be including a `sessionInfo()` chunk to try to make it easier to reproduce the work, as well as to diagnose any problems.

```
sessionInfo()
#> R version 4.3.1 (2023-06-16)
#> Platform: aarch64-apple-darwin20 (64-bit)
#> Running under: macOS Ventura 13.6.1
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> time zone: America/New_York
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets
#> [6] methods   base
#>
```

```
#> other attached packages:
#>  [1] lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0
#>  [4] dplyr_1.1.2     purrr_1.0.1     readr_2.1.4
#>  [7] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3
#> [10] tidyverse_2.0.0
#>
#> loaded via a namespace (and not attached):
#>  [1] gtable_0.3.4      compiler_4.3.1    tidyselect_1.2.0
#>  [4] scales_1.2.1      yaml_2.3.7        fastmap_1.1.1
#>  [7] R6_2.5.1          generics_0.1.3    knitr_1.43
#> [10] bookdown_0.37     munsell_0.5.0     pillar_1.9.0
#> [13] tzdb_0.4.0        rlang_1.1.1       utf8_1.2.3
#> [16] stringi_1.7.12    xfun_0.39         timechange_0.2.0
#> [19] cli_3.6.1         withr_2.5.0       magrittr_2.0.3
#> [22] digest_0.6.33     grid_4.3.1        rstudioapi_0.15.0
#> [25] hms_1.1.3         lifecycle_1.0.3   vctrs_0.6.3
#> [28] evaluate_0.21     glue_1.6.2        fansi_1.0.4
#> [31] colorspace_2.1-0  rmarkdown_2.23    tools_4.3.1
#> [34] pkgconfig_2.0.3   htmltools_0.5.6
```

Chapter 1

Data import and set up

In the original **R Opus**, HGH presented datasets from actual research conducted in her lab, that she worked with throughout. I think this is a great approach, and with her permission we're going to replicate her work, almost step for step, on the same datasets. Here, we'll review how to get them loaded and familiarize ourselves with them.

1.1 Exploring our data

Most of the **R Opus** concerns 4 datasets. Let's import them and inspect them. We're going to use the `tidyverse` set of packages for basic data import and wrangling.

```
library(tidyverse)
library(here) # easy file navigation

descriptive_data <- read_csv(here("data/torriDAFinal.csv"))
consumer_data <- read_csv(here("data/torriconsFinal.csv"))
sorting_data <- read_csv(here("data/sorting_r1.csv"))
missing_data_example <-read_csv(here("data/torrimiss.csv"))
```

We've now created four data tables (called "tibbles" in the `tidyverse` parlance) with our data, which you can see in your `Environment`. Let's look at each of these.

1.1.1 The descriptive data

```
glimpse(descriptive_data)
```

```
## Rows: 336
## Columns: 23
## $ NJ              <dbl> 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1400, ~
## $ ProductName     <chr> "C_MERLOT", "C_SYRAH", "C_ZINFANDEL", "C_REFOSCO", "I_~
## $ NR              <dbl> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, ~
## $ Red_berry       <dbl> 5.1, 5.6, 4.9, 5.0, 3.3, 5.7, 2.9, 3.2, 0.1, 1.6, 4.5,~
## $ Dark_berry      <dbl> 5.8, 1.9, 2.6, 1.9, 7.2, 3.6, 5.1, 6.0, 0.1, 0.7, 2.9,~
## $ Jam             <dbl> 2.1, 3.9, 1.4, 7.8, 0.5, 8.7, 8.7, 4.0, 0.2, 0.0, 0.3,~
## $ Dried_fruit     <dbl> 4.7, 1.2, 5.9, 0.6, 5.8, 1.9, 0.4, 0.7, 2.9, 6.4, 2.4,~
## $ Artificial_frui <dbl> 1.0, 7.9, 0.8, 6.6, 0.7, 7.4, 6.2, 4.1, 0.1, 0.1, 0.1,~
## $ Chocolate       <dbl> 2.9, 1.0, 2.0, 6.4, 2.1, 3.3, 3.4, 3.6, 0.2, 1.0, 0.2,~
## $ Vanilla         <dbl> 5.0, 8.3, 2.7, 5.5, 1.3, 6.9, 8.1, 4.8, 2.0, 0.8, 1.9,~
## $ Oak             <dbl> 5.0, 2.3, 5.6, 3.6, 2.1, 1.5, 1.8, 2.6, 3.0, 5.4, 6.1,~
## $ Burned          <dbl> 1.4, 1.8, 1.9, 3.2, 5.6, 0.2, 0.4, 4.7, 7.5, 5.1, 0.3,~
## $ Leather         <dbl> 2.3, 3.5, 4.3, 0.3, 6.5, 1.5, 4.1, 6.5, 0.7, 0.8, 0.2,~
## $ Earthy          <dbl> 0.6, 1.0, 0.6, 0.2, 4.7, 0.3, 0.5, 1.9, 0.7, 3.0, 1.3,~
## $ Spicy           <dbl> 3.2, 0.7, 1.4, 2.9, 0.7, 3.1, 0.7, 1.4, 0.3, 3.2, 3.1,~
## $ Pepper          <dbl> 5.4, 3.0, 4.1, 0.9, 2.8, 1.6, 3.6, 4.5, 0.1, 2.0, 0.9,~
## $ Grassy          <dbl> 2.1, 0.6, 3.6, 1.8, 3.8, 0.9, 2.3, 0.8, 0.1, 1.3, 0.4,~
## $ Medicinal       <dbl> 0.4, 2.2, 1.7, 0.2, 2.6, 0.5, 0.2, 3.8, 0.1, 2.1, 0.1,~
## $ `Band-aid`      <dbl> 0.4, 0.4, 0.1, 0.2, 5.1, 1.2, 0.2, 6.2, 0.1, 1.1, 0.1,~
## $ Sour            <dbl> 5.0, 9.7, 7.8, 8.3, 7.6, 7.2, 5.9, 6.3, 5.7, 6.4, 5.4,~
## $ Bitter          <dbl> 5.9, 5.2, 3.5, 3.0, 1.9, 9.8, 2.9, 0.2, 0.6, 2.9, 0.1,~
## $ Alcohol         <dbl> 9.0, 7.2, 4.7, 8.9, 2.8, 8.7, 1.6, 7.0, 1.6, 5.4, 4.9,~
## $ Astringent      <dbl> 8.7, 8.3, 5.0, 7.8, 5.9, 8.0, 2.6, 4.2, 5.5, 5.1, 5.9,~
```

The `descriptive_data` file, read from `torriDAFinal.csv`, contains results from a descriptive analysis (DA) of wines, I believe from California and Italy (more on that in a minute). Each row contains the observation of multiple variables (columns) from a single judge (`NJ`) on a single wine (`ProductName`) in a single replicate (`NR`). That means we have 20 measured, quantitative sensory variables (`Red_berry:Astringent`), and the other three variables (judge, wine, rep) are ID or classifying variables. These data are *not* tidy; each row has multiple variables.

We can learn a little more about these data before we move on: let's figure out how many samples, how many judges, and how many reps we have:

```
# There are 14 judges, each of whom completed the same number of observations
descriptive_data %>%
  count(NJ, sort = TRUE)
```

```
## # A tibble: 14 x 2
##       NJ     n
##    <dbl> <int>
##  1  1331    24
##  2  1400    24
##  3  1401    24
##  4  1402    24
##  5  1404    24
##  6  1405    24
##  7  1408    24
##  8  1409    24
##  9  1412    24
## 10  1413    24
## 11  1414    24
## 12  1415    24
## 13  1416    24
## 14  1417    24
```

```
# There are 3 reps (labeled as 7, 8, and 9), balanced
descriptive_data %>%
  count(NR)
```

```
## # A tibble: 3 x 2
##      NR     n
##   <dbl> <int>
## 1     7   112
## 2     8   112
## 3     9   112
```

```
# There are 8 products, 4 from Italy and 4 from California
descriptive_data %>%
  count(ProductName)
```

```
## # A tibble: 8 x 2
##   ProductName     n
##   <chr>       <int>
## 1 C_MERLOT       42
## 2 C_REFOSCO      42
## 3 C_SYRAH        42
## 4 C_ZINFANDEL    42
## 5 I_MERLOT       42
## 6 I_PRIMITIVO    42
## 7 I_REFOSCO      42
## 8 I_SYRAH        42
```

The `%>%` operator is called **the pipe**, and it takes whatever is on its left side and passes it on to the right side. You can learn more about the pipe, but you can also just experiment. When you see it in code, you can read it as "...and then...", so above we'd say something like "take `descriptive_data` **and then** count the number of times each `NJ` value occurs". The other thing to know about piping, that you will see in some code in the **R Opus**, is the use of the period, `.`, as a "pronoun" for whatever is getting passed from the previous line. This lets you tell the pipe where in the line you want the output from the previous/left side to end up:

```
 # Take the descriptive data
descriptive_data %>%
# AND THEN run 1-way ANOVA, with descriptive_data passed to the `data =` argument of a
  aov(Jam ~ ProductName, data = .) %>%
# AND THEN print the readable summary of the results of aov()
  summary()
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## ProductName   7  280.6   40.08   7.882 7.74e-09 ***
## Residuals   328 1667.9    5.08
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Turns out that `Jam` attribute might vary by product. We'll see.

1.1.2 Consumer data

Let's look at the data in `consumer_data`:

```
glimpse(consumer_data)
```

```
## Rows: 106
## Columns: 13
## $ Judge            <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16~
## $ `Wine Frequency` <dbl> 2, 3, 3, 3, 2, 3, 2, 2, 1, 2, 2, 2, 1, 1, 1, 4, 1, 1,~
## $ `IT frequency`   <dbl> 3, 4, 1, 2, 3, 3, 4, 3, 3, 4, 3, 3, 3, 3, 3, 1, 3, 4,~
## $ Gender           <dbl> 2, 2, 1, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2,~
## $ Age              <dbl> 22, 21, 25, 24, 31, 60, 24, 25, 26, 40, 40, 22, 23, 2~
## $ C_MERLOT         <dbl> 5, 8, 8, 4, 4, 4, 8, 5, 7, 7, 7, 8, 6, 4, 4, 2, 3, 7,~
## $ C_SYRAH          <dbl> 4, 5, 6, 6, 6, 4, 6, 4, 5, 1, 8, 8, 4, 2, 4, 1, 8, 6,~
## $ C_ZINFANDEL      <dbl> 3, 8, 6, 3, 6, 4, 7, 6, 6, 4, 6, 7, 6, 8, 6, 7, 9, 7,~
## $ C_REFOSCO        <dbl> 8, 5, 8, 4, 7, 4, 6, 6, 3, 6, 3, 7, 8, 4, 6, 1, 8, 8,~
## $ I_MERLOT         <dbl> 7, 4, 7, 8, 2, 3, 8, 7, 6, 5, 3, 3, 3, 3, 4, 1, 6, 7,~
```

```
## $ I_SYRAH          <dbl> 2, 8, 5, 4, 6, 3, 6, 4, 6, 8, 2, 4, 8, 7, 3, 4, 4, 8,~
## $ I_PRIMITIVO      <dbl> 4, 4, 7, 9, 5, 4, 4, 4, 6, 1, 1, 2, 7, 3, 4, 2, 7, 6,~
## $ I_REFOSCO        <dbl> 6, 6, 7, 2, 7, 1, 7, 3, 3, 2, 4, 3, 7, 5, 4, 4, 6, 7,~
```

Here we have consumer acceptance testing data on the same 8 wines from the DA. The rows look like they represent consumers, identified by the `Judge` column. Let's check to see if they're all unique:

```
consumer_data %>%
# The sort = TRUE argument would put anyone with multiple rows at the top
  count(Judge, sort = TRUE)
```

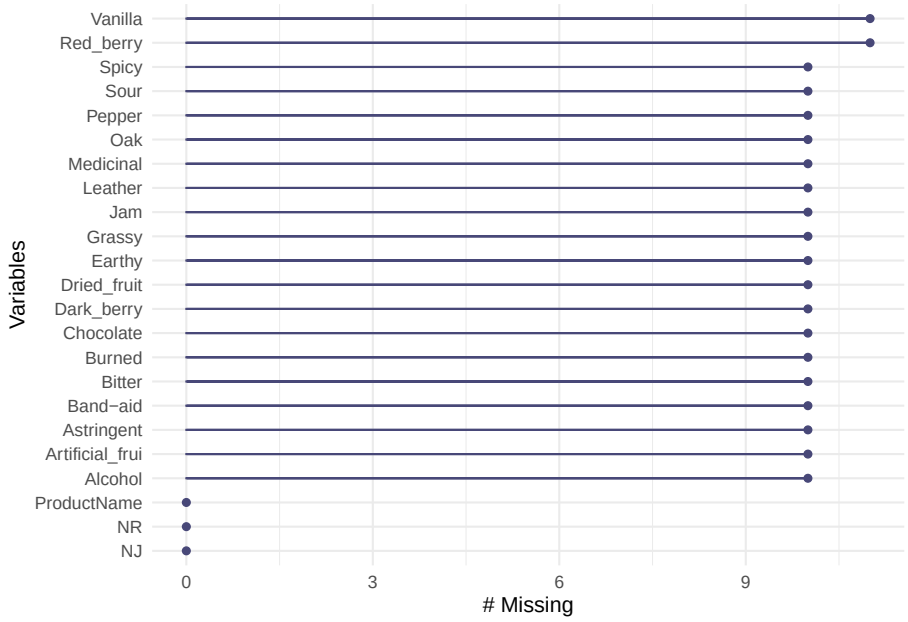
```
## # A tibble: 106 x 2
##    Judge     n
##    <dbl> <int>
##  1     1     1
##  2     2     1
##  3     3     1
##  4     4     1
##  5     5     1
##  6     6     1
##  7     7     1
##  8     8     1
##  9     9     1
## 10    10     1
## # i 96 more rows
```

Then we have some consumption data (`Wine Frequency` and `IT Frequency`) and some demographic data (`Gender` and `Age`), and then 8 columns representing each judge's rating for each wine. Doesn't look like we have any replication or any incomplete data. Again, this is not tidy data–each row contains multiple observations beyond the classifying/identifying data. We'll come back to that.

1.1.3 The other data frames

The other two data frames we're going to pay less attention to for now. The first, `missing_data_example`, is the same data frame as `descriptive_data` but with missing data introduced on purpose for didactic purposes. We can use the `naniar` package to quickly learn a litle bit about the missingness:

```
library(naniar)

naniar::gg_miss_var(x = missing_data_example)
```



We could also use the useful `skimr` package to give us an informative data summary that would tell us about missingness:

```
library(skimr)
```

```
##
## Attaching package: 'skimr'
```

```
## The following object is masked from 'package:naniar':
##
##     n_complete
```

```
skim(missing_data_example) %>%
  # This is purely to allow the skimr::skim() to be rendered in PDF, ignore otherwise
  knitr::kable()
```

skim_type	skim_variable	n_missing	complete_rate	character.min	character.max	character.empty	c
character	ProductName	0	1.0000000	7	11	0	
numeric	NJ	0	1.0000000	NA	NA	NA	
numeric	NR	0	1.0000000	NA	NA	NA	
numeric	Red_berry	11	0.9672619	NA	NA	NA	
numeric	Dark_berry	10	0.9702381	NA	NA	NA	
numeric	Jam	10	0.9702381	NA	NA	NA	
numeric	Dried_fruit	10	0.9702381	NA	NA	NA	
numeric	Artificial_frui	10	0.9702381	NA	NA	NA	
numeric	Chocolate	10	0.9702381	NA	NA	NA	
numeric	Vanilla	11	0.9672619	NA	NA	NA	
numeric	Oak	10	0.9702381	NA	NA	NA	
numeric	Burned	10	0.9702381	NA	NA	NA	
numeric	Leather	10	0.9702381	NA	NA	NA	
numeric	Earthy	10	0.9702381	NA	NA	NA	
numeric	Spicy	10	0.9702381	NA	NA	NA	
numeric	Pepper	10	0.9702381	NA	NA	NA	
numeric	Grassy	10	0.9702381	NA	NA	NA	
numeric	Medicinal	10	0.9702381	NA	NA	NA	
numeric	Band-aid	10	0.9702381	NA	NA	NA	
numeric	Sour	10	0.9702381	NA	NA	NA	
numeric	Bitter	10	0.9702381	NA	NA	NA	
numeric	Alcohol	10	0.9702381	NA	NA	NA	
numeric	Astringent	10	0.9702381	NA	NA	NA	

Because the only difference between this data table and the `descriptive_data` table is the introduced missingness we're not going to pay much attention to it until we get to dealing with missing data.

The `sorting_data` data frame contains results from a sorting task in which 15 subjects (1 per column) sorted the same wines into disjoint groups according to their own criteria. If in a column two wines have the same label (say, `G1`), that subject put them in the same group. We'll get to how to deal with this kind of data later on, so we're not going to go into more depth here.

```
sorting_data
```

```
## # A tibble: 8 x 16
##   wine      `263` `1331` `1400` `1401` `1402` `1404` `1405` `1408` `1409` `1412`
##   <chr>     <chr> <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
## 1 I_REFOSCO G6    G3     G5     G4     G5     G4     G2     G2     G1     G1
## 2 I_MERLOT  G1    G3     G4     G3     G2     G3     G4     G3     G1     G2
## 3 I_SYRAH   G5    G4     G2     G1     G1     G2     G1     G4     G1     G2
## 4 I_PRIMIT~ G2    G1     G3     G1     G4     G5     G3     G4     G3     G2
## 5 C_SYRAH   G3    G2     G4     G4     G5     G6     G1     G3     G2     G4
## 6 C_REFOSCO G4    G3     G1     G2     G5     G5     G2     G1     G2     G3
```

```
## 7 C_MERLOT  G1    G2     G3     G3     G3     G1     G4     G3     G2     G2
## 8 C_ZINFAN~ G1    G5     G3     G3     G2     G1     G1     G3     G1     G2
## # i 5 more variables: `1413` <chr>, `1414` <chr>, `1415` <chr>, `1416` <chr>,
## #   `1417` <chr>
```

1.2 Wrangling/tidying data

Whenever we do data analysis, about 90% of the effort is getting the data into the right shape for the analysis at hand. We're going to be employing many approaches to this task throughout the **R Opus**, but here we'll do a quick preparatory analysis with our two untidy data frames.

The `pivot_longer()` function in `dplyr` is built to take data in the many-observations-per-row format (often called "wide" data) and bring it into a longer, "tidier" representation that is often easier to work with. For the `descriptive_data`, a natural format that would be easier to work with is one in which all the descriptors are pivoted into two columns: one labeling the descriptor by name, and the other giving the rated value. We'd do that as follows:

```
descriptive_data_tidy <-
  descriptive_data %>%
  pivot_longer(cols = -c(NJ, NR, ProductName),
               names_to = "descriptor",
               values_to = "rating")

descriptive_data_tidy
```

```
## # A tibble: 6,720 x 5
##       NJ ProductName    NR descriptor      rating
##    <dbl> <chr>       <dbl> <chr>            <dbl>
##  1  1331 C_MERLOT        7 Red_berry          5.1
##  2  1331 C_MERLOT        7 Dark_berry         5.8
##  3  1331 C_MERLOT        7 Jam                2.1
##  4  1331 C_MERLOT        7 Dried_fruit        4.7
##  5  1331 C_MERLOT        7 Artificial_frui    1
##  6  1331 C_MERLOT        7 Chocolate          2.9
##  7  1331 C_MERLOT        7 Vanilla            5
##  8  1331 C_MERLOT        7 Oak                5
##  9  1331 C_MERLOT        7 Burned             1.4
## 10  1331 C_MERLOT        7 Leather            2.3
## # i 6,710 more rows
```

Now we have kept our 3 ID or classifying variables and created a 4th, `descriptor`, which together uniquely describe each numeric observation, now

called `rating`. In our newly tidied dataset, the first line give the `rating`–the observation–for the judge with code `1331`, in rep `7`, for the wine `C_MERLOT`, for the `Red_berry` descriptor. It's `5.1`.

To motivate this kind of work, let's quickly talk about the "split-apply-combine" workflow this enables. Now that we've separated out our classifying variables, we can do work with them much more easily. For example, let's say we want to generate a means table comparing the wines on the `Jam` variable, which we know from our example above seems to vary per wine. We can now do this very easily:

```
descriptive_data_tidy %>%

# First we will use filter() to get only Jam observations

  filter(descriptor == "Jam") %>%

# Then we will use group_by() to identify our wines as the relevant classifying variable

  group_by(ProductName) %>%

# Finally, we'll use the summarize() function to generate a quick per-wine summary

  summarize(mean_jam_score = mean(rating),
            se_jam_score = sd(rating) / sqrt(n()),
            lower_95 = mean_jam_score - 1.96 * se_jam_score,
            upper_95 = mean_jam_score + 1.96 * se_jam_score)
```

```
## # A tibble: 8 x 5
##   ProductName mean_jam_score se_jam_score lower_95 upper_95
##   <chr>                <dbl>        <dbl>    <dbl>    <dbl>
## 1 C_MERLOT              1.37        0.262    0.857     1.89
## 2 C_REFOSCO             1.03        0.238    0.565     1.50
## 3 C_SYRAH               1.75        0.378    1.00      2.49
## 4 C_ZINFANDEL           1.98        0.394    1.21      2.75
## 5 I_MERLOT              0.843       0.185    0.480     1.21
## 6 I_PRIMITIVO           3.61        0.475    2.68      4.54
## 7 I_REFOSCO             1.54        0.282    0.982     2.09
## 8 I_SYRAH               3.10        0.453    2.21      3.98
```

Our use of all of the intermediate steps–filtering, grouping, summarizing–depends on us having gotten the data into a tidy format. This will sometimes *not* work well–especially when we're looking at pairwise relationships like covariance or correlation–but it will often be a step in wrangling results.

We can do the same thing with our `consumer_data`. In this case, we have 5 ID

variables, and 8 observations, one for each wine. Let's do the same thing with pivot_longer:

```
consumer_data_tidy <-
  consumer_data %>%

# We can use ":" to select a continuous range of columns

  pivot_longer(cols = C_MERLOT:I_REFOSCO,
               names_to = "wine",
               values_to = "rating")

consumer_data_tidy
```

```
## # A tibble: 848 x 7
##    Judge `Wine Frequency` `IT frequency` Gender   Age wine        rating
##    <dbl>            <dbl>          <dbl>  <dbl> <dbl> <chr>        <dbl>
##  1     1                2              3      2    22 C_MERLOT         5
##  2     1                2              3      2    22 C_SYRAH          4
##  3     1                2              3      2    22 C_ZINFANDEL      3
##  4     1                2              3      2    22 C_REFOSCO        8
##  5     1                2              3      2    22 I_MERLOT         7
##  6     1                2              3      2    22 I_SYRAH          2
##  7     1                2              3      2    22 I_PRIMITIVO      4
##  8     1                2              3      2    22 I_REFOSCO        6
##  9     2                3              4      2    21 C_MERLOT         8
## 10     2                3              4      2    21 C_SYRAH          5
## # i 838 more rows
```

Again, we've created a new ID/classifying column called `wine` and a new observation column called `rating`. Now the 6 ID columns uniquely specify an observation. Let's do something fun with this: is there a relationship between `Gender` and `rating`?

```
consumer_data_tidy %>%
  lm(rating ~ Gender, data = .) %>%
  summary()
```

```
##
## Call:
## lm(formula = rating ~ Gender, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -4.3137 -1.3137  0.6863  1.6863  3.7091
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.26809    0.21653  24.330   <2e-16 ***
## Gender       0.02282    0.13852   0.165    0.869
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.015 on 846 degrees of freedom
## Multiple R-squared:  3.207e-05,  Adjusted R-squared:  -0.00115
## F-statistic: 0.02713 on 1 and 846 DF,  p-value: 0.8692
```

Looks like there isn't, which is probably evidence of good sampling–I can't think why there would be such a relationship.

1.3 Wrap up and summary

At this point, we've already done quite a bit of work. We've practiced importing and inspecting data, and gotten a bit of practice with tidying data and applying the split-apply-combine data analysis steps. We're going to be moving on to applying Analysis of Variance (ANOVA) to these data next.

1.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] skimr_2.1.5     naniar_1.0.0    here_1.0.1      lubridate_1.9.2
##  [5] forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1
##  [9] readr_2.1.4     tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.3        generics_0.1.3    stringi_1.7.12    hms_1.1.3
##  [5] digest_0.6.33     magrittr_2.0.3    evaluate_0.21     grid_4.3.1
##  [9] timechange_0.2.0  bookdown_0.37     fastmap_1.1.1     jsonlite_1.8.7
## [13] rprojroot_2.0.3   fansi_1.0.4       scales_1.2.1      cli_3.6.1
## [17] rlang_1.1.1       crayon_1.5.2      bit64_4.0.5       munsell_0.5.0
## [21] base64enc_0.1-3   repr_1.1.6        withr_2.5.0       yaml_2.3.7
## [25] tools_4.3.1       parallel_4.3.1    tzdb_0.4.0        colorspace_2.1-0
## [29] vctrs_0.6.3       R6_2.5.1          lifecycle_1.0.3   bit_4.0.5
## [33] vroom_1.6.3       pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.4
## [37] glue_1.6.2        visdat_0.6.0      highr_0.10        xfun_0.39
## [41] tidyselect_1.2.0  rstudioapi_0.15.0 knitr_1.43        farver_2.1.1
## [45] htmltools_0.5.6   labeling_0.4.3    rmarkdown_2.23    compiler_4.3.1
```

Chapter 2

Analysis of Variance

So far we've imported and inspected our data. In case that got left behind, let's do it again (as well as setting up the basic packages we're going to use).

```
library(tidyverse)
library(here)

descriptive_data <- read_csv(here("data/torriDAFinal.csv"))
consumer_data <- read_csv(here("data/torriconsFinal.csv"))
```

2.1 Univariate analysis with the linear model

In the original **R Opus**, HGH made sure that the classifying variables in the `descriptive_data` were set to the `factor` data type. There is a *looong* history of factors in `R`, and you can get a better crash course than I am going to give in some introductory course like Stat545. Suffice it to say here that the `factor` data type is a labeled integer, and that it ~~is~~ used to be the default way that `R` imports strings. It has some desirable characteristics, but it can often be non-intuitive for those learning `R`. Much of base `R` has now evolved to silently handle `character` data as `factor` when necessary, so unless you're a hardcore `R` user you probably don't need to worry about this too much.

That being said, let's take the opportunity to follow along with HGH's intentions in order to learn some more data wrangling. In this case, HGH says that the `NJ`, `NR`, and `ProductName` variables in `descriptive_data` should be `factor` data type, so let's make it so (this is definitely a good idea for `NJ` and `NR`, which are stored as numerals, as otherwise `R` will treat them as continuous numeric variables).

```
descriptive_data <-
  descriptive_data %>%
  mutate(NJ = as.factor(NJ),
         NR = as.factor(NR),
         ProductName = as.factor(ProductName))

glimpse(descriptive_data)
```

```
## Rows: 336
## Columns: 23
## $ NJ              <fct> 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1400, ~
## $ ProductName     <fct> C_MERLOT, C_SYRAH, C_ZINFANDEL, C_REFOSCO, I_MERLOT, I~
## $ NR              <fct> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, ~
## $ Red_berry       <dbl> 5.1, 5.6, 4.9, 5.0, 3.3, 5.7, 2.9, 3.2, 0.1, 1.6, 4.5,~
## $ Dark_berry      <dbl> 5.8, 1.9, 2.6, 1.9, 7.2, 3.6, 5.1, 6.0, 0.1, 0.7, 2.9,~
## $ Jam             <dbl> 2.1, 3.9, 1.4, 7.8, 0.5, 8.7, 8.7, 4.0, 0.2, 0.0, 0.3,~
## $ Dried_fruit     <dbl> 4.7, 1.2, 5.9, 0.6, 5.8, 1.9, 0.4, 0.7, 2.9, 6.4, 2.4,~
## $ Artificial_frui <dbl> 1.0, 7.9, 0.8, 6.6, 0.7, 7.4, 6.2, 4.1, 0.1, 0.1, 0.1,~
## $ Chocolate       <dbl> 2.9, 1.0, 2.0, 6.4, 2.1, 3.3, 3.4, 3.6, 0.2, 1.0, 0.2,~
## $ Vanilla         <dbl> 5.0, 8.3, 2.7, 5.5, 1.3, 6.9, 8.1, 4.8, 2.0, 0.8, 1.9,~
## $ Oak             <dbl> 5.0, 2.3, 5.6, 3.6, 2.1, 1.5, 1.8, 2.6, 3.0, 5.4, 6.1,~
## $ Burned          <dbl> 1.4, 1.8, 1.9, 3.2, 5.6, 0.2, 0.4, 4.7, 7.5, 5.1, 0.3,~
## $ Leather         <dbl> 2.3, 3.5, 4.3, 0.3, 6.5, 1.5, 4.1, 6.5, 0.7, 0.8, 0.2,~
## $ Earthy          <dbl> 0.6, 1.0, 0.6, 0.2, 4.7, 0.3, 0.5, 1.9, 0.7, 3.0, 1.3,~
## $ Spicy           <dbl> 3.2, 0.7, 1.4, 2.9, 0.7, 3.1, 0.7, 1.4, 0.3, 3.2, 3.1,~
## $ Pepper          <dbl> 5.4, 3.0, 4.1, 0.9, 2.8, 1.6, 3.6, 4.5, 0.1, 2.0, 0.9,~
## $ Grassy          <dbl> 2.1, 0.6, 3.6, 1.8, 3.8, 0.9, 2.3, 0.8, 0.1, 1.3, 0.4,~
## $ Medicinal       <dbl> 0.4, 2.2, 1.7, 0.2, 2.6, 0.5, 0.2, 3.8, 0.1, 2.1, 0.1,~
## $ `Band-aid`      <dbl> 0.4, 0.4, 0.1, 0.2, 5.1, 1.2, 0.2, 6.2, 0.1, 1.1, 0.1,~
## $ Sour            <dbl> 5.0, 9.7, 7.8, 8.3, 7.6, 7.2, 5.9, 6.3, 5.7, 6.4, 5.4,~
## $ Bitter          <dbl> 5.9, 5.2, 3.5, 3.0, 1.9, 9.8, 2.9, 0.2, 0.6, 2.9, 0.1,~
## $ Alcohol         <dbl> 9.0, 7.2, 4.7, 8.9, 2.8, 8.7, 1.6, 7.0, 1.6, 5.4, 4.9,~
## $ Astringent      <dbl> 8.7, 8.3, 5.0, 7.8, 5.9, 8.0, 2.6, 4.2, 5.5, 5.1, 5.9,~
```

Great. We got it. Now, when we do something like regression (with `lm()`) or ANOVA (with `aov()`) using those variables as predictors, `R` will treat them as discrete, unordered levels rather than the numbers that they are encoded as.

The other trick that HGH does in her original **R Opus** is to run linear models on a matrix of outcomes. We're going to take a slightly different approach using the "split-apply-combine" approach that the `nest()` function gives us access to in `dplyr`.

```
# First, we will retidy our data
descriptive_data_tidy <-
  descriptive_data %>%
  pivot_longer(cols = -c(NJ, NR, ProductName),
               names_to = "descriptor",
               values_to = "rating")

nested_AOV_res <-
  descriptive_data_tidy %>%
  nest(data = -descriptor) %>%
  mutate(anova_res = map(.x = data,
                         .f = ~aov(rating ~ (NJ + ProductName + NR)^2, data = .x)))
```

This functional programming is rather advanced, so let me walk through what we did:

1. We used `nest()` to make a bunch of sub-tables, each of which contained the data only for a single descriptor rating. We have a table for `Red_berry`, a table for `Jam`, and so on.
2. We used `mutate()` to make a new column to store our ANOVA results in, which we created through the use of `map()`, as discussed below:
3. We used `map()` to take each of the nested subtables, now called `data`, and pass it to the `aov()` function. This works because each ANOVA model has the exact same formula: `rating ~ (NJ + ProductName + NR)^2`. In `R` formula syntax, this means we are modeling the variability in `rating` as a the outcome of judge (`NJ`), wine (`ProductName`), and rep (`NR`) *for each individual descriptor*. The "one-sided `~`" syntax we use in the `.f =` line defines this function, which is then applied to each of the nested subtables we built in the previous step. By tidying the data, we are able to do this in this succinct but somewhat intimidating syntax. "Split, apply, combine."

This whole workflow is based on Hadley Wickham's work, and you can find more details here. The outcome is a "tibble of tibbles", taking advantage of the fact that `R` lets us store arbitrary data into the cells of a `tibble` object.

```
nested_AOV_res
```

```
## # A tibble: 20 x 3
##    descriptor      data               anova_res
##    <chr>           <list>             <list>
##  1 Red_berry       <tibble [336 x 4]> <aov>
##  2 Dark_berry      <tibble [336 x 4]> <aov>
##  3 Jam             <tibble [336 x 4]> <aov>
##  4 Dried_fruit     <tibble [336 x 4]> <aov>
```

```
##  5 Artificial_frui <tibble [336 x 4]> <aov>
##  6 Chocolate       <tibble [336 x 4]> <aov>
##  7 Vanilla         <tibble [336 x 4]> <aov>
##  8 Oak             <tibble [336 x 4]> <aov>
##  9 Burned          <tibble [336 x 4]> <aov>
## 10 Leather         <tibble [336 x 4]> <aov>
## 11 Earthy          <tibble [336 x 4]> <aov>
## 12 Spicy           <tibble [336 x 4]> <aov>
## 13 Pepper          <tibble [336 x 4]> <aov>
## 14 Grassy          <tibble [336 x 4]> <aov>
## 15 Medicinal       <tibble [336 x 4]> <aov>
## 16 Band-aid        <tibble [336 x 4]> <aov>
## 17 Sour            <tibble [336 x 4]> <aov>
## 18 Bitter          <tibble [336 x 4]> <aov>
## 19 Alcohol         <tibble [336 x 4]> <aov>
## 20 Astringent      <tibble [336 x 4]> <aov>
```

If we dig into this structure, we can see that we're creating, in parallel, a bunch of model fits:

```
# The first fitted model
nested_AOV_res$anova_res[[1]]
```

```
## Call:
##    aov(formula = rating ~ (NJ + ProductName + NR)^2, data = .x)
##
## Terms:
##                       NJ ProductName       NR NJ:ProductName    NJ:NR
## Sum of Squares  597.8637     73.1578   2.8577       659.0318  97.1707
## Deg. of Freedom       13           7        2             91       26
##                 ProductName:NR Residuals
## Sum of Squares         52.2499  652.8217
## Deg. of Freedom             14       182
##
## Residual standard error: 1.89392
## Estimated effects may be unbalanced
```

We can continue to use `map()` to get summaries from these models. We use the broom::tidy() function in order to get the same results as the typical `summary()` function, but in a manipulable `tibble`/`data.frame` object instead of in a `summary` object.

```
da_aov_res <-
  nested_AOV_res %>%
  mutate(aov_summaries = map(.x = anova_res,
```

```
                              .f = ~broom::tidy(.x))) %>%

  # unnest() does the reverse of nest(): it takes a nested subtable and expands it
  # back into the data frame, duplicating all of the classifying variables
  # necessary to uniquely identify it.

  unnest(aov_summaries) %>%
  select(-data, -anova_res)

da_aov_res
```

```
## # A tibble: 140 x 7
##    descriptor term              df  sumsq meansq statistic   p.value
##    <chr>      <chr>          <dbl>  <dbl>  <dbl>     <dbl>     <dbl>
##  1 Red_berry  NJ                13 598.    46.0     12.8    9.35e-20
##  2 Red_berry  ProductName        7  73.2   10.5      2.91   6.52e- 3
##  3 Red_berry  NR                 2   2.86   1.43     0.398  6.72e- 1
##  4 Red_berry  NJ:ProductName    91 659.     7.24     2.02   3.18e- 5
##  5 Red_berry  NJ:NR             26  97.2    3.74     1.04   4.16e- 1
##  6 Red_berry  ProductName:NR    14  52.2    3.73     1.04   4.15e- 1
##  7 Red_berry  Residuals        182 653.     3.59    NA     NA
##  8 Dark_berry NJ                13 813.    62.5     15.5    2.50e-23
##  9 Dark_berry ProductName        7 126.    18.0      4.47   1.28e- 4
## 10 Dark_berry NR                 2  18.9    9.47     2.35   9.81e- 2
## # i 130 more rows
```

We now have a table of all of the model results for running a 3-way ANOVA with two-way interactions on each of the individual sensory descriptors. We can use this format to easily look for results that are significant. For example, let's filter down to only rows where the model `term` represents variation from the wine (e.g., `ProductName`), and there is what is typically described as a signficant p-value (i.e., $p < 0.05$).

```
naive_significance <-
  da_aov_res %>%
  filter(term == "ProductName",
         p.value < 0.05) %>%
  select(descriptor, p.value)

naive_significance
```

```
## # A tibble: 17 x 2
##    descriptor       p.value
##    <chr>              <dbl>
```

```
##  1 Red_berry       6.52e- 3
##  2 Dark_berry      1.28e- 4
##  3 Jam             7.02e-16
##  4 Artificial_frui 3.23e-11
##  5 Chocolate       4.88e- 6
##  6 Vanilla         1.58e- 6
##  7 Oak             1.58e- 4
##  8 Burned          2.65e-26
##  9 Leather         6.56e- 9
## 10 Earthy          3.77e- 4
## 11 Pepper          7.55e- 3
## 12 Grassy          5.14e- 3
## 13 Medicinal       2.67e-11
## 14 Band-aid        2.77e-11
## 15 Sour            6.02e- 3
## 16 Bitter          8.07e- 3
## 17 Alcohol         1.91e- 4
```

We see that there are 17 descriptors that fit this criteria! For these descriptors, ratings vary significantly for different wines.

2.2 Pseudo-mixed ANOVA

In the original **R Opus**, HGH promotes the use of "pseudo-mixed" ANOVA in order to account for added variability from lack of judge consistency. Specifically, we will see that several of our models show *significant interaction terms* between wines and judges or wines and replicates. Let's find these products:

```
da_aov_res %>%
  filter(term %in% c("NJ:ProductName", "ProductName:NR"),
         p.value < 0.05)
```

```
## # A tibble: 20 x 7
##    descriptor      term              df sumsq meansq statistic  p.value
##    <chr>           <chr>          <dbl> <dbl>  <dbl>     <dbl>    <dbl>
##  1 Red_berry       NJ:ProductName    91 659.    7.24      2.02 3.18e- 5
##  2 Dark_berry      NJ:ProductName    91 704.    7.73      1.92 1.04e- 4
##  3 Jam             NJ:ProductName    91 480.    5.27      2.04 2.38e- 5
##  4 Jam             ProductName:NR    14  64.0   4.57      1.77 4.60e- 2
##  5 Dried_fruit     NJ:ProductName    91 346.    3.80      1.54 7.17e- 3
##  6 Artificial_frui NJ:ProductName    91 369.    4.05      2.14 7.71e- 6
##  7 Chocolate       NJ:ProductName    91 207.    2.28      2.00 4.12e- 5
##  8 Vanilla         NJ:ProductName    91 334.    3.67      2.09 1.33e- 5
```

```
##  9 Oak            NJ:ProductName    91 273.    3.00      1.67 1.74e- 3
## 10 Burned         NJ:ProductName    91 543.    5.97      4.18 1.21e-16
## 11 Burned         ProductName:NR    14  39.8   2.84      1.99 2.07e- 2
## 12 Leather        NJ:ProductName    91 338.    3.71      2.50 8.27e- 8
## 13 Earthy         NJ:ProductName    91 169.    1.86      1.76 6.46e- 4
## 14 Spicy          NJ:ProductName    91 232.    2.55      2.18 4.33e- 6
## 15 Pepper         NJ:ProductName    91 267.    2.94      1.64 2.46e- 3
## 16 Grassy         NJ:ProductName    91 198.    2.18      2.11 1.00e- 5
## 17 Medicinal      NJ:ProductName    91 366.    4.03      2.07 1.63e- 5
## 18 Medicinal      ProductName:NR    14  50.3   3.60      1.85 3.42e- 2
## 19 Band-aid       NJ:ProductName    91 395.    4.34      2.74 4.04e- 9
## 20 Sour           NJ:ProductName    91 390.    4.29      1.61 3.43e- 3
```

For these descriptors, the amount of variance explained by the interaction between the judge and the wine (`NJ:ProductName`) or the wine and the rep (`ProductName:Rep`) is significantly larger than the residual variation. But these interactions themselves are an undesirable source of variation: different judges are using these descriptors differently on the same wines, or descriptors are being used differently on the same wines in different reps!

The **pseudo-mixed model** is a conservative approach to accounting for this extra, undesirable error. Without getting too into the weeds on the linear model, the basic intution is that, since these interactions, which represents undesirable variability, are significantly larger than the error term for each of these 5 descriptors, we will substitute the interaction in our normal linear model for the error term when calculating and evaluating our F-statistic, which as you may recall is the ratio of variance from a known source (in this case our wines, `ProductName`) and the error variance.

How are we going to do this? With some good, old-fashioned functional programming:

```
get_pseudomixed_results <- function(aov_res){

  # The relevant ANOVA results for NJ:ProductName are in the 4th row
  product_by_judge <- aov_res[4, ]
  # The relevant ANOVA results for ProductName:NR are in the 6th row
  product_by_rep <- aov_res[6, ]
  # The relevant main effect for ProductName
  product_effect <- aov_res[2, ]

  # If neither of the interactions is significant,
  if(product_by_judge$p.value > 0.05 & product_by_rep$p.value > 0.05){
    return(tibble(sig_effect = product_effect$term,
                  f_stat = product_effect$statistic,
                  df_error = aov_res$df[7],
```

```
                  p_val = product_effect$p.value))
  }
  if(product_by_judge$p.value < product_by_rep$p.value){
    return(tibble(sig_effect = product_by_judge$term,
                  f_stat = product_effect$meansq / product_by_judge$meansq,
                  df_error = product_by_judge$df,
                  p_val = 1 - pf(q = f_stat, df1 = product_effect$df, df2 = df_error)))
  }else{
    return(tibble(sig_effect = product_by_rep$term,
                  f_stat = product_effect$meansq / product_by_rep$meansq,
                  df_error = product_by_rep$df,
                  p_val = 1- pf(q = f_stat, df1 = product_effect$df, df2 = df_error)))
  }
}

da_aov_res %>%
  filter(descriptor == "Red_berry") %>%
  get_pseudomixed_results()
```

```
## # A tibble: 1 x 4
##   sig_effect     f_stat df_error p_val
##   <chr>           <dbl>    <dbl> <dbl>
## 1 NJ:ProductName   1.44       91 0.198
```

This function will take a `tibble` of 3-way ANOVA results from our analysis, look at the relevant interactions, and, if they are large enough, run a pseudomixed analysis. If both of the interactions are significant it will test the largest (most significant) interaction as the one to examine. It will then return the results of that modified, pseudomixed ANOVA in a nice table that tells us the relevant details: what effect was tested, the F-statistic and degrees of freedom for the denominator (the numerator degrees of freedom will remain the same), and the new p-value.

In the case of `Red_berry` above, the `NJ:ProductName` interaction is the most significant, and results in a non-significant pseudomixed test.

NB: Because I used numeric indexing (e.g., `aov_res[6, ]`) to write this function "quick and dirty", it would need to be modified if I had a different linear model defined for my ANOVA (the model specified as `rating ~ (NJ + ProductName + NR)^2` in my `aov()` call above), or even if I reordered the terms in *that* model.

We use the same split-apply-combine workflow with `nest()` and `map()` to first make subtables for each of our descriptors' original ANOVA results, and then run our new function on them.

```
da_aov_res %>%
  nest(data = -descriptor) %>%
  mutate(pseudomixed_res = map(data, ~get_pseudomixed_results(.x))) %>%
  unnest(pseudomixed_res) %>%
  filter(p_val < 0.05)
```

```
## # A tibble: 13 x 6
##    descriptor      data              sig_effect     f_stat df_error       p_val
##    <chr>           <list>            <chr>           <dbl>    <dbl>       <dbl>
##  1 Dark_berry      <tibble [7 x 6]> NJ:ProductName   2.33        91 0.0311
##  2 Jam             <tibble [7 x 6]> NJ:ProductName   7.60        91 0.000000336
##  3 Artificial_frui <tibble [7 x 6]> NJ:ProductName   4.99        91 0.0000819
##  4 Chocolate       <tibble [7 x 6]> NJ:ProductName   2.88        91 0.00912
##  5 Vanilla         <tibble [7 x 6]> NJ:ProductName   2.97        91 0.00753
##  6 Oak             <tibble [7 x 6]> NJ:ProductName   2.62        91 0.0163
##  7 Burned          <tibble [7 x 6]> NJ:ProductName   6.83        91 0.00000162
##  8 Leather         <tibble [7 x 6]> NJ:ProductName   3.37        91 0.00306
##  9 Earthy          <tibble [7 x 6]> NJ:ProductName   2.30        91 0.0335
## 10 Medicinal       <tibble [7 x 6]> NJ:ProductName   5.18        91 0.0000544
## 11 Band-aid        <tibble [7 x 6]> NJ:ProductName   3.91        91 0.000899
## 12 Bitter          <tibble [7 x 6]> ProductName      2.83       182 0.00807
## 13 Alcohol         <tibble [7 x 6]> ProductName      4.32       182 0.000191
```

From these results, we see that almost all of our products have significant interactions, and that we've lost a few significant descriptors: we're down to 13/20 being significant, from 17/20 from our naive evaluation. Furthermore, when we had significant interactions, they were all wine-by-judge interactions: believe it or not, this is good news. As noted by Brockhoff et al. [2015], even trained judges are almost never aligned in scale use, so the pseudomixed model is good insurance against this variability. A larger wine-by-rep interaction would be more concerning: it would indicate that the product may have changed in a particular presentation (perhaps it was exposed to oxygen?), and this would warrant a reconsideration of the results.

2.3 Mean comparison (post-hoc testing)

In the original **R Opus**, HGH demonstrates how to calculate Fisher's Least Significant Difference (Fisher's LSD), but I discourage the use of this test, even though it is arguable that the significant ANOVA result means that there is some protection against familywise error Rencher [2002] The Tukey's Honestly Significant Difference (HSD) is probably preferable, and as a reviewer I push authors to use this test.

We're going to go back to our `nested_AOV_res` object because we don't need the sum-of-squares/ANOVA table output, we need the actual model fit results. HGH uses the `agricolae` package's `HSD.test()` function, and we're going to wrap that up in a convenience function to let us easily capture the results and drop them into a `tibble` for easy access and plotting.

```
library(agricolae)

tidy_hsd <- function(anova_res, treatment = "ProductName"){

  hsd_res <- HSD.test(anova_res, trt = treatment)

  return(
    hsd_res$means %>%
      as_tibble(rownames = "wine") %>%
      left_join(hsd_res$groups %>%
                  as_tibble(rownames = "wine")) %>%
      arrange(groups)
  )

}

# We can test this on the Red_berry descriptor
nested_AOV_res$anova_res[[1]] %>%
  tidy_hsd()
```

```
## Joining with `by = join_by(wine, rating)`
```

```
## # A tibble: 8 x 11
##   wine        rating   std     r    se   Min   Max   Q25   Q50   Q75 groups
##   <chr>        <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
## 1 I_PRIMITIVO   3.85  2.96    42 0.292     0   9.2 1.42   3.1   6.2  a
## 2 C_ZINFANDEL   3.08  2.37    42 0.292     0   8.2 0.725  2.85  4.9  ab
## 3 I_MERLOT      2.79  2.55    42 0.292     0   8   0.5    2.6   4.92 ab
## 4 I_SYRAH       3.17  2.59    42 0.292     0   8.6 1.25   2.45  5.5  ab
## 5 C_MERLOT      2.46  2.39    42 0.292     0   8   0.5    1.6   4.3  b
## 6 C_REFOSCO     2.47  2.30    42 0.292     0   7   0.325  1.8   4.35 b
## 7 C_SYRAH       2.46  2.48    42 0.292     0   8.6 0.5    1.6   3.35 b
## 8 I_REFOSCO     2.48  2.36    42 0.292     0   7.7 0.425  2.1   3.58 b
```

NB, again: Once again I have written this function to be quick and dirty–I have hardcoded some of the information about the structure of our analysis into the function, so if we change how our `aov()` is fit or our column names are defined, this will no longer work.

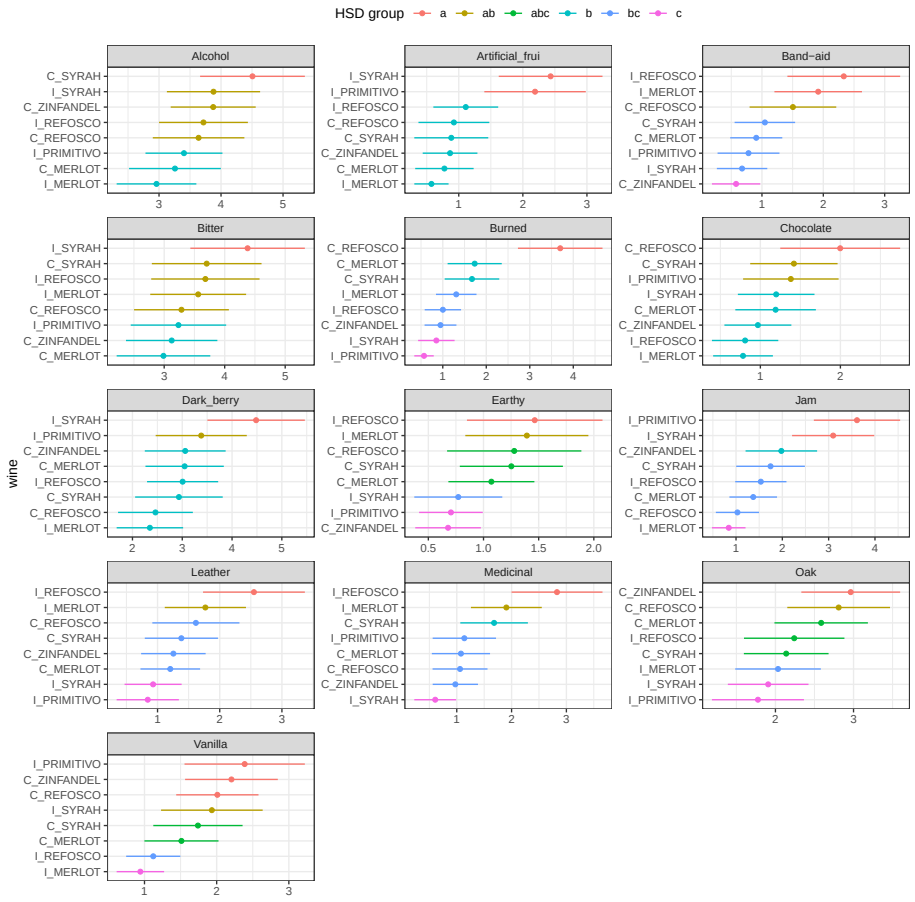
This function returns a nice table of our wines arranged by their group membership in an HSD test. We can use this to get mean separations for all of our significant attributes with a bit of data wrangling.

```
# First, we'll get a list of significant descriptors from pseudomixed ANOVA
significant_descriptors <-
  da_aov_res %>%
  nest(data = -descriptor) %>%
  mutate(pseudomixed_res = map(data, ~get_pseudomixed_results(.x))) %>%
  unnest(pseudomixed_res) %>%
  filter(p_val < 0.05) %>%
  pull(descriptor)

# Then we'll take our original models and only run HSD on the significant ones

hsd_tbl <-
  nested_AOV_res %>%
  filter(descriptor %in% significant_descriptors) %>%
  mutate(hsd_res = map(anova_res, ~tidy_hsd(.x))) %>%
  select(-data, -anova_res) %>%
  unnest(hsd_res)

hsd_tbl
```

```
## # A tibble: 104 x 12
##    descriptor wine        rating   std     r    se   Min   Max   Q25   Q50   Q75
##    <chr>      <chr>        <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Dark_berry I_SYRAH       4.48  3.24    42 0.310     0   9.9 1.58   4.1   7.82
##  2 Dark_berry I_PRIMITIVO   3.38  3.03    42 0.310     0   9.6 0.55   2.8   5.38
##  3 Dark_berry C_MERLOT      3.05  2.60    42 0.310     0   8.3 0.65   2.3   5.08
##  4 Dark_berry C_REFOSCO     2.46  2.48    42 0.310     0   8.6 0.4    1.7   4.22
##  5 Dark_berry C_SYRAH       2.93  2.91    42 0.310     0   8.9 0.7    2.05  4.15
##  6 Dark_berry C_ZINFANDEL   3.06  2.68    42 0.310     0   9.2 0.75   2.6   4.7
##  7 Dark_berry I_MERLOT      2.35  2.21    42 0.310     0   7.2 0.325  1.65  4.12
##  8 Dark_berry I_REFOSCO     3.01  2.36    42 0.310     0   8.5 1.1    2.4   5.02
##  9 Jam        I_PRIMITIVO   3.61  3.08    42 0.248     0   9.8 0.95   2.5   6.35
## 10 Jam        I_SYRAH       3.10  2.93    42 0.248     0   9.5 0.475  1.9   5.15
## # i 94 more rows
## # i 1 more variable: groups <chr>
```

Again, using the same split-apply-combine workflow (but we started with an already nested table), we get a set of results suitable for reporting in a paper or plotting. Let's take a look at how to make a basic visualization of these results using `ggplot2`.

```
library(tidytext)

hsd_tbl %>%
  # Here we define a standard normal confidence interval
  mutate(interval = 1.96 * (std / sqrt(r))) %>%
  # Here we use the tidytext::reorder_within() function to order samples
  # separately across facets, getting nice visualizations
  mutate(wine = reorder_within(wine, by = rating, within = descriptor)) %>%
  ggplot(aes(x = rating, y = wine, color = groups)) +
  geom_point() +
  geom_segment(aes(x = rating - interval,
                   xend = rating + interval,
                   y = wine, yend = wine,
                   color = groups),
               inherit.aes = FALSE) +
  scale_y_reordered() +
  facet_wrap(~descriptor, ncol = 3, scales = "free") +
  theme_bw() +
  theme(legend.position = "top") +
  labs(x = NULL) +
  guides(color = guide_legend(title = "HSD group", nrow = 1))
```



2.4 BONUS: Bayesian approaches to ANOVA

As I have worked longer in our field and with applied statistics, I've gotten less satisfied with "Null Hypothesis Statistics Testing" (NHST), and more interested in a general Bayesian framework. I'm not nearly expert enough to give a rigorous definition of Bayesian statistics, but the gist is that Bayesian statistics combines our prior knowledge (from theory, previous experience, observation) with data from an experiment to give us estimates about our certainty around parameters of interest. Essentially, instead of rejecting a (usually implausible) H_0 as a consequence of finding data incompatible with that hypothesis, Bayesian statistics evaluates what range of *parameters* are compatible with observed data. This, to me, seems more of a satisfying use of statistical analysis.

Put less generally, for our observed data from subjects rating wines' sensory attributes, we would be able to use a Bayesian approach to not just determine

whether the observed means are "significantly different" (at some given level of α, usually $\alpha = 0.05$), but to give *plausible estimates of the underlying descriptor means for each wine*. This seems worthwhile to me.

So if Bayesian statistics are so great, why isn't everyone using them? There are two reasons (in my understanding):

1. Many statisticians, particularly those influenced by Fisher, Neyman, and Pearson (generally called "Frequentists") reject the requirement in Bayesian modeling for providing an estimate of *prior* knowledge, since it is based on opinion. I think this is silly, because it pretends that our experimental approach (science, writ broadly) doesn't integrate prior, subjective knowledge with rigorous data collection. For more on this (silly) objection, consider the excellently titled *The Theory That Would Not Die*.
2. Application of Bayes theorem requires numerical integration of complicated integrands which, without modern computational methods, was essentially impossible. Thus, even enthusiastic Bayesians couldn't actually apply the approach to many practical problems. This barrier has been progressively broken down, as computational power has increased and better, more user-friendly tools have been developed.

Following point 2, I will be using a combination of *Stan* (a standalone tool for Bayesian estimation) and the `R` packages associated with Stan, particularly `brms` ("brooms") and `tidybayes`, to work through this section. Their installation requires extra steps that can be found at the Stan website.

There are many, many, many excellent introductions to Bayesian statistics available, so I am not going to embarrass myself by trying to stumble through an explanation. For my own benefit, instead, I am going to try to work through an application of simple Bayesian methods to the multi-way ANOVAs we ran above. This is based on material from A. Solomon Kurz's [2023] truly excellent walkthrough to modern `R`-based approaches to Bayesian modeling. Let's see how it goes!

First, we load the extra required packages.

```
library(brms)
library(tidybayes)
```

Then we're going to use the approach to setting up ANOVA-like models in Stan that A. Solomon Kurz details in his excellent online companion to Kruschke's *Doing Bayesian Data Analysis* Kruschke [2014].

We are first going to apply this approach to a single outcome: `Bitter`. We know that above we found a significant difference between Bitter ratings for the various wines. Let's let a Bayesian approach explore that result for us.

```
# Let's make a less unwieldy tibble to work with

bayes_data <-
  descriptive_data_tidy %>%
  filter(descriptor == "Bitter")
bayes_data
```

```
## # A tibble: 336 x 5
##    NJ    ProductName NR    descriptor rating
##    <fct> <fct>       <fct> <chr>       <dbl>
##  1 1331  C_MERLOT    7     Bitter        5.9
##  2 1331  C_SYRAH     7     Bitter        5.2
##  3 1331  C_ZINFANDEL 7     Bitter        3.5
##  4 1331  C_REFOSCO   7     Bitter        3
##  5 1331  I_MERLOT    7     Bitter        1.9
##  6 1331  I_SYRAH     7     Bitter        9.8
##  7 1331  I_PRIMITIVO 7     Bitter        2.9
##  8 1331  I_REFOSCO   7     Bitter        0.2
##  9 1400  C_MERLOT    7     Bitter        0.6
## 10 1400  C_SYRAH     7     Bitter        2.9
## # i 326 more rows
```

```
# and we'll store some information about our data so that we can use it in our
# brm() call later:

mean_y <- mean(bayes_data$rating)
sd_y <- sd(bayes_data$rating)
```

Finally, we're going to take a function from Kurz that gives parameters for a gamma distribution with a specific mode and standard deviation; to learn about why we're doing this consult Kurz or Kruschke.

```
gamma_a_b_from_omega_sigma <- function(mode, sd) {
  if (mode <= 0) stop("mode must be > 0")
  if (sd   <= 0) stop("sd must be > 0")
  rate <- (mode + sqrt(mode^2 + 4 * sd^2)) / (2 * sd^2)
  shape <- 1 + mode * rate
  return(list(shape = shape, rate = rate))
}

shape_rate <- gamma_a_b_from_omega_sigma(mode = sd_y / 2, sd = sd_y * 2)
```

We store all of this information into a `stanvar` object, which lets us send this information to Stan through `brms`.

```
stanvars <-
  stanvar(mean_y, name = "mean_y") +
  stanvar(sd_y, name = "sd_y") +
  stanvar(shape_rate$shape, name = "alpha") +
  stanvar(shape_rate$rate, name = "beta")

# If you're curious, stanvar() creates lists of named variables that can be
# passed off to Stan for processing; we could calculate these directly in our
# call to brm() but it would be messier looking and potentially more fragile
# (e.g., if we change our model and forget to change the calculation, that would
# be bad).

str(stanvars)
```

```
## List of 4
##  $ mean_y:List of 6
##   ..$ name    : chr "mean_y"
##   ..$ sdata   : num 3.49
##   ..$ scode   : chr "real mean_y;"
##   ..$ block   : chr "data"
##   ..$ position: chr "start"
##   ..$ pll_args: chr "data real mean_y"
##  $ sd_y  :List of 6
##   ..$ name    : chr "sd_y"
##   ..$ sdata   : num 2.76
##   ..$ scode   : chr "real sd_y;"
##   ..$ block   : chr "data"
##   ..$ position: chr "start"
##   ..$ pll_args: chr "data real sd_y"
##  $ alpha :List of 6
##   ..$ name    : chr "alpha"
##   ..$ sdata   : num 1.28
##   ..$ scode   : chr "real alpha;"
##   ..$ block   : chr "data"
##   ..$ position: chr "start"
##   ..$ pll_args: chr "data real alpha"
##  $ beta  :List of 6
##   ..$ name    : chr "beta"
##   ..$ sdata   : num 0.205
##   ..$ scode   : chr "real beta;"
##   ..$ block   : chr "data"
##   ..$ position: chr "start"
##   ..$ pll_args: chr "data real beta"
##  - attr(*, "class")= chr "stanvars"
```

OK! As per usual, my attempt to explain (to myself, mostly) what I am doing is much longer than I intended! Let's get to modeling.

We are going to develop a model that is equivalent to the 3-way ANOVA we proposed for each variable: we want to allow 2-way interactions between all main effects, as well. It will be slightly uglier because we're going to be using syntax that is common for non-linear or repeated-effects modeling in `R`. To save typing, I am going to omit the interactions between `NR` and the other factors, as (happily) they were never large, and are not of theoretical interest.

```
b_bitter <-
  brm(data = bayes_data,
      family = gaussian,
      formula = rating ~ 1 + (1 | NJ) + (1 | NR) + (1 | ProductName) + (1 | NJ:ProductName),
      prior = c(prior(normal(mean_y, sd_y * 5), class = Intercept),
                prior(gamma(alpha, beta), class = sd),
                prior(cauchy(0, sd_y), class = sigma)),
      stanvars = stanvars, seed = 2, chains = 4, cores = 4, iter = 4000,
      warmup = 2000, control = list(adapt_delta = 0.999, max_treedepth = 13),
      file = here("fits/fit_02_bitter"))
```

This resulted in at least one diagnostic warning that, after inspection, seems trivial enough to ignore for now (only a few divergent transitions with good $\hat{R}$ and ESS values), as can be seen in the summary and diagnostic plots below:

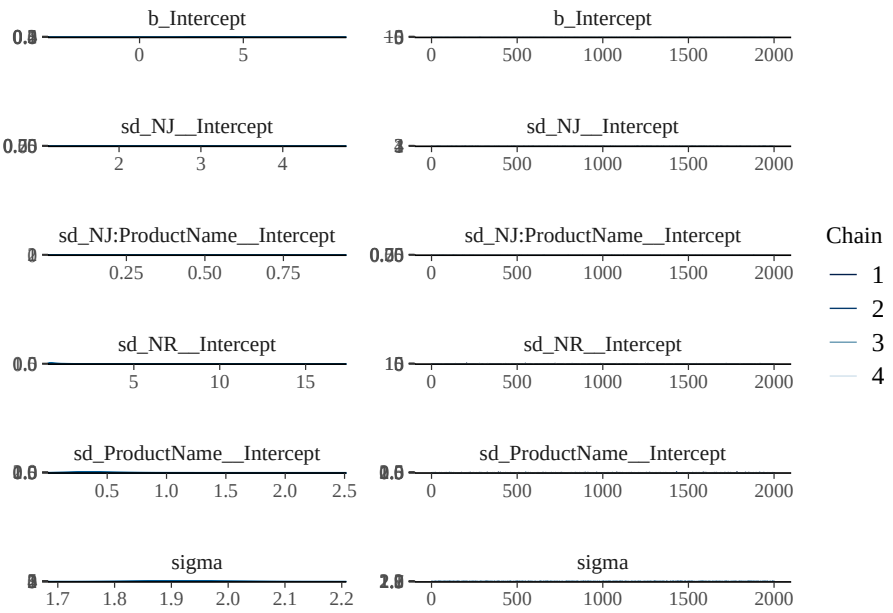
```
b_bitter
```

```
## Warning: There were 3 divergent transitions after warmup. Increasing
## adapt_delta above 0.999 may help. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rating ~ 1 + (1 | NJ) + (1 | NR) + (1 | ProductName) + (1 | NJ:ProductName)
##    Data: bayes_data (Number of observations: 336)
##   Draws: 4 chains, each with iter = 4000; warmup = 2000; thin = 1;
##          total post-warmup draws = 8000
##
## Group-Level Effects:
## ~NJ (Number of levels: 14)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     2.21      0.50     1.49     3.42 1.00     1576     2794
##
## ~NJ:ProductName (Number of levels: 112)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
```

```
## sd(Intercept)     0.22      0.14     0.02     0.55 1.00     3609     3932
##
## ~NR (Number of levels: 3)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.74      1.15     0.03     4.04 1.00     2065     3155
##
## ~ProductName (Number of levels: 8)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.42      0.23     0.07     0.97 1.00     2221     2446
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     3.50      0.95     1.79     5.45 1.00     1495     2367
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.92      0.08     1.78     2.09 1.00     8538     5282
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
plot(b_bitter, N = 6, widths = c(2, 3))
```



We can examine the actual "draws" from the joint posterior using the

`as_draws_df()` function and doing some wrangling.

```
draws <-
  b_bitter %>%
  as_draws_df()

draws
```

```
## # A draws_df: 2000 iterations, 4 chains, and 145 variables
##    b_Intercept sd_NJ__Intercept sd_NJ:ProductName__Intercept sd_NR__Intercept
## 1          3.6              2.5                         0.526           0.0733
## 2          2.3              2.7                         0.173           0.0247
## 3          3.7              1.5                         0.041           0.0055
## 4          3.3              1.6                         0.107           0.1484
## 5          3.6              2.0                         0.169           0.0308
## 6          3.3              2.1                         0.336           0.2021
## 7          3.0              1.8                         0.079           0.0586
## 8          3.2              2.0                         0.312           0.4505
## 9          3.4              2.1                         0.375           0.2390
## 10         4.4              1.5                         0.150           1.2597
##    sd_ProductName__Intercept sigma r_NJ[1331,Intercept] r_NJ[1400,Intercept]
## 1                       0.30   1.9                -0.27                -1.85
## 2                       0.59   2.0                 0.46                -0.81
## 3                       0.42   1.9                -0.55                -1.53
## 4                       0.61   1.9                -0.84                -1.96
## 5                       0.33   1.9                -0.72                -2.19
## 6                       0.38   2.0                -0.22                -2.09
## 7                       0.41   1.8                 0.20                -2.08
## 8                       0.27   2.1                -0.45                -1.58
## 9                       0.42   1.8                -0.85                -2.63
## 10                      0.82   2.0                -0.37                -2.58
## # ... with 7990 more draws, and 137 more variables
## # ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

You'll note that there are a *lot* of variables. We are estimating (as I understand it) deflections from the grand mean for all of the levels of all of our main effects. We can summarize these using the `posterior_summary()` function:

```
posterior_summary(b_bitter) %>%
  as_tibble(rownames = "parameter") %>%
  mutate(across(where(is.numeric), ~round(., 2)))
```

```
## # A tibble: 145 x 5
##    parameter                    Estimate Est.Error  Q2.5 Q97.5
##    <chr>                           <dbl>     <dbl> <dbl> <dbl>
```

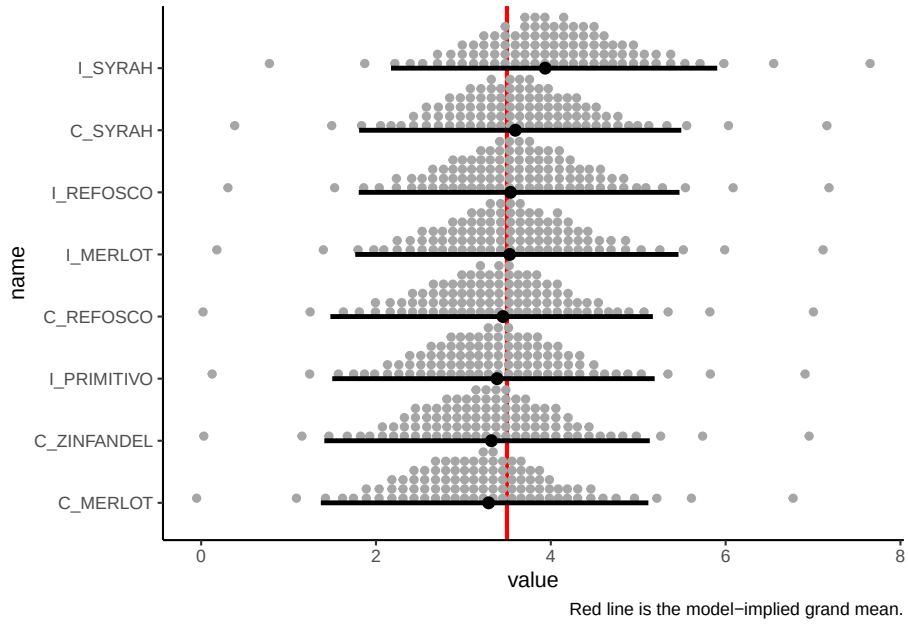
```
##  1 b_Intercept                      3.5       0.95  1.79  5.45
##  2 sd_NJ__Intercept                 2.21      0.5   1.49  3.42
##  3 sd_NJ:ProductName__Intercept     0.22      0.14  0.02  0.55
##  4 sd_NR__Intercept                 0.74      1.15  0.03  4.04
##  5 sd_ProductName__Intercept        0.42      0.23  0.07  0.97
##  6 sigma                            1.92      0.08  1.78  2.09
##  7 r_NJ[1331,Intercept]            -0.44      0.72 -1.85  1.03
##  8 r_NJ[1400,Intercept]            -1.78      0.73 -3.22 -0.32
##  9 r_NJ[1401,Intercept]             2.34      0.72  0.96  3.8
## 10 r_NJ[1402,Intercept]             2.31      0.72  0.92  3.74
## # i 135 more rows
```

For a sanity check, notice that the estimate for our grand mean (`b_Intercept`) is the observed value.

```
mean_y
```

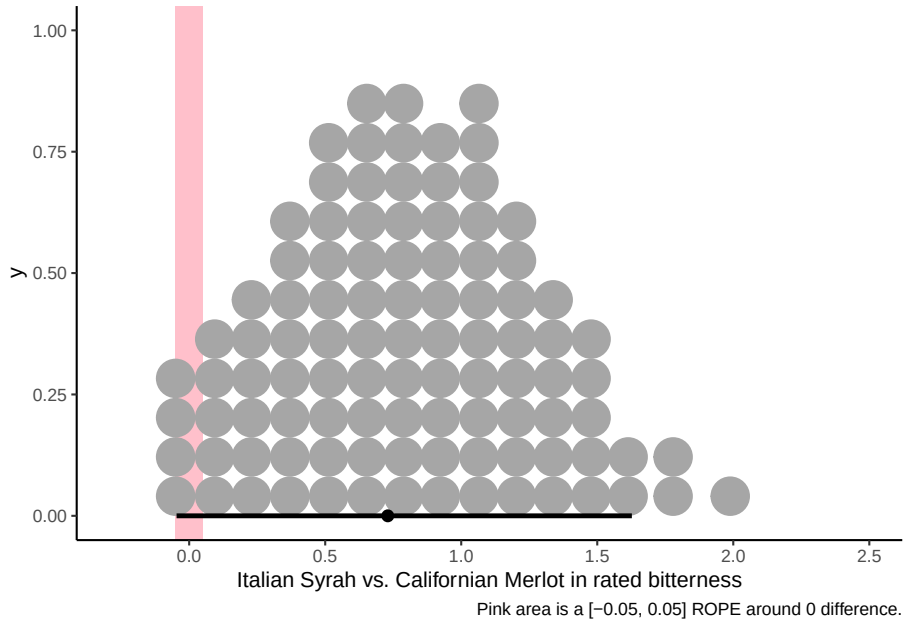
```
## [1] 3.494643
```

```
nd <-
  bayes_data %>%
  distinct(ProductName)

# we use `re_formula` to marginalize across the other factors

f <-
  fitted(b_bitter,
         newdata = nd,
         re_formula = ~ (1 | ProductName),
         summary = FALSE) %>%
  as_tibble() %>%
  set_names(nd$ProductName)

# We can look at the plausible 95% HDI for each wine's rated bitterness.

f %>%
  pivot_longer(everything()) %>%
  mutate(name = fct(name) %>% fct_reorder(.x = value, .fun = mean)) %>%
  ggplot(aes(x = value, y = name)) +
  geom_vline(xintercept = fixef(b_bitter)[, 1], size = 1, color = "red") +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95, quantiles = 100) +
  theme_classic() +
  labs(caption = "Red line is the model-implied grand mean.")
```



It sure looks like these wines are not easy to discriminate. One advantage of Bayesian approaches is that the paradigm of investigation allows us to investigate the posterior draws to explore things like pairwise comparisons with no need to worry about ideas like "family-wise error" (because we are not in a hypothesis-testing framework).

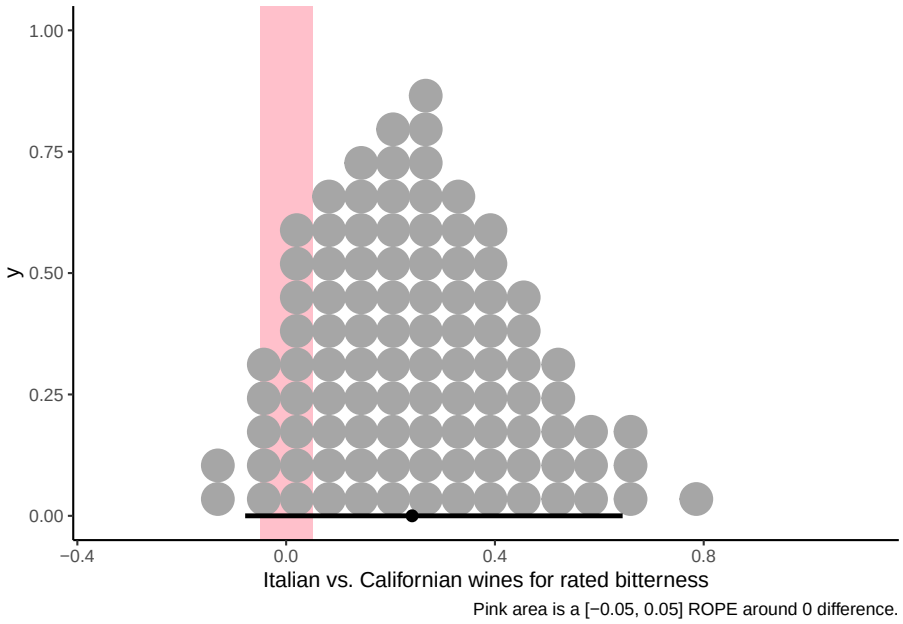
```
# Let's see if the Italian Syrah is plausibly more bitter than the Californian Merlot:

f %>%
  transmute(diff = I_SYRAH - C_MERLOT) %>%
  ggplot(aes(x = diff)) +
  geom_rect(aes(xmin = -0.05, xmax = 0.05,
                ymin = -Inf, ymax = Inf),
            size = 1, fill = "pink", alpha = 1/3) +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95, quantiles = 100) +
  theme_classic() +
  labs(x = "Italian Syrah vs. Californian Merlot in rated bitterness",
       caption = "Pink area is a [-0.05, 0.05] ROPE around 0 difference.")
```



It looks like the posterior differences between the most-bitter and least-bitter wines don't exclude a small "region of plausible equivalence" around 0 from the 95% area of highest posterior density, so even though there is a modal difference of around 0.75 points in bitterness between these wines, we can't entirely dismiss the possibility that there is no difference in bitterness. That being said, there is at least as much posterior probability density around 1.5 points of bitterness difference, so it seems unreasonable to me to claim that these wines are exactly the same in their level of bitterness–it just probably isn't that large a difference!

We also have the freedom to explore any contrasts that might be of interest. For example, we might be interested (for either exploratory data analysis based on the HDIs above or because of *a priori* theory) in knowing whether Italian wines are on average more bitter than California wines. We can do so simply:

```
f %>%
  transmute(diff = (I_SYRAH + I_MERLOT + I_PRIMITIVO + I_REFOSCO) / 4 -
              (C_MERLOT + C_SYRAH + C_ZINFANDEL + C_REFOSCO) / 4) %>%
  ggplot(aes(x = diff)) +
  geom_rect(aes(xmin = -0.05, xmax = 0.05,
                ymin = -Inf, ymax = Inf),
            size = 1, fill = "pink", alpha = 1/3) +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95, quantiles = 100) +
  theme_classic() +
  labs(x = "Italian vs. Californian wines for rated bitterness",
       caption = "Pink area is a [-0.05, 0.05] ROPE around 0 difference.")
```



Unsurprisingly, given the figures above, we see that there is some possibility of a very small amount of overall bitterness, but we'd be pretty hesitant to make broad statements when a ROPE around 0 is so clearly within the HDI for this contrast.

Finally, for my own interest, I am curious how `brms` would do using defaults in fitting the same ANOVA-like model.

```
b_bitter_default <-
  brm(data = bayes_data,
      family = gaussian,
      formula = rating ~ 1 + (1 | NJ) + (1 | NR) + (1 | ProductName) + (1 | NJ:ProductName),
      seed = 2, chains = 4, cores = 4, control = list(adapt_delta = 0.999, max_treedepth = 13),
      file = here("fits/fit_02_bitter_default"))
```

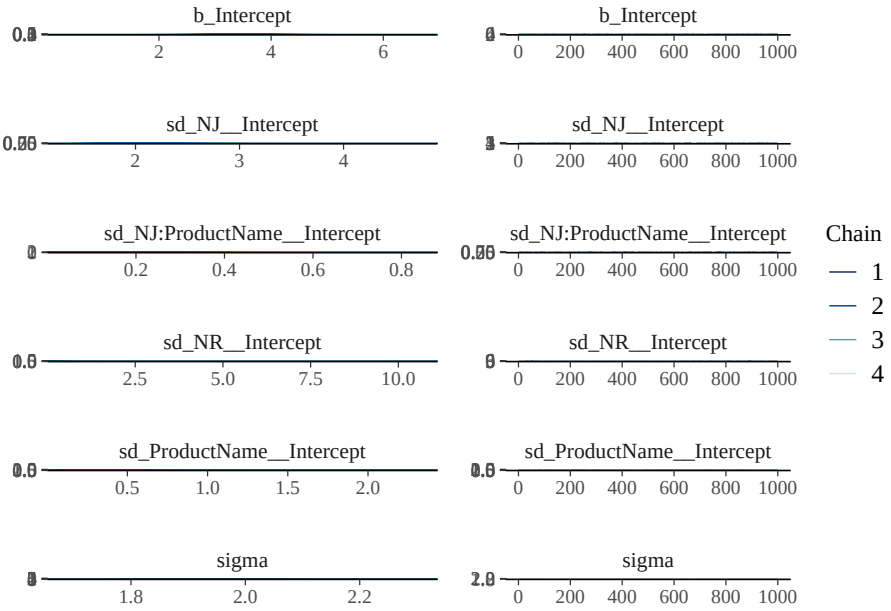
Interestingly, this model seemed to take much less time to fit and didn't have the issues with transitions. Let's look at the chains and the parameter estimates.

```
b_bitter_default
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rating ~ 1 + (1 | NJ) + (1 | NR) + (1 | ProductName) + (1 | NJ:ProductName)
##    Data: bayes_data (Number of observations: 336)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
```

```
##          total post-warmup draws = 4000
##
## Group-Level Effects:
## ~NJ (Number of levels: 14)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     2.16      0.48     1.45     3.31 1.01      880     1568
##
## ~NJ:ProductName (Number of levels: 112)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.19      0.14     0.01     0.52 1.00     1335     1765
##
## ~NR (Number of levels: 3)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.47      0.68     0.01     2.31 1.00     1116     1994
##
## ~ProductName (Number of levels: 8)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.39      0.22     0.05     0.92 1.00     1321     1441
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     3.45      0.68     2.07     4.74 1.01      914     1456
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.93      0.08     1.78     2.08 1.00     5824     3043
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
plot(b_bitter_default, N = 6, width = c(2, 3))
```



It all looks ok, but it seems like the estimates for some parameters are a bit less precise, probably because we allowed the model to use uninformed priors. We can see how that was done using the `get_prior()` function on the formula and data, as so:

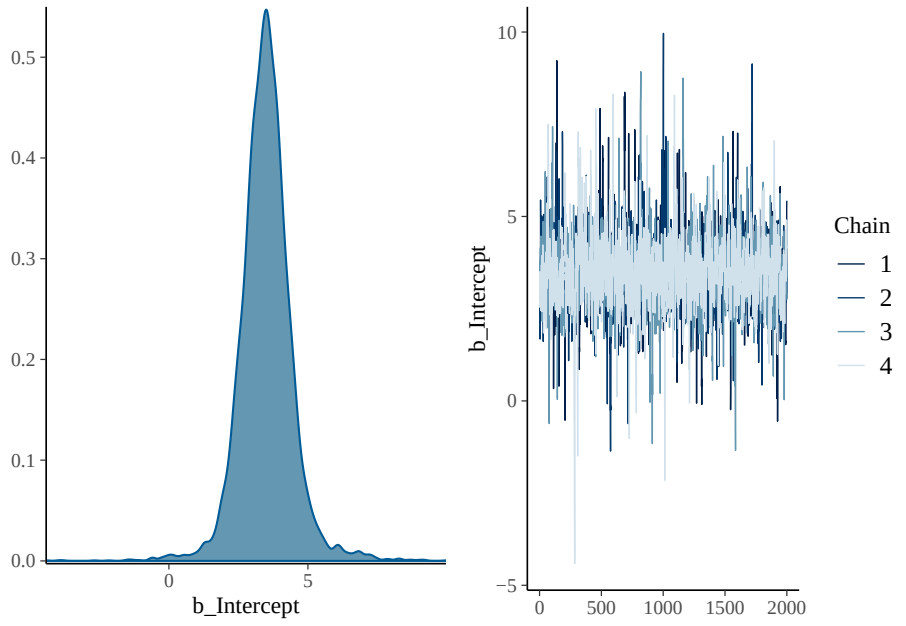
```
get_prior(formula = rating ~ 1 + (1 | NJ) + (1 | NR) + (1 | ProductName) + (1 | NJ:ProductName),
          data = bayes_data)
```

```
##                 prior     class      coef          group resp dpar nlpar lb ub
##  student_t(3, 3, 3.3) Intercept
##  student_t(3, 0, 3.3)        sd                                          0
##  student_t(3, 0, 3.3)        sd                     NJ                   0
##  student_t(3, 0, 3.3)        sd Intercept           NJ                   0
##  student_t(3, 0, 3.3)        sd           NJ:ProductName                 0
##  student_t(3, 0, 3.3)        sd Intercept NJ:ProductName                 0
##  student_t(3, 0, 3.3)        sd                     NR                   0
##  student_t(3, 0, 3.3)        sd Intercept           NR                   0
##  student_t(3, 0, 3.3)        sd              ProductName                 0
##  student_t(3, 0, 3.3)        sd Intercept    ProductName                 0
##  student_t(3, 0, 3.3)     sigma                                          0
##        source
##       default
##       default
##  (vectorized)
##  (vectorized)
```

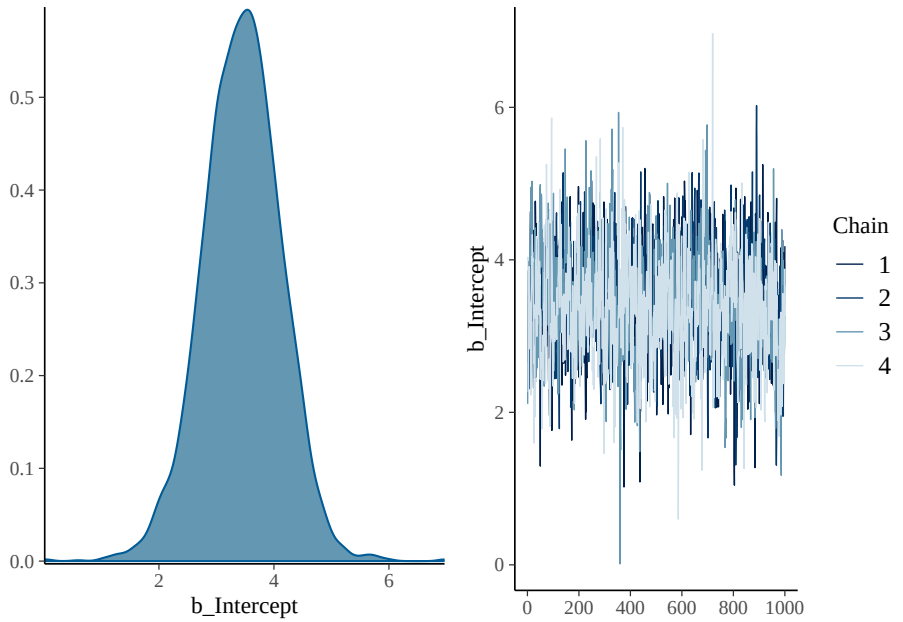
```
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##  (vectorized)
##       default
```

It looks like, by default, `brm()` uses very broad Student's t priors for all the parameters; I'm no expert but I think this is what are called "uninformed priors" and so will be almost entirely influenced by the data. For example, if we compare our original fit following Kurz's example where we use a normal distribution with `mean` and `sd` influenced by our data, we see a distinct difference in parameter estimates for the intercept:

```
# The original model with specified priors
plot(b_bitter, variable = "b_Intercept")
```



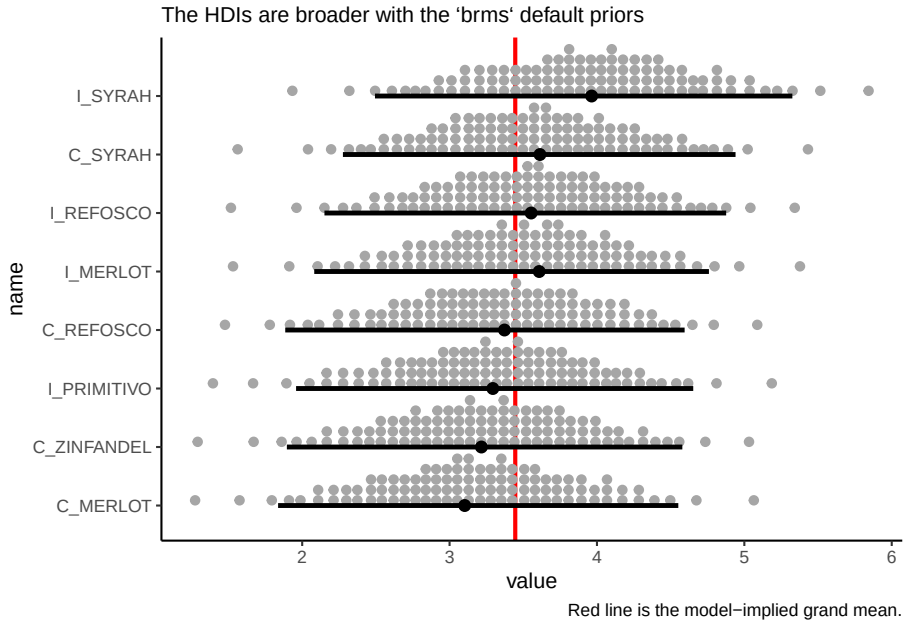

```
# The model with default priors
plot(b_bitter_default, variable = "b_Intercept")
```



So we lose some precision in exchange for speed. Let's look at how the default model predicts mean separation:

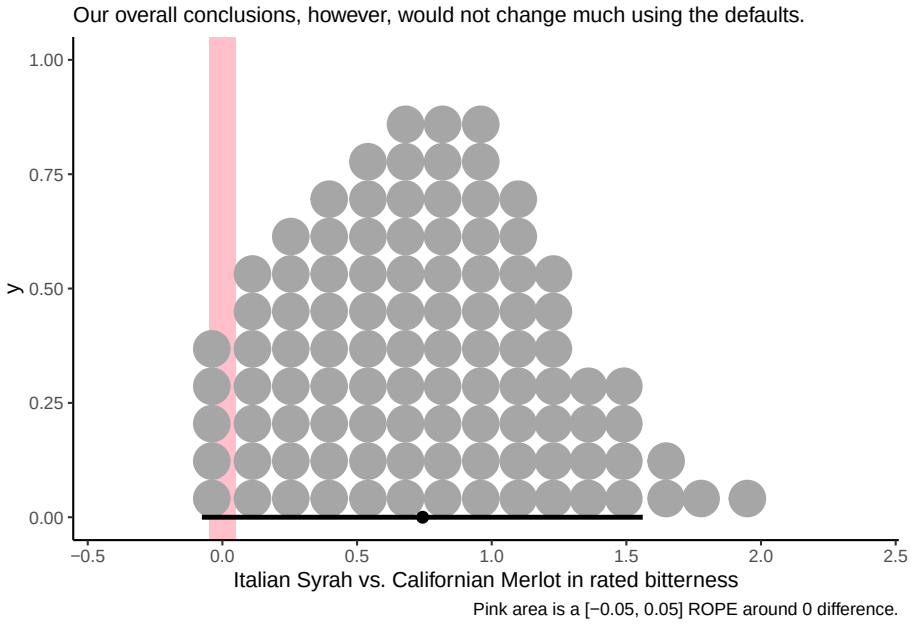
```
nd <-
  bayes_data %>%
  distinct(ProductName)

# we use `re_formula` to marginalize across the other factors

f <-
  fitted(b_bitter_default,
         newdata = nd,
         re_formula = ~ (1 | ProductName),
         summary = FALSE) %>%
  as_tibble() %>%
  set_names(nd$ProductName)

# We can look at the plausible 95% HDI for each wine's rated bitterness.

f %>%
  pivot_longer(everything()) %>%
  mutate(name = fct(name) %>% fct_reorder(.x = value, .fun = mean)) %>%
  ggplot(aes(x = value, y = name)) +
  geom_vline(xintercept = fixef(b_bitter_default)[, 1], size = 1, color = "red") +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95, quantiles = 100) +
```

```
  theme_classic() +
  labs(caption = "Red line is the model-implied grand mean.",
       subtitle = "The HDIs are broader with the `brms` default priors")
```



And we can do the same kind of check with the individual differences:

```
# Let's see if the Italian Syrah is plausibly more bitter than the Californian Merlot:

f %>%
  transmute(diff = I_SYRAH - C_MERLOT) %>%
  ggplot(aes(x = diff)) +
  geom_rect(aes(xmin = -0.05, xmax = 0.05,
                ymin = -Inf, ymax = Inf),
            size = 1, fill = "pink", alpha = 1/3) +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95, quantiles = 100) +
  theme_classic() +
  labs(x = "Italian Syrah vs. Californian Merlot in rated bitterness",
       caption = "Pink area is a [-0.05, 0.05] ROPE around 0 difference.",
       subtitle = "Our overall conclusions, however, would not change much using the de
```



So in this case, it seems like the speed of using `brms` defaults, at least for exploratory data analysis, may be worthwhile.

I might try to incorporate some more Bayesian tools as we go along, as I find them very interesting. If you want to learn more about these, I recommend the sources listed above and cited in the text here.

2.5 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
```

```
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] tidybayes_3.0.6 brms_2.20.1     Rcpp_1.0.11     tidytext_0.4.1
##  [5] agricolae_1.3-6 here_1.0.1      lubridate_1.9.2 forcats_1.0.0
##  [9] stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1     readr_2.1.4
## [13] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##   [1] tensorA_0.36.2       rstudioapi_0.15.0    magrittr_2.0.3
##   [4] estimability_1.4.1   farver_2.1.1         rmarkdown_2.23
##   [7] vctrs_0.6.3          base64enc_0.1-3      htmltools_0.5.6
##  [10] distributional_0.3.2 haven_2.5.3          broom_1.0.5
##  [13] janeaustenr_1.0.0    StanHeaders_2.26.27  htmlwidgets_1.6.2
##  [16] tokenizers_0.3.0     plyr_1.8.8           emmeans_1.8.7
##  [19] zoo_1.8-12           igraph_1.5.0.1       mime_0.12
##  [22] lifecycle_1.0.3      pkgconfig_2.0.3      colourpicker_1.3.0
##  [25] Matrix_1.6-0         R6_2.5.1             fastmap_1.1.1
##  [28] shiny_1.7.5          digest_0.6.33        klaR_1.7-2
##  [31] colorspace_2.1-0     ps_1.7.5             rprojroot_2.0.3
##  [34] crosstalk_1.2.0      SnowballC_0.7.1      labeling_0.4.3
##  [37] fansi_1.0.4          timechange_0.2.0     abind_1.4-5
##  [40] compiler_4.3.1       bit64_4.0.5          withr_2.5.0
##  [43] backports_1.4.1      inline_0.3.19        shinystan_2.6.0
##  [46] pkgbuild_1.4.2       highr_0.10           MASS_7.3-60
##  [49] gtools_3.9.4         loo_2.6.0            tools_4.3.1
##  [52] httpuv_1.6.11        threejs_0.3.3        quadprog_1.5-8
##  [55] glue_1.6.2           questionr_0.7.8      callr_3.7.3
##  [58] nlme_3.1-162         promises_1.2.1       grid_4.3.1
##  [61] checkmate_2.2.0      cluster_2.1.4        reshape2_1.4.4
##  [64] generics_0.1.3       gtable_0.3.4         labelled_2.12.0
##  [67] tzdb_0.4.0           hms_1.1.3            utf8_1.2.3
##  [70] pillar_1.9.0         ggdist_3.3.0         markdown_1.8
##  [73] vroom_1.6.3          posterior_1.4.1      later_1.3.1
##  [76] lattice_0.21-8       AlgDesign_1.2.1      bit_4.0.5
##  [79] tidyselect_1.2.0     miniUI_0.1.1.1       knitr_1.43
##  [82] arrayhelpers_1.1-0   gridExtra_2.3        bookdown_0.37
##  [85] stats4_4.3.1         xfun_0.39            bridgesampling_1.1-2
##  [88] matrixStats_1.0.0    DT_0.28              rstan_2.21.8
##  [91] stringi_1.7.12       yaml_2.3.7           evaluate_0.21
##  [94] codetools_0.2-19     cli_3.6.1            RcppParallel_5.1.7
##  [97] shinythemes_1.2.0    xtable_1.8-4         munsell_0.5.0
## [100] processx_3.8.2       coda_0.19-4          svUnit_1.0.6
```

```
## [103] parallel_4.3.1       rstantools_2.3.1.1   ellipsis_0.3.2
## [106] prettyunits_1.1.1    dygraphs_1.1.1.6     bayesplot_1.10.0
## [109] Brobdingnag_1.2-9    mvtnorm_1.2-2        scales_1.2.1
## [112] xts_0.13.1           crayon_1.5.2         combinat_0.0-8
## [115] rlang_1.1.1          shinyjs_2.1.0
```

Chapter 3

Dealing with missing data

In this short section, we'll look at "single imputation" using the Expectation Maximization approach. First, let's get our data imported and our packages set up.

```
library(tidyverse)
library(here)

descriptive_data <- read_csv(here("data/torriDAFinal.csv"))
missing_data_example <-read_csv(here("data/torrimiss.csv"))
```

3.1 Exploring missing data

In the original **R Opus**, HGH used the `missmda` package, from Susan Josse and Francois Husson, which to my understanding uses a PCA approach to do single imputation. Reading through their documentation, they note that this is equivalent to the "expectation maximization" (EM) approach to single imputation. We're going to use a slightly more modern package that combines approaches from a number of different missing-data packages, called `simputation`. It gives standard syntax for a number of different formula-based methods for imputation. We're going to also use the `skimr` package to easily see where we have missing data, and the `naniar` package to visualize.

```
library(simputation)
library(skimr)
library(naniar)
```

First off, let's take a look at our `missing_data_example` data set using the `skim()` method.

```
skim(missing_data_example) %>%
  # This is purely to allow the skimr::skim() to be rendered in PDF, ignore otherwise
  knitr::kable()
```

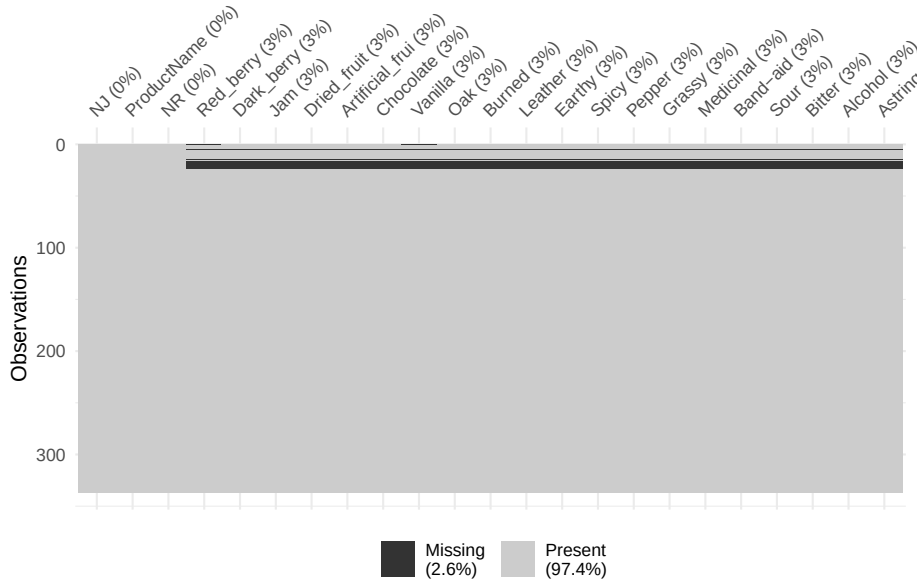
skim_type	skim_variable	n_missing	complete_rate	character.min	character.max	charact
character	ProductName	0	1.0000000	7	11	
numeric	NJ	0	1.0000000	NA	NA	
numeric	NR	0	1.0000000	NA	NA	
numeric	Red_berry	11	0.9672619	NA	NA	
numeric	Dark_berry	10	0.9702381	NA	NA	
numeric	Jam	10	0.9702381	NA	NA	
numeric	Dried_fruit	10	0.9702381	NA	NA	
numeric	Artificial_frui	10	0.9702381	NA	NA	
numeric	Chocolate	10	0.9702381	NA	NA	
numeric	Vanilla	11	0.9672619	NA	NA	
numeric	Oak	10	0.9702381	NA	NA	
numeric	Burned	10	0.9702381	NA	NA	
numeric	Leather	10	0.9702381	NA	NA	
numeric	Earthy	10	0.9702381	NA	NA	
numeric	Spicy	10	0.9702381	NA	NA	
numeric	Pepper	10	0.9702381	NA	NA	
numeric	Grassy	10	0.9702381	NA	NA	
numeric	Medicinal	10	0.9702381	NA	NA	
numeric	Band-aid	10	0.9702381	NA	NA	
numeric	Sour	10	0.9702381	NA	NA	
numeric	Bitter	10	0.9702381	NA	NA	
numeric	Alcohol	10	0.9702381	NA	NA	
numeric	Astringent	10	0.9702381	NA	NA	

This shows us that we have (induced) missingness in all of our outcome variables (the sensory descriptors), but our predictors (`ProductName`, `NJ`, and `NR`) have no missingness, which is good. It also shows us that we need to `mutate()` the latter two of those 3 variables into factors, as right now `R` will treat them as numeric predictors.

```
# Here we mutate the first 3 variables to all be factors
missing_data_example <-
  missing_data_example %>%
  mutate_at(vars(1:3), ~as.factor(.))

# let's also fix this in our complete dataset before we forget
descriptive_data <-
  descriptive_data %>%
  mutate_at(vars(1:3), ~as.factor(.))
```

We can then get a look at the patternness of missingness using `naniar`, which has a bunch of utilities for easily visualizing missing data.

```
missing_data_example %>%
  vis_miss()
```




```
# We see above that HGH just replaced some rows with NAs.  In this case, the
# data is NOT missing "completely at random", as we can confirm with a
# statistical test.
missing_data_example %>%
  mcar_test()
```

```
## # A tibble: 1 x 4
##   statistic    df  p.value missing.patterns
##       <dbl> <dbl>    <dbl>            <int>
## 1      57.7    24 0.000133                3
```

As we know that every row in our dataset is a "case" rather than an observation (i.e., our data is *wide* rather than *long*), we can use the `miss_case_summary()` function from `naniar` to give us a bit more info about how the missingness was introduced:

```
missing_data_example %>%
  miss_case_summary() %>%
  filter(n_miss > 0)
```

```
## # A tibble: 11 x 3
##     case n_miss pct_miss
##    <int>  <int>    <dbl>
##  1     6     20    87.0
##  2    15     20    87.0
##  3    17     20    87.0
##  4    18     20    87.0
##  5    19     20    87.0
##  6    20     20    87.0
##  7    21     20    87.0
##  8    22     20    87.0
##  9    23     20    87.0
## 10    24     20    87.0
## 11     1      2     8.70
```

The problem of missing data is a complex one, and if you have large quantities of missing data you will need to account for it in your reports. In this case, however, we're just going to accept the missingness (perhaps those subjects missed those sessions) and move on.

3.2 Imputing missing data

I am certainly no expert on missing data, but after doing some reading into the package HGH used in the original **R Opus**, `missMDA`, my understanding is that this package uses a Principal Component Analysis (PCA)-based approach to imputation which is both useful for the unique problems that sensory scientists typically tackle (e.g., dealing with highly multivariate data with high uncertainty and high multicollinearity) and is competitive with current methods. Therefore, we'll use this package for imputation here, as well as showing some alternatives below.

```
library(missMDA)

# We can first estimate the number of components to use in our imputation.
# NB: This may take a few minutes.
estim_ncpFAMD(missing_data_example, verbose = FALSE)
```

```
## $ncp
```

```
## [1] 5
##
## $criterion
##          0          1          2          3          4          5
## 0.07008772 0.05759964 0.05118433 0.04795663 0.04574107 0.04444935
```

```
# Then we run the imputation itself with the recommended number of components.
# In contrast to the original R Opus, which used ncp = 2, this function
# recommends we use ncp = 5.

missing_data_imputed <- imputeFAMD(missing_data_example, ncp = 5)

missing_data_imputed$completeObs %>%
  as_tibble()
```

```
## # A tibble: 336 x 23
##    NJ    ProductName NR    Red_berry Dark_berry   Jam Dried_fruit
##    <fct> <fct>       <fct>     <dbl>      <dbl> <dbl>       <dbl>
##  1 1331  C_MERLOT    7          4.71       5.8   2.1         4.7
##  2 1331  C_SYRAH     7          5.6        1.9   3.9         1.2
##  3 1331  C_ZINFANDEL 7          4.9        2.6   1.4         5.9
##  4 1331  C_REFOSCO   7          5          1.9   7.8         0.6
##  5 1331  I__MERLOT   7          3.3        7.2   0.5         5.8
##  6 1331  I_SYRAH     7          4.02       5.01  4.41        2.86
##  7 1331  I_PRIMITIVO 7          2.9        5.1   8.7         0.4
##  8 1331  I_REFOSCO   7          3.2        6     4           0.7
##  9 1400  C_MERLOT    7          0.1        0.1   0.2         2.9
## 10 1400  C_SYRAH     7          1.6        0.7   0           6.4
## # i 326 more rows
## # i 16 more variables: Artificial_frui <dbl>, Chocolate <dbl>, Vanilla <dbl>,
## #   Oak <dbl>, Burned <dbl>, Leather <dbl>, Earthy <dbl>, Spicy <dbl>,
## #   Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>, `Band-aid` <dbl>, Sour <dbl>,
## #   Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

We see that values have been filled in. Let's compare them to our original data visually.

```
descriptive_data
```

```
## # A tibble: 336 x 23
##    NJ    ProductName NR    Red_berry Dark_berry   Jam Dried_fruit
##    <fct> <fct>       <fct>     <dbl>      <dbl> <dbl>       <dbl>
##  1 1331  C_MERLOT    7           5.1        5.8   2.1         4.7
##  2 1331  C_SYRAH     7           5.6        1.9   3.9         1.2
```

```
##  3 1331  C_ZINFANDEL 7           4.9        2.6   1.4         5.9
##  4 1331  C_REFOSCO   7           5          1.9   7.8         0.6
##  5 1331  I_MERLOT    7           3.3        7.2   0.5         5.8
##  6 1331  I_SYRAH     7           5.7        3.6   8.7         1.9
##  7 1331  I_PRIMITIVO 7           2.9        5.1   8.7         0.4
##  8 1331  I_REFOSCO   7           3.2        6     4           0.7
##  9 1400  C_MERLOT    7           0.1        0.1   0.2         2.9
## 10 1400  C_SYRAH     7           1.6        0.7   0           6.4
## # i 326 more rows
## # i 16 more variables: Artificial_frui <dbl>, Chocolate <dbl>, Vanilla <dbl>,
## #   Oak <dbl>, Burned <dbl>, Leather <dbl>, Earthy <dbl>, Spicy <dbl>,
## #   Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>, `Band-aid` <dbl>, Sour <dbl>,
## #   Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

In the first row, both `Red_berry` and `Vanilla` were imputed. These get reasonably close. Let's compare the results of an ANOVA on the `Red_berry` variable for the imputed and the original data.

```
# Imputed data
aov(Red_berry ~ (ProductName + NJ + NR)^2,
    data = missing_data_imputed$completeObs) %>%
  summary()
```

```
##                 Df Sum Sq Mean Sq F value   Pr(>F)
## ProductName      7   84.4   12.06   3.546  0.00135 **
## NJ              13  599.8   46.14  13.571  < 2e-16 ***
## NR               2    2.9    1.46   0.429  0.65189
## ProductName:NJ  91  639.7    7.03   2.068 1.76e-05 ***
## ProductName:NR  14   47.9    3.42   1.005  0.44992
## NJ:NR           26   93.0    3.58   1.052  0.40305
## Residuals      182  618.8    3.40
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Complete data
aov(Red_berry ~ (ProductName + NJ + NR)^2,
    data = descriptive_data) %>%
  summary()
```

```
##                 Df Sum Sq Mean Sq F value   Pr(>F)
## ProductName      7   73.2   10.45   2.914  0.00652 **
## NJ              13  597.9   45.99  12.821  < 2e-16 ***
## NR               2    2.9    1.43   0.398  0.67201
## ProductName:NJ  91  659.0    7.24   2.019 3.18e-05 ***
```

```
## ProductName:NR  14   52.2    3.73   1.040  0.41539
## NJ:NR           26   97.2    3.74   1.042  0.41565
## Residuals      182  652.8    3.59
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You'll notice that the p-value for the imputed data is actually lower! But this is an artifact, as we have not adjusted our degrees of freedom–we need to reduce our numerator degrees of freedom to account for the number of imputed values, as those values are based on the other, observed values. This will reverse this trend. We'd see something similar in the rest of the data set.

3.3 Alternative approaches to imputation

While it is powerful, `missMDA` is not very fast, and may not be the best tool for imputation in all cases. In general, the `simputation` package has a really easy to use interface, which you can learn about by using `vignette("intro", package = "simputation")`. It offers a lot of competitive methods. For example, let's use a multiple regression model to predict missing variables in the dataset.

Something I've found with `simputation` is that it is flummoxed by non-syntactic names in `R`, such as `Band-aid` (you cannot typically use "-" in a name in `R`). We're going to quickly fix the names of our dataset in order to make it play nice before we proceed.

```
missing_data_imputed_lm <-
  missing_data_example %>%
  rename_all(~str_replace_all(., "-", "_")) %>%
  impute_lm(formula = . ~ ProductName + NR + NJ)
```

If we run the same checks as before on `Red_berry`, let's see what we get:

```
# Imputed data
aov(Red_berry ~ (ProductName + NJ + NR)^2,
    data = missing_data_imputed_lm) %>%
  summary()
```

```
##                 Df Sum Sq Mean Sq F value   Pr(>F)
## ProductName      7   90.1   12.87   3.798 0.000714 ***
## NJ              13  614.3   47.25  13.948  < 2e-16 ***
## NR               2    3.4    1.68   0.496 0.609491
## ProductName:NJ  91  638.7    7.02   2.072 1.67e-05 ***
## ProductName:NR  14   48.3    3.45   1.018 0.437183
```

```
## NJ:NR           26   89.3    3.43   1.014 0.451877
## Residuals      182  616.6    3.39
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Complete data
aov(Red_berry ~ (ProductName + NJ + NR)^2,
    data = descriptive_data) %>%
  summary()
```

```
##                 Df Sum Sq Mean Sq F value   Pr(>F)
## ProductName      7   73.2   10.45   2.914  0.00652 **
## NJ              13  597.9   45.99  12.821  < 2e-16 ***
## NR               2    2.9    1.43   0.398  0.67201
## ProductName:NJ  91  659.0    7.24   2.019 3.18e-05 ***
## ProductName:NR  14   52.2    3.73   1.040  0.41539
## NJ:NR           26   97.2    3.74   1.042  0.41565
## Residuals      182  652.8    3.59
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Why is our p-value even smaller? If you think about it, we've filled in missing values *using a linear model like the one we are analyzing*. Therefore, our hypothesis–that ratings for descriptors depend on the wine, the judge, and the rep–is leaking into our imputation. This is one reason it's critical to adjust our degrees of freedom for any hypothesis tests we use on these imputed data... or to use non-parametric models for imputation (like the ones from `missMDA`) that don't encode our hypotheses in the first place.

3.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack
##
## locale:
```

```
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] missMDA_1.18      naniar_1.0.0      skimr_2.1.5       simputation_0.2.8
##  [5] here_1.0.1        lubridate_1.9.2   forcats_1.0.0     stringr_1.5.0
##  [9] dplyr_1.1.2       purrr_1.0.1       readr_2.1.4       tidyr_1.3.0
## [13] tibble_3.2.1      ggplot2_3.4.3     tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.0     farver_2.1.1         fastmap_1.1.1
##  [4] digest_0.6.33        rpart_4.1.19         timechange_0.2.0
##  [7] estimability_1.4.1   lifecycle_1.0.3      cluster_2.1.4
## [10] multcompView_0.1-9   survival_3.5-5       magrittr_2.0.3
## [13] compiler_4.3.1       rlang_1.1.1          tools_4.3.1
## [16] utf8_1.2.3           yaml_2.3.7           knitr_1.43
## [19] labeling_0.4.3       htmlwidgets_1.6.2    bit_4.0.5
## [22] scatterplot3d_0.3-44 repr_1.1.6           norm_1.0-11.1
## [25] withr_2.5.0          nnet_7.3-19          grid_4.3.1
## [28] fansi_1.0.4          jomo_2.7-6           xtable_1.8-4
## [31] colorspace_2.1-0     mice_3.16.0          emmeans_1.8.7
## [34] scales_1.2.1         iterators_1.0.14     MASS_7.3-60
## [37] flashClust_1.01-2    cli_3.6.1            mvtnorm_1.2-2
## [40] rmarkdown_2.23       crayon_1.5.2         generics_0.1.3
## [43] rstudioapi_0.15.0    tzdb_0.4.0           minqa_1.2.5
## [46] splines_4.3.1        parallel_4.3.1       base64enc_0.1-3
## [49] vctrs_0.6.3          boot_1.3-28.1        Matrix_1.6-0
## [52] glmnet_4.1-8         jsonlite_1.8.7       bookdown_0.37
## [55] hms_1.1.3            mitml_0.4-5          bit64_4.0.5
## [58] visdat_0.6.0         ggrepel_0.9.3        FactoMineR_2.8
## [61] foreach_1.5.2        gower_1.0.1          glue_1.6.2
## [64] pan_1.9              nloptr_2.0.3         codetools_0.2-19
## [67] DT_0.28              shape_1.4.6          stringi_1.7.12
## [70] gtable_0.3.4         lme4_1.1-34          munsell_0.5.0
## [73] pillar_1.9.0         htmltools_0.5.6      R6_2.5.1
## [76] doParallel_1.0.17    rprojroot_2.0.3      vroom_1.6.3
## [79] evaluate_0.21        lattice_0.21-8       highr_0.10
## [82] backports_1.4.1      leaps_3.1            broom_1.0.5
## [85] Rcpp_1.0.11          nlme_3.1-162         coda_0.19-4
## [88] xfun_0.39            pkgconfig_2.0.3
```

Chapter 4

MANOVA (Multivariate Analysis of Variance)

In this extremely brief section we'll apply a multivariate ANOVA (MANOVA) to our data. MANOVA analyzes changes in the mean vectors of outcome variables based on categorical factors. We'll spend slightly more time than HGH did in the original **R Opus** on these concepts, although *only* slightly.

First, let's reload our packages and data:

```
library(tidyverse)
library(here)

descriptive_data <- read_csv(here("data/torriDAFinal.csv")) %>%

  # Don't forget to change the independent variables here to factors...

  mutate_at(.vars = vars(1:3), ~as.factor(.))
```

4.1 Running MANOVA

The syntax for MANOVA in `R` is very similar to that for ANOVA. However, as far as I know there isn't a *great* way to use a tidy approach with `map()` and `nest()` to get `manova()` to work; from what I can tell, this has to do with `R`'s formula syntax: we can't easily specify the outcome variable (on the left side of the `~` in the formula) using the named elements of a typical `R` formula as we did in the ANOVA section. There, we could write a formula like `rating ~ ProductName` (I'm simplifying to save myself typing). `R` is able to interpret a single name in

the formula syntax, but it will not parse something like `Red_berry:Astringent ~ ProductName`, which would be necessary for MANOVA.

Instead, we have to supply the multivariate outcomes as a (named) matrix *in the same (case or wide) order* as the categorical predictors. Since we don't really need this for anything else, we'll use nested function calls.

```
manova_res <-
  manova(as.matrix(descriptive_data[, 4:23]) ~ (ProductName + NJ + NR)^2,
         data = descriptive_data)
```

We can get a summary MANOVA table using the typical `summary()` function.

```
summary(manova_res, test = "W")
```

```
##                  Df   Wilks approx F num Df den Df    Pr(>F)
## ProductName       7 0.04243   4.7575    140 1093.6 < 2.2e-16 ***
## NJ               13 0.00001  12.5493    260 1819.7 < 2.2e-16 ***
## NR                2 0.64784   1.9756     40  326.0 0.0007056 ***
## ProductName:NJ   91 0.00000   1.7583   1820 3331.1 < 2.2e-16 ***
## ProductName:NR   14 0.22673   0.9414    280 1916.9 0.7387371
## NJ:NR            26 0.02215   1.3926    520 2672.6 1.873e-07 ***
## Residuals       182
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `test = "W"` argument asks for Wilk's Λ, which is for whatever reason the preferred test in sensory evaluation (HGH also prefers it). The default test provided by `R` is Pillai's V. Only in rare edge cases will the two tests disagree, which you are unlikely to encounter.

In this particular case, we see that there are significant differences in group mean vectors for wine (`ProductName`) and judge (`NJ`), as well as an interaction effect (`ProductName:NJ`).

4.2 What does MANOVA test?

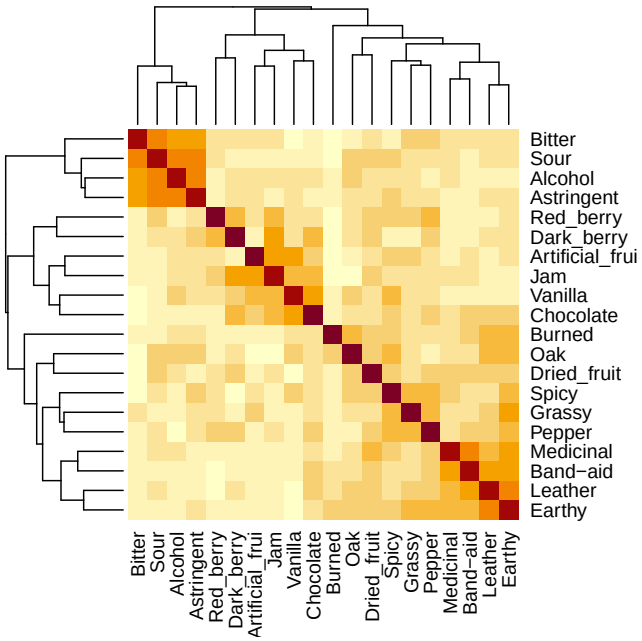
This isn't a statistics course. But I do feel that HGH didn't give much detail in the original **R Opus**, so let's take a second to think about MANOVA (and practice some wrangling while we're at it).

In typical ANOVA, we are interested in whether knowing the categorical predictor variables tells us something about expected value for an observation. In our case, we might want to know if knowing the wine identity (`C_ZINFANDEL`

for example) tells us something about the expected level of a sensory descriptor (for example, `Red_berry`). With elaboration, this is always the question we are answering with ANOVA.

What are we measuring in MANOVA? Well, if we treat our data in the wide, case-based way, we are asking whether the categorical predictor variables tell us anything about *all of the expected values, simultaneously.* This makes a lot of sense for our particular case: we might expect correlations among variables, and so rather than investigating variables one-by-one, as we did when we ran ANOVA, we want to take advantage of this *correlation structure.*

```
descriptive_data[, 4:23] %>%
  cor() %>%
  heatmap(revC = TRUE)
```



Without getting too outside our remit in the **R Opus**, MANOVA attemps to explain the variation of individual observations from their *group mean-vectors.* A "mean-vector" is just the mean of all dependent variables for a particular set of classifying categorical predictors. So, for our data, the dependent variables are all the sensory variables. We (generally) are interested in the variation around the **wine** mean-vectors. We can see these easily using the split-apply-combine approach:

```
wine_mean_vectors <-
  descriptive_data %>%
```

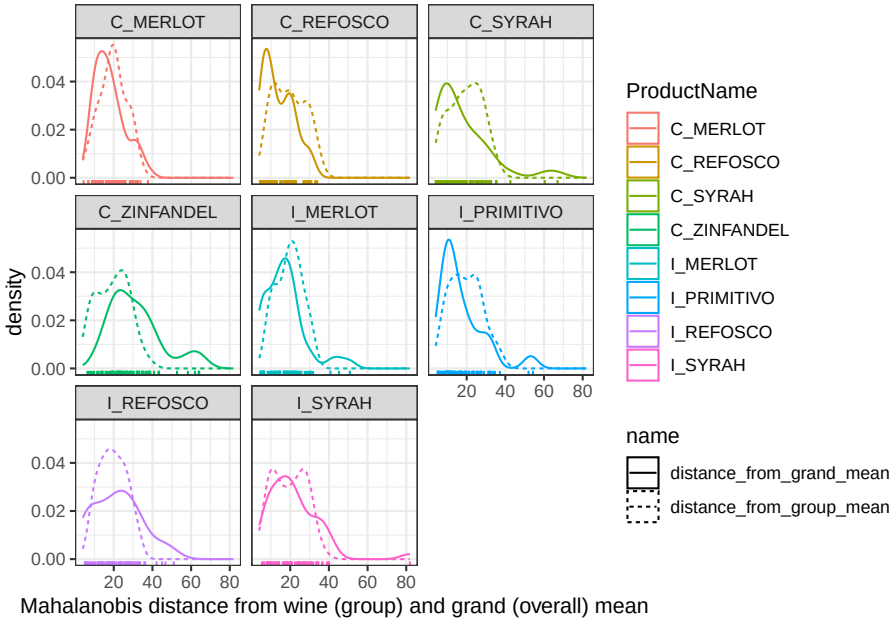
```
  group_by(ProductName) %>%
  summarize_if(.predicate = ~is.numeric(.),
               .funs = ~mean(.))

wine_mean_vectors
```

```
## # A tibble: 8 x 21
##   ProductName Red_berry Dark_berry   Jam Dried_fruit Artificial_frui Chocolate
##   <fct>           <dbl>      <dbl> <dbl>       <dbl>           <dbl>     <dbl>
## 1 C_MERLOT         2.46       3.05 1.37         1.86           0.776     1.19
## 2 C_REFOSCO        2.47       2.46 1.03         1.42           0.924     2.00
## 3 C_SYRAH          2.46       2.93 1.75         1.68           0.883     1.42
## 4 C_ZINFANDEL      3.08       3.06 1.98         2.06           0.864     0.969
## 5 I_MERLOT         2.79       2.35 0.843        1.85           0.574     0.783
## 6 I_PRIMITIVO      3.85       3.38 3.61         1.44           2.19      1.38
## 7 I_REFOSCO        2.48       3.01 1.54         1.87           1.11      0.810
## 8 I_SYRAH          3.17       4.48 3.10         2.16           2.43      1.20
## # i 14 more variables: Vanilla <dbl>, Oak <dbl>, Burned <dbl>, Leather <dbl>,
## #   Earthy <dbl>, Spicy <dbl>, Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>,
## #   `Band-aid` <dbl>, Sour <dbl>, Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

We want to know whether an observation from a particular wine is significantly "closer" to that wine's mean-vector than it is to other wine's mean-vectors. We test this using the Mahalanobis Distance, which is a generalization of the normalized distance (or z-score) from univariate statistics. Intuitively, we ask about the (multi-dimensional) distance of an observation from a mean-vector, divided by the covariance of the observations (the multivariate equivalent of standard deviation). Without getting into mathematics, we can ask about this distance using the `mahalanobis()` function.

```
distance_from_grand_mean <-
  descriptive_data[, -(1:3)] %>%
  as.matrix() %>%
  mahalanobis(x = ., center = colMeans(.), cov = cov(.))

descriptive_data %>%
  select(-NJ, -NR) %>%
  nest(data = -ProductName) %>%
  mutate(data = map(data, ~as.matrix(.x)),
         distance_from_group_mean = map(data, ~mahalanobis(x = .x, center = colMeans(.x
  select(-data) %>%
  unnest(distance_from_group_mean) %>%
  mutate(distance_from_grand_mean = distance_from_grand_mean) %>%
  pivot_longer(-ProductName) %>%
```

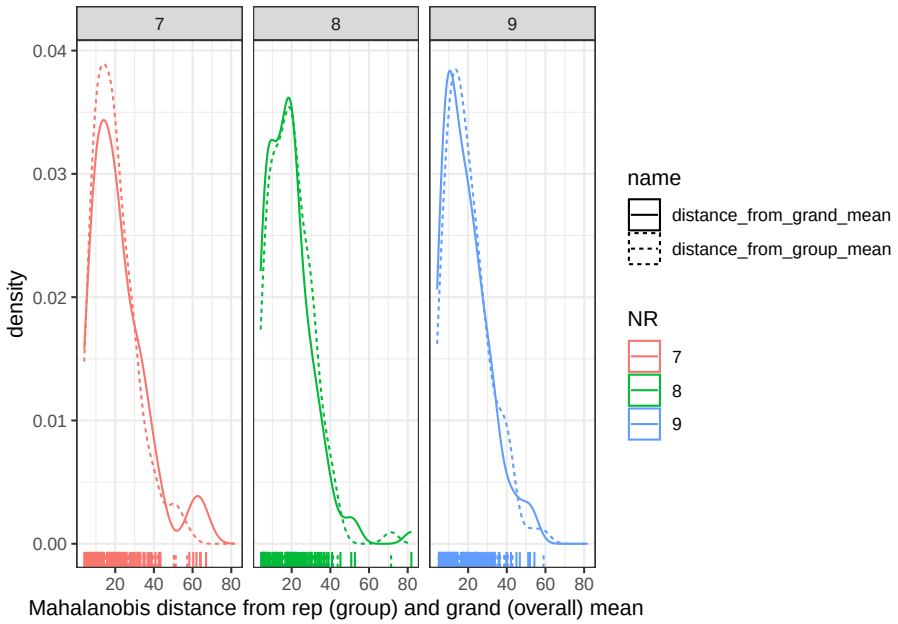
```
  ggplot(aes(x = value, color = ProductName, linetype = name)) +
  geom_density() +
  geom_rug() +
  theme_bw() +
  labs(x = "Mahalanobis distance from wine (group) and grand (overall) mean") +
  facet_wrap(~ProductName)
```



In this plot, we're able to see how far each individual row (observation) is, in a standardized, multivariate space, from the group mean (the dashed line) and the grand mean (the solid line). We can see that our groups are not necessarily well-defined: often, the grand mean seems to do a better job of describing our samples than the group mean for the specific wine. This might be a sign of our panelists needing further training, but recall that this (simple) visualization doesn't account for variance stemming from replicate and individual judge scaling behavior, nuisance factors that we actually included in the real MANOVA we ran above. When we include these, we find that in fact group membership (the type of wine) is important for predicting the sample mean vector.

```
descriptive_data %>%
  select(-ProductName, -NJ) %>%
  nest(data = -NR) %>%
  mutate(data = map(data, ~as.matrix(.x)),
         distance_from_group_mean = map(data, ~mahalanobis(x = .x, center = colMeans(.x), cov = c
  select(-data) %>%
```

```
  unnest(distance_from_group_mean) %>%
  mutate(distance_from_grand_mean = distance_from_grand_mean) %>%
  pivot_longer(-NR) %>%
  ggplot(aes(x = value, color = NR, linetype = name)) +
  geom_density() +
  geom_rug() +
  theme_bw() +
  labs(x = "Mahalanobis distance from rep (group) and grand (overall) mean") +
  facet_wrap(~NR)
```



Here we can see that the distribution is almost identical, meaning that knowing what replication (`NR`) the observation comes from gives us almost no new information. This is good news, because we don't *want* the replication to predict anything about the sensory quality of our wines!

We would display the same plot for our judges (`NJ`), but some of our judges are TOO repeatable: they have singular covariance matrices (we can tell because the determinant of their covariance matrices is 0), which indicates that the rank of their covariance matrix is less than the number of attributes. This in turn leads to a situation in which their product descriptions are lower-dimensional than the data.

```
descriptive_data %>%
  select(-ProductName, -NR) %>%
  nest(data = -NJ) %>%
```

```
  mutate(data = map(data, ~as.matrix(.x)),
         covariance_matrix = map(data, ~cov(.x)),
         determinant = map_dbl(covariance_matrix, ~det(.x)))
```

```
## # A tibble: 14 x 4
##    NJ    data             covariance_matrix determinant
##    <fct> <list>           <list>                   <dbl>
##  1 1331  <dbl [24 x 20]> <dbl [20 x 20]>      5.21e+ 2
##  2 1400  <dbl [24 x 20]> <dbl [20 x 20]>      1.86e- 6
##  3 1401  <dbl [24 x 20]> <dbl [20 x 20]>      1.57e-10
##  4 1402  <dbl [24 x 20]> <dbl [20 x 20]>      1.56e-16
##  5 1404  <dbl [24 x 20]> <dbl [20 x 20]>      0
##  6 1405  <dbl [24 x 20]> <dbl [20 x 20]>      5.16e- 1
##  7 1408  <dbl [24 x 20]> <dbl [20 x 20]>      2.73e-21
##  8 1409  <dbl [24 x 20]> <dbl [20 x 20]>      5.75e-13
##  9 1412  <dbl [24 x 20]> <dbl [20 x 20]>      3.09e- 3
## 10 1413  <dbl [24 x 20]> <dbl [20 x 20]>      0
## 11 1414  <dbl [24 x 20]> <dbl [20 x 20]>      1.68e- 8
## 12 1415  <dbl [24 x 20]> <dbl [20 x 20]>      5.55e+ 5
## 13 1416  <dbl [24 x 20]> <dbl [20 x 20]>      7.86e+ 1
## 14 1417  <dbl [24 x 20]> <dbl [20 x 20]>      1.31e- 4
```

Let's take a quick look at subject 1404's ratings:

```
descriptive_data %>%
  filter(NJ == "1404")
```

```
## # A tibble: 24 x 23
##    NJ    ProductName NR    Red_berry Dark_berry   Jam Dried_fruit
##    <fct> <fct>       <fct>     <dbl>      <dbl> <dbl>       <dbl>
##  1 1404  C_MERLOT    7           4          5.6   0           0
##  2 1404  C_SYRAH     7           0.5        7.5   0           4.4
##  3 1404  C_ZINFANDEL 7           4.9        7.9   0.5         1.9
##  4 1404  C_REFOSCO   7           0.3        1.8   0           0
##  5 1404  I_MERLOT    7           0.7        5.1   0           5.2
##  6 1404  I_SYRAH     7           5.2        4     4           2.9
##  7 1404  I_PRIMITIVO 7           6.4        8.8   7.6         1
##  8 1404  I_REFOSCO   7           3.7        4.8   3.8         3.9
##  9 1404  C_MERLOT    8           3.4        8.3   3.5         4.6
## 10 1404  C_SYRAH     8           3          8.8   3           4.3
## # i 14 more rows
## # i 16 more variables: Artificial_frui <dbl>, Chocolate <dbl>, Vanilla <dbl>,
## #   Oak <dbl>, Burned <dbl>, Leather <dbl>, Earthy <dbl>, Spicy <dbl>,
## #   Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>, `Band-aid` <dbl>, Sour <dbl>,
## #   Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

Because this subject never used "Chocolate" or "Band-aid" descriptors, they end up with a rank-deficient covariance matrix:

```
descriptive_data %>%
  filter(NJ == "1404") %>%
  select(-(1:3)) %>%
  cov() %>%
  round(2)
```

```
##                 Red_berry Dark_berry   Jam Dried_fruit Artificial_frui
## Red_berry            5.42       3.66  5.73        0.65            1.81
## Dark_berry           3.66       6.83  4.92        2.96            0.82
## Jam                  5.73       4.92  8.92        1.95            2.48
## Dried_fruit          0.65       2.96  1.95        3.61            0.22
## Artificial_frui      1.81       0.82  2.48        0.22            1.65
## Chocolate            0.00       0.00  0.00        0.00            0.00
## Vanilla              0.38       0.91 -0.48        0.22           -0.10
## Oak                 -0.28      -0.02 -2.02       -0.56           -0.66
## Burned              -4.74      -4.98 -5.14       -3.26           -1.65
## Leather             -0.29      -0.31 -0.28       -0.08           -0.09
## Earthy              -0.15      -0.20 -0.35       -0.18           -0.05
## Spicy                0.10      -0.30 -0.58       -0.19           -0.21
## Pepper              -0.53      -0.10 -1.45       -0.33           -0.53
## Grassy               0.14       0.07  0.12       -0.06            0.05
## Medicinal           -1.20      -1.46 -0.80        1.31           -0.39
## Band-aid             0.00       0.00  0.00        0.00            0.00
## Sour                -0.33       0.15 -0.72        0.81           -0.34
## Bitter              -0.10      -0.13 -0.38       -0.04           -0.20
## Alcohol             -0.29       0.37 -0.64       -0.12           -0.05
## Astringent          -0.38       0.19 -0.45        0.55           -0.26
##                 Chocolate Vanilla   Oak Burned Leather Earthy Spicy Pepper
## Red_berry               0    0.38 -0.28  -4.74   -0.29  -0.15  0.10  -0.53
## Dark_berry              0    0.91 -0.02  -4.98   -0.31  -0.20 -0.30  -0.10
## Jam                     0   -0.48 -2.02  -5.14   -0.28  -0.35 -0.58  -1.45
## Dried_fruit             0    0.22 -0.56  -3.26   -0.08  -0.18 -0.19  -0.33
## Artificial_frui         0   -0.10 -0.66  -1.65   -0.09  -0.05 -0.21  -0.53
## Chocolate               0    0.00  0.00   0.00    0.00   0.00  0.00   0.00
## Vanilla                 0    1.47  1.25  -0.72   -0.04  -0.19  0.25   0.47
## Oak                     0    1.25  2.66  -0.15   -0.03   0.02  0.47   0.66
## Burned                  0   -0.72 -0.15  10.72    0.44   0.17 -0.51   1.08
## Leather                 0   -0.04 -0.03   0.44    0.08  -0.01 -0.05   0.04
## Earthy                  0   -0.19  0.02   0.17   -0.01   0.17  0.05   0.03
## Spicy                   0    0.25  0.47  -0.51   -0.05   0.05  0.45   0.12
## Pepper                  0    0.47  0.66   1.08    0.04   0.03  0.12   0.68
## Grassy                  0    0.02 -0.01  -0.13   -0.01   0.00  0.04  -0.02
```

```
## Medicinal               0    0.11 -0.16  -2.25    0.16  -0.36  0.11  -0.56
## Band-aid                0    0.00  0.00   0.00    0.00   0.00  0.00   0.00
## Sour                    0    0.31  0.41  -2.06    0.01  -0.09  0.35   0.06
## Bitter                  0    0.10  0.29  -0.41   -0.01  -0.06  0.32  -0.02
## Alcohol                 0    0.39  0.59   0.61   -0.02  -0.03 -0.07   0.31
## Astringent              0    0.16  0.59  -0.18   -0.02  -0.01 -0.01   0.07
##                  Grassy Medicinal Band-aid  Sour Bitter Alcohol Astringent
## Red_berry          0.14     -1.20        0 -0.33  -0.10   -0.29      -0.38
## Dark_berry         0.07     -1.46        0  0.15  -0.13    0.37       0.19
## Jam                0.12     -0.80        0 -0.72  -0.38   -0.64      -0.45
## Dried_fruit       -0.06      1.31        0  0.81  -0.04   -0.12       0.55
## Artificial_frui    0.05     -0.39        0 -0.34  -0.20   -0.05      -0.26
## Chocolate          0.00      0.00        0  0.00   0.00    0.00       0.00
## Vanilla            0.02      0.11        0  0.31   0.10    0.39       0.16
## Oak               -0.01     -0.16        0  0.41   0.29    0.59       0.59
## Burned            -0.13     -2.25        0 -2.06  -0.41    0.61      -0.18
## Leather           -0.01      0.16        0  0.01  -0.01   -0.02      -0.02
## Earthy             0.00     -0.36        0 -0.09  -0.06   -0.03      -0.01
## Spicy              0.04      0.11        0  0.35   0.32   -0.07      -0.01
## Pepper            -0.02     -0.56        0  0.06  -0.02    0.31       0.07
## Grassy             0.02     -0.06        0  0.03   0.03   -0.02      -0.05
## Medicinal         -0.06      6.46        0  1.82   0.41   -0.66       0.04
## Band-aid           0.00      0.00        0  0.00   0.00    0.00       0.00
## Sour               0.03      1.82        0  1.64   0.41   -0.27       0.17
## Bitter             0.03      0.41        0  0.41   0.76    0.01       0.05
## Alcohol           -0.02     -0.66        0 -0.27   0.01    0.59       0.21
## Astringent        -0.05      0.04        0  0.17   0.05    0.21       0.52
```

The rows (columns) for "Chocolate" and "Band-aid" are exact scaled multiples of each other (they are in fact both entirely 0), and so this matrix is of rank less than its dimensionality, leading to a singularity problem.

4.3 BONUS: Hierarchical Bayes instead of MANOVA

In the section on ANOVA, we explored applying an ANOVA-like model to our rating data. But we really have a set of correlated dependent variables:

```
# For correlation tibbles
library(corrr)

descriptive_data %>%
  select(-c(1:3)) %>%
  correlate(quiet = TRUE)
```

```
## # A tibble: 20 x 21
##    term       Red_berry Dark_berry     Jam Dried_fruit Artificial_frui Chocolate
##    <chr>          <dbl>      <dbl>   <dbl>       <dbl>           <dbl>     <dbl>
##  1 Red_berry    NA          0.339   0.286        0.231           0.169     0.143
##  2 Dark_berry    0.339     NA       0.420        0.294           0.111     0.359
##  3 Jam           0.286      0.420  NA            0.247           0.438     0.385
##  4 Dried_fru~    0.231      0.294   0.247       NA               0.145     0.256
##  5 Artificia~    0.169      0.111   0.438        0.145          NA         0.285
##  6 Chocolate     0.143      0.359   0.385        0.256           0.285    NA
##  7 Vanilla       0.122      0.206   0.404        0.119           0.399     0.455
##  8 Oak           0.113      0.183   0.0261       0.320           0.0649    0.201
##  9 Burned       -0.0493    -0.0555 -0.0655       0.149           0.0127    0.107
## 10 Leather       0.0668     0.184   0.116        0.340           0.0953    0.324
## 11 Earthy        0.139      0.210   0.124        0.354           0.143     0.308
## 12 Spicy         0.254      0.107   0.146        0.301           0.268     0.285
## 13 Pepper        0.319      0.307   0.204        0.329           0.166     0.344
## 14 Grassy        0.246      0.0908  0.157        0.222           0.284     0.238
## 15 Medicinal     0.0721     0.0880  0.142        0.351           0.0739    0.221
## 16 Band-aid      0.0513     0.112   0.146        0.292           0.125     0.289
## 17 Sour          0.193      0.157   0.0909       0.308           0.103     0.105
## 18 Bitter        0.0727     0.115   0.116        0.119           0.0968    0.0363
## 19 Alcohol       0.0960     0.210   0.160        0.221           0.160     0.162
## 20 Astringent    0.143      0.236   0.145        0.196           0.185     0.102
## # i 14 more variables: Vanilla <dbl>, Oak <dbl>, Burned <dbl>, Leather <dbl>,
## #   Earthy <dbl>, Spicy <dbl>, Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>,
## #   `Band-aid` <dbl>, Sour <dbl>, Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

I wonder if we can use a Bayesian hierarchical approach to simultaneously model all of the descriptor ratings. We can do this by treating `descriptor` as a factor in an ANOVA-like model:

```
rating ~ 1 + (1 | descriptor) + (1 | ProductName) + (1 | NJ) + (1 | NR) +
  (1 | ProductName:NJ) + (1 | ProductName:descriptor) + (1 | NJ:descriptor)
```

This is only kind of theoretically justified. We are implying that all descriptors have some hierarchical source of variation. In one way, this makes no sense, but in another it isn't entirely crazy. Let's see if this model if fittable.

```
# First, get the data into the right shape:

descriptive_data_tidy <-
  descriptive_data %>%
  pivot_longer(cols = -c(NJ, NR, ProductName),
               names_to = "descriptor",
               values_to = "rating")
```

```
library(brms)
library(tidybayes)

# Note: this took about an hour to run on my early-2020 MacBook.  It's a silly
# idea that I am exploring, and so probably not worth that time!
b_all <-
  brm(data = descriptive_data_tidy,
      family = gaussian,
      formula = rating ~ 1 + (1 | descriptor) + (1 | ProductName) + (1 | NJ) + (1 | NR) +
        (1 | ProductName:NJ) + (1 | ProductName:descriptor) + (1 | NJ:descriptor),
      iter = 4000, warmup = 1000, chains = 4, cores = 4, seed = 4,
      control = list(adapt_delta = 0.999, max_treedepth = 13),
      file = here("fits/fit_04_all_descriptors"))
```

We can inspect the overall model fit, although plotting posterior draws for all parameters would take up so much space that we are not going to do so. Let's look at the overall model summary:

```
b_all
```

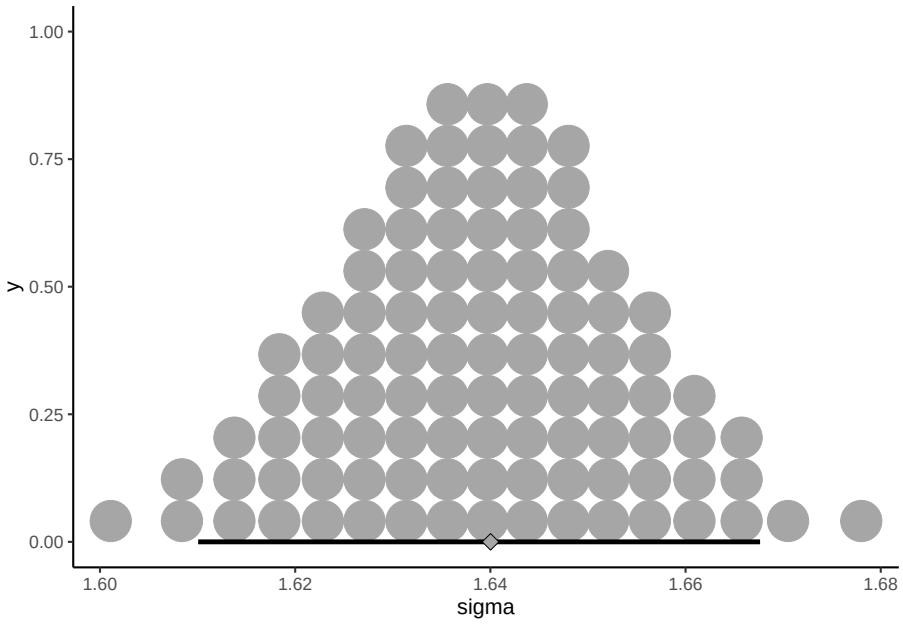
```
## Warning: There were 2 divergent transitions after warmup. Increasing
## adapt_delta above 0.999 may help. See
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rating ~ 1 + (1 | descriptor) + (1 | ProductName) + (1 | NJ) + (1 | NR) + (1 | Produc
##    Data: descriptive_data_tidy (Number of observations: 6720)
##   Draws: 4 chains, each with iter = 4000; warmup = 1000; thin = 1;
##          total post-warmup draws = 12000
##
## Group-Level Effects:
## ~descriptor (Number of levels: 20)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     1.21      0.23     0.84     1.75 1.00     2099     3868
##
## ~NJ (Number of levels: 14)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     1.11      0.25     0.73     1.71 1.00     2178     4357
##
## ~NJ:descriptor (Number of levels: 280)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     1.01      0.05     0.92     1.12 1.00     2721     4913
##
```

```
## ~NR (Number of levels: 3)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.15      0.29     0.00     0.92 1.00     2345     4231
##
## ~ProductName (Number of levels: 8)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.08      0.07     0.00     0.26 1.00     1830     4353
##
## ~ProductName:descriptor (Number of levels: 160)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.49      0.04     0.42     0.57 1.00     3718     6474
##
## ~ProductName:NJ (Number of levels: 112)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.13      0.04     0.04     0.20 1.00     2299     1574
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     2.09      0.44     1.21     2.96 1.00     1434     2814
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.64      0.01     1.61     1.67 1.00    10759     9035
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

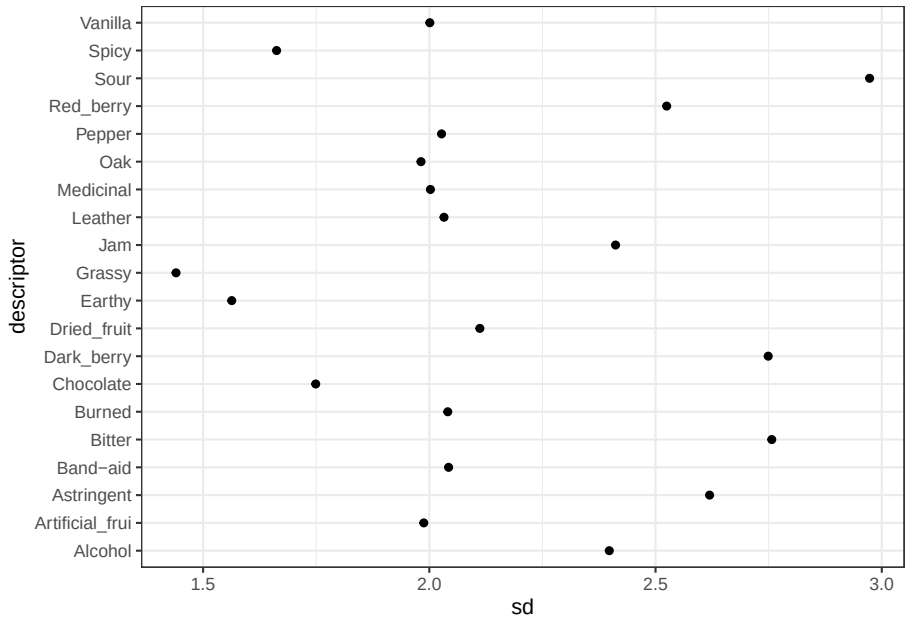
I am genuinely surprised that we were able to fit a model with only 2 divergent transitions. Looking at $\hat{R}$ and ESS, it appears that we don't have huge problems with the estimation part of the task. The question is whether this approach makes any sense. Let's take a look at the overall estimated standard deviation, σ_{rating}, which is held constant across all groups:

```
draws <- as_draws_df(b_all)

draws %>%
  ggplot(aes(x = sigma)) +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95,
                    shape = 23, stroke = 1/4, point_size = 3, slab_size = 1/4,
                    quantiles = 100) +
  theme_classic()
```



We can see that the standard deviation is pretty large, which makes sense with the approach we've chosen, because the individual standard deviation in different subgroups of observations (within descriptors, within subjects, and within reps) is quite large (for example, with the descriptors):

```
descriptive_data_tidy %>%
  group_by(descriptor) %>%
  summarize(sd = sd(rating)) %>%
  ggplot(aes(x = sd, y = descriptor)) +
  geom_point() +
  theme_bw()
```



Some of our descriptors have observed standard deviations less than 1.5, and some are almost 3! That's a pretty big spread. There are ways to model variance heterogeneously, and this would be a better approach in this situation, but since this is mostly exploration on my part, and it already took ~1 hour to run this model, I am not going to run a more intensive model. Check out that link if you want to learn more.

We can see that a consequence of this hierarchical model is that the overall mean rating tendency influences the estimates for subgroups. For example, our more extreme mean ratings get more shrunk back towards the grand mean for all ratings:

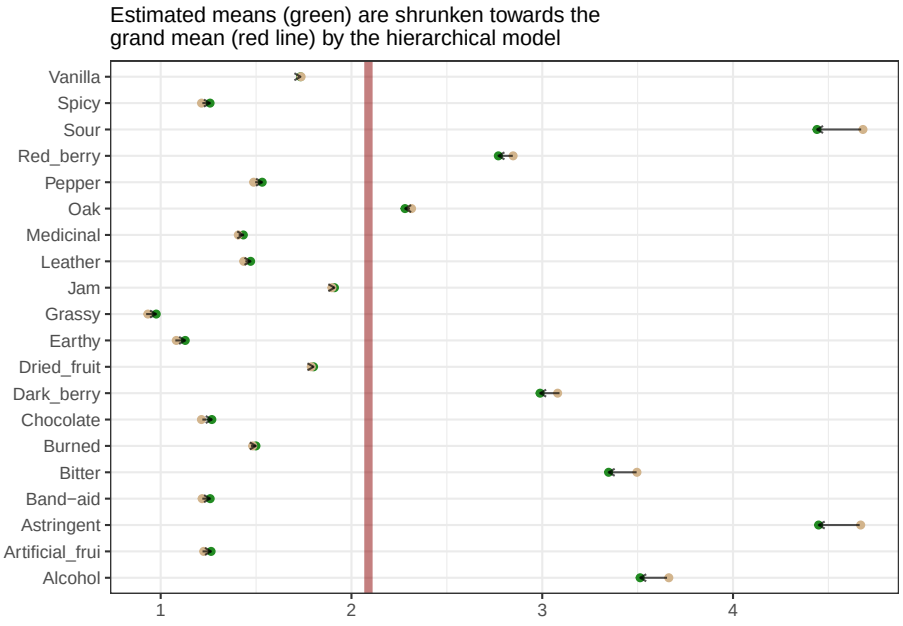
```
# First, we'll reformulate our model to look at the main effect of descriptors

# We need to tell the `fitted()` function that we are requesting posterior draws
# for each distinct descriptor group mean (mu) parameter
nd <- descriptive_data_tidy %>% distinct(descriptor)

f <-
  fitted(b_all,
         re_formula = ~ (1 | descriptor),
         newdata = nd) %>%
  as_tibble() %>%
  bind_cols(nd) %>%
  select(descriptor, Estimate) %>%
  left_join(descriptive_data_tidy %>%
```

```
              group_by(descriptor) %>%
              summarize(Observed = mean(rating)))
```

```
## Joining with `by = join_by(descriptor)`
```

```
f %>%
  ggplot(aes(y = descriptor)) +
  geom_vline(xintercept = fixef(b_all)[, 1],
             color = "darkred", size = 2, alpha = 1/2) +
  geom_point(aes(x = Estimate), color = "forestgreen") +
  geom_point(aes(x = Observed), color = "tan",
             position = position_jitter(height = 0, width = 0.01, seed = 4)) +
  geom_segment(aes(yend = descriptor, x = Observed, xend = Estimate),
               arrow = arrow(length = unit(x = 0.05, units = "in"), ends = "last"),
               alpha = 2/3) +
  theme_bw() +
  labs(x = NULL, y = NULL,
       subtitle = "Estimated means (green) are shrunken towards the\ngrand mean (red line) by the
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Whether this is a reasonable consequence depends on our assumptions about the data. I think there might be something to this approach for helping to combat heterogeneity in rating approaches among subjects, but I am not really enough of an expert in this area to determine if that really makes sense.

What we can do with the current model is use it investigate useful contrasts. We take the same approach as above to get the marginalized posterior across the descriptors and the wines.

```
nd <-
  descriptive_data_tidy %>%
  distinct(descriptor, ProductName) %>%
  mutate(col_names = str_c(ProductName, ":", descriptor))


f <-
  nd %>%
  fitted(b_all,
         newdata = .,
         # We need to marginalize across both `NJ` and `NR` for this
         re_formula = ~ (1 | ProductName) + (1 | descriptor) + (1 | ProductName:descrip
         summary = FALSE) %>%
  as_tibble() %>%
  set_names(nd %>% pull(col_names))
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
## `.name_repair` is omitted as of tibble 2.0.0.
## i Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
head(f)
```

```
## # A tibble: 6 x 160
##   `C_MERLOT:Red_berry` `C_MERLOT:Dark_berry` `C_MERLOT:Jam`
##                  <dbl>                 <dbl>          <dbl>
## 1                 2.54                  3.36           2.26
## 2                 2.50                  3.42           1.40
## 3                 2.45                  3.68           1.41
## 4                 3.07                  3.78           1.75
## 5                 2.90                  3.85           2.04
## 6                 3.05                  3.91           1.74
## # i 157 more variables: `C_MERLOT:Dried_fruit` <dbl>,
## #   `C_MERLOT:Artificial_frui` <dbl>, `C_MERLOT:Chocolate` <dbl>,
```

```
## #   `C_MERLOT:Vanilla` <dbl>, `C_MERLOT:Oak` <dbl>, `C_MERLOT:Burned` <dbl>,
## #   `C_MERLOT:Leather` <dbl>, `C_MERLOT:Earthy` <dbl>, `C_MERLOT:Spicy` <dbl>,
## #   `C_MERLOT:Pepper` <dbl>, `C_MERLOT:Grassy` <dbl>,
## #   `C_MERLOT:Medicinal` <dbl>, `C_MERLOT:Band-aid` <dbl>,
## #   `C_MERLOT:Sour` <dbl>, `C_MERLOT:Bitter` <dbl>, ...
```

This gets us a rather large table of the estimates for each descriptor at each wine. These are the estimated mean parameters for the "simple effects" that can let us examine contrasts. This full data table is too large to be wieldy, so we can trim it down for individual inquiries. For example, let's examine `Vanilla` and `Chocolate`, since they are often confusable, and see if there is a difference in the way `rating` changes for them in across two different wines… let's say `C_MERLOT` and `I_MERLOT`.

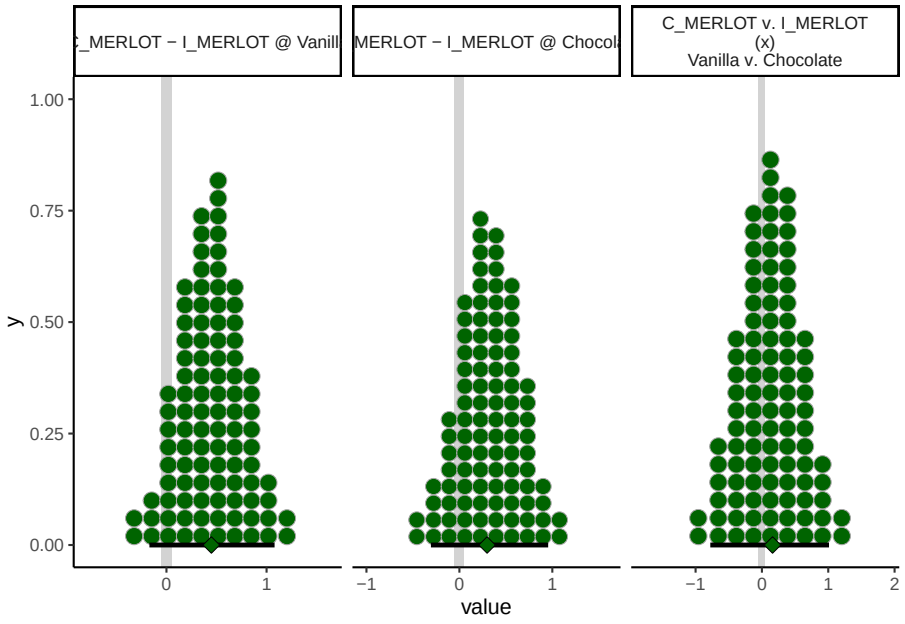
```
interactions <-
  f %>%
  transmute(`C_MERLOT - I_MERLOT @ Vanilla` = `C_MERLOT:Vanilla` - `I_MERLOT:Vanilla`,
            `C_MERLOT - I_MERLOT @ Chocolate` = `C_MERLOT:Chocolate` - `I_MERLOT:Chocolate`) %>%
  mutate(`C_MERLOT v. I_MERLOT\n(x)\nVanilla v. Chocolate` = `C_MERLOT - I_MERLOT @ Vanilla` - `C

# What does this look like?
head(interactions)
```

```
## # A tibble: 6 x 3
##   `C_MERLOT - I_MERLOT @ Vanilla` C_MERLOT - I_MERLOT @~1 C_MERLOT v. I_MERLOT~2
##                             <dbl>                  <dbl>                  <dbl>
## 1                         -0.0537                  0.626                 -0.679
## 2                          0.816                   0.285                  0.531
## 3                          0.292                   0.103                  0.189
## 4                          0.442                   0.612                 -0.170
## 5                          0.259                   0.413                 -0.154
## 6                          0.542                   0.162                  0.380
## # i abbreviated names: 1: `C_MERLOT - I_MERLOT @ Chocolate`,
## #   2: `C_MERLOT v. I_MERLOT\n(x)\nVanilla v. Chocolate`
```

Now we can plot this, with just a little bit more wrangling:

```
levels <- colnames(interactions)

interactions %>%
  pivot_longer(everything()) %>%
  mutate(name = factor(name, levels = levels)) %>%
  ggplot(aes(x = value)) +
  # Throw in a ROPE of [-0.05, 0.05] around 0
  geom_rect(aes(xmin = -0.05, xmax = 0.05,
```

```
                ymin = -Inf, ymax = Inf),
            color = "transparent", fill = "lightgrey") +
  stat_dotsinterval(point_interval = mode_hdi, .width = 0.95,
                    shape = 23, stroke = 1/4, point_size = 3, slab_size = 1/4,
                    quantiles = 100, fill = "darkgreen") +
  facet_wrap(~name, scales = "free_x") +
  theme_classic()
```



In retrospect, this isn't a very interesting comparison: it turns out both the simple effects and (unsurprisingly, given the former statement) contrast effect both have HDIs that firmly include a ROPE around 0. Again, this was more of an exercise in exploring this tool than something I was expecting to find necessarily immediately useful.

4.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
```

```
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] tidybayes_3.0.6 brms_2.20.1     Rcpp_1.0.11     corrr_0.4.4
##  [5] here_1.0.1      lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0
##  [9] dplyr_1.1.2     purrr_1.0.1     readr_2.1.4     tidyr_1.3.0
## [13] tibble_3.2.1    ggplot2_3.4.3   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##   [1] gridExtra_2.3        inline_0.3.19        rlang_1.1.1
##   [4] magrittr_2.0.3       matrixStats_1.0.0    compiler_4.3.1
##   [7] loo_2.6.0            callr_3.7.3          vctrs_0.6.3
##  [10] reshape2_1.4.4       quadprog_1.5-8       arrayhelpers_1.1-0
##  [13] pkgconfig_2.0.3      crayon_1.5.2         fastmap_1.1.1
##  [16] backports_1.4.1      ellipsis_0.3.2       labeling_0.4.3
##  [19] utf8_1.2.3           threejs_0.3.3        promises_1.2.1
##  [22] rmarkdown_2.23       markdown_1.8         tzdb_0.4.0
##  [25] ps_1.7.5             bit_4.0.5            xfun_0.39
##  [28] highr_0.10           later_1.3.1          prettyunits_1.1.1
##  [31] parallel_4.3.1       R6_2.5.1             dygraphs_1.1.1.6
##  [34] StanHeaders_2.26.27  stringi_1.7.12       estimability_1.4.1
##  [37] bookdown_0.37        rstan_2.21.8         knitr_1.43
##  [40] zoo_1.8-12           base64enc_0.1-3      bayesplot_1.10.0
##  [43] httpuv_1.6.11        Matrix_1.6-0         igraph_1.5.0.1
##  [46] timechange_0.2.0     tidyselect_1.2.0     rstudioapi_0.15.0
##  [49] abind_1.4-5          yaml_2.3.7           codetools_0.2-19
##  [52] miniUI_0.1.1.1       processx_3.8.2       pkgbuild_1.4.2
##  [55] lattice_0.21-8       plyr_1.8.8           shiny_1.7.5
##  [58] withr_2.5.0          bridgesampling_1.1-2 posterior_1.4.1
##  [61] coda_0.19-4          evaluate_0.21        RcppParallel_5.1.7
##  [64] ggdist_3.3.0         xts_0.13.1           pillar_1.9.0
##  [67] tensorA_0.36.2       checkmate_2.2.0      DT_0.28
##  [70] stats4_4.3.1         shinyjs_2.1.0        distributional_0.3.2
##  [73] generics_0.1.3       vroom_1.6.3          rprojroot_2.0.3
```

```
##  [76] hms_1.1.3           rstantools_2.3.1.1   munsell_0.5.0
##  [79] scales_1.2.1        gtools_3.9.4         xtable_1.8-4
##  [82] glue_1.6.2          emmeans_1.8.7        tools_4.3.1
##  [85] shinystan_2.6.0     colourpicker_1.3.0   mvtnorm_1.2-2
##  [88] grid_4.3.1          crosstalk_1.2.0      colorspace_2.1-0
##  [91] nlme_3.1-162        cli_3.6.1            svUnit_1.0.6
##  [94] fansi_1.0.4         Brobdingnag_1.2-9    gtable_0.3.4
##  [97] digest_0.6.33       htmlwidgets_1.6.2    farver_2.1.1
## [100] htmltools_0.5.6     lifecycle_1.0.3      mime_0.12
## [103] bit64_4.0.5         shinythemes_1.2.0
```

Chapter 5

Canonical Variate Analysis

In the analysis of sensory data, many of the steps are concerned with "dimensionality reduction": finding optimal, low-dimensional representations of sensory data, which is typically highly multivariate. The definition of "optimal" will vary, depending on the particular method chosen. In this section, we're going to focus on **Canonical Variate Analysis (CVA)**, which HGH tends to prefer for sensory data over the more common Principal Components Analysis (we'll get to that, too!).

First, we'll start with loading our data and the relevant libraries. Note that here we will be loading a new library: `candisc`. This is the library that HGH uses in the original **R Opus**, and as far as I can tell it remains the best way to conduct CVA. A good alternative would be Beaton & Abdi's `MExPosition`, but that package appears to be defunct as of the time of this writing (late 2022), and uses a closely related approach called Barycentric Discriminant Analysis instead of the typical CVA.

```
# Attempted fix for the rgl error documented here: https://stackoverflow.com/a/66127391/2554330

#options(rgl.useNULL = TRUE)
library(rgl)

library(tidyverse)
library(candisc) # this is new
library(here)

descriptive_data <- read_csv(here("data/torriDAFinal.csv")) %>%
  mutate_at(.vars = vars(1:3), ~as.factor(.))
```

5.1 What is CVA?

Canonical Variate Analysis is also often called (Linear) Discriminant Analysis. It is closely related to Canonical Correlation Analysis, and is even sometimes called Fisher's Linear Descriminant Analysis (phew). The basic goal of this analysis is to find linear combinations of variables that best separate groups of observations. In this way, we could say that CVA/LDA is a "supervised" learning method, in that we need to provide a vector of group IDs for the observations (rows) in order to allow the algorithm to find a combinaton of variables to separate them.

CVA builds on the same matrix-math calculations that underly MANOVA, and so, according to Rencher **CITE Rencher 2012**, we can think of CVA as the multivariate equivalent of univariate post-hoc testing, like Tukey's HSD. In post-hoc testing, we calculate various intervals around group means in order to see if we can separate observations from multiple groups according to their group IDs. We do much the same with CVA, except that we use group mean *vectors*.

Recall that we had a significant 3-way MANOVA for our data:

```
manova_res <-
  manova(as.matrix(descriptive_data[, 4:23]) ~ (ProductName + NJ + NR)^2,
         data = descriptive_data)

summary(manova_res, test = "W")
```

```
##                  Df   Wilks approx F num Df den Df    Pr(>F)
## ProductName       7 0.04243   4.7575    140 1093.6 < 2.2e-16 ***
## NJ               13 0.00001  12.5493    260 1819.7 < 2.2e-16 ***
## NR                2 0.64784   1.9756     40  326.0 0.0007056 ***
## ProductName:NJ   91 0.00000   1.7583   1820 3331.1 < 2.2e-16 ***
## ProductName:NR   14 0.22673   0.9414    280 1916.9 0.7387371
## NJ:NR            26 0.02215   1.3926    520 2672.6 1.873e-07 ***
## Residuals       182
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We are interested in building on our intuition about Mahalanobis distances, etc, and seeing if we can use, in our case, `ProductName` to separate our observations in multidimensional descriptor space. Specifically, we have a 20-dimensional data set:

```
descriptive_data %>%
  glimpse()
```

```
## Rows: 336
## Columns: 23
## $ NJ              <fct> 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1400, ~
## $ ProductName     <fct> C_MERLOT, C_SYRAH, C_ZINFANDEL, C_REFOSCO, I_MERLOT, I~
## $ NR              <fct> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, ~
## $ Red_berry       <dbl> 5.1, 5.6, 4.9, 5.0, 3.3, 5.7, 2.9, 3.2, 0.1, 1.6, 4.5,~
## $ Dark_berry      <dbl> 5.8, 1.9, 2.6, 1.9, 7.2, 3.6, 5.1, 6.0, 0.1, 0.7, 2.9,~
## $ Jam             <dbl> 2.1, 3.9, 1.4, 7.8, 0.5, 8.7, 8.7, 4.0, 0.2, 0.0, 0.3,~
## $ Dried_fruit     <dbl> 4.7, 1.2, 5.9, 0.6, 5.8, 1.9, 0.4, 0.7, 2.9, 6.4, 2.4,~
## $ Artificial_frui <dbl> 1.0, 7.9, 0.8, 6.6, 0.7, 7.4, 6.2, 4.1, 0.1, 0.1, 0.1,~
## $ Chocolate       <dbl> 2.9, 1.0, 2.0, 6.4, 2.1, 3.3, 3.4, 3.6, 0.2, 1.0, 0.2,~
## $ Vanilla         <dbl> 5.0, 8.3, 2.7, 5.5, 1.3, 6.9, 8.1, 4.8, 2.0, 0.8, 1.9,~
## $ Oak             <dbl> 5.0, 2.3, 5.6, 3.6, 2.1, 1.5, 1.8, 2.6, 3.0, 5.4, 6.1,~
## $ Burned          <dbl> 1.4, 1.8, 1.9, 3.2, 5.6, 0.2, 0.4, 4.7, 7.5, 5.1, 0.3,~
## $ Leather         <dbl> 2.3, 3.5, 4.3, 0.3, 6.5, 1.5, 4.1, 6.5, 0.7, 0.8, 0.2,~
## $ Earthy          <dbl> 0.6, 1.0, 0.6, 0.2, 4.7, 0.3, 0.5, 1.9, 0.7, 3.0, 1.3,~
## $ Spicy           <dbl> 3.2, 0.7, 1.4, 2.9, 0.7, 3.1, 0.7, 1.4, 0.3, 3.2, 3.1,~
## $ Pepper          <dbl> 5.4, 3.0, 4.1, 0.9, 2.8, 1.6, 3.6, 4.5, 0.1, 2.0, 0.9,~
## $ Grassy          <dbl> 2.1, 0.6, 3.6, 1.8, 3.8, 0.9, 2.3, 0.8, 0.1, 1.3, 0.4,~
## $ Medicinal       <dbl> 0.4, 2.2, 1.7, 0.2, 2.6, 0.5, 0.2, 3.8, 0.1, 2.1, 0.1,~
## $ `Band-aid`      <dbl> 0.4, 0.4, 0.1, 0.2, 5.1, 1.2, 0.2, 6.2, 0.1, 1.1, 0.1,~
## $ Sour            <dbl> 5.0, 9.7, 7.8, 8.3, 7.6, 7.2, 5.9, 6.3, 5.7, 6.4, 5.4,~
## $ Bitter          <dbl> 5.9, 5.2, 3.5, 3.0, 1.9, 9.8, 2.9, 0.2, 0.6, 2.9, 0.1,~
## $ Alcohol         <dbl> 9.0, 7.2, 4.7, 8.9, 2.8, 8.7, 1.6, 7.0, 1.6, 5.4, 4.9,~
## $ Astringent      <dbl> 8.7, 8.3, 5.0, 7.8, 5.9, 8.0, 2.6, 4.2, 5.5, 5.1, 5.9,~
```

We can't see in 20 dimensions! So how does CVA help us not only separate our observations, but visualize them? It turns out, without digging too far into the matrix algebra, that the following are properties of CVA:

1. For k groups and p variables, CVA will find $min(k-1, p-1)$ linear functions of the variables that best separate the groups.

2. The solution turns out to be an eigendecomposition of the MANOVA calculations.
3. Because it is an eigendecomposition, we can rank the functions by their associated eigenvalues, and say that the function associated with the largest eigenvalue best separates the groups.
4. We can treat the functions, which map observation vectors to single values using a linear combination of variables, as cartesian axes to plot variables against each other.
5. **NB:** In fact, unlike PCA and some other approaches, CVA does not in general produce orthogonal linear functions, and so we are technically distorting the space a bit by plotting canonical variate function 1 as the x-axis, canonical variate function 2 as the y-axis, and so on, but in prac-

tice the distortion is reported to be minor [Rencher, 2002, @heymannSensory2017].

OK, back outside of the math digression, let me resummarize: we can use CVA to find combinations of the original variables that best separate our observations into their assigned group IDs. Because this is often the exact task we want to accomplish with a Descriptive Analysis, CVA is a great tool to use to describe variation in samples' descriptive profiles according to the important categorical variable(s) that identify them.

5.2 Application of CVA

In the original **R Opus**, HGH recommends using CVA only on one-way MANOVA, and we ran a 3-way MANOVA. This makes sense if you think in analogy to univariate post-hoc tests: these are sometimes called "marginal means" tests because we marginalize across the other factors to collapse our data to only a single categorical predictor in which we're interested. However, we are able to run post-hoc tests on multi-way ANOVA, and we can do the same thing in MANOVA/CVA land. Let's compare the approach for the two.

First, let's run a one-way MANOVA with just ProductName as the predictor and then send it off to CVA (using the `candisc()` function, whose name comes from "canonical discriminant analysis"):

```
manova_res_oneway <-
  manova(as.matrix(descriptive_data[, 4:23]) ~ ProductName,
         data = descriptive_data)

cva_res_oneway <- candisc(manova_res_oneway)
cva_res_oneway
```

```
##
## Canonical Discriminant Analysis for ProductName:
##
##     CanRsq Eigenvalue Difference Percent Cumulative
## 1 0.381405   0.616566    0.22367 44.2257     44.226
## 2 0.282071   0.392895    0.22367 28.1820     72.408
## 3 0.121339   0.138095    0.22367  9.9054     82.313
## 4 0.109005   0.122340    0.22367  8.7754     91.089
## 5 0.059767   0.063566    0.22367  4.5595     95.648
## 6 0.041480   0.043275    0.22367  3.1041     98.752
## 7 0.017100   0.017397    0.22367  1.2479    100.000
##
## Test of H0: The canonical correlations in the
```

```
## current row and all that follow are zero
##
##   LR test stat approx F numDF   denDF   Pr(> F)
## 1      0.30799  2.85884   140 2063.54 < 2.2e-16 ***
## 2      0.49788  2.02435   114 1791.99 3.834e-09 ***
## 3      0.69349  1.31764    90 1513.26    0.0276 *
## 4      0.78926  1.12136    68 1226.69    0.2378
## 5      0.88582  0.80761    48  931.73    0.8224
## 6      0.94213  0.63333    30  628.00    0.9372
## 7      0.98290  0.39144    14  315.00    0.9769
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The likelihood-ratio tests given in the summary from the `cva_res_oneway` object gives tests derived from the F-statistic approximation to the Wilk's Λ results from the original MANOVA (another reason we prefer to use that statistic).

However, compare these results to those from our original MANOVA:

```
cva_res <- candisc(manova_res, term = "ProductName")
cva_res
```

```
##
## Canonical Discriminant Analysis for ProductName:
##
##     CanRsq Eigenvalue Difference  Percent Cumulative
## 1 0.699341   2.326032     1.0647 47.21757     47.218
## 2 0.557787   1.261351     1.0647 25.60496     72.823
## 3 0.340601   0.516533     1.0647 10.48543     83.308
## 4 0.281808   0.392386     1.0647  7.96528     91.273
## 5 0.191329   0.236597     1.0647  4.80282     96.076
## 6 0.128956   0.148048     1.0647  3.00531     99.081
## 7 0.043294   0.045254     1.0647  0.91863    100.000
##
## Test of H0: The canonical correlations in the
## current row and all that follow are zero
##
##   LR test stat approx F numDF   denDF   Pr(> F)
## 1      0.04243   8.9770   140 2063.54 < 2.2e-16 ***
## 2      0.14113   6.3682   114 1791.99 < 2.2e-16 ***
## 3      0.31914   4.4633    90 1513.26 < 2.2e-16 ***
## 4      0.48398   3.6644    68 1226.69 < 2.2e-16 ***
## 5      0.67389   2.7546    48  931.73 5.162e-09 ***
## 6      0.83333   1.9980    30  628.00  0.001389 **
## 7      0.95671   1.0182    14  315.00  0.434537
## ---
```

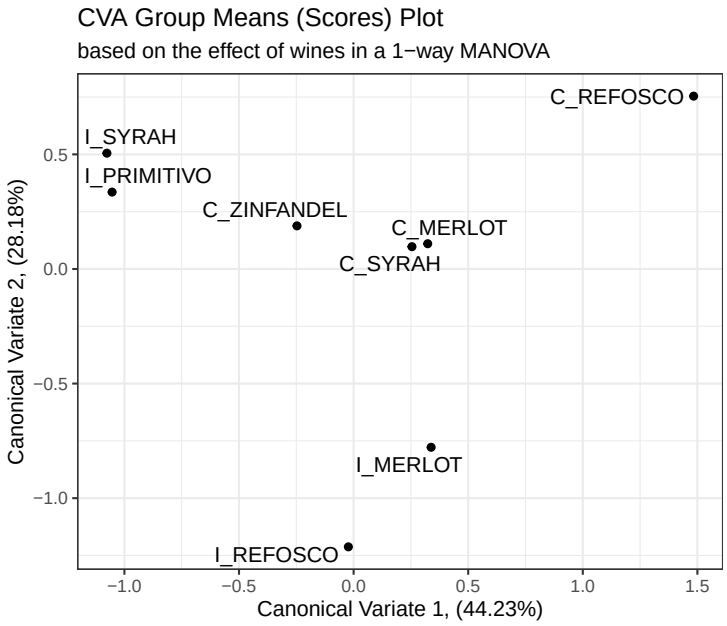
```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

While the exact numbers are different (and we see more significant dimensions according to the LR test), the actual amount of variation explained by the canonical variates is very close. Furthermore, we can visualize the results to contrast them.

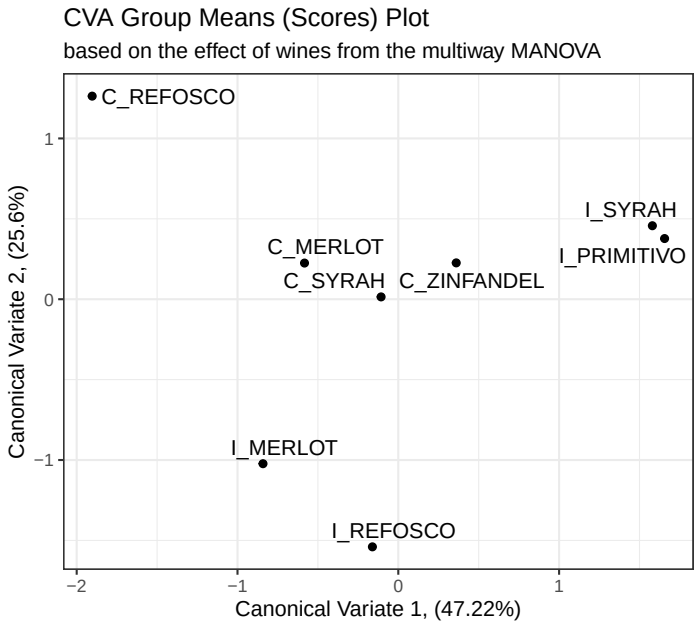
5.2.1 Basic visualization of CVA

We'll start by examining the placement of the group means in the resultant "CVA-space", which is just the scores (linear combinations of original variables) for the first and second canonical variate functions for each product mean vector.

```
# Here is the plot of canonical variates 1 and 2 from the one-way approach
scores_p_1 <-
  cva_res_oneway$means %>%
  as_tibble(rownames = "product") %>%
  ggplot(aes(x = Can1, y = Can2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = product)) +
  theme_bw() +
  labs(x = paste0("Canonical Variate 1, (", round(cva_res_oneway$pct[1], 2), "%)"),
       y = paste0("Canonical Variate 2, (", round(cva_res_oneway$pct[2], 2), "%)"),
       title = "CVA Group Means (Scores) Plot",
       subtitle = "based on the effect of wines in a 1-way MANOVA") +
  coord_equal()

scores_p_1
```




```
# And here is the plot of the canonical variates 1 and 2 from the multiway MANOVA
scores_p_2 <-
  cva_res$means %>%
  as_tibble(rownames = "product") %>%
  ggplot(aes(x = Can1, y = Can2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = product)) +
  theme_bw() +
  labs(x = paste0("Canonical Variate 1, (", round(cva_res$pct[1], 2), "%)"),
       y = paste0("Canonical Variate 2, (", round(cva_res$pct[2], 2), "%)"),
       title = "CVA Group Means (Scores) Plot",
       subtitle = "based on the effect of wines from the multiway MANOVA") +
  coord_equal()

scores_p_2
```



Besides a reversal of the x-axis (a trivial reflection which does not change meaning), the product mean separation appears to be identical in these two solutions, and the proportion of explained variation remains almost identical. We'll keep up this comparison as we investigate the rest of the solution.

Next, we want to examine how the original variables are related to the resulting "CVA-space". This is usually visualized as a vector plot, so we're going to follow convention. This will make our `ggplot2` calls a little messsier, but I think it is worth it.

In the `candisc()` output, the `$coeffs.raw` list holds the linear combinations of variables that contribute to each dimension:

```
cva_res_oneway$coeffs.raw %>%
  round(2) %>%
  as_tibble(rownames = "descriptor")
```

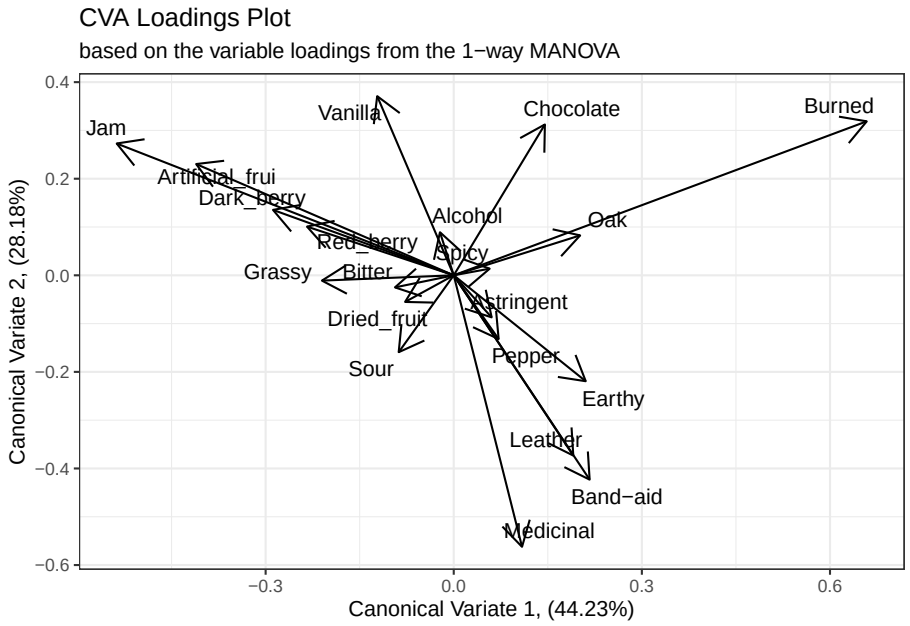
```
## # A tibble: 20 x 8
##    descriptor       Can1  Can2  Can3  Can4  Can5  Can6  Can7
##    <chr>           <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Red_berry        0.01  0.03 -0.04 -0.1  -0.21 -0.06 -0.18
##  2 Dark_berry      -0.12  0.02  0.16  0.05  0.23 -0.03  0.1
##  3 Jam             -0.22  0.09 -0.05 -0.11 -0.15 -0.03  0.11
##  4 Dried_fruit     -0.06  0.03 -0.1   0.2   0.17  0.03 -0.14
##  5 Artificial_frui -0.18  0.04  0.28 -0.02  0.09 -0.1   0.18
##  6 Chocolate        0.3   0.26  0.1  -0.11 -0.1   0.13  0.07
```

```
##  7 Vanilla         -0.03  0.05 -0.18 -0.07 -0.08 -0.05 -0.17
##  8 Oak              0.04  0.1  -0.26  0.15  0    -0.25  0.2
##  9 Burned           0.32  0.26  0.24 -0.06  0    -0.05  0.05
## 10 Leather         -0.02 -0.22 -0.02 -0.12  0.11 -0.32 -0.07
## 11 Earthy           0.1   0    -0.12 -0.17 -0.01  0.47  0.07
## 12 Spicy            0.02  0.09  0.07  0.06  0.1  -0.22 -0.3
## 13 Pepper           0.08 -0.09 -0.17  0.27 -0.11  0.12  0.23
## 14 Grassy          -0.31  0     0.16  0.18 -0.09  0.22 -0.16
## 15 Medicinal        0.02 -0.29 -0.02 -0.25 -0.02  0.04  0.15
## 16 Band-aid         0.07 -0.13  0.22  0.05 -0.1  -0.09 -0.16
## 17 Sour            -0.08 -0.07  0.11 -0.05 -0.06 -0.04  0.14
## 18 Bitter           0.02  0.13  0.06  0.06  0.09 -0.04 -0.23
## 19 Alcohol          0.02  0.06 -0.22 -0.28  0.23  0.16 -0.1
## 20 Astringent       0.1  -0.13  0.11  0.16 -0.06 -0.02  0.02
```

This tell us that to get an observation's (call it i) score on Canonical Variate 1, we'd take the following linear sum: $Can1_i = -0.01 * Red\_berry_i + -0.12 * Dark\_berry_i + ... + -0.22 * Astringent_i$. We do this for every observation to find their score in the space.

To visualize this, rather than plotting the raw coefficients, we typically plot the correlations between the raw coefficients and the original variables, which are stored in the `$structure` list. We can consider the "loadings" or "structure coefficients" themselves as coordinates in the same CVA space, which gives us an idea of how each variable contributes to the dimensions.

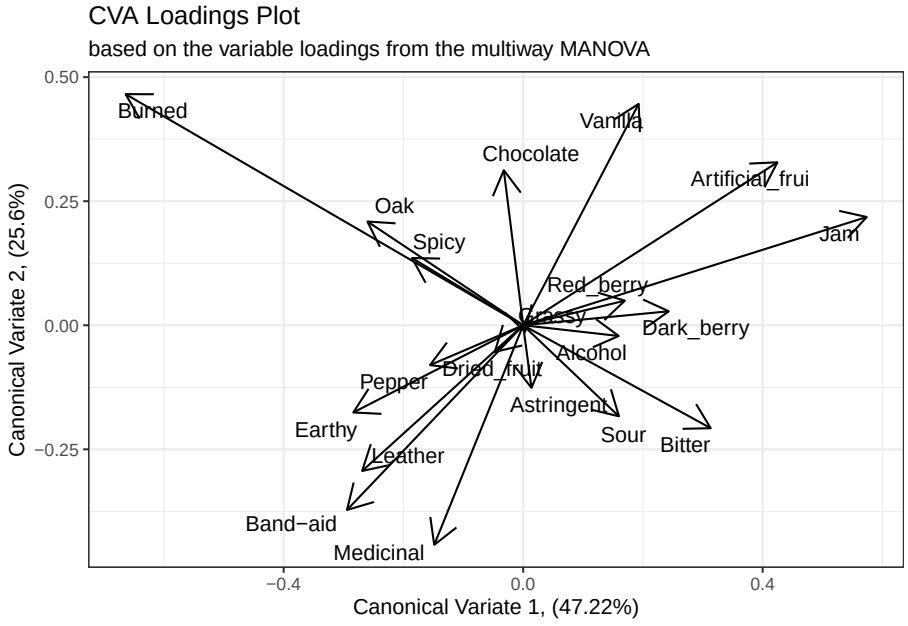
```
cva_res_oneway$structure %>%
  as_tibble(rownames = "descriptor") %>%
  ggplot(aes(x = Can1, y = Can2)) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.2, units = "in"), ends = "first")) +
  ggrepel::geom_text_repel(aes(label = descriptor)) +
  labs(x = paste0("Canonical Variate 1, (", round(cva_res_oneway$pct[1], 2), "%)"),
       y = paste0("Canonical Variate 2, (", round(cva_res_oneway$pct[2], 2), "%)"),
       title = "CVA Loadings Plot",
       subtitle = "based on the variable loadings from the 1-way MANOVA") +
  theme_bw()
```



We can see from this plot that the first canonical variate has large positive contributions from "Burned" characteristics and smaller contributions from "Oak", and strong negative contributions from fruit-related terms like "Jam", "Artificial_frui [sic]", etc. The second canonical variate has large positive contributions from both "Vanilla" and "Chocolate", and negative contributions from flaw/complex terms like "Medicinal", "Band-aid", etc.

If we examine the results from the multiway MANOVA we'll see similar results, with the exception of a reversed x-axis (first canonical variate).

```
cva_res$structure %>%
  as_tibble(rownames = "descriptor") %>%
  ggplot(aes(x = Can1, y = Can2)) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.2, units = "in"), ends = "first")) +
  ggrepel::geom_text_repel(aes(label = descriptor)) +
  labs(x = paste0("Canonical Variate 1, (", round(cva_res$pct[1], 2), "%)"),
       y = paste0("Canonical Variate 2, (", round(cva_res$pct[2], 2), "%)"),
       title = "CVA Loadings Plot",
       subtitle = "based on the variable loadings from the multiway MANOVA") +
  theme_bw()
```



5.2.2 An aside on exporting R plots

In the original **R Opus**, HGH used RStudio's "Export" pane to get files from the `Plots` pane to a format that could be edited in PowerPoint. I tend to discourage this approach, because it produces overall lower resolution results. In addition, as you will have noticed I use `ggplot2` almost exclusively for visualizations, and the `ggplot2::ggsave()` function gives easy functional access to `R`'s somewhat byzantine `device` interface, making saving images directly much more feasible.

To use `ggsave()`, you can either save an executed `ggplot2` plot as an object and pass it to the function, or let the default `ggsave(plot = last_plot())` behavior grab the current visual you're looking at in the `Plots` pane. Either way, you then need to specify a destination file *with extension* (e.g., `/path/to/your/image.png`), which `ggsave()` will parse to get the correct output format. I tend to output as `.png`, but you might also want to output `.jpg` or `.tif`, depending on the usecase. Use `?ggsave` for more details.

You will also need to specify arguments for `height =` and `width =`, as well as a `units =`, which takes a character string indicating pixels, inches, centimeters, etc. Finally, the `scale =` argument is equivalent to the base `R` `cex =` arguments: it dictates the degree to which the plot will be "zoomed". `scale =` values that are greather than 1 will cause the plot to "zoom out" (elements in the plot will appear smaller), and values less than 1 will cause a "zoom in" effect.

An example of this workflow, the following (not executed) code will take the

last plot of the CVA loadings for the multiway ANOVA, save it as an object called `p`, and then save that plot to my home folder:

```
p <-
  cva_res$structure %>%
  as_tibble(rownames = "descriptor") %>%
  ggplot(aes(x = Can1, y = Can2)) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.2, units = "in"), ends = "first")) +
  ggrepel::geom_text_repel(aes(label = descriptor)) +
  labs(x = paste0("Canonical Variate 1, (", round(cva_res$pct[1], 2), "%)"),
       y = paste0("Canonical Variate 2, (", round(cva_res$pct[2], 2), "%)"),
       title = "CVA Loadings Plot",
       subtitle = "based on the variable loadings from the multiway MANOVA") +
  theme_bw()

ggsave(filename = "~/my_new_plot.png",
       plot = p, height = 6, width = 8,
       units = "in", scale = 1.2)
```

5.3 Plotting uncertainty in CVA

One of the advantages of CVA is that it is based on raw data, which PCA is *typically* not [an exception is so-called Tucker-1 PCA; Dettmar et al. [2020]]. So far we've only visualized the *group* means for the wines in our CVA plots, but we can investigate the `$scores` data frame in the CVA results to see the raw observation scores, and use them to plot confidence ellipses.

```
cva_res_oneway$scores %>%
  as_tibble()
```

```
## # A tibble: 336 x 8
##    ProductName   Can1   Can2   Can3   Can4     Can5   Can6    Can7
##    <fct>        <dbl>  <dbl>  <dbl>  <dbl>    <dbl>  <dbl>   <dbl>
##  1 C_MERLOT     0.378  1.23  -2.16   1.33   1.25    -0.177 -1.38
##  2 C_SYRAH     -1.43  -0.233  0.130 -1.97  -0.577   -1.42   0.478
##  3 C_ZINFANDEL -0.499 -0.177 -1.58   1.38  -0.00298 -0.807 -0.0839
##  4 C_REFOSCO   -0.352  3.45   0.848 -2.56  -1.22    -0.368  0.740
##  5 I_MERLOT     0.882 -1.40   2.42   0.508  0.715    0.535  0.153
##  6 I_SYRAH     -2.34   2.32   0.679 -1.85   0.0947  -1.17  -1.81
##  7 I_PRIMITIVO -3.27   1.06   0.445 -0.766 -1.51    -1.44   1.60
##  8 I_REFOSCO    0.881 -0.986  1.14  -2.60  -0.0514  -1.25   1.56
##  9 C_MERLOT     2.39   1.16   0.794  0.678 -0.340   -0.469  1.06
```

```
## 10 C_SYRAH      1.95   0.928 -1.09   0.847  1.01     0.644 -0.155
## # i 326 more rows
```

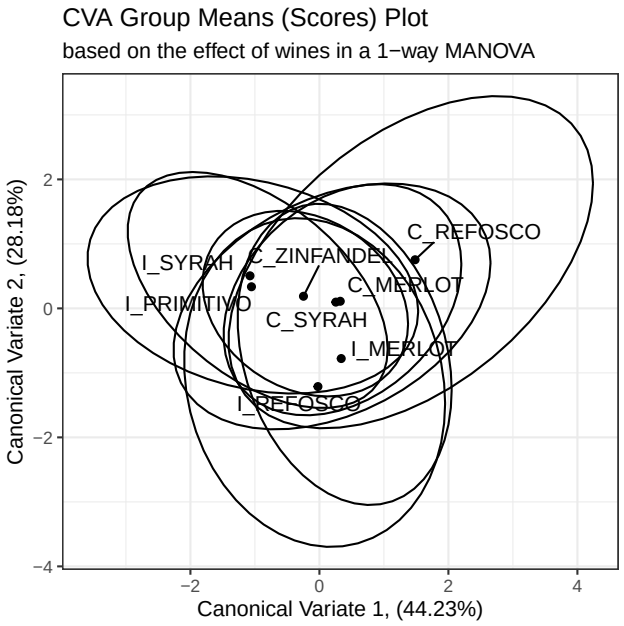
```
cva_res$scores %>%
  as_tibble()
```

```
## # A tibble: 336 x 10
##    ProductName NJ    NR      Can1   Can2   Can3    Can4    Can5   Can6    Can7
##    <fct>       <fct> <fct>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
##  1 C_MERLOT    1331  7     -0.478  1.14  -0.778 -0.633  -2.51   -1.06   2.46
##  2 C_SYRAH     1331  7      3.53   0.810  4.58  -2.30    0.403  -2.94   0.312
##  3 C_ZINFANDEL 1331  7     -0.551  0.137 -0.675  0.0954 -1.37   -2.47   0.864
##  4 C_REFOSCO   1331  7      2.19   5.17   4.06  -1.05    1.40   -0.988  0.209
##  5 I_MERLOT    1331  7     -3.59  -1.79   4.29   0.980  -2.15   -0.613 -0.0964
##  6 I_SYRAH     1331  7      5.89   2.58   4.66  -0.838  -0.0944 -1.30   3.14
##  7 I_PRIMITIVO 1331  7      5.28   2.20   3.59   1.27    2.51   -2.61  -1.12
##  8 I_REFOSCO   1331  7     -1.73  -0.271  5.68  -0.973   1.75   -1.96  -0.896
##  9 C_MERLOT    1400  7     -3.98   2.69  -0.498  0.0528 -0.804  -1.02  -1.57
## 10 C_SYRAH     1400  7     -3.49   1.72  -1.04  -1.39   -2.38   -0.338  0.563
## # i 326 more rows
```

We can add these to our mean-scores plots using the `ggplot2` layering capabilities–another example of using `ggplot2` over base R graphics.

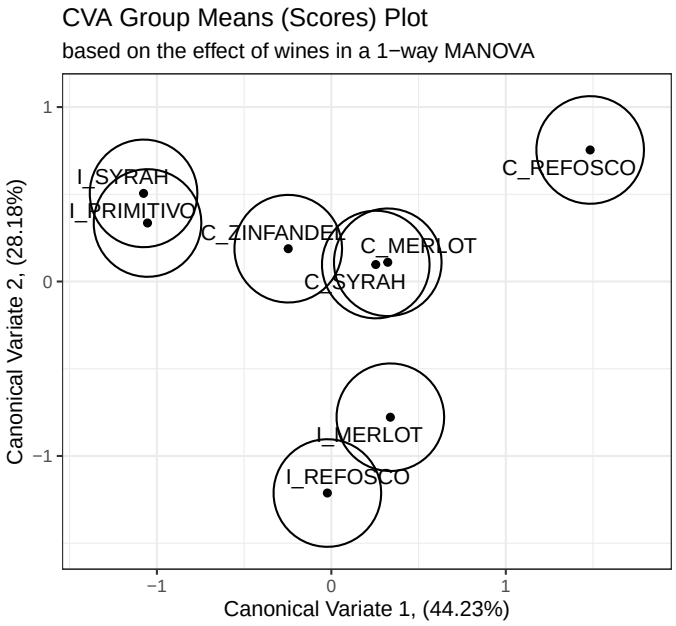
```
scores_p_1 +
  stat_ellipse(data = cva_res_oneway$scores,
               mapping = aes(group = ProductName),
               type = "t") +
  coord_fixed()
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```




```
# This is in some contrast to the original R Opus, in which HGH used 95% confidence ci

scores_p_1 +
  stat_ellipse(data = cva_res_oneway$scores,
               mapping = aes(group = ProductName),
               type = "euclid",
               level = 2 / sqrt(table(cva_res_oneway$scores[,1]))[1]) +
  coord_fixed()
```

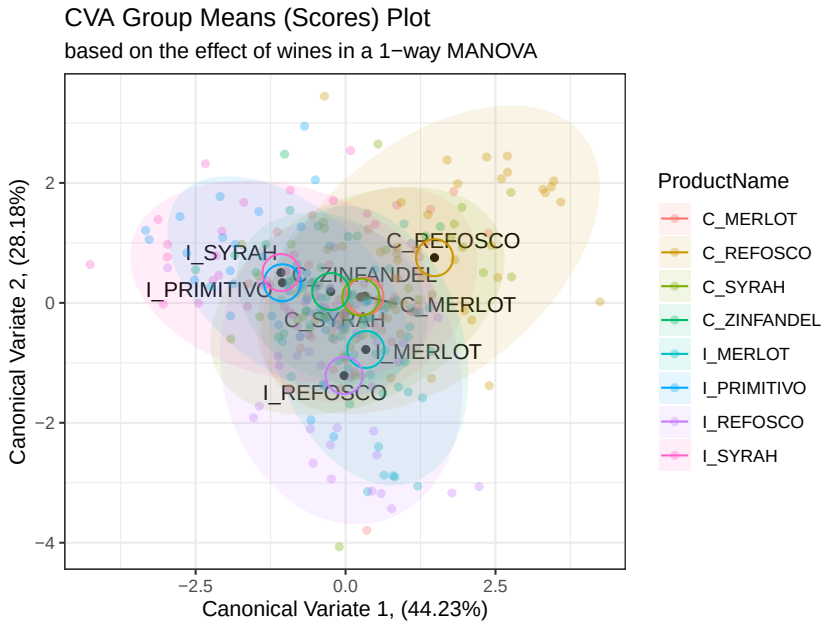
```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```



While I don't have immediate access to the original reference HGH cites, the calculation provided in the original **R Opus** and repeated above draws a circle of confidence based on the number of observations in each group, which to my mind seems likely to underestimate the overlap between samples. On the other hand, my simple confidence ellipses are for the raw observations, rather than the "barycentric" (mean vector) confidence intervals that the method given by HGH hopes to provide. We can look at this by looking at the actual observations plotted in the CVA space.

```
scores_p_1 +
  geom_point(data = cva_res_oneway$scores,
             aes(color = ProductName),
             alpha = 1/3) +
  stat_ellipse(data = cva_res_oneway$scores,
               mapping = aes(fill = ProductName),
               geom = "polygon",
               alpha = 0.1) +
  stat_ellipse(data = cva_res_oneway$scores,
               mapping = aes(group = ProductName, color = ProductName),
               type = "euclid",
               level = 2 / sqrt(table(cva_res_oneway$scores[,1]))[1]) +
  coord_fixed()
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```



When we plot the actual observations, it is apparent that the proposed confidence circles don't do a good job of capturing the observed variability in the data. They are probably overgenerous in their estimation of the separability of the individual observations, as from what I can tell the radius of the circles is simply a function of the number of observations. This would decrease as the number of observations go up, which makes sense as more observations will give more stable mean vectors, but it doesn't really take advantage of the raw data. On the other hand, the confidence intervals based on raw observations (which are no longer barycentric) don't tell us much about mean separation–they are more akin to predicting the orientation of other new observations for each wine in the CVA space.

In the original **R Opus**, HGH provides a function from Helene Hopfer, Peter Buffon, and Vince Buffalo to calculate barycentric confidence ellipses based on, I believe, a similar method as that given in Peltier et al. [2015]. These functions rely on assumptions of bivariate normality for mean scores on the represented CVA axes, which is reasonable, but I think the function is quite opaque. As noted in Peltier et al. [2015], a resampling approach could be used, so we're going to take this approach to illustrate such an approach. Resampling, which involves perturbing the original data randomly and then examining its stability, is a nonparametric approach that is useful because it makes very few assumptions about either our data or analysis.

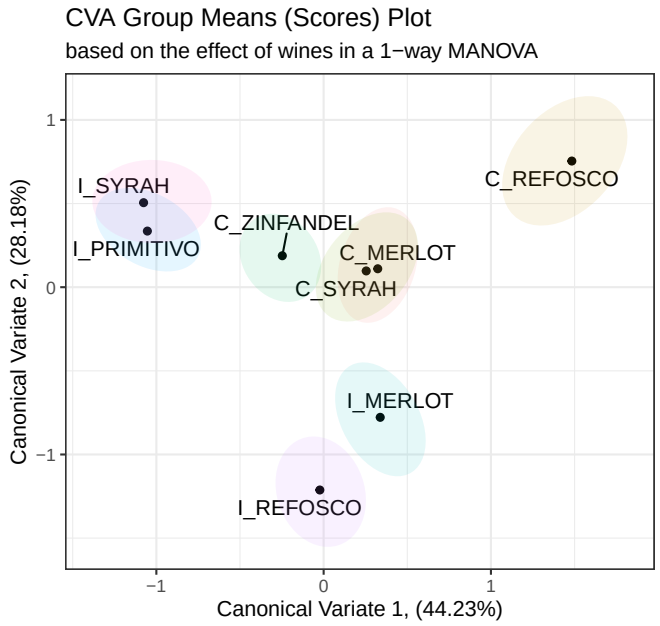
```
# This is still going to be a bit complicated, but I will walk through the steps
# in my thinking.
```

```
# First, we will write two functions that just calculate the canonical variate
# scores for the means of our data.  These are based on the equations from
# Rencher 2002.
get_cva_1 <- function(means){

  cva_res_oneway$coeffs.raw[, 1] %*% means

}

get_cva_2 <- function(means){

  cva_res_oneway$coeffs.raw[, 2] %*% means

}

# Then, we write a convenience function that will resample our data with
# replacement, calculate centered means for our samples, and then project them
# into our original CVA space using the convenience functions we wrote above.
get_bootstrapped_means <- function(){

  descriptive_data %>%
  select(-NJ, -NR) %>%
  group_by(ProductName) %>%
    # Here we resample each wine - we draw 42 new observations for each wine
    # with replacement
  slice_sample(prop = 1, replace = TRUE) %>%
    # We calculate the mean for the newly drawn samples
  summarize_if(is.numeric, ~mean(.)) %>%
    # And then we center the means (by subtracting the column means)
  mutate_if(is.numeric, ~. - mean(.)) %>%
    # Finally, we apply our projection functions (the linear combinations from
    # our CVA) to project the bootstrapped means into our new space
  nest(data = -ProductName) %>%
  mutate(means = map(data, ~as.matrix(.x) %>% t()),
         Can1 = map_dbl(means, ~get_cva_1(.x)),
         Can2 = map_dbl(means, ~get_cva_2(.x)),
         ) %>%
    select(ProductName, Can1, Can2)

}

# For example, here is the result of our function
get_bootstrapped_means()
```

```
## # A tibble: 8 x 3
##   ProductName    Can1    Can2
##   <fct>         <dbl>   <dbl>
## 1 C_MERLOT     0.299   0.0649
## 2 C_REFOSCO    1.34    0.983
## 3 C_SYRAH      0.242   0.0848
## 4 C_ZINFANDEL -0.0792  0.118
## 5 I_MERLOT     0.446  -0.847
## 6 I_PRIMITIVO -0.979   0.222
## 7 I_REFOSCO   -0.0958 -1.14
## 8 I_SYRAH     -1.17    0.515
```

```
# In order to generate a set of resampled confidence ellipses for our results,
# we'll run this 100 times for speed (for accuracy, 1000x would be better) and use
# stat_ellipse() to draw those.

bootstrapped_cva_means <-
  tibble(boot_id = 1:100) %>%
  mutate(bootstrapped_data = map(boot_id, ~get_bootstrapped_means())) %>%
  unnest(bootstrapped_data)

scores_p_1 +
  stat_ellipse(data = bootstrapped_cva_means,
               aes(fill = ProductName),
               geom = "polygon",
               alpha = 0.1) +
  coord_fixed() +
  theme(legend.position = "none")
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```



While this requires a bit of computation time (around 5 seconds on my 2017 iMac, writing in 2022), this is still pretty tractable. It would take about 1 minute to do a "publishable" bootstrap of 1000 samples, which is still fine. As we'll see, this resampling approach will generalize easily to other needs.

This is a more conservative result than that given by the bivariate normal ellipses in the original **R Opus**, although the actual conclusions of pairwise sample discrimination are identical.

5.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] here_1.0.1      candisc_0.8-6   heplots_1.6.0   broom_1.0.5
##  [5] car_3.1-2       carData_3.0-5   lubridate_1.9.2 forcats_1.0.0
##  [9] stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1     readr_2.1.4
## [13] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3   tidyverse_2.0.0
## [17] rgl_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.3        generics_0.1.3    stringi_1.7.12    hms_1.1.3
##  [5] digest_0.6.33     magrittr_2.0.3    evaluate_0.21     grid_4.3.1
##  [9] timechange_0.2.0  bookdown_0.37     fastmap_1.1.1     rprojroot_2.0.3
## [13] jsonlite_1.8.7    ggrepel_0.9.3     backports_1.4.1   fansi_1.0.4
## [17] scales_1.2.1      abind_1.4-5       cli_3.6.1         crayon_1.5.2
## [21] rlang_1.1.1       bit64_4.0.5       munsell_0.5.0     base64enc_0.1-3
## [25] withr_2.5.0       yaml_2.3.7        parallel_4.3.1    tools_4.3.1
## [29] tzdb_0.4.0        colorspace_2.1-0  vctrs_0.6.3       R6_2.5.1
## [33] lifecycle_1.0.3   bit_4.0.5         htmlwidgets_1.6.2 vroom_1.6.3
## [37] MASS_7.3-60       pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.4
## [41] Rcpp_1.0.11       glue_1.6.2        highr_0.10        xfun_0.39
## [45] tidyselect_1.2.0  rstudioapi_0.15.0 knitr_1.43        farver_2.1.1
## [49] htmltools_0.5.6   labeling_0.4.3    rmarkdown_2.23    compiler_4.3.1
```

Chapter 6

Principal Components Analysis (PCA)

While HGH is a strong proponent of Canonical Variate Analysis (CVA), it is hard to argue that Principal Components Analysis is not the "workhorse" tool for analysis of multivariate sensory datasets. In fact, PCA is probably the most common tool for multivariate analysis, hard stop. This is because it is so closely related to eigendecomposition and Singular Value Decomposition (SVD).

We'll start with setting up our data as usual. This time, we'll use the `FactoMineR` package for PCA because of its comprehensive outputs. You could also use the base `R` function `princomp()`, but it provides fewer sensory-specific options.

```
library(tidyverse)
library(FactoMineR) # this is new
library(here)

descriptive_data <- read_csv(here("data/torriDAFinal.csv")) %>%
  mutate_at(.vars = vars(1:3), ~as.factor(.))
```

6.1 What does PCA do?

As a reminder, in a dataset with multiple variables measured on the same observations, PCA finds *linear combinations* of the original variables that maximally explain the covariance among the original variables. Thus, the criterion maximized by PCA is explanation of (co)variance. We often describe this as "amount of variation explained" by the subsequent, new linear combinations. In highly

correlated data, which is usually the case in sensory data, PCA is very effective at finding a few new linear combinations of the original variables that explain most of the observed variance.

Typically, PCA is conducted on mean vectors.

Speaking of CVA, recall that CVA finds linear combinations of variables that best explain *mean-vector* separation in the original data. Thus, while CVA operates on raw data, it is looking to separate the mean-vectors or "barycenters" of the data. PCA "knows" less information about the data: in fact, PCA is an "unsupervised" learning approach, in comparison to the "supervision" provided by the group IDs in CVA. PCA is also generally lax about assumptions: the main assumption is that the observed variables are continuous. If they are categorical, it is probably more appropriate to use (multiple) Correspondence Analysis.

PCA is equivalent to the eigendecomposition of the covariance matrix of our original variables. If we standardize our variables (convert them to z-scores, or equivalently center our variables and standardize them to have unit variance) we will be conducting eigendecomposition on a correlation matrix. Thus, you will often see discussion of "correlation" vs "covariance" PCA. There are times in which one or the other is most appropriate for sensory results, but here we'll focus on correlation PCA.

A final note on PCA is that, because of the nature of eigendecomposition, the components are mutually orthogonal: they are uncorrelated and, geometrically, form right angles in the multivariate space.

For more details on PCA, I strongly recommend Hervé Abdi's excellent and detailed chapter on the topic, available on his website (I think A77 is the entry in his pub list).

6.2 Let's do PCA!

HGH starts the original **R Opus** with a function to take column and group means of the data in `descriptive_data` that she calls `mtable()`, for "means table". Luckily for us, this functionality is a basic part of the `tidyverse`, and we've already used this approach in previous sections. So let's generate a means table using `tidyverse`:

```
descriptive_means <-
  descriptive_data %>%
  # What defines the group for which we want means?
  group_by(ProductName) %>%
  # Then we take the group means for every numeric variable (ignore NR, NJ)
  summarize_if(is.numeric, ~mean(.))
```

```
descriptive_means
```

```
## # A tibble: 8 x 21
##   ProductName Red_berry Dark_berry   Jam Dried_fruit Artificial_frui Chocolate
##   <fct>           <dbl>      <dbl> <dbl>       <dbl>           <dbl>     <dbl>
## 1 C_MERLOT         2.46       3.05 1.37         1.86           0.776     1.19
## 2 C_REFOSCO        2.47       2.46 1.03         1.42           0.924     2.00
## 3 C_SYRAH          2.46       2.93 1.75         1.68           0.883     1.42
## 4 C_ZINFANDEL      3.08       3.06 1.98         2.06           0.864     0.969
## 5 I_MERLOT         2.79       2.35 0.843        1.85           0.574     0.783
## 6 I_PRIMITIVO      3.85       3.38 3.61         1.44           2.19      1.38
## 7 I_REFOSCO        2.48       3.01 1.54         1.87           1.11      0.810
## 8 I_SYRAH          3.17       4.48 3.10         2.16           2.43      1.20
## # i 14 more variables: Vanilla <dbl>, Oak <dbl>, Burned <dbl>, Leather <dbl>,
## #   Earthy <dbl>, Spicy <dbl>, Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>,
## #   `Band-aid` <dbl>, Sour <dbl>, Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

The `FactoMineR::PCA()` function is great, but it also tries to do way too much. One of its annoying habits is a desire to give you a lot of plots you don't want. So be sure to use the `graph = FALSE` argument so you have more control over plotting. It also uses an older standard which relies on storing observation IDs in `row.names`–this isn't great programming practice and we have to explicitly do so. Following HGH, we are going to conduct a covariance PCA by setting `scale.unit = FALSE`.

```
means_pca <-
  descriptive_means %>%
  column_to_rownames("ProductName") %>%
  PCA(scale.unit = FALSE, graph = FALSE)

means_pca
```

```
## **Results for the Principal Component Analysis (PCA)**
## The analysis was performed on 8 individuals, described by 20 variables
## *The results are available in the following objects:
##
##    name               description
## 1  "$eig"             "eigenvalues"
## 2  "$var"             "results for the variables"
## 3  "$var$coord"       "coord. for the variables"
## 4  "$var$cor"         "correlations variables - dimensions"
## 5  "$var$cos2"        "cos2 for the variables"
## 6  "$var$contrib"     "contributions of the variables"
## 7  "$ind"             "results for the individuals"
```

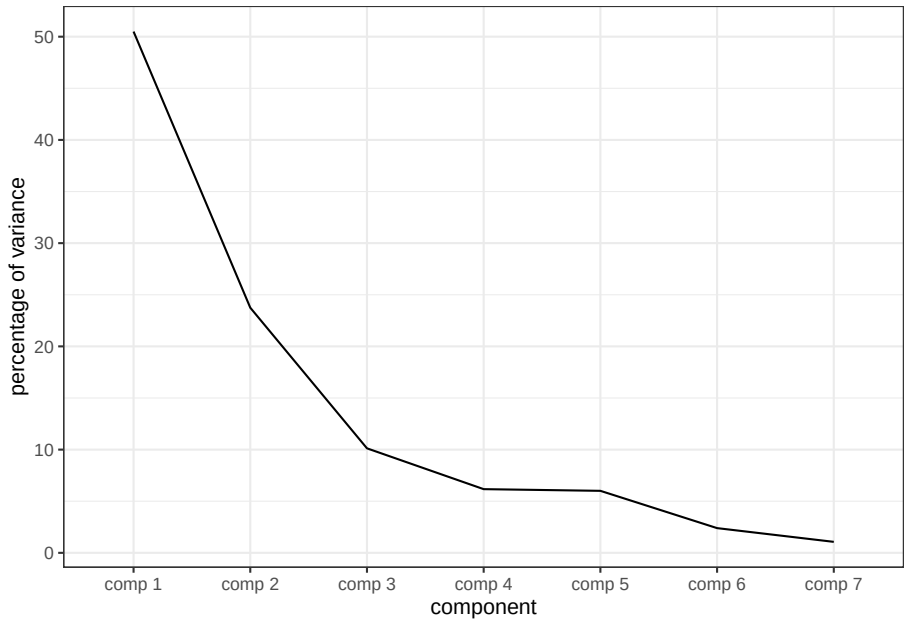
```
## 8  "$ind$coord"       "coord. for the individuals"
## 9  "$ind$cos2"        "cos2 for the individuals"
## 10 "$ind$contrib"     "contributions of the individuals"
## 11 "$call"            "summary statistics"
## 12 "$call$centre"     "mean of the variables"
## 13 "$call$ecart.type" "standard error of the variables"
## 14 "$call$row.w"      "weights for the individuals"
## 15 "$call$col.w"      "weights for the variables"
```

The nice thing about `PCA()` is that it gives a well-structured list of results that we can do a lot with. First off, let's make a quick "scree plot", describing the variance explained by each of the principal components.

```
# Here are the actual numeric results.
means_pca$eig
```

```
##        eigenvalue percentage of variance cumulative percentage of variance
## comp 1 2.65480006              50.496451                           50.49645
## comp 2 1.24849044              23.747301                           74.24375
## comp 3 0.53223903              10.123618                           84.36737
## comp 4 0.32433613               6.169136                           90.53651
## comp 5 0.31577131               6.006227                           96.54273
## comp 6 0.12584413               2.393657                           98.93639
## comp 7 0.05591818               1.063609                          100.00000
```

```
# And now we can plot
means_pca$eig %>%
  # Note that we need to use `rownames=` to capture the rownames from PCA()
  as_tibble(rownames = "component") %>%
  ggplot(aes(x = component, y = `percentage of variance`, group = 1)) +
  geom_line() +
  theme_bw()
```



We can see that the "elbow" here occurs at the third or fourth component, but we're going to only examine the first 2 components. Keep in mind that this means some variation might not be apparent.

The so-called "loadings" in a PCA are the weights of the linear combination of original variables that make up each component. We access them in the `$var$coord` table in the results from `PCA()`.

```
means_pca$var$coord
```

```
##                      Dim.1       Dim.2        Dim.3         Dim.4
## Red_berry        0.392823276  0.01215638 -0.121346852  0.0825451624
## Dark_berry       0.520232898  0.08267522  0.199523469 -0.1864158541
## Jam              0.888495244  0.04433424  0.073820850  0.1720938632
## Dried_fruit      0.050212647  0.10009820 -0.019690630 -0.1664223919
## Artificial_frui  0.572309456  0.05751255  0.238851293 -0.0038853644
## Chocolate       -0.003475125 -0.30028443  0.170378617  0.0938653125
## Vanilla          0.314478467 -0.30349435  0.018229231  0.1585583125
## Oak             -0.193340853 -0.22147313 -0.071362551 -0.0086868023
## Burned          -0.586302217 -0.61025517  0.337991587 -0.0004488918
## Leather         -0.382862924  0.27891331  0.103734732  0.0930246719
## Earthy          -0.257608328  0.09856678  0.079461250  0.0145072272
## Spicy           -0.049183026 -0.02852180  0.031255722 -0.0507807926
## Pepper          -0.153720418  0.07222224 -0.133757042 -0.2058682953
## Grassy           0.162497315  0.10272633 -0.002515019 -0.1016081718
```

```
## Medicinal       -0.365073070  0.48842639  0.031051449  0.2422723514
## Band-aid        -0.432054073  0.33143313  0.129721753  0.0777846772
## Sour             0.040307189  0.34354213  0.186317776  0.0065371842
## Bitter           0.126467903  0.18466794  0.285051403 -0.1458235340
## Alcohol          0.082923788 -0.04864986  0.221354178  0.1389696519
## Astringent      -0.143108623  0.12443684  0.217756015 -0.1479925430
##                        Dim.5
## Red_berry        0.157522934
## Dark_berry      -0.097453593
## Jam              0.079902737
## Dried_fruit     -0.135993046
## Artificial_frui  0.168622742
## Chocolate        0.078682732
## Vanilla         -0.039447653
## Oak             -0.178016939
## Burned           0.135757837
## Leather         -0.022575394
## Earthy           0.040315580
## Spicy            0.050007545
## Pepper           0.124049258
## Grassy           0.109061848
## Medicinal       -0.003582952
## Band-aid         0.175578114
## Sour            -0.064779269
## Bitter          -0.029549547
## Alcohol         -0.320004547
## Astringent       0.007052281
```

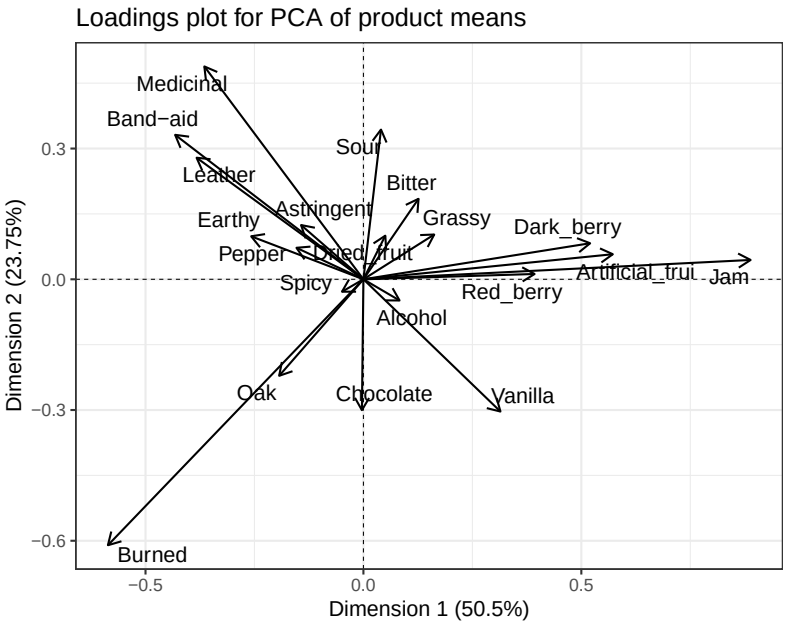
Technically, the number of dimensions we *could* get from a PCA is equal to the $min(n-1, k)$, where n is the number of product means we have and k is the number of measured variables, but in practice we won't typically examine more than 3-4 components/dimensions, as having to examine more for our purposes would indicate that PCA may not be the right tool for dimension reduction.

```
p_loadings <-
  means_pca$var$coord %>%
  as_tibble(rownames = "descriptor") %>%
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_hline(yintercept = 0, linetype = "dashed", size = 1/4) +
  geom_vline(xintercept = 0, linetype = "dashed", size = 1/4) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.1, "in"), ends = "first")) +
  ggrepel::geom_text_repel(aes(label = descriptor)) +
  theme_bw() +
  coord_fixed() +
  labs(title = "Loadings plot for PCA of product means",
```

```
       x = paste0("Dimension 1 (", round(means_pca$eig[1, 2], 2), "%)"),
       y = paste0("Dimension 2 (", round(means_pca$eig[2, 2], 2), "%)"))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
p_loadings
```



Typically, we'd interpret this by looking at variables that are very "close" (make acute angles to) the x- or y-axes (which are the first and second components, respectively), and that are very long. The magnitude of the vector indicates the coefficient for that variable in the linear combination making up each of the two principal components that serve as axes of the displayed space. Therefore, we'd say that the fruity flavors (e.g., `Jam`) are loading strongly and positively on the first component, and almost not at all to the second component, and that the strongest negative contributions come from `Burned`, which is also strongly negatively loaded on the second component. For the second component, `Medicinal` and `Band-aid`, with other associated flavors, are strongly positively loaded (and negatively loaded on the first dimension), whereas `Sour` loads positively almost entirely on the first component, but not as strongly. There are many other observations we could make from this plot. It is worth comparing it to the

loading/coefficient plot for the CVA; you will see that while the values of the coefficients are not identical, the patterns are very similar.

As a side note (because I had to puzzle this out myself), the `$var$coord` matrix is *not* the raw **Q** loadings matrix from singular value decomposition (SVD), which we'd use to find the scores for the original observations. Rather, if we write the SVD as $\mathbf{X} = \mathbf{P}\,\mathbf{Q}^T$, it is $\mathbf{Q}^T$. In effect, the "coordinates" given in `$var$coord` are expanded by the size of the singular value for the associated principal component. We can find the raw loadings (coefficient), **Q** in the `$svd$V` matrix in the results from the `PCA()` function.

We find scores for our product means by using the linear combinations described by the loadings. So, we could by hand calculate:

```
means_pca$svd$V
```

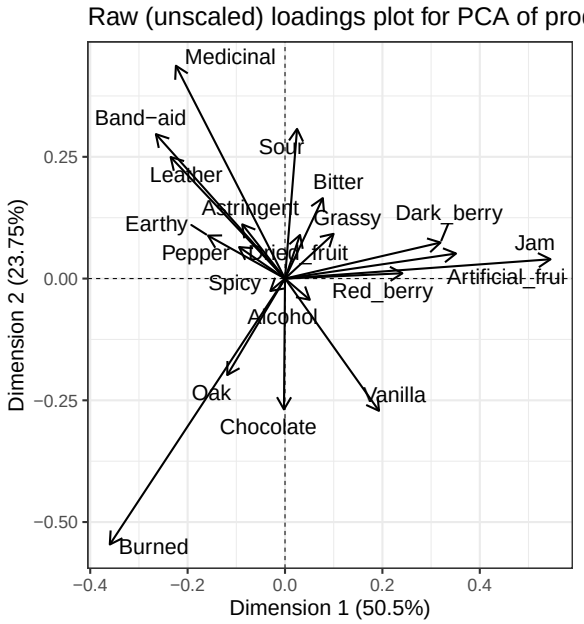
```
##               [,1]        [,2]         [,3]          [,4]         [,5]
##  [1,]  0.241091170  0.01087957 -0.166331750  0.1449419026  0.280322177
##  [2,]  0.319287490  0.07399166  0.273489481 -0.3273295222 -0.173424927
##  [3,]  0.545304646  0.03967772  0.101187224  0.3021813905  0.142192051
##  [4,]  0.030817486  0.08958466 -0.026990210 -0.2922227956 -0.242008358
##  [5,]  0.351248932  0.05147188  0.327396655 -0.0068223514  0.300074997
##  [6,] -0.002132821 -0.26874488  0.233540243  0.1648190709  0.140020974
##  [7,]  0.193007864 -0.27161766  0.024987051  0.2784141773 -0.070199632
##  [8,] -0.118660923 -0.19821131 -0.097817601 -0.0152532458 -0.316792574
##  [9,] -0.359836842 -0.54615870  0.463289578 -0.0007882138  0.241589787
## [10,] -0.234978108  0.24961842  0.142190582  0.1633429815 -0.040174364
## [11,] -0.158104412  0.08821409  0.108918596  0.0254733899  0.071744163
## [12,] -0.030185567 -0.02552609  0.042842635 -0.0891665178  0.088991638
## [13,] -0.094344296  0.06463658 -0.183342562 -0.3614862644  0.220753620
## [14,]  0.099731024  0.09193675 -0.003447371 -0.1784148375  0.194082562
## [15,] -0.224059772  0.43712587  0.042562636  0.4254085222 -0.006376094
## [16,] -0.265168660  0.29662196  0.177811338  0.1365829175  0.312452529
## [17,]  0.024738115  0.30745913  0.255388262  0.0114787092 -0.115278869
## [18,]  0.077618350  0.16527185  0.390723762 -0.2560530483 -0.052585316
## [19,]  0.050893606 -0.04354006  0.303413126  0.2440182460 -0.569468640
## [20,] -0.087831418  0.11136696  0.298481076 -0.2598616336  0.012549988
```

This tells us that, to get a mean vector i's score on principal component 1 (Dimension 1), we would calculate $PC_i = 0.24{*}Red\_berry_i{+}...{-}0.09{*}Astringent_i$, and so on (note that because this is a raw matrix, it has no row names or descriptive columns; I am getting the loadings for specific descriptors by looking back at what the 1st and 20th descriptors are in our data set and matching them up manually).

Before we leave this topic, I want to point out one more thing: if we plot the raw loadings, we'll come to the same conclusions:

```
means_pca$svd$V %>%
  as_tibble() %>%
  bind_cols(descriptor = row.names(means_pca$var$coord)) %>%
  rename(Dim.1 = V1, Dim.2 = V2) %>%
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_hline(yintercept = 0, linetype = "dashed", size = 1/4) +
  geom_vline(xintercept = 0, linetype = "dashed", size = 1/4) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.1, "in"), ends = "first")) +
  ggrepel::geom_text_repel(aes(label = descriptor)) +
  theme_bw() +
  coord_fixed() +
  labs(title = "Raw (unscaled) loadings plot for PCA of product means",
       x = paste0("Dimension 1 (", round(means_pca$eig[1, 2], 2), "%)"),
       y = paste0("Dimension 2 (", round(means_pca$eig[2, 2], 2), "%)"))
```

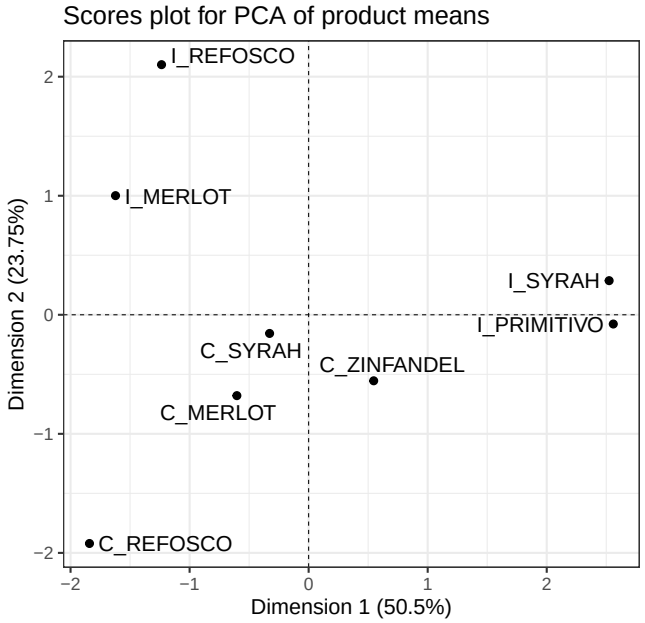
```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
## `.name_repair` is omitted as of tibble 2.0.0.
## i Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



OK, with all that said, if we multiply our means-vector ratings (mean-centered

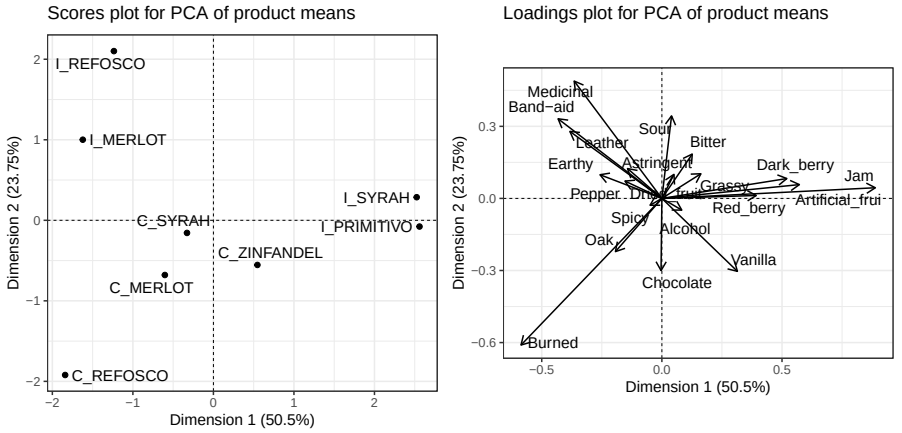
for each column) by the loadings we just spent a while getting, we get the *scores* for our mean vectors in the PCA space. These are stored in the `$ind$coord` matrix.

```
p_scores <-
  means_pca$ind$coord %>%
  as_tibble(rownames = "product") %>%
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_hline(yintercept = 0, linetype = "dashed", size = 1/4) +
  geom_vline(xintercept = 0, linetype = "dashed", size = 1/4) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = product)) +
  theme_bw() +
  coord_fixed() +
  labs(title = "Scores plot for PCA of product means",
       x = paste0("Dimension 1 (", round(means_pca$eig[1, 2], 2), "%)"),
       y = paste0("Dimension 2 (", round(means_pca$eig[2, 2], 2), "%)"))

p_scores
```



We interpret this plot by noting the spatial separation of sample mean-vectors, as well as noting the proximity to the axes, which we interpret by their loadings from variables. In order to facilitate this second task, it is often helpful to have the loadings and scores plots side by side. We can accomplish this using the nifty `patchwork` package, which lets us arrange saved plots however we want.

```
library(patchwork)

p_scores + p_loadings + plot_layout(widths = c(2, 2))
```



By looking at these plots together, we can see that the first dimension, which we previously noted separated fruity flavors from medicinal and burnt flavors, is driven by a separation of the Italian Primitivo and Syrah samples from the other samples. The second dimension, which is separating the medicinal and burnt flavors, is interestingly also separating the Californian and Italian wines made from two sets of the same grape, Refosco and Merlot.

6.3 In-depth interpretation

There are several ways that we can get more information about the structure of the PCA solution. First, we will follow HGH and investigate the *correlations* between the variables and each of the first two components. HGH used the `FactoMineR::dimdesc()` function, but I find this prints out too much to look good here. We can access the correlations directly using `$var$cor` in the output of the `PCA()` function. I'll turn it into a tibble to make display easier.

```
# The most highly correlated variables with Dimension 1
means_pca$var$cor %>%
  as_tibble(rownames = "descriptor") %>%
  select(1:3) %>%
  arrange(-Dim.1) %>%
  slice(1:3, 18:20)
```

```
## # A tibble: 6 x 3
```

```
##   descriptor       Dim.1  Dim.2
##   <chr>            <dbl>  <dbl>
## 1 Jam              0.972 0.0485
## 2 Artificial_frui  0.882 0.0886
## 3 Dark_berry       0.849 0.135
## 4 Band-aid        -0.726 0.557
## 5 Leather         -0.750 0.547
## 6 Earthy          -0.864 0.330
```

```
# The most highly correlated variables with Dimension 2
means_pca$var$cor %>%
  as_tibble(rownames = "descriptor") %>%
  select(1:3) %>%
  arrange(-Dim.2) %>%
  slice(1:3, 18:20)
```

```
## # A tibble: 6 x 3
##   descriptor    Dim.1  Dim.2
##   <chr>         <dbl>  <dbl>
## 1 Sour        0.0997   0.850
## 2 Medicinal  -0.554    0.741
## 3 Band-aid   -0.726    0.557
## 4 Vanilla     0.660   -0.637
## 5 Burned     -0.636   -0.662
## 6 Chocolate  -0.00940 -0.812
```

These are *also* called loadings **REF Abdi 2010**. The semantics of PCA are a nightmare. We can see that these reflect our interpretation from the loadings plot pretty accurately.

HGH then calculated the "communalities" for variables on the first and second dimensions of the PCA solution. I had to do some digging to figure out what she meant by this, but she described them as:

> Communality is the sum of the squared loadings for the number of dimensions that you would like to keep.

We know from ***REF Abdi 2010*** *that the sum of squared loadings (in the sense of* correlations*) for each dimension should sum to 1, so this allows us to speak of "proportion of explained variance" for each variable and each dimension. But we have to remember the semantic overloadings here, so if we want to see this property we will need to look at the* `$svd$V` matrix again. The `$var$cor` matrix, remember, stores this scaled by the original variables (maybe this is instead storing covariances). We can see this pretty easily:

```
# The "cor" matrix doesn't seem to really be storing what we think of as
# correlations.
means_pca$var$cor %>%
  as_tibble(rownames = "descriptor") %>%
  pivot_longer(-descriptor) %>%
  mutate(squared_loading = value ^ 2) %>%
  group_by(name) %>%
  summarize(total = sum(squared_loading))
```

```
## # A tibble: 5 x 2
##   name  total
##   <chr> <dbl>
## 1 Dim.1  7.46
## 2 Dim.2  4.58
## 3 Dim.3  2.45
## 4 Dim.4  2.06
## 5 Dim.5  1.88
```

```
# Whereas the SVD list behaves as expected.
means_pca$svd$V %>%
  as_tibble() %>%
  pivot_longer(everything()) %>%
  mutate(value = value ^ 2) %>%
  group_by(name) %>%
  summarize(total = sum(value))
```

```
## # A tibble: 5 x 2
##   name  total
##   <chr> <dbl>
## 1 V1     1.00
## 2 V2     1
## 3 V3     1.00
## 4 V4     1.00
## 5 V5     1
```

6.3.1 Correlations or (squared) loadings or contributions

The correlations between the variables and the components are also unfortunately called "*loadings*" (cf. Abdi & Williams 2010 **REF** for more info), but these are distinct (although related, argh) from the "loadings" we discussed as the elements of the **Q** matrix from SVD.

We could directly calculate these, but we can make use of the `dimdesc()` convenience function

```
dimdesc(means_pca, axes = c(1, 2))
```

```
## $Dim.1
##
## Link between the variable and the continuous variables (R-square)
## ================================================================================
##                  correlation      p.value
## Jam                0.9723336 5.184944e-05
## Artificial_frui    0.8819989 3.752736e-03
## Dark_berry         0.8492159 7.630518e-03
## Red_berry          0.8418534 8.752510e-03
## Band-aid          -0.7259831 4.144504e-02
## Leather           -0.7504104 3.195742e-02
## Earthy            -0.8635921 5.713937e-03
##
## $Dim.2
##
## Link between the variable and the continuous variables (R-square)
## ================================================================================
##            correlation     p.value
## Sour         0.8501775 0.007491159
## Medicinal    0.7412352 0.035345156
## Chocolate   -0.8121299 0.014329245
```

HGH calls these squared correlations the "communalities" of the variables, which should(?) sum to 1. This is not the case here because we have a typical $n < p$ problem: there are more variables than observations (when we run a PCA on the means of the products across panelists and reps).

There are a number of other ways to interpret the over-loaded "*loadings*"–the role of the variables in determining the new principal-components space in PCA results. To return to the progression of the **R Opus**, let's follow HGH in examining the **Contributions** and the **Communalities** in the PCA results.

According to ***REF Abdi & Williams 2010, p.8-9***,

> …the importance of an observation for a component can be obtained by the ratio of the squared factor score of this observation by the eigenvalue associated with that component.

and

> The value of a contribution is between 0 and 1 and, for a given component, the sum of the contributions of all observations is equal to

1. The larger the value of the contribution, the more the observation contributes to the component. A useful heuristic is to base the interpretation of a component on the observations whose contribution is larger than the average contribution (i.e., observations whose contribution is larger than $1/I$).

While Abdi and Williams are talking about the *contribution* of an observation, since PCA is (largely) agnostic about the role of observation (product) and variable (descriptor), `FactoMineR::PCA()` will return contributions for both products and descriptors, found as usual in the `ind` and `var` sub-lists of the results. We don't care so much about the products' contributions, in this case, but we do care about the variables'. We can find and print them:

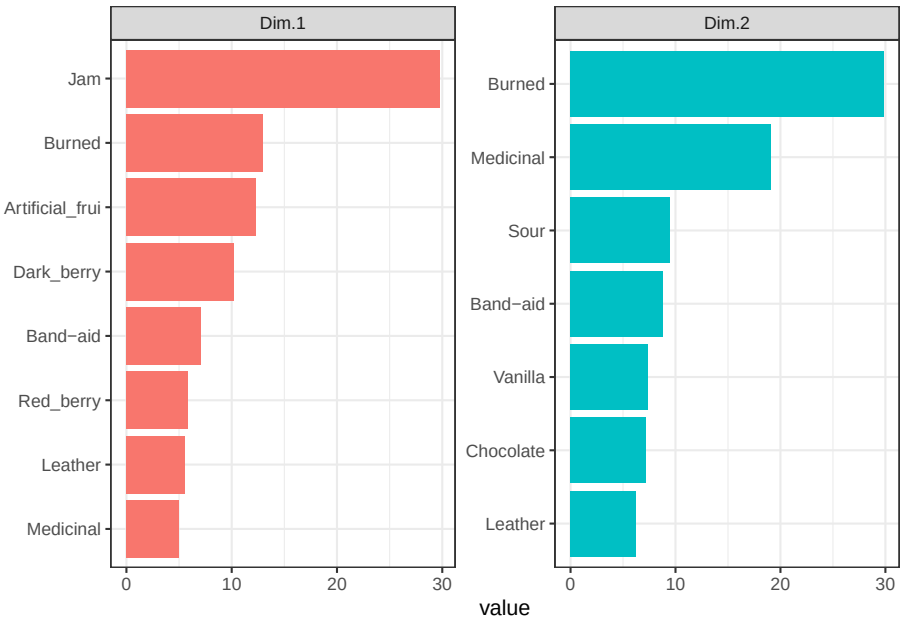
```
means_pca$var$contrib %>% round(3)
```

```
##                  Dim.1  Dim.2  Dim.3  Dim.4  Dim.5
## Red_berry        5.812  0.012  2.767  2.101  7.858
## Dark_berry      10.194  0.547  7.480 10.714  3.008
## Jam             29.736  0.157  1.024  9.131  2.022
## Dried_fruit      0.095  0.803  0.073  8.539  5.857
## Artificial_frui 12.338  0.265 10.719  0.005  9.005
## Chocolate        0.000  7.222  5.454  2.717  1.961
## Vanilla          3.725  7.378  0.062  7.751  0.493
## Oak              1.408  3.929  0.957  0.023 10.036
## Burned          12.948 29.829 21.464  0.000  5.837
## Leather          5.521  6.231  2.022  2.668  0.161
## Earthy           2.500  0.778  1.186  0.065  0.515
## Spicy            0.091  0.065  0.184  0.795  0.792
## Pepper           0.890  0.418  3.361 13.067  4.873
## Grassy           0.995  0.845  0.001  3.183  3.767
## Medicinal        5.020 19.108  0.181 18.097  0.004
## Band-aid         7.031  8.798  3.162  1.865  9.763
## Sour             0.061  9.453  6.522  0.013  1.329
## Bitter           0.602  2.731 15.267  6.556  0.277
## Alcohol          0.259  0.190  9.206  5.954 32.429
## Astringent       0.771  1.240  8.909  6.753  0.016
```

Note that `FactoMineR` seems to scale the contributions to a percentage (e.g., multiply by 100), rather than returning contributions in the range $[0, 1]$. Following Abdi & Williams' suggestion above, we can do a little wrangling to see important contributions visually:

```
# First we make a tibble
means_pca$var$contrib %>%
```

```
  as_tibble(rownames = "descriptor") %>%
  # Then we select the first 2 dimensions (for ease)
  select(descriptor, Dim.1, Dim.2) %>%
  # For both plotting and filtering, long-type data will be easier to work with
  pivot_longer(-descriptor) %>%
  # We can now choose only contributions > 100 / # of descriptors (that is, 20)
  filter(value > 100 / 20) %>%
  # We use some convenience functions from `tidytext` to make our facets nicer
  mutate(descriptor = factor(descriptor) %>%
           tidytext::reorder_within(by = value, within = name)) %>%
  # And now we plot!
  ggplot(aes(x = descriptor, y = value)) +
  geom_col(aes(fill = name), show.legend = FALSE) +
  tidytext::scale_x_reordered(NULL) +
  coord_flip() +
  theme_bw() +
  facet_wrap(~name, scales = "free")
```



We can see that for PC1, contributions seem to be coming from a lot of fruity flavors, as well as some influence from complex flavors that I would attribute to possible Brettanomyces influence in some of the wines. In PC2, there appears to be instead more influence of oak ("Chocolate" and "Vanilla") as well as the same Brettanomyces flavors. Note that contributions, as squared measurements, are always positive - these are *absolute* measures of influence on the dimensions.

HGH says in the original **R Opus** that

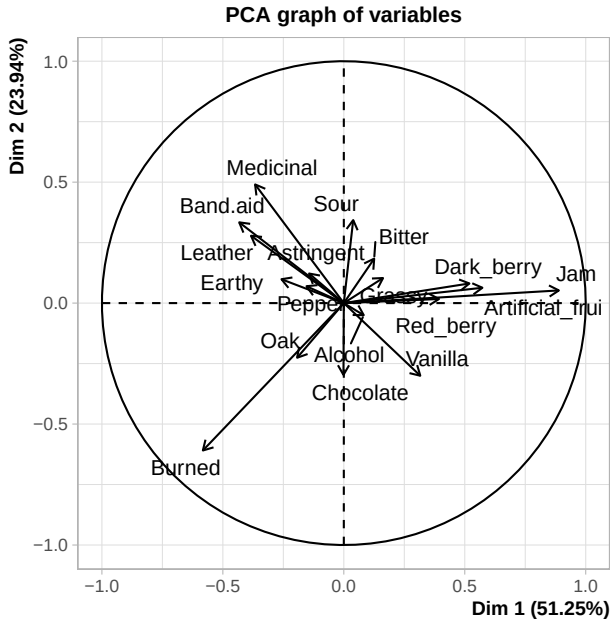
> Communality is the sum of the squared loadings for the number of dimensions that you would like to keep.

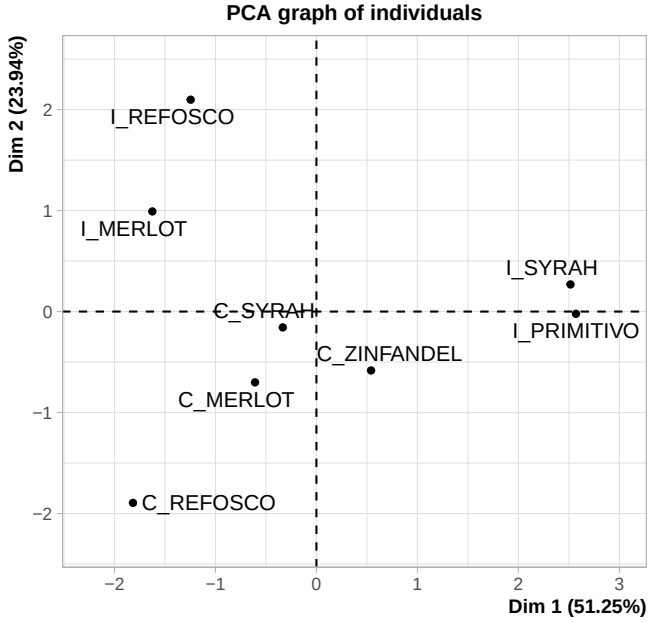
I am not sure I quite follow what she did, as she then goes on to examine the contributions, which as we've described are the squared loadings divided by the eigenvalues so that they sum to 1. In **REF Abdi & Williams** they don't discuss *communality*, and if I remember properly the concept is more frequently applied to Factor Analysis **REF Rencher 2012**, so we'll leave it for now. I think that this is a great example of how closely overlapping concepts can get confusing in the world of components-based methods, since to my understanding Factor Analysis, in some of its simpler forms, can be derived directly from PCA but with different assumptions mapped onto the steps.

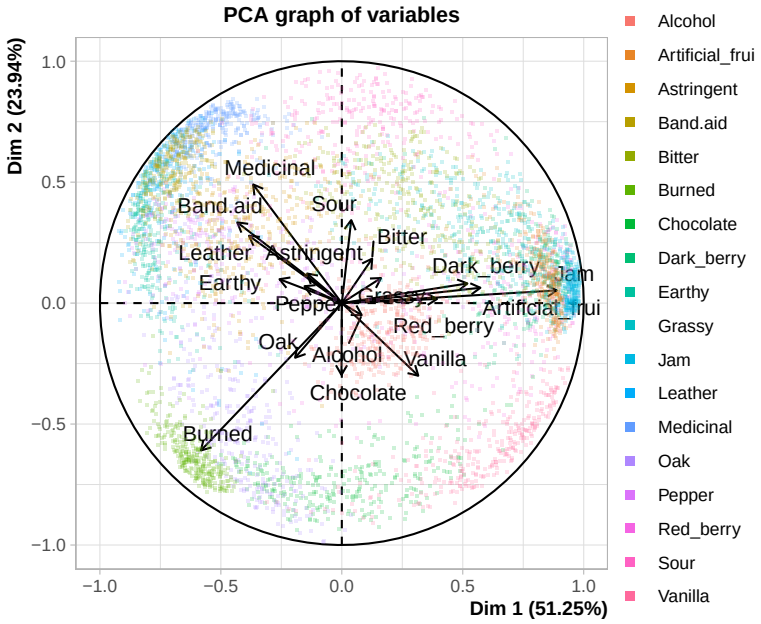
6.4 PCA with resampling for confidence intervals

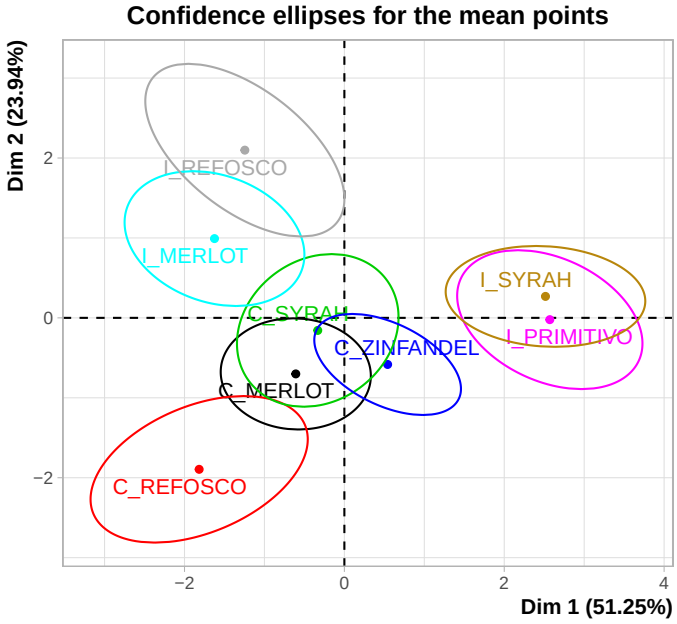
In the original **R Opus**, HGH uses the `SensoMineR::panellipse()` function to generate confidence ellipses for the product mean vectors in PCA.

```
panellipse_res <-
  # We have to reimport the data because panellipse() doesn't behave well with
  # tibble() formats.
  SensoMineR::panellipse(donnee = read.csv(here("data/torriDAFinal.csv")),
                         col.p = 2, col.j = 1, firstvar = 4, scale.unit = FALSE)
```


PCA graph of variables
Dim 2 (23.94%)
Dim 1 (51.25%)
Medicinal
Band.aid
Sour
Bitter
Leather
Astringent
Earthy
Dark_berry
Jam
Red_berry
Oak
Alcohol
Vanilla
Chocolate
Burned
1.0
0.5
0.0
−0.5
−1.0

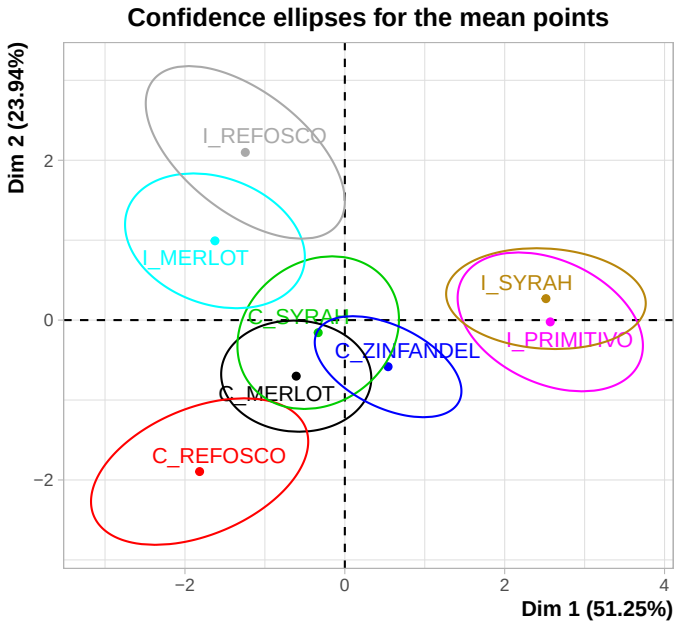

PCA graph of individuals
Dim 2 (23.94%)
Dim 1 (51.25%)
I_REFOSCO
I_MERLOT
I_SYRAH
C_SYRAH
I_PRIMITIVO
C_ZINFANDEL
C_MERLOT
C_REFOSCO
2
1
0
−1
−2
−2
−1
0
1
2
3





I'm not a huge fan of `panellipse()` because it's pretty opaque. I can't find the documentation on what it's doing, and there doesn't seem to be an associated journal article. It doesn't really document how it is resampling or give easily understood descriptions of what the results (both numerical and graphical) it is producing *mean*. Here is the plot that HGH uses for the original **R Opus**:

```
panellipse_res$graph$plotIndEll
```



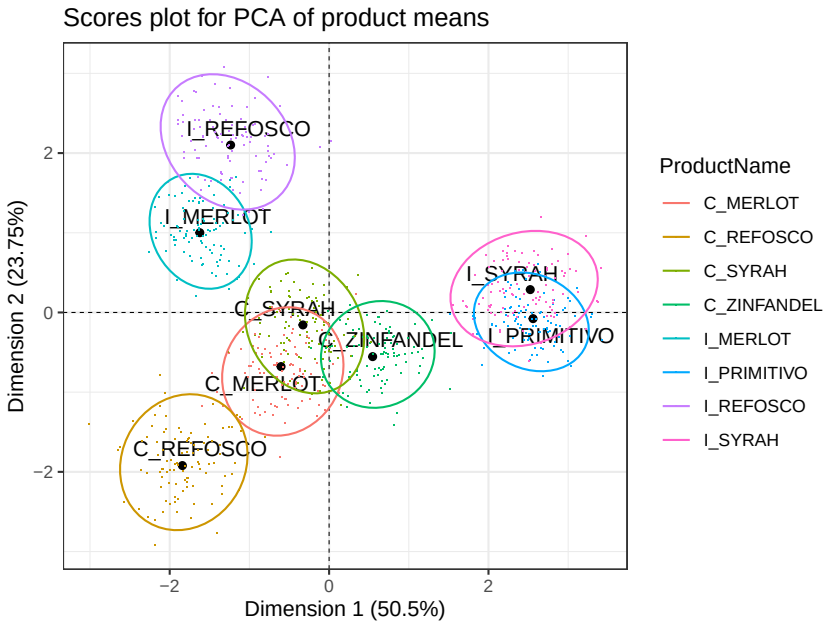
The confidence ellipses are definitely being drawn around 95% of the resampled results from the bootstrapping procedure, but I'm not sure if this is a bootstrap based on, for example, the "partial bootstrap" or the "truncated bootstrap". We will us a naive approach to producing a partial bootstrap in order to do some resampling and compare it.

It is also worth noting that the plot produced here is different than that produced in the original **R Opus**, so eithere there is simulation variability (probably) or the underlying program has changed between 2015 and now (also possible). The overall conclusions are not greatly different but the overlapping areas can vary quite dramatically.

The basic approach (which we saw back in the CVA section of the **R Opus**) is to draw a new set of bootstrapped observations for each product: we need 42 observations per product to calculate a new mean. We then can use the projection function from our original PCA solution to project these results into our space; in a nod to the truncated bootstrap approach **REF Peltier et al** we will use only the first 2 dimensions of the projection function to get the results so as not to overfit. Finally, we'll draw ellipses around our results to represent variability.

```
get_bootstrapped_pca_means <- function(){
```

```
  descriptive_data %>%
    select(-NJ, -NR) %>%
    group_by(ProductName) %>%
    # Here we resample each wine - we draw 42 new observations for each wine
    # with replacement
    slice_sample(prop = 1, replace = TRUE) %>%
    # We calculate the mean for the newly drawn samples
    summarize_if(is.numeric, ~mean(.)) %>%
    # And then we center the means (by subtracting the column means)
    mutate_if(is.numeric, ~. - mean(.)) %>%
    nest(data = -ProductName) %>%
    mutate(means = map(data, ~as.matrix(.x)),
           pc1 = map_dbl(means, ~ .x %*% means_pca$svd$V[, 1]),
           pc2 = map_dbl(means, ~ .x %*% means_pca$svd$V[, 2])) %>%
    select(-data, -means)

}

# Make it reproducible
set.seed(6)

pca_boots <-
  tibble(boot_id = 1:100) %>%
  mutate(bootstrapped_data = map(boot_id, ~get_bootstrapped_pca_means())) %>%
  unnest(bootstrapped_data)

p_scores +
  geom_point(data = pca_boots,
             inherit.aes = FALSE,
             aes(x = pc1, y = pc2, color = ProductName), shape = ".") +
  stat_ellipse(data = pca_boots, inherit.aes = FALSE,
               aes(x = pc1, y = pc2, color = ProductName))
```



Our results are pretty close, but not exactly the same. It seems like our method of generating bootstrapped scores (via resampling followed by projection via the **Q** matrix from SVD) is potentially more liberal in product separation than that from the `panellipse()` function. Perhaps `panellipse()` is using the "truncated bootstrap" approach **REF Peltier**, which solves a full PCA with the resampled data, then aligns it with the original observed space via Generalized Procrustes Analysis, then repeats that process a large number (e.g., 1000) times..

6.5 Comparison of products with PCA

HGH then used the `panellipse()` results to get Hotelling's T^2 stats for each set of products. I believe that Hotelling's T^2 is a generalization of the t-distribution to multivariate data. These were accessed from the outputs of `panellipse()`, which we stored in `panellipse_res`.

```
names(panellipse_res)
```

```
## [1] "eig"         "coordinates" "hotelling"   "graph"       "correl"
```

The `SensoMineR::coltable()` function HGH used is a visualization function for this kind of output, let's take a look.

```
SensoMineR::coltable(panellipse_res$hotelling, main.title = "Hotelling's T2 for all products")
```

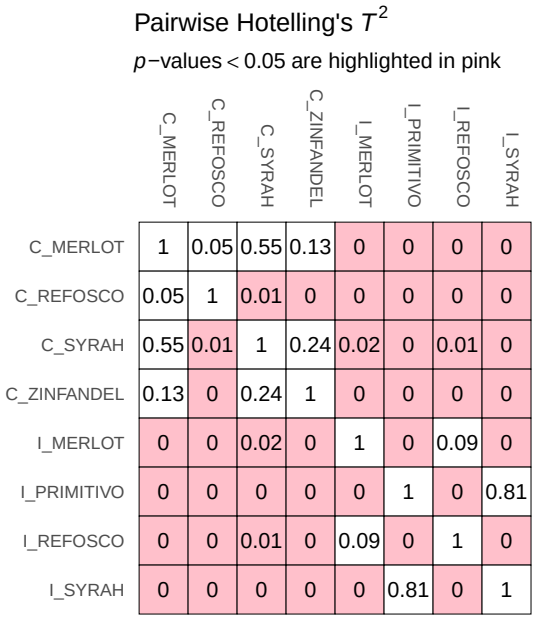
Hotelling's T2 for all products

	C_MERLOT	C_REFOSCO	C_SYRAH	C_ZINFANDEL	I_MERLOT	I_PRIMITIVO	I_REFOSCO	I_SYRAH
C_MERLOT	1	0.05215	0.5451	0.1273	0.003156	8.821e−05	0.0002263	0.000104
C_REFOSCO	0.05215	1	0.01246	0.002463	7.147e−05	7.915e−06	5.228e−06	6.472e−06
C_SYRAH	0.5451	0.01246	1	0.2356	0.02246	0.001006	0.005852	0.001447
C_ZINFANDEL	0.1273	0.002463	0.2356	1	0.0006153	0.003376	0.0002946	0.002035
I_MERLOT	0.003156	7.147e−05	0.02246	0.0006153	1	1.375e−05	0.08825	1.923e−05
I_PRIMITIVO	8.821e−05	7.915e−06	0.001006	0.003376	1.375e−05	1	8.722e−05	0.8095
I_REFOSCO	0.0002263	5.228e−06	0.005852	0.0002946	0.08825	8.722e−05	1	0.0001061
I_SYRAH	0.000104	6.472e−06	0.001447	0.002035	1.923e−05	0.8095	0.0001061	1

Let's practice how to make a similar table from this kind of data. The actual `panellipse_res$hotelling` object is just a square matrix. We can use this as input for something like the `geom_tile()` function with the right kind of wrangling.

```
# First wrangle

panellipse_res$hotelling %>%
  as_tibble(rownames = "x") %>%
  pivot_longer(-x, names_to = "y") %>%
  mutate(color = if_else(value < 0.05, "pink", "white")) %>%

  # Now plot

  ggplot(aes(x = x, y = y)) +
  geom_tile(aes(fill = color),
            color = "black") +
  geom_text(aes(label = round(value, 2))) +
  scale_fill_manual(values = c("pink", "white")) +
  coord_fixed() +

  # Everything after this is just making the plot look nice
```

```
  scale_x_discrete(position = "top") +
  scale_y_discrete(limits = rev) +
  labs(x = NULL,
       y = NULL,
       title = bquote("Pairwise Hotelling's"~italic(T)^2),
       subtitle = bquote(italic(p)*"-values"<0.05~"are highlighted in pink")) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 270, hjust = 1),
        legend.position = "none",
        axis.ticks = element_blank(),
        panel.grid = element_blank())
```



Notice the `ggplot2` syntax above is kind of complicated, but that's because I did it all at once, and I wanted to do a lot of minor things like remove axis ticks, so as to replicate the plot from the `panellipse()` function closely. Notice that I don't think the degrees of freedom for the Hotelling's T^2 plot.

As a bonus, we will quickly look into how to conduct Hotelling's T^2 tests ourselves, and then leave the world of PCA (for now) to turn to methods for cluster analysis.

```
# We need pairs of products - if we wanted to make all pairwise comparisons it
# would be possible to do so using, for example, nested `for()` loops or some
# kind of list-table structure
```

```
hotelling_demo_data <-
  descriptive_data %>%
  filter(ProductName %in% c("C_MERLOT", "C_REFOSCO")) %>%
  select(-NJ, -NR)

DescTools::HotellingsT2Test(formula = as.matrix(hotelling_demo_data[, -1]) ~ ProductName,
                            data = hotelling_demo_data)
```

```
##
##  Hotelling's two sample T2-test
##
## data:  as.matrix(hotelling_demo_data[, -1]) by ProductName
## T.2 = 2.4053, df1 = 20, df2 = 63, p-value = 0.004345
## alternative hypothesis: true location difference is not equal to c(0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

These results are not the same as those given in the `panellipse()` output; I suspect after reading `?panellipse` that this is because internally that function is running a Hotelling's T^2 test on the PCA results, rather than on the raw data, but I am not sure and I am not willing to try to interpret the under-the-hood code. If you know, please reach out and let me know!

6.6 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
```

```
##
## other attached packages:
##  [1] patchwork_1.1.2 here_1.0.1      FactoMineR_2.8  lubridate_1.9.2
##  [5] forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1
##  [9] readr_2.1.4     tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] Exact_3.2           tidyselect_1.2.0    rootSolve_1.8.2.4
##  [4] farver_2.1.1        fastmap_1.1.1       janeaustenr_1.0.0
##  [7] digest_0.6.33       timechange_0.2.0    estimability_1.4.1
## [10] lifecycle_1.0.3     cluster_2.1.4       multcompView_0.1-9
## [13] tokenizers_0.3.0    lmom_3.0            magrittr_2.0.3
## [16] compiler_4.3.1      rlang_1.1.1         tools_4.3.1
## [19] utf8_1.2.3          tidytext_0.4.1      yaml_2.3.7
## [22] data.table_1.14.8   knitr_1.43          labeling_0.4.3
## [25] htmlwidgets_1.6.2   bit_4.0.5           scatterplot3d_0.3-44
## [28] plyr_1.8.8          KernSmooth_2.23-21  expm_0.999-7
## [31] withr_2.5.0         grid_4.3.1          fansi_1.0.4
## [34] SensoMineR_1.26     AlgDesign_1.2.1     xtable_1.8-4
## [37] e1071_1.7-13        colorspace_2.1-0    emmeans_1.8.7
## [40] scales_1.2.1        gtools_3.9.4        MASS_7.3-60
## [43] flashClust_1.01-2   cli_3.6.1           mvtnorm_1.2-2
## [46] rmarkdown_2.23      crayon_1.5.2        generics_0.1.3
## [49] rstudioapi_0.15.0   httr_1.4.6          reshape2_1.4.4
## [52] tzdb_0.4.0          readxl_1.4.3        gld_2.6.6
## [55] proxy_0.4-27        parallel_4.3.1      cellranger_1.1.0
## [58] vctrs_0.6.3         boot_1.3-28.1       Matrix_1.6-0
## [61] bookdown_0.37       hms_1.1.3           bit64_4.0.5
## [64] ggrepel_0.9.3       glue_1.6.2          DT_0.28
## [67] stringi_1.7.12      gtable_0.3.4        munsell_0.5.0
## [70] pillar_1.9.0        htmltools_0.5.6     R6_2.5.1
## [73] rprojroot_2.0.3     vroom_1.6.3         evaluate_0.21
## [76] lattice_0.21-8      highr_0.10          SnowballC_0.7.1
## [79] leaps_3.1           DescTools_0.99.50   class_7.3-22
## [82] Rcpp_1.0.11         coda_0.19-4         xfun_0.39
## [85] pkgconfig_2.0.3
```

Chapter 7

Cluster analysis

The goal of any cluster analysis is to find groups ("clusters") of observations that are "close to" each other in some sense, so as to reveal underlying structure in the data: typically, we would want to know that groups of more than one observation are very "close" so that we can speak about the group instead of the original observations. In most sensory evaluation, "close" is usually taken to mean "similar", as the definitions of "close" we will operationalize are based on the sensory descriptors, so that observations that are "close" to each other will, in some sense, have similar sensory profiles.

We could also use cluster analysis to explore possible hypotheses, if we have some hypotheses about the underlying group structure that exists–for example, if we think that wines made from the same grape would be more similar to each other, we'd expect those wines to show up in the same group. We'll explore this more as we look at our results.

We start by loading our results, as before. We will also define a tibble of product means, which will be our main data input to start with.

```
library(tidyverse)
library(here)
library(factoextra) # this is new

# I've decided to use the `across()` syntax instead of the scoped mutate_*()
# functions, for documentation check out ?across()
descriptive_data <- read_csv(here("data/torriDAFinal.csv")) %>%
  mutate(across(1:3, ~as.factor(.)))

descriptive_means <-
  descriptive_data %>%
  group_by(ProductName) %>%
```

```
  summarize(across(where(is.numeric), ~mean(.)))
```

We're going to mostly use the built in `stats::hclust()` function for most of this workflow, but do know that this is the simplest (and perhaps not best) clustering tool available in `R`. It will do for our purposes.

In the **R Opus**, HGH *scales* the mean data to have zero-means and unit-variance. This choice (it is not necessary for calculation) means that all descriptors will have equal impact on our estimates of proximity for the purpose of clustering. We'll follow along.

```
descriptive_means_scaled <-
  descriptive_means %>%
  # This line is desnse - notice the "lambda" ("~") function that uses multiple
  # references to the same column: we are subtracting the column mean and
  # dividing by the column sd for each column.
  mutate(across(where(is.numeric), ~ (. - mean(.)) / sd(.)))
```

Now we're ready to think about "close". As the word implies, we're going to examine the distances among all of our products. The built in function in `R` to calculate distance is `stats::dist()`. This will serve our purposes well.

```
descriptive_distance <-
  descriptive_means_scaled %>%
  # We need to remember to move our variable column to the "rownames" attribute
  # so that the older function keeps it (and doesn't try to find distances
  # between character vectors)
  column_to_rownames("ProductName") %>%
  dist(method = "euclidean")

descriptive_distance
```

```
##             C_MERLOT C_REFOSCO  C_SYRAH C_ZINFANDEL I_MERLOT I_PRIMITIVO
## C_REFOSCO   5.083950
## C_SYRAH     4.525147  5.472269
## C_ZINFANDEL 3.909011  5.856348 4.817566
## I_MERLOT    4.503404  6.405151 6.153529    6.522035
## I_PRIMITIVO 6.129930  7.853063 5.998705    5.543431 7.842528
## I_REFOSCO   6.272963  6.992652 5.408757    6.917554 4.452965    8.514098
## I_SYRAH     6.911105  8.427799 6.359281    6.170264 7.546229    5.747512
##             I_REFOSCO
## C_REFOSCO
## C_SYRAH
## C_ZINFANDEL
## I_MERLOT
```

```
## I_PRIMITIVO
## I_REFOSCO
## I_SYRAH      7.537424
```

The `dist()` function produces a lower-triangular matrix of the distances between each pair of mean vectors. Because we selected `method = "euclidean"` the distance is calculated as the typical (L2) norm: the square-root of the the sum of the squared differences between each attribute mean for the two products. Other common options are available, see `?dist`.

Technically, all (mathematical) distances must fulfill 4 properties:

1. For any object, $dist(a, a) = 0$ (the distance of an object to itself is always 0)
2. For all pairs of objects, $dist(a, b) \geq 0$ (all distances are positive or 0)
3. For any pair of objects, $dist(a, b) = dist(b, a)$ (distane is symmetric)
4. For any three objects, $dist(a, b) + dist(b, c) \geq dist(a, c)$ (the triangle inequality - I like the Wikipedia description as "intermediate objects can't speed you up")

Enough about distance! Let's get on with it. We can see that our distance matrix is all positive entries that give us some idea of "how close" each pair of objects is. Smaller numbers indicate proximity.

7.1 Hierarchical clustering on distances

One of the major families of clustering is called "hierarchical" clustering (HCA: Hierarchical Clustering Analysis). In plain language, hierarchical methods are iterative methods that start with the assumption that each object (sample) starts in its own group and then, for each step in the process, the two "closest" objects are merged into a group. Different strategies for calculating distance (between the merged groups, as we will typically stick with Euclidean distance as our base metric for singlets) and different methods for making the merge define the different hierarchical clustering approaches.

In the original **R Opus**, HGH demonstrates 4 different HCA methods. We'll look at each briefly.

7.1.1 Ward's Method

Ward's Method is probably the most commonly used (and intellectually satisfying approach). For Ward's method, to quote Rencher [2002, 466]:

> Ward's method, also called the incremental sum of squares method, uses the within cluster (squared) distances and the between-cluster (squared) distances… Ward's method joins the two clusters A and B that minimize the increase in [the difference between the new AB cluster's within-distance and the old A and B within-distances].

Ward's method, qualitatively, tends to find balanced clusters that result from the merge of smaller clusters.

```
cluster_ward <-
  descriptive_distance %>%
  hclust(method = "ward.D2")
```

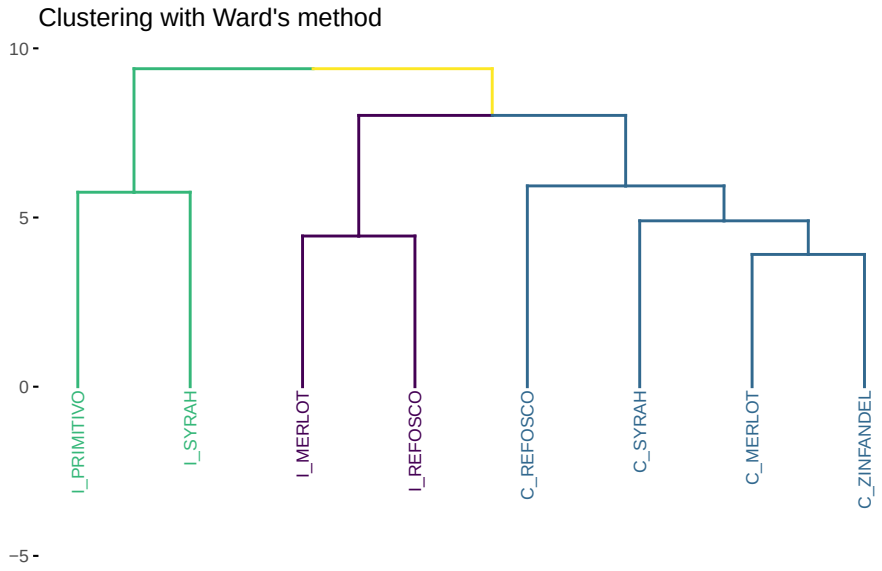
Note that the original **R Opus** used `method = "ward"`–according to the documentation in `?hclust`:

> Two different algorithms are found in the literature for Ward clustering. The one used by option "ward.D" (equivalent to the only Ward option "ward" in R versions 3.0.3) does not implement Ward's (1963) clustering criterion, whereas option "ward.D2" implements that criterion (Murtagh and Legendre 2014). With the latter, the dissimilarities are squared before cluster updating. Note that agnes(, *method="ward") corresponds to hclust(*, "ward.D2").

```
p_ward <-
  cluster_ward %>%
  fviz_dend(k = 3, cex = 2/3)
```

We're going to use the `factoextra::fviz_dend()` function for drawing our clusters. Long-story-short, the native `ggplot2` functions for plotting tree- and graph-like structures like "dendrograms" (the tree plots that are common for HCA results) don't really exist, and `fviz_dend()` is going to do a lot of heavy lifting for us in the background by giving us a basic dendrogram `ggplot2` object that we can alter as we see fit using more familiar syntax.

```
p_ward <-
  p_ward +
  # Stretch out the y-axis so that we can see the labels
  expand_limits(y = -6) +
  # Clean up the messiness
  labs(title = "Clustering with Ward's method", y = NULL) +
  scale_color_viridis_d()

p_ward
```



Notice that we had to *tell* the program (in this case `fviz_dend()`) how many groups we wanted to label separately (`k = 3`). We're following HGH here. In general, while there are methods for attempting to determine the "right" number of groups from HCA, this involves "researcher degrees of freedom" (i.e., "good judgment").

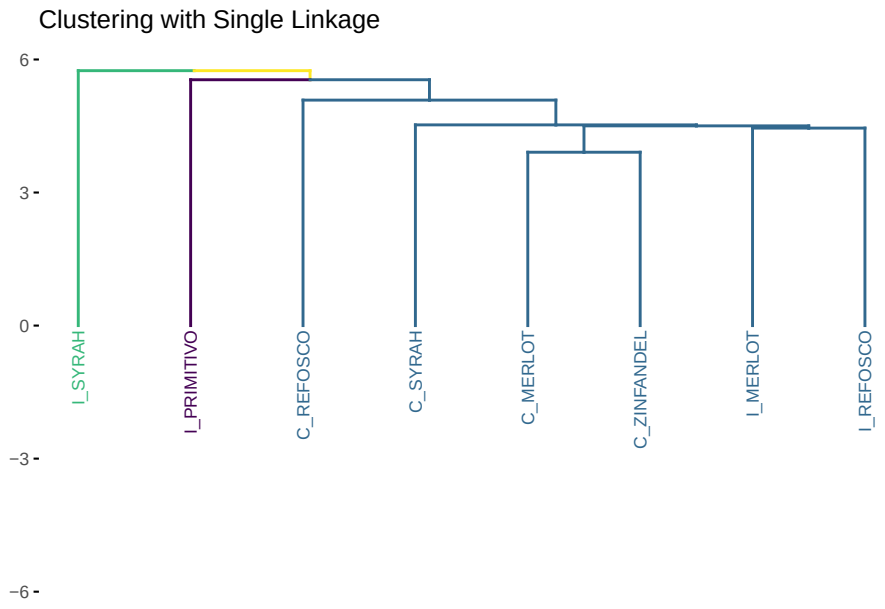
7.1.2 Single linkage

Single linkage is also called "nearest neighbor" clustering. In single-linkage, the key element is that the distance between any two *clusters* A and B is defined as the *minimum* distance between a point a in A and a point b in B. Single linkage is, therefore, "greedy", because big clusters will tend to get bigger: there is a greater chance that a large cluster will have a small distance between *some* point within it and another point outside it.

```
cluster_single <-
  descriptive_distance %>%
  hclust(method = "single")
```

We can use the same number of groups (`k = 3`) so we can have a consistent comparison among the methods.

```
p_single <-
  cluster_single %>%
  fviz_dend(k = 3, cex = 2/3) +
  expand_limits(y = -6) +
  scale_color_viridis_d() +
  labs(y = NULL, title = "Clustering with Single Linkage")

p_single
```



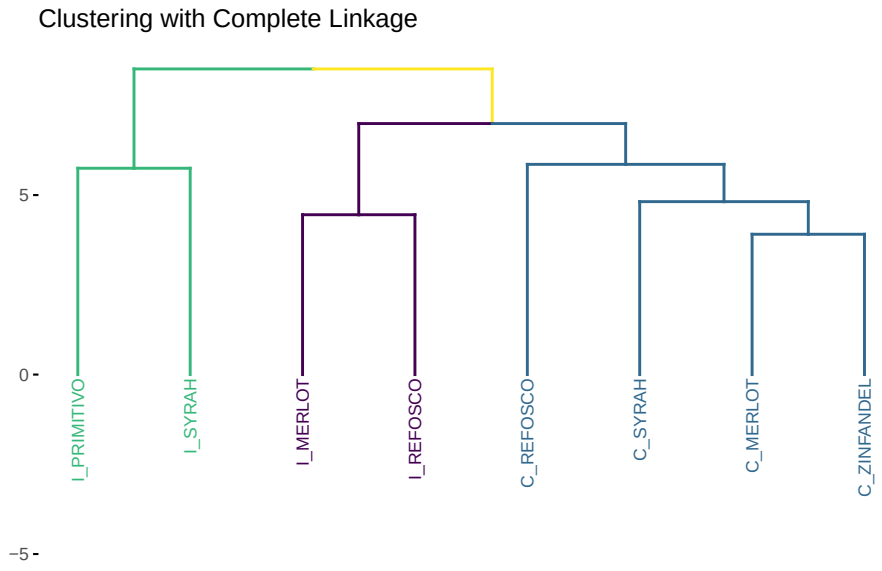
Notice the "greediness": large single group keeps adding a single new observation at each step of the algorithm, resulting in this characteristic "step" pattern. For most situations, single linkage is a not a recommended approach for clustering.

7.1.3 Complete Linkage

The complete linkage approach is just the opposite of the single linkage method: the distance between two clusters is the *maximum* distance between all two points a and b in clusters A and B, respectively. With this definition, the same iterative approach is carried out and the two closest clusters are merged at each step.

```
cluster_complete <-
  descriptive_distance %>%
```

```
  hclust(method = "complete")

p_complete <-
  cluster_complete %>%
  fviz_dend(k = 3, cex = 2/3) +
  expand_limits(y = -6) +
  labs(y = NULL, title = "Clustering with Complete Linkage") +
  scale_color_viridis_d()

p_complete
```



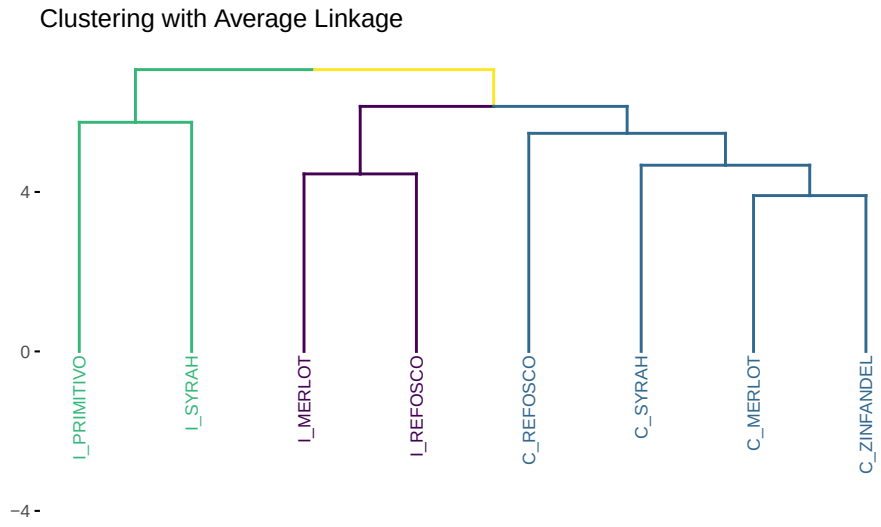
Intuitively, complete linkage avoids the "greediness" problem of single linkage. It is the default method used in `hclust()`: see `?hclust`.

7.1.4 Average Linkage

As the name implies, in the average linkage method, the distance between two clusters is defined as the average distance between all objects a_i in A and all objects b_j in B.

```
cluster_average <-
  descriptive_distance %>%
  hclust(method = "average")
```

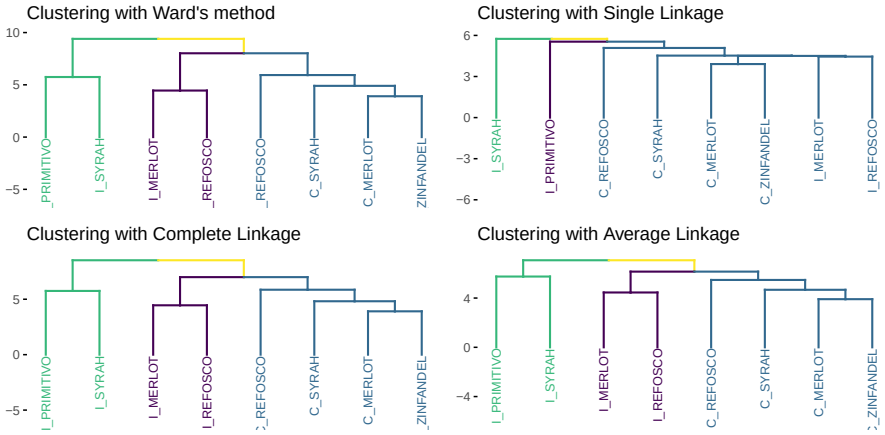
```
p_average <-
  cluster_average %>%
  fviz_dend(k = 3, cex = 2/3) +
  expand_limits(y = -6) +
  labs(title = "Clustering with Average Linkage", y = NULL) +
  scale_color_viridis_d()

p_average
```



7.1.5 Comparing methods

Let's look at the results of our cluster analyses side by side.

```
library(patchwork)

(p_ward + p_single) / (p_complete + p_average)
```



Only single linkage gives us very different results; the others are a matter of scaling. This could be quite different if we had a larger number of more dissimilar objects - recall our distance matrix:

```
descriptive_distance
```

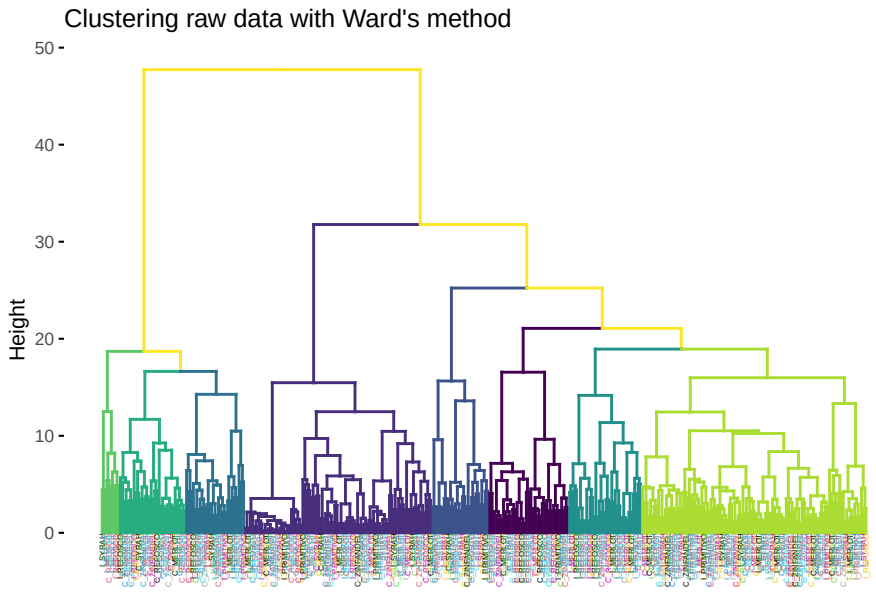
```
##             C_MERLOT C_REFOSCO  C_SYRAH C_ZINFANDEL I_MERLOT I_PRIMITIVO
## C_REFOSCO   5.083950
## C_SYRAH     4.525147  5.472269
## C_ZINFANDEL 3.909011  5.856348 4.817566
## I_MERLOT    4.503404  6.405151 6.153529    6.522035
## I_PRIMITIVO 6.129930  7.853063 5.998705    5.543431 7.842528
## I_REFOSCO   6.272963  6.992652 5.408757    6.917554 4.452965    8.514098
## I_SYRAH     6.911105  8.427799 6.359281    6.170264 7.546229    5.747512
##             I_REFOSCO
## C_REFOSCO
## C_SYRAH
## C_ZINFANDEL
## I_MERLOT
## I_PRIMITIVO
## I_REFOSCO
## I_SYRAH      7.537424
```

Not actually that much variation!

We can do the same thing with our individual observations.

```
individual_distances <-
  descriptive_data %>%
  unite(NJ, ProductName, NR, col = "ID") %>%
```

```
  mutate(across(where(is.numeric), ~ (. - mean(.)) / sd(.))) %>%
  column_to_rownames("ID") %>%
  dist()

clusters_individual <-
  individual_distances %>%
  hclust(method = "ward.D2")

# Here we drop the original unique IDs for just the ProductName
clusters_individual$labels <- str_extract(clusters_individual$labels, "[A-Z]_[A-Z]+")

p <- clusters_individual %>%
  fviz_dend(cex = 1/4,
            k = 8,
            label_cols = clusters_individual$labels %>% as.factor() %>% as.numeric()) +
  labs(title = "Clustering raw data with Ward's method") +
  scale_color_viridis_d()

p
```



We tried looking for 8 clusters (since there are 8 wines) and giving each label (the individual row observation representing a single judge rating a single wine sample) the color of the rated wine. It is obvious that the wines don't cluster together based on their sample ID.

7.2 Using cluster IDs

Above we played a little bit to see if, for the raw data, clustering provided a structure that mirrored product ID. But that was more for demonstration purposes than because it was a good data-analysis practice. More standard in sensory evaluation workflow would be to conduct a clustering analysis and then determine whether that cluster "explained" sensory variation in the base data.

From our original clustering results, we can pull out a tibble that tells us which sample belongs to which cluster:

```
# We use `cutree()` with `k = 3` to get 3 groups from the "tree" (clustering)
groups_from_ward <-
  cluster_ward %>%
  cutree(k = 3) %>%
  as_tibble(rownames = "ProductName") %>%
  rename(ward_group = value) %>%
  # It will be helpful to treat group membership as a nominal variable
  mutate(ward_group = factor(ward_group))

# The `dplyr::left_join()` function matches up the two tibbles based on the
# value of a shared column(s): in this case, `ProductName`
descriptive_data <-
  descriptive_data %>%
  left_join(groups_from_ward)

glimpse(descriptive_data)
```

```
## Rows: 336
## Columns: 24
## $ NJ              <fct> 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1331, 1400, ~
## $ ProductName     <chr> "C_MERLOT", "C_SYRAH", "C_ZINFANDEL", "C_REFOSCO", "I_~
## $ NR              <fct> 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, ~
## $ Red_berry       <dbl> 5.1, 5.6, 4.9, 5.0, 3.3, 5.7, 2.9, 3.2, 0.1, 1.6, 4.5,~
## $ Dark_berry      <dbl> 5.8, 1.9, 2.6, 1.9, 7.2, 3.6, 5.1, 6.0, 0.1, 0.7, 2.9,~
## $ Jam             <dbl> 2.1, 3.9, 1.4, 7.8, 0.5, 8.7, 8.7, 4.0, 0.2, 0.0, 0.3,~
## $ Dried_fruit     <dbl> 4.7, 1.2, 5.9, 0.6, 5.8, 1.9, 0.4, 0.7, 2.9, 6.4, 2.4,~
## $ Artificial_frui <dbl> 1.0, 7.9, 0.8, 6.6, 0.7, 7.4, 6.2, 4.1, 0.1, 0.1, 0.1,~
## $ Chocolate       <dbl> 2.9, 1.0, 2.0, 6.4, 2.1, 3.3, 3.4, 3.6, 0.2, 1.0, 0.2,~
## $ Vanilla         <dbl> 5.0, 8.3, 2.7, 5.5, 1.3, 6.9, 8.1, 4.8, 2.0, 0.8, 1.9,~
## $ Oak             <dbl> 5.0, 2.3, 5.6, 3.6, 2.1, 1.5, 1.8, 2.6, 3.0, 5.4, 6.1,~
## $ Burned          <dbl> 1.4, 1.8, 1.9, 3.2, 5.6, 0.2, 0.4, 4.7, 7.5, 5.1, 0.3,~
## $ Leather         <dbl> 2.3, 3.5, 4.3, 0.3, 6.5, 1.5, 4.1, 6.5, 0.7, 0.8, 0.2,~
## $ Earthy          <dbl> 0.6, 1.0, 0.6, 0.2, 4.7, 0.3, 0.5, 1.9, 0.7, 3.0, 1.3,~
## $ Spicy           <dbl> 3.2, 0.7, 1.4, 2.9, 0.7, 3.1, 0.7, 1.4, 0.3, 3.2, 3.1,~
```

```
## $ Pepper          <dbl> 5.4, 3.0, 4.1, 0.9, 2.8, 1.6, 3.6, 4.5, 0.1, 2.0, 0.9,~
## $ Grassy          <dbl> 2.1, 0.6, 3.6, 1.8, 3.8, 0.9, 2.3, 0.8, 0.1, 1.3, 0.4,~
## $ Medicinal       <dbl> 0.4, 2.2, 1.7, 0.2, 2.6, 0.5, 0.2, 3.8, 0.1, 2.1, 0.1,~
## $ `Band-aid`      <dbl> 0.4, 0.4, 0.1, 0.2, 5.1, 1.2, 0.2, 6.2, 0.1, 1.1, 0.1,~
## $ Sour            <dbl> 5.0, 9.7, 7.8, 8.3, 7.6, 7.2, 5.9, 6.3, 5.7, 6.4, 5.4,~
## $ Bitter          <dbl> 5.9, 5.2, 3.5, 3.0, 1.9, 9.8, 2.9, 0.2, 0.6, 2.9, 0.1,~
## $ Alcohol         <dbl> 9.0, 7.2, 4.7, 8.9, 2.8, 8.7, 1.6, 7.0, 1.6, 5.4, 4.9,~
## $ Astringent      <dbl> 8.7, 8.3, 5.0, 7.8, 5.9, 8.0, 2.6, 4.2, 5.5, 5.1, 5.9,~
## $ ward_group      <fct> 1, 1, 1, 1, 2, 3, 3, 2, 1, 1, 1, 1, 2, 3, 3, 2, 1, 1, ~
```

Now we have a `factor` identifying which cluster (from an HCA with Ward's Method and 3 groups) each wine belongs to. We can use this structure as a new possible input for M/ANOVA, like we did in MANOVA (Multivariate Analysis of Variance).

```
cluster_manova <-
  manova(formula = as.matrix(descriptive_data[, 4:23]) ~ ward_group,
         data = descriptive_data)

# We get a "significant" 1-way MANOVA result for the effect of cluster
# membership on overall sensory profile.
summary(cluster_manova, test = "W")
```

```
##             Df   Wilks approx F num Df den Df    Pr(>F)
## ward_group   2 0.52896   5.8867     40    628 < 2.2e-16 ***
## Residuals  333
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Let's review how to use nest() and map() functions to tidily apply 1-way ANOVA
# to each variable.

descriptive_data %>%
  # We are going to ignore (marginalize) reps and judges and even products
  select(-NR, -NJ, -ProductName) %>%
  pivot_longer(-ward_group,
               names_to = "descriptor",
               values_to = "rating") %>%
  nest(data = -descriptor) %>%
  # run 1-way ANOVAs
  mutate(anova_results = map(.x = data,
                             .f = ~aov(rating ~ ward_group, data = .))) %>%
  # use broom::tidy() to get tibble-ized summaries out of the `aov` objects
  transmute(descriptor,
            tidy_summary = map(anova_results, broom::tidy)) %>%
```

```
  unnest(everything()) %>%
  filter(term  != "Residuals",
         p.value < 0.05) %>%
  mutate(across(where(is.numeric), ~round(., 3))) %>%
  print(n = 13)
```

```
## # A tibble: 13 x 7
##    descriptor      term          df sumsq meansq statistic p.value
##    <chr>           <chr>      <dbl> <dbl>  <dbl>     <dbl>   <dbl>
##  1 Red_berry       ward_group     2  49.7  24.9       3.97   0.02
##  2 Dark_berry      ward_group     2  81.5  40.8       5.54   0.004
##  3 Jam             ward_group     2 243.  122.       23.7    0
##  4 Artificial_frui ward_group     2 134.   66.9      18.7    0
##  5 Chocolate       ward_group     2  20.6  10.3       3.41   0.034
##  6 Vanilla         ward_group     2  59.6  29.8       7.74   0.001
##  7 Oak             ward_group     2  37.5  18.8       4.89   0.008
##  8 Burned          ward_group     2 106.   53.1      13.7    0
##  9 Leather         ward_group     2  70.2  35.1       8.90   0
## 10 Earthy          ward_group     2  20.0  10.0       4.18   0.016
## 11 Grassy          ward_group     2  16.6   8.28      4.07   0.018
## 12 Medicinal       ward_group     2 109.   54.4      14.7    0
## 13 Band-aid        ward_group     2  96.5  48.3      12.4    0
```

We find that 13 descriptors have significantly different means for the 3 different clusters. This seems to contradict HGH's results–she found 7 in the original **R Opus**. I am not quite sure what the difference could be here; when I reran the code from the actual original **R Opus** my results match the tidy workflow shown above (not shown). I wonder if some data got lost somewhere in the original?

7.3 K-means clustering

In hierarchical clustering, we use an iterative process to merge our items into groups–if we let the process run long enough we end up with a single group. We decide on the "right" number of groups by examining results like the *dendrogram* produced from the set of merges throughout the process (e.g., we apply `cutree(k = 3)`).

If we had some idea *a priori* of how many groups we were expecting, we could use some alternative processes. The most popular–and possibly intuitive–of these is *k-means clustering*. The *k* in "k-means" stands in for the same thing it does in the `k =` argument in `cutree()`: how many groups are we looking for. However, in k-means clustering, we don't proceed hierarchically. Instead,

we start with some initial k "seeds" - the initial 1-item groups. These can be chosen at random or purposively, although if the former it is important to note that the choice of seed can be influential on the solution, and so multiple runs of k-means clustering may produce different results.

Once the k seeds are chosen, every other item that is to be clustered is assigned to the group for which it is closest to the seed (again, "closest" can be defined by any distance metric). Once a group has more than 1 item, its centroid (the "mean" in the name "k-means") replaces the initial seed for distance measurements. After an initial run, it is possible that items are in the "wrong" group–that a new, different group centroid is closer to them than is their current group centroid. These items are re-assigned, centroids are re-calculated, and the procedure iterates until no items are re-assigned. This reassignment is not possible with hierarchical methods, and so k-means can sometimes find better results. As might be imagined, different choices of methods for calculating centroid and for distance can affect the results of k-means clustering.

A common workflow is to use HCA to get an idea of the appropriate k for k-means clustering, and then to use k-means clustering to get a final, optimized clustering solution. We will take this approach, using `k = 3`.

```
clusters_kmeans <-
  descriptive_means_scaled %>%
  column_to_rownames("ProductName") %>%
  kmeans(centers = 3)

clusters_kmeans$cluster
```

```
##    C_MERLOT   C_REFOSCO     C_SYRAH C_ZINFANDEL    I_MERLOT I_PRIMITIVO
##           1           1           1           1           2           3
##   I_REFOSCO     I_SYRAH
##           2           3
```

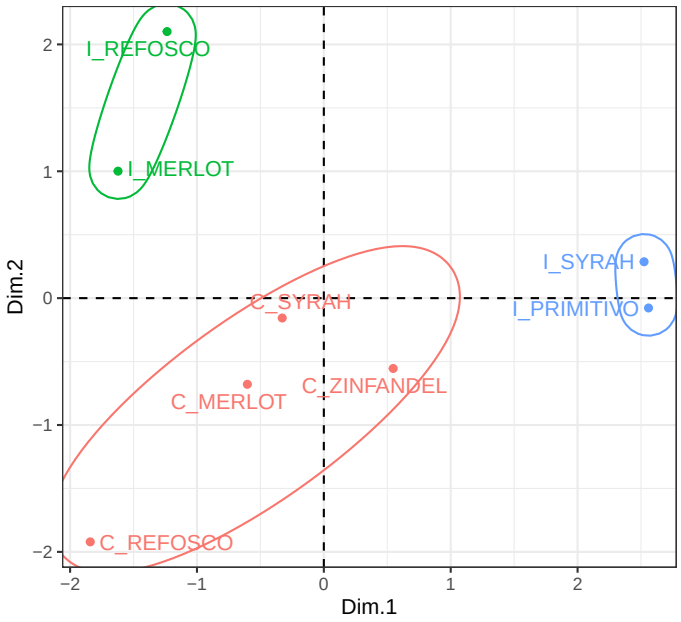
```
clusters_kmeans$centers %>%
  as_tibble(rownames = "cluster")
```

```
## # A tibble: 3 x 21
##   cluster Red_berry Dark_berry    Jam Dried_fruit Artificial_frui Chocolate
##   <chr>       <dbl>      <dbl>  <dbl>       <dbl>           <dbl>     <dbl>
## 1 1          -0.456     -0.328 -0.379     -0.139           -0.515     0.444
## 2 2          -0.423     -0.629 -0.729      0.255           -0.544    -1.07
## 3 3           1.34       1.29   1.49       0.0234           1.58      0.179
## # i 14 more variables: Vanilla <dbl>, Oak <dbl>, Burned <dbl>, Leather <dbl>,
## #   Earthy <dbl>, Spicy <dbl>, Pepper <dbl>, Grassy <dbl>, Medicinal <dbl>,
## #   `Band-aid` <dbl>, Sour <dbl>, Bitter <dbl>, Alcohol <dbl>, Astringent <dbl>
```

HGH shows how to replicate the same workflow as we did above with the HCA results: using the group assignments to run a 1-way M/ANOVA to determine which attributes vary significantly. I am going to leave this as an exercise to the reader, but we'll instead look at how we might visualize these results, since we can't draw a dendrogram in the same way as an HCA.

We'll first run a simple means PCA to get the product score plot, and then we'll draw some hulls around the groups.

```
means_pca <-
  descriptive_means %>%
  column_to_rownames("ProductName") %>%
  FactoMineR::PCA(scale.unit = FALSE, graph = FALSE)

# Join the PCA scores to the k-means cluster IDs
p1 <-
  means_pca$ind$coord %>%
  as_tibble(rownames = "ProductName") %>%
  left_join(
    clusters_kmeans$cluster %>%
      as_tibble(rownames = "ProductName")
  ) %>%
  mutate(value = factor(value)) %>%
  # And plot!
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_hline(yintercept = 0, linetype = 2) +
  geom_vline(xintercept = 0, linetype = 2) +
  geom_point(aes(color = value)) +
  ggrepel::geom_text_repel(aes(color = value, label = ProductName)) +
  ggforce::geom_mark_ellipse(aes(color = value)) +
  theme_bw() +
  coord_equal() +
  theme(legend.position = "none")
p1
```



Looks like those California wines end up grouped together even though there is a fair bit of spread. I looked into the PC3 to see if that explained it, but they are even more spread there (not shown–see if you can make the small code tweak to do so). I think this is a case where arguably 3 clusters is not a good solution. We might consider examining some measures of cluster fit, perhaps through something like the `NbClust` package, to determine if we should consider other solutions.

7.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] patchwork_1.1.2  factoextra_1.0.7 here_1.0.1       lubridate_1.9.2
##  [5] forcats_1.0.0    stringr_1.5.0    dplyr_1.1.2      purrr_1.0.1
##  [9] readr_2.1.4      tidyr_1.3.0      tibble_3.2.1     ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.0     viridisLite_0.4.2    farver_2.1.1
##  [4] viridis_0.6.4        fastmap_1.1.1        tweenr_2.0.2
##  [7] digest_0.6.33        timechange_0.2.0     estimability_1.4.1
## [10] lifecycle_1.0.3      cluster_2.1.4        multcompView_0.1-9
## [13] magrittr_2.0.3       compiler_4.3.1       rlang_1.1.1
## [16] tools_4.3.1          utf8_1.2.3           yaml_2.3.7
## [19] knitr_1.43           ggsignif_0.6.4       labeling_0.4.3
## [22] htmlwidgets_1.6.2    bit_4.0.5            scatterplot3d_0.3-44
## [25] abind_1.4-5          withr_2.5.0          grid_4.3.1
## [28] polyclip_1.10-4      fansi_1.0.4          ggpubr_0.6.0
## [31] xtable_1.8-4         colorspace_2.1-0     emmeans_1.8.7
## [34] scales_1.2.1         MASS_7.3-60          flashClust_1.01-2
## [37] cli_3.6.1            mvtnorm_1.2-2        rmarkdown_2.23
## [40] crayon_1.5.2         generics_0.1.3       rstudioapi_0.15.0
## [43] tzdb_0.4.0           ggforce_0.4.1        parallel_4.3.1
## [46] vctrs_0.6.3          carData_3.0-5        bookdown_0.37
## [49] car_3.1-2            hms_1.1.3            bit64_4.0.5
## [52] rstatix_0.7.2        ggrepel_0.9.3        FactoMineR_2.8
## [55] dendextend_1.17.1    glue_1.6.2           DT_0.28
## [58] stringi_1.7.12       gtable_0.3.4         munsell_0.5.0
## [61] pillar_1.9.0         htmltools_0.5.6      R6_2.5.1
## [64] rprojroot_2.0.3      vroom_1.6.3          evaluate_0.21
## [67] lattice_0.21-8       highr_0.10           backports_1.4.1
## [70] leaps_3.1            broom_1.0.5          Rcpp_1.0.11
## [73] coda_0.19-4          gridExtra_2.3        xfun_0.39
## [76] pkgconfig_2.0.3
```

Chapter 8

Multidimensional Scaling (MDS)

Multidimensional scaling (MDS) is a method of representing the distances among a set of items or observations. These distances can come originally from a high-dimensional space (such as the 20-dimensional space that defines our DA data), or from a low-dimensional space. Regardless, the goal of MDS is to find a low-dimensional (2- or 3-dimensional, usually) representation of the items in which the distances among the items is a close approximation of the original distance matrix.

You may notice that MDS operates on the same input data as the clustering techniques we just explored in the previous chapter. In clustering, the goal is not to find a metric or spatial arrangement of objects, but instead to determine some smaller number of *clusters* or *groups* that summarize or explain the observed distances among our samples. In MDS, the goal is to provide a visual and spatial representation of the distances.

In the **R Opus**, HGH applied MDS to the DA data. We'll explore the same approach once we review the key concept of metric and non-metric MDS.

8.1 Metric vs non-metric

Without going into details, MDS comes in two "flavors": metric and non-metric. In **metric MDS**, the distances among objects are taken as "metric" or measurement data: the units themselves are meaningful (another way of saying this is that they are *interval* or *ratio* data). In this case, the underlying algorithm for MDS is very similar to PCA, in which an eigendecomposition provides the optimal solution for finding a low-dimensional representation that explains the

majority of the variations in distances among samples. Most sensory data is suitable for metric MDS, and in fact the DISTATIS method we will use next is predicated on a proof that distances even among sorting groups are metric (specifically, Euclidean).

For non-metric MDS, the assumption of the distances being meaningful metrics is relaxed; rather, we assume that the provided distances are only meaningful insofar as they represent ranks (they are *ordinal* data). In this case, non-metric MDS uses a gradient-descent approach to find an approximation by extracting orthogonal dimensions while minimizing a criterion called $STRESS$. Typically, we won't have to concern ourselves too much with non-metric MDS, but it could be useful when examining results from, for example, flash profiling (in which samples are ranked but not rated).

8.2 Metric MDS

As usual, we begin by loading our required packages and the data.

```
library(tidyverse)
library(here)

descriptive_data <-
  read_csv(here("data/torriDAFinal.csv")) %>%
  # We'll make some factors as usual
  mutate(across(1:3, ~as.factor(.)))

descriptive_means <-
  descriptive_data %>%
  # Get the means for each wine and descriptor
  group_by(ProductName) %>%
  summarize(across(where(is.numeric), ~mean(.))) %>%
  # Scale the data to unit variance
  mutate(across(where(is.numeric), ~(. - mean(.)) / sd(.)))
```

Note that as in the previous chapter, HGH scales (normalizes) the data to $\bar{x} = 0$ and $s = 1$, which is a choice rather than a necessity for these methods. We will follow for consistency.

We then need to get the distances among all of our observations.

```
descriptive_distances <-
  descriptive_means %>%
  # get the rownames as we go back to older functions
  column_to_rownames("ProductName") %>%
  dist(method = "euclidean")
```

The `cmdscale()` function in the base `R stats` package will do metric MDS for us.

```
metric_mds <-
  descriptive_distances %>%
  # `k` defines the # of dimensions to extract, `eig` returns the eigenvalues
  # for the spectral (eigen)decomposition
  cmdscale(k = 2, eig = TRUE)

metric_mds
```

```
## $points
##                     [,1]       [,2]
## C_MERLOT    -0.66017528 -1.4236819
## C_REFOSCO   -2.05394351 -3.2971758
## C_SYRAH     -0.08165706 -0.4392567
## C_ZINFANDEL  1.20071498 -1.5563951
## I_MERLOT    -2.87771919  1.5668345
## I_PRIMITIVO  4.11149279 -0.4593610
## I_REFOSCO   -3.14226246  2.8789771
## I_SYRAH      3.50354973  2.7300588
##
## $eig
## [1] 5.343721e+01 3.392126e+01 1.755089e+01 1.363281e+01 1.225350e+01
## [6] 7.141861e+00 2.062458e+00 1.796329e-15
##
## $x
## NULL
##
## $ac
## [1] 0
##
## $GOF
## [1] 0.6239891 0.6239891
```

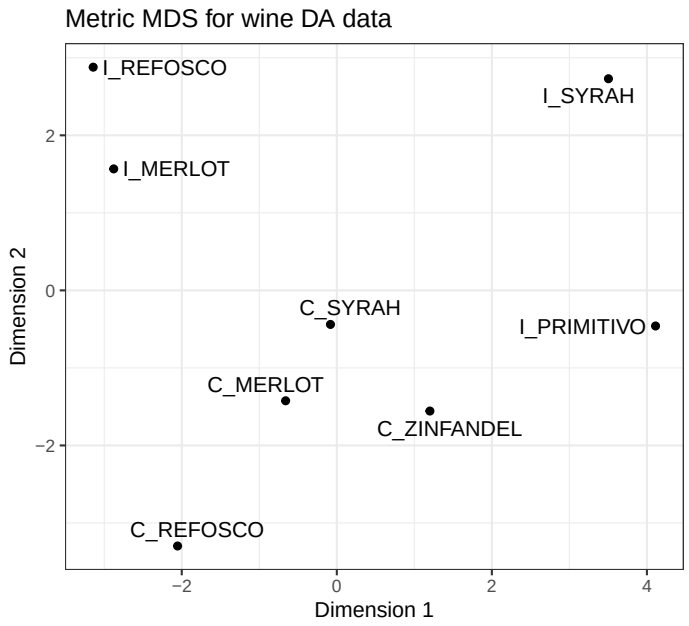
At this point you are familiar with our plot and wrangle approach - we're going to get that `$points` table and pipe it into `ggplot2` to make a nicer looking plot.

```
p_metric <-
  metric_mds$points %>%
  as_tibble(rownames = "sample") %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = sample)) +
  labs(x = "Dimension 1", y = "Dimension 2",
```

```
       title = "Metric MDS for wine DA data") +
  coord_fixed() +
  theme_bw()
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
## `.name_repair` is omitted as of tibble 2.0.0.
## i Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

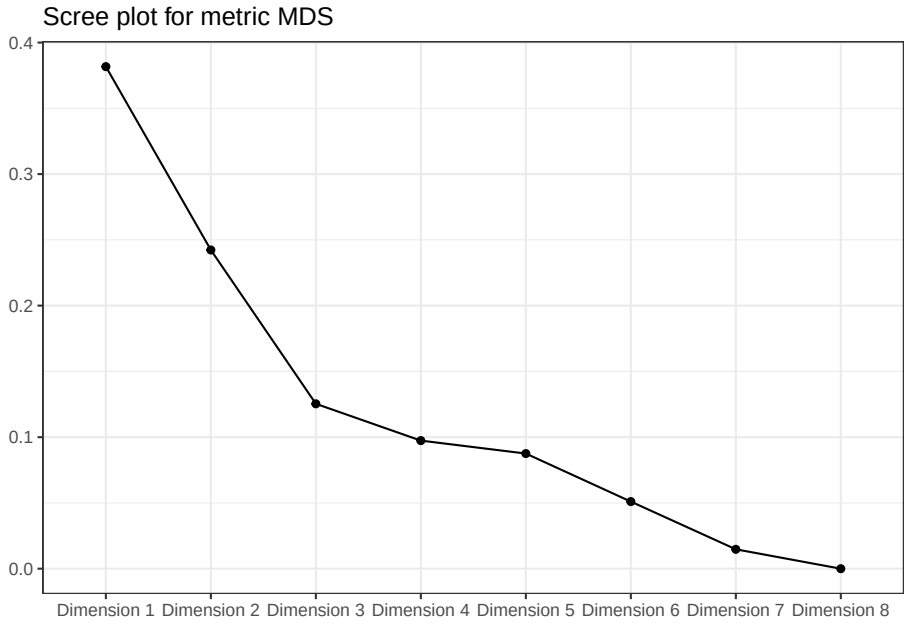
```
p_metric
```



We can also examine a scree-plot of the eigenvalues to get a (qualitative) view of this solution.

```
metric_mds$eig %>%
  as_tibble() %>%
  mutate(dim = str_c("Dimension ", row_number()),
         value = value / sum(value)) %>%
  ggplot(aes(x = dim, y = value)) +
  geom_point() +
  geom_line(aes(group = 1)) +
  theme_bw() +
```

```
  labs(x = NULL, y = NULL,
       title = "Scree plot for metric MDS")
```



We can see that, arguably, a 2-dimensional solution is not great for capturing the distances among these samples. See that bend *after* the 3rd dimension? That's where we'd often want to put a cut. But this is a little academic - in practice we often only look at the first dimensions.

Let's compare the distances from our MDS solution to the actual distances:

```
# Actual distances
descriptive_distances
```

```
##             C_MERLOT C_REFOSCO  C_SYRAH C_ZINFANDEL I_MERLOT I_PRIMITIVO
## C_REFOSCO   5.083950
## C_SYRAH     4.525147  5.472269
## C_ZINFANDEL 3.909011  5.856348 4.817566
## I_MERLOT    4.503404  6.405151 6.153529    6.522035
## I_PRIMITIVO 6.129930  7.853063 5.998705    5.543431 7.842528
## I_REFOSCO   6.272963  6.992652 5.408757    6.917554 4.452965    8.514098
## I_SYRAH     6.911105  8.427799 6.359281    6.170264 7.546229    5.747512
##             I_REFOSCO
## C_REFOSCO
## C_SYRAH
```

```
## C_ZINFANDEL
## I_MERLOT
## I_PRIMITIVO
## I_REFOSCO
## I_SYRAH      7.537424
```

```
# Approximate distances
metric_mds$points %>%
  dist(method = "euclidean")
```

```
##             C_MERLOT C_REFOSCO  C_SYRAH C_ZINFANDEL I_MERLOT I_PRIMITIVO
## C_REFOSCO   2.335074
## C_SYRAH     1.141830  3.472408
## C_ZINFANDEL 1.865617  3.690951 1.700728
## I_MERLOT    3.722995  4.933275 3.441274    5.136944
## I_PRIMITIVO 4.868134  6.787179 4.193198    3.110645 7.276988
## I_REFOSCO   4.967256  6.271308 4.514198    6.207574 1.338544    7.985078
## I_SYRAH     5.881341  8.198371 4.785214    4.865875 6.486423    3.246844
##             I_REFOSCO
## C_REFOSCO
## C_SYRAH
## C_ZINFANDEL
## I_MERLOT
## I_PRIMITIVO
## I_REFOSCO
## I_SYRAH      6.647480
```

Here we can see that the distances are indeed only approximate (and not that good an approximation).

Would this get better if we went up to 3 dimensions? We can answer that pretty easily:

```
descriptive_distances %>%
  cmdscale(k = 3, eig = TRUE) %>%
  .$points %>%
  dist()
```

```
##             C_MERLOT C_REFOSCO  C_SYRAH C_ZINFANDEL I_MERLOT I_PRIMITIVO
## C_REFOSCO   3.882087
## C_SYRAH     3.867494  3.522813
## C_ZINFANDEL 2.323006  4.070842 2.869323
## I_MERLOT    3.775222  6.182909 5.523805    5.516157
## I_PRIMITIVO 4.910980  7.217195 5.183820    3.196724 7.387510
## I_REFOSCO   5.696284  6.279119 4.604395    6.364382 3.667013    8.267098
```

```
## I_SYRAH     6.453639  8.210405 4.896535    5.029587 7.269797    3.818459
##             I_REFOSCO
## C_REFOSCO
## C_SYRAH
## C_ZINFANDEL
## I_MERLOT
## I_PRIMITIVO
## I_REFOSCO
## I_SYRAH      6.648776
```

It does look like we're getting closer to the original distances as we go up in dimensionality - the algorithm is able to find "space" to place the points further apart.

Let's examine one more thing that will lead us to *non*-metric MDS: does our 2-dimensional MDS solution capture the correct *rank order* of distances? In other words, is each pairwise distance in the MDS approximation in the same *order* as it would be in the actual distance matrix?

```
# I am getting sick of manually tidying distance matrices, so we're going to
# enlist the `widyr` package to automate this.
library(widyr)

# Of course this creates a new set of needs, including getting our data into a
# tidy format to start with
tidy_descriptive_distances <-
  descriptive_means %>%
  pivot_longer(-ProductName,
               names_to = "descriptor",
               values_to = "rating") %>%
  arrange(ProductName) %>%
  # The distance between `ProductName`, based on `descriptor`, with values
  # stored in `rating`
  pairwise_dist(item = ProductName, feature = descriptor, value = rating,
                upper = FALSE, method = "euclidean")

# We'll do the same thing with our MDS results
tidy_mds_distances <-
  metric_mds$points %>%
  as_tibble(rownames = "product") %>%
  arrange(product) %>%
  pivot_longer(-product) %>%
  pairwise_dist(item = product, feature = name, value = value,
                upper = FALSE, method = "euclidean")
```

We can now look at the ranks of pairwise distances and see how many are

misaligned.

```
# We will rank the pairwise distances from the original data
tidy_descriptive_distances %>%
  unite(item1, item2, col = "items", sep = " <--> ") %>%
  transmute(observed_rank = items,
            distance_rank = dense_rank(distance)) %>%
  arrange(distance_rank) %>%
  # And now we'll line this up with the MDS results
  left_join(
    tidy_mds_distances %>%
      unite(item1, item2, col = "items", sep = " <--> ") %>%
      transmute(mds_rank = items,
                distance_rank = dense_rank(distance)) %>%
      arrange(distance_rank),
    # This is the line that lines them up (`by = `)
    by = "distance_rank"
  ) %>%
  # And finally we'll check which are the same
  mutate(match = observed_rank == mds_rank) %>%
  count(match)
```

```
## # A tibble: 2 x 2
##   match     n
##   <lgl> <int>
## 1 FALSE    21
## 2 TRUE      7
```

That's actually very bad! We see only 1/4 of the original distances captured by the 2-dimensional MDS solution. Yikes.

Let's take a look and see if a non-metric MDS can do better.

8.3 Non-metric MDS

In non-metric MDS, we assume that the observed distances are non-metric: they only represent an *ordination* of the distances among the items. An example would be, in our actual data, that the observed distances represent estimates or opinions from our panelists:

```
tidy_descriptive_distances %>%
  arrange(distance)
```

```
## # A tibble: 28 x 3
##    item1       item2       distance
##    <fct>       <fct>          <dbl>
##  1 C_MERLOT    C_ZINFANDEL     3.91
##  2 I_MERLOT    I_REFOSCO       4.45
##  3 C_MERLOT    I_MERLOT        4.50
##  4 C_MERLOT    C_SYRAH         4.53
##  5 C_SYRAH     C_ZINFANDEL     4.82
##  6 C_MERLOT    C_REFOSCO       5.08
##  7 C_SYRAH     I_REFOSCO       5.41
##  8 C_REFOSCO   C_SYRAH         5.47
##  9 C_ZINFANDEL I_PRIMITIVO     5.54
## 10 I_PRIMITIVO I_SYRAH         5.75
## # i 18 more rows
```

The closest samples are `C_MERLOT` and `C_ZINFANDEL`, and the second closest are `I_MERLOT` and `I_REFOSCO`. If our distances are merely ordinations, we can only say that–we can't compare the actual "differences of distances" as we could if these were metric. In the metric case, we could subtract the distance between `C_MERLOT` and `C_ZINFANDEL` and `I_MERLOT` and `I_REFOSCO` ($\approx 4.5 - 3.9 \approx 0.6$) and say that the difference between those distances is larger than that between the second and third smallest distances (`I_MERLOT` and `I_REFOSCO` vs `C_MERLOT` and `I_MERLOT`, which is $\approx 4.5 - 4.45 \approx 0$). So we could say something like "the difference between the two closest pairs of samples is quite large, but the difference between the next two closest is approximately the same". We can't do that with an ordination, because we only know the relative ranks:

```
tidy_descriptive_distances %>%
  transmute(item1, item2,
            distance_rank = dense_rank(distance)) %>%
  arrange(distance_rank)
```

```
## # A tibble: 28 x 3
##    item1       item2       distance_rank
##    <fct>       <fct>               <int>
##  1 C_MERLOT    C_ZINFANDEL             1
##  2 I_MERLOT    I_REFOSCO               2
##  3 C_MERLOT    I_MERLOT                3
##  4 C_MERLOT    C_SYRAH                 4
##  5 C_SYRAH     C_ZINFANDEL             5
##  6 C_MERLOT    C_REFOSCO               6
##  7 C_SYRAH     I_REFOSCO               7
##  8 C_REFOSCO   C_SYRAH                 8
##  9 C_ZINFANDEL I_PRIMITIVO             9
## 10 I_PRIMITIVO I_SYRAH                10
```

```
## # i 18 more rows
```

We can no longer know if the difference between 1st and 2nd place (so to speak, of `distance_rank`) is the same as the difference between 2nd and 3rd place, and so on. We only know that `C_MERLOT` and `C_ZINFANDEL` are closer than `I_MERLOT` and `I_REFOSCO`, and so on.

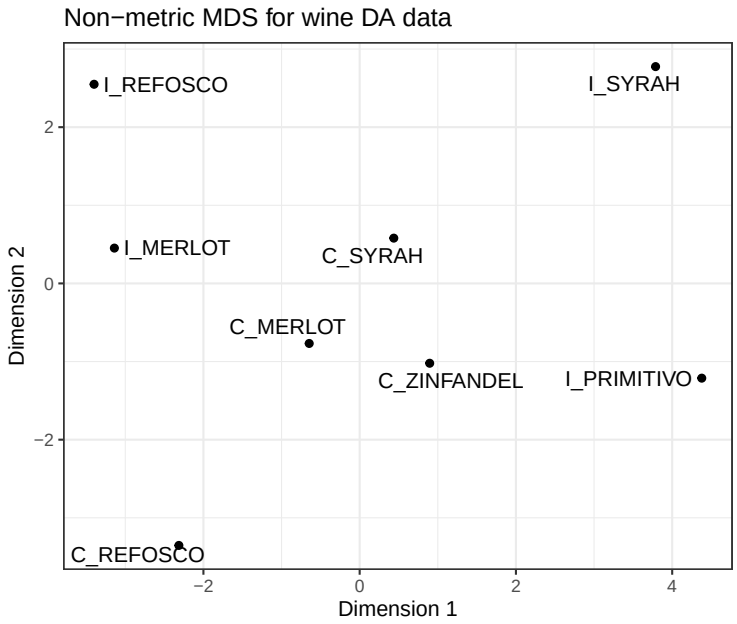
OK, with that out of the way, let's take a look at non-metric MDS. For this, we will use the `MASS` package. `MASS` is bundled with `R`, but is not loaded by default. It contains many useful core functions for statistics, especially non-linear and multivariate stats. You can learn more about it at the authors' website.

```
library(MASS)

# the `isoMDS()` function will be what we use.

nonmetric_mds <-
  isoMDS(d = descriptive_distances, k = 2)
```

```
## initial  value 9.444389
## iter   5 value 7.065995
## iter  10 value 6.940918
## final  value 6.928604
## converged
```

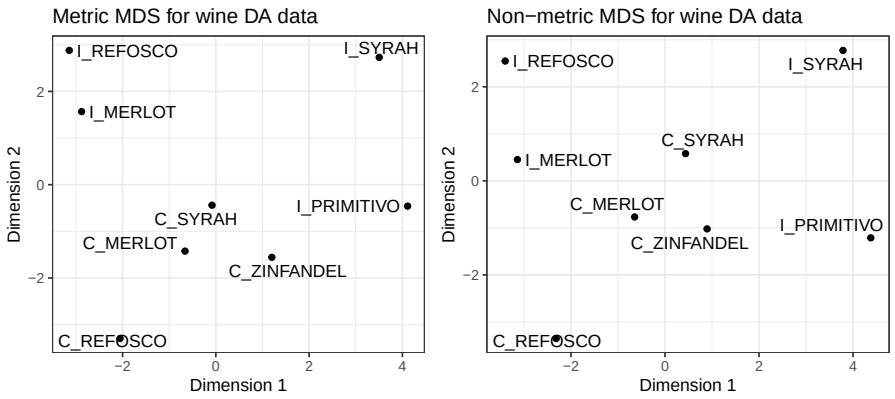
As mentioned, non-metric MDS is an iterative optimization problem. We are told that the solution has "converged" because the $STRESS$ value is no longer going down with subsequent iterations. Let's look at our results.

```
p_nonmetric <-
  nonmetric_mds$points %>%
  as_tibble(rownames = "sample") %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = sample)) +
  labs(x = "Dimension 1", y = "Dimension 2",
       title = "Non-metric MDS for wine DA data") +
  coord_fixed() +
  theme_bw()

p_nonmetric
```



I'd like to compare our metric and non-metric configurations. To do so we can use the `patchwork` package, as we have before.

```
library(patchwork)

p_metric + p_nonmetric
```



Note that the configurations are quite similar, but with some notable sifts (e.g.,

the positioning of `I_MERLOT` pops out). This makes sense: if you read the `?isoMDS` documentation, you'll see that the initial configuration for the non-metric MDS *is* the metric MDS solution, so this approach starts with our metric MDS configuration and improves on it.

Let's see if we've improved the representation of our relative ranks for distances:

```
tidy_nonmetric_mds_distances <-
  nonmetric_mds$points %>%
  as_tibble(rownames = "product") %>%
  arrange(product) %>%
  pivot_longer(-product) %>%
  pairwise_dist(item = product, feature = name, value = value,
                upper = FALSE, method = "euclidean")

tidy_descriptive_distances %>%
  unite(item1, item2, col = "items", sep = " <--> ") %>%
  transmute(observed_rank = items,
            distance_rank = dense_rank(distance)) %>%
  arrange(distance_rank) %>%
  # And now we'll line this up with the MDS results
  left_join(
    tidy_nonmetric_mds_distances %>%
      unite(item1, item2, col = "items", sep = " <--> ") %>%
      transmute(mds_rank = items,
                distance_rank = dense_rank(distance)) %>%
      arrange(distance_rank),
    # This is the line that lines them up (`by = `)
    by = "distance_rank"
  ) %>%
  # And finally we'll check which are the same
  mutate(match = observed_rank == mds_rank) %>%
  count(match)
```

```
## # A tibble: 2 x 2
##   match     n
##   <lgl> <int>
## 1 FALSE    25
## 2 TRUE      3
```

It certainly doesn't seem like we have! Yikes! I think a higher-dimensional solution is certainly required to adequately capture the configuration of these products.

8.4 Wrap up

To be fair, saying "Yikes!" is probably a little overblown. If we examined the tables above I used to compare the observed and estimated ranks in more detail, we'd see that the variation is quite small, for example, for the non-metric data:

```
## # A tibble: 28 x 4
##    observed_rank                distance_rank mds_rank                      match
##    <chr>                               <int> <chr>                         <lgl>
##  1 C_MERLOT <--> C_ZINFANDEL               1 C_MERLOT <--> C_ZINFANDEL     TRUE
##  2 I_MERLOT <--> I_REFOSCO                 2 C_SYRAH <--> C_ZINFANDEL      FALSE
##  3 C_MERLOT <--> I_MERLOT                  3 C_MERLOT <--> C_SYRAH         FALSE
##  4 C_MERLOT <--> C_SYRAH                   4 I_MERLOT <--> I_REFOSCO       FALSE
##  5 C_SYRAH <--> C_ZINFANDEL                5 C_MERLOT <--> I_MERLOT        FALSE
##  6 C_MERLOT <--> C_REFOSCO                 6 C_MERLOT <--> C_REFOSCO       TRUE
##  7 C_SYRAH <--> I_REFOSCO                  7 C_ZINFANDEL <--> I_PRIMITIVO FALSE
##  8 C_REFOSCO <--> C_SYRAH                  8 C_SYRAH <--> I_MERLOT         FALSE
##  9 C_ZINFANDEL <--> I_PRIMITIVO            9 C_REFOSCO <--> I_MERLOT       FALSE
## 10 I_PRIMITIVO <--> I_SYRAH               10 C_REFOSCO <--> C_ZINFANDEL    FALSE
## # i 18 more rows
```

As we can see, while indeed there are rank reversals, they are relatively minor: it isn't like samples that are close in the actual distance matrix are ending up being very far in the non-metric MDS 2-dimensional solution.

8.5 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
```

```
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] patchwork_1.1.2 MASS_7.3-60     widyr_0.1.5     here_1.0.1
##  [5] lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2
##  [9] purrr_1.0.1     readr_2.1.4     tidyr_1.3.0     tibble_3.2.1
## [13] ggplot2_3.4.3   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.3        generics_0.1.3    lattice_0.21-8    stringi_1.7.12
##  [5] hms_1.1.3         digest_0.6.33     magrittr_2.0.3    evaluate_0.21
##  [9] grid_4.3.1        timechange_0.2.0  bookdown_0.37     fastmap_1.1.1
## [13] plyr_1.8.8        Matrix_1.6-0      rprojroot_2.0.3   ggrepel_0.9.3
## [17] backports_1.4.1   fansi_1.0.4       scales_1.2.1      cli_3.6.1
## [21] rlang_1.1.1       crayon_1.5.2      bit64_4.0.5       munsell_0.5.0
## [25] withr_2.5.0       yaml_2.3.7        tools_4.3.1       parallel_4.3.1
## [29] reshape2_1.4.4    tzdb_0.4.0        colorspace_2.1-0  broom_1.0.5
## [33] vctrs_0.6.3       R6_2.5.1          lifecycle_1.0.3   bit_4.0.5
## [37] vroom_1.6.3       pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.4
## [41] glue_1.6.2        Rcpp_1.0.11       highr_0.10        xfun_0.39
## [45] tidyselect_1.2.0  rstudioapi_0.15.0 knitr_1.43        farver_2.1.1
## [49] htmltools_0.5.6   labeling_0.4.3    rmarkdown_2.23    compiler_4.3.1
```

Chapter 9

DISTATIS

DISTATIS is the "discriminant sorting" version of STATIS, which according to Abdi et al. [2012, 125] is a French acronym for

> "Structuration des Tableaux à Trois Indices de la Statistique" (which can, approximately, be translated as "structuring three way statistical tables").

Whereas STATIS is a methodology that is closely related to Multiple Factor Analysis [MFA: Abdi et al., 2013], DISTATIS is a slght elaboration of the method primarily for analyzing the results of **sorting tasks**. In a sorting task, subjects receive a set of samples all at the same time (simultaneous sample presentation) and are asked to sort them into groups. Typically, subjects are not given specific criteria for sorting, although variations of the method do exist that specify this. Also typically, with n samples, subjects are told to make between 2 and $n-1$ groups (i.e., they cannot say all samples are identical or all samples are different). With this simple set of instructions, a set of 15-25 subjects can evaluate a set of samples in a relatively efficient fashion. The results are often not as well-defined or discriminating as traditional profiling methods (Descriptive Analysis, for the most part), but as a "first look" into the structure of a sample set, sorting tasks are often quite useful.

Two further points are worth mentioning:

1. DISTATIS can be used on any dissimilarity data, not just that from sorting tasks. It also does not require that the dissimilarity data be *binary* (as it is in a sorting task: in a group or not in a group). For example, DISTATIS can be used with the results of hierarchical or multiple sorting [Courcoux et al., 2023]. Recently, "free linking" tasks, in which pairwise-similarity is judged among a set of samples (as opposed to group-wise similarity),

have also been analyzed successfully with DISTATIS: these results are also non-binary similarities.
2. DISTATIS is closely related to MDS: it is an eigendecomposition on a set of dissimilarities. It differs primarily in data-preprocessing and in the generation of additional analytical outputs, which are often useful. The biplots produced by MDS and DISTATIS will usually be almost identical (see below).

With that said, let's launch into our application of DISTATIS.

9.1 The dataset: sorting wines by color

We're actually going to use another dataset for this analysis, not the results of the DA we've been analyzing so far. We're going to load the `sorting_r1` dataset, which according to HGH is

> color data that came from sorting the wines into similarly colored groups.

Thus, the data pertain to the same wine, but is sorted into groups *by appearance* by 15 panelists.

```
library(tidyverse)
library(here)

sorting_data <- read_csv(here("data/sorting_r1.csv"))

sorting_data
```

```
## # A tibble: 8 x 16
##   wine      `263` `1331` `1400` `1401` `1402` `1404` `1405` `1408` `1409` `1412`
##   <chr>     <chr> <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>  <chr>
## 1 I_REFOSCO G6    G3     G5     G4     G5     G4     G2     G2     G1     G1
## 2 I_MERLOT  G1    G3     G4     G3     G2     G3     G4     G3     G1     G2
## 3 I_SYRAH   G5    G4     G2     G1     G1     G2     G1     G4     G1     G2
## 4 I_PRIMIT~ G2    G1     G3     G1     G4     G5     G3     G4     G3     G2
## 5 C_SYRAH   G3    G2     G4     G4     G5     G6     G1     G3     G2     G4
## 6 C_REFOSCO G4    G3     G1     G2     G5     G5     G2     G1     G2     G3
## 7 C_MERLOT  G1    G2     G3     G3     G3     G1     G4     G3     G2     G2
## 8 C_ZINFAN~ G1    G5     G3     G3     G2     G1     G1     G3     G1     G2
## # i 5 more variables: `1413` <chr>, `1414` <chr>, `1415` <chr>, `1416` <chr>,
## #   `1417` <chr>
```

It looks like the data are presented as follows: each row corresponds to 1 of the 8 wine samples we've been dealing with so far, and each column represents a panelist (coded numerically), with the cells showing which group they assigned the wine to. So, for example, panelist `263` assigned `I_REFOSCO` to group `G6`, whereas panelist `1331` assigned the same wine to group `G3`. You'll notice, of course, that the problem with these data are that group numbers are dependent on panelist: it doesn't matter what actual number is assigned, but rather *what other wines each panelist thinks are in the same group*. We need some way to show this data: this is exactly transforming groups into dissimilarities.

We are going to use the `DistatisR` package for much of the analysis in this section. It has a utility function, `DistanceFromSort()`, that will transform this kind of grouping information into dissimilarities: symmetrical matrices for each panelist that show dissimilarity. Let's check it out:

```
library(DistatisR)

sorting_dissimilarities <-
  sorting_data %>%
  # As is often the case, we need to move our ID column to `rownames`
  column_to_rownames("wine") %>%
  DistanceFromSort()

# Let's take a look at what we have now
str(sorting_dissimilarities)
```

```
##  num [1:8, 1:8, 1:15] 0 1 1 1 1 1 1 1 1 0 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:8] "I_REFOSCO" "I_MERLOT" "I_SYRAH" "I_PRIMITIVO" ...
##   ..$ : chr [1:8] "I_REFOSCO" "I_MERLOT" "I_SYRAH" "I_PRIMITIVO" ...
##   ..$ : chr [1:15] "263" "1331" "1400" "1401" ...
```

```
# And here is a "slice" of the sorting "brick" (3D array or tensor)
sorting_dissimilarities[, , 1]
```

```
##             I_REFOSCO I_MERLOT I_SYRAH I_PRIMITIVO C_SYRAH C_REFOSCO C_MERLOT
## I_REFOSCO           0        1       1           1       1         1        1
## I_MERLOT            1        0       1           1       1         1        0
## I_SYRAH             1        1       0           1       1         1        1
## I_PRIMITIVO         1        1       1           0       1         1        1
## C_SYRAH             1        1       1           1       0         1        1
## C_REFOSCO           1        1       1           1       1         0        1
## C_MERLOT            1        0       1           1       1         1        0
## C_ZINFANDEL         1        0       1           1       1         1        0
##             C_ZINFANDEL
```

```
## I_REFOSCO             1
## I_MERLOT              0
## I_SYRAH               1
## I_PRIMITIVO           1
## C_SYRAH               1
## C_REFOSCO             1
## C_MERLOT              0
## C_ZINFANDEL           0
```

In order to deal with these data, the `DistanceFromSort()` function produces a "brick" of data (more typically called an "array" or a "tensor" in the machine-learning literature). An array is a stack of matrices: the 3rd dimension, in this case, indexes subjects, so that `sorting_dissimilarities[, , 1]` gets the dissimilarity judgments of the first subject.

These matrices are symmetric and contain only `0` and `1`: `0` when the row and column wines are in the same group, and `1` when they are not. Abdi et al. [2007] have a demonstration that this in fact constitutes a Euclidean distance.

For subject #1, it looks like most wines were different, with only one group of 2 wines being similar by color (`C_MERLOT` and `C_ZINFANDEL`).

You'll notice that this process is much easier than the one used in the original **R Opus**. The `DistatisR` package has matured (it incorporates the code file that HGH references) and so much of the preprocessing is now fully automatic.

9.2 DISTATIS demonstrated

I am not going to go into all of the steps of DISTATIS [see instead Abdi et al., 2007], but briefly: DISTATIS first uses multivariate statistics to evaluate how similar all the subjects' sorts are to each other, then weights each subject by their similarity to the overall consensus, uses these weights to sum up the individual results into a consensus dissimilarity, and finally uses an eigendecomposition on the weighted sum to produce a map of (dis)similarities (this final step is the MDS-like part).

This multistep approach means that not only do we have the consensus results, but we have access to results that tell us how much consensus there actually is, as well as the ability to generate bootstrapped confidence intervals based on individuals.

Luckily, the `DistatisR` package makes it quite easy to run DISTATIS.

```
distatis_results <- distatis(sorting_dissimilarities)

# We will also generate some bootstrapped results while we're at it so as to be
```

```
# able to plot confidence ellipses.
distatis_boots <- BootFromCompromise(sorting_dissimilarities)
```

```
## [1] Starting Full Bootstrap. Iterations #:
## [2] 1000
```

Before we proceed, let's look at what we've done:

```
distatis_results
```

```
## **Results for DiSTATIS**
## The analysis was performed on  15 individuals, on 8 sorted items.
## *The results are available in the following objects:
##
##   name         description
## 1 "$res4Cmat"  "Results from the C matrix (see notation)"
## 2 "$res4Splus" "Results from the S+ matrix (see notation)"
## 3 "$compact"   "a boolean. TRUE if compact results used."
```

You'll notice that the output from `distatis()` is a complex list object that is reasonably well-annotated (similar to what can be found in the `FactoMineR` package outputs). We'll dig into this in a minute.

```
distatis_boots %>% str
```

```
##  num [1:8, 1:3, 1:1000] -0.3729 0.3443 -0.0877 -0.283 0.0734 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:8] "I_REFOSCO" "I_MERLOT" "I_SYRAH" "I_PRIMITIVO" ...
##   ..$ : chr [1:3] "Factor 1" "Factor 2" "Factor 3"
##   ..$ : NULL
```

```
distatis_boots[, , 1]
```

```
##                Factor 1    Factor 2    Factor 3
## I_REFOSCO   -0.37290008 -0.18993078  0.53541992
## I_MERLOT     0.34434618 -0.03381469  0.19121393
## I_SYRAH     -0.08767685  0.36744534 -0.20347707
## I_PRIMITIVO -0.28296349  0.39322921 -0.06458306
## C_SYRAH      0.07342328 -0.34201698 -0.41756383
## C_REFOSCO   -0.41454335 -0.25278523 -0.02365847
## C_MERLOT     0.38346274 -0.04243239 -0.12752523
## C_ZINFANDEL  0.35685156  0.10030552  0.11017381
```

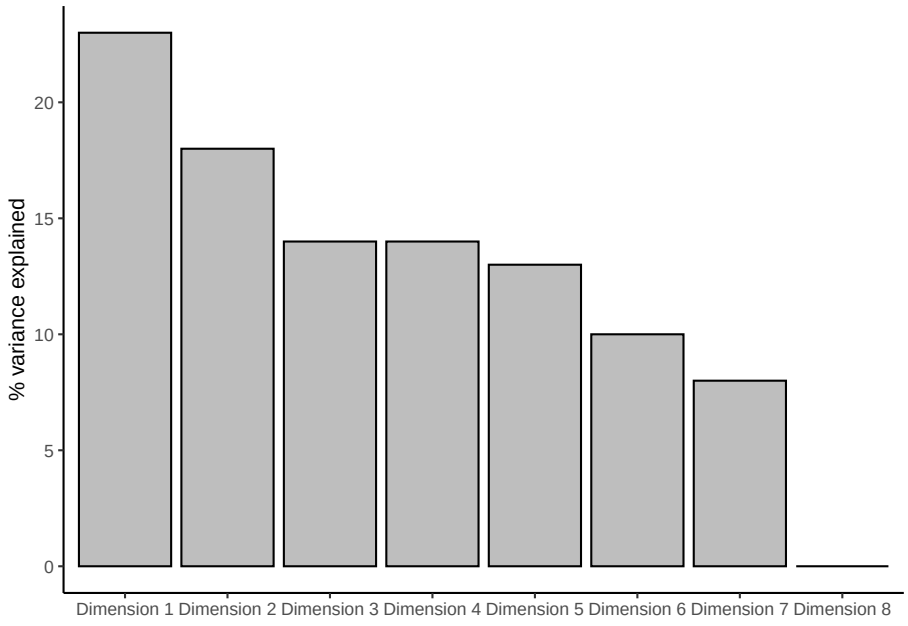
The `BootFromCompromise()` function gives us another array: in this case a set of factor scores for the samples on the first 3 factors/components, over 1000 (the default) bootstrapped iterations. Each slice is a single bootstrapped iteration.

Finally, let's examine our results. The output from `distatis()` stores results about the *samples* in the `$res4Splus` list, and results about the judges in the `$res4Cmat` list. These are named after the matrices defined in Abdi et al. [2007]: the $\mathbf{C}$ matrix is the matrix of RV coefficients between judges (a measure of multidimensional similarity), and the $\mathbf{S}_+$ matrix contains the compromise matrix of dissimilarities (after double-centering, weighting as I described above, and summing).

Let's follow along in the order that HGH did in her workflow. She starts by examining first the overall explained variances. These are found in the `distatis_results$res4Splus$tau` vector.

```
distatis_results$res4Splus$tau %>%
  as_tibble(rownames = "dim") %>%
  mutate(dim = str_c("Dimension ", dim)) %>%
  ggplot(aes(x = dim, y = value)) +
  geom_col(fill = "grey", color = "black", size = 1/2) +
  theme_classic() +
  labs(x = NULL, y = "% variance explained")
```

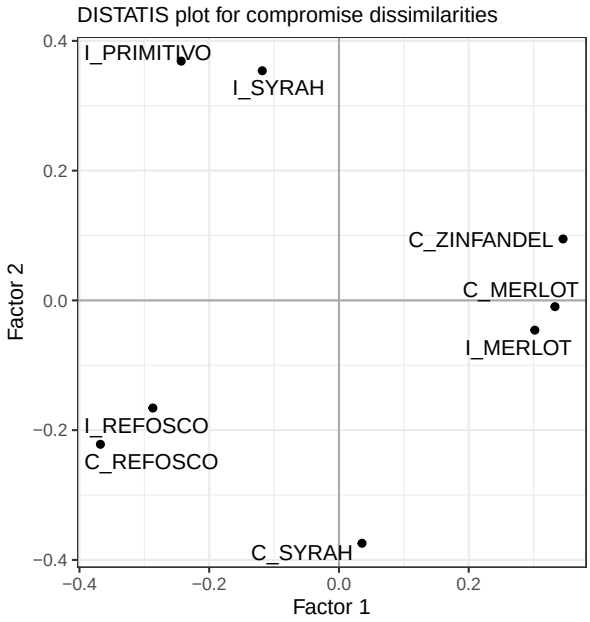
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



These results look the same as HGH's (good news!), and we can see our first two dimensions explain about 41% of the total variation in the sorts; this is fairly common for sorting, in which getting very high explanation in 2 dimensions is unlikely.

Let's look at how the products project into those first 2 dimensions.

```
p_splus <-
  distatis_results$res4Splus$F %>%
  as_tibble(rownames = "sample") %>%
  ggplot(aes(x = `Factor 1`, y = `Factor 2`)) +
  geom_vline(xintercept = 0, color = "darkgrey") +
  geom_hline(yintercept = 0, color = "darkgrey") +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = sample)) +
  coord_equal() +
  theme_bw() +
  labs(subtitle = "DISTATIS plot for compromise dissimilarities")

p_splus
```

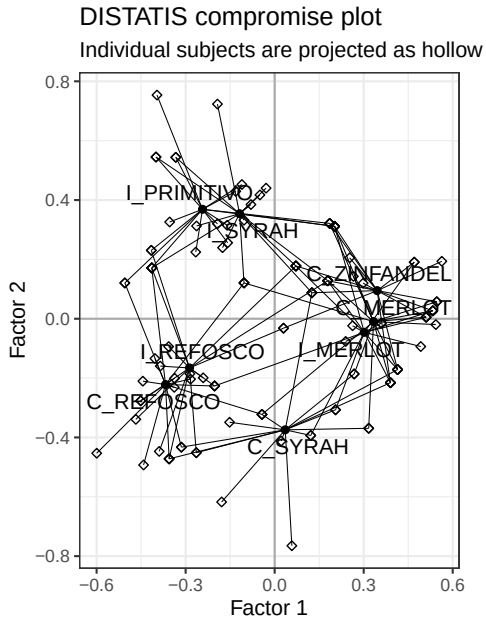


These results are the same as HGH's, again, which is good news.

HGH plotted the individual subjects as projected into this space. It's a little annoying to do, and I do not think it is typically particularly illuminating, but we'll do it here for practice.

```
p_splus_augmented <- p_splus
for(i in 1:dim(distatis_results$res4Splus$PartialF)[3]){
  augmented_data <-
    distatis_results$res4Splus$PartialF[, , i] %>%
    as_tibble(rownames = "sample") %>%
    # and then add the compromise barycenters so we can draw lines to them
    bind_cols(
      distatis_results$res4Splus$F %>%
        as_tibble() %>%
        rename(xend = 1, yend = 2)
    )

  p_splus_augmented <-
    p_splus_augmented +
    geom_point(data = augmented_data,
               shape = 23) +
    geom_segment(data = augmented_data,
                 aes(x = `Factor 1`, y = `Factor 2`,
                     xend = xend, yend = yend),
                 size = 1/4)
```

```
}

p_splus_augmented +
  labs(title = "DISTATIS compromise plot",
       subtitle = "Individual subjects are projected as hollow diamonds")
```

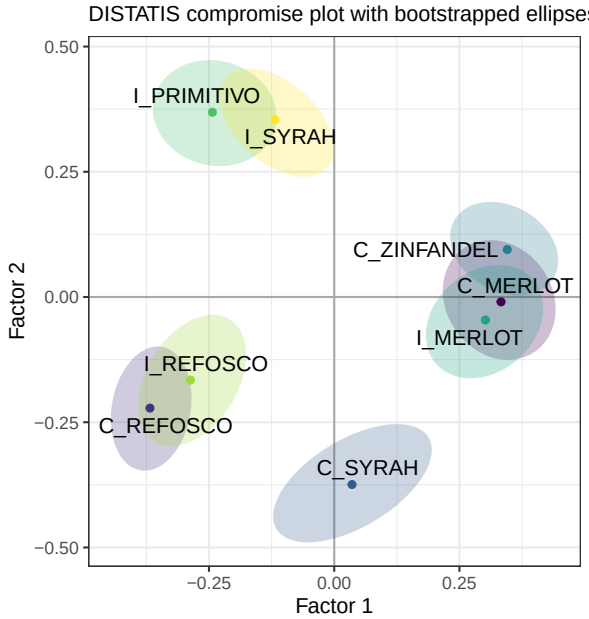


We can see how each wine's position is actually the (weighted) barycenter of all the subject's judgments about where that wine should be grouped. We could make this more easily readable by doing some work with colors and shapes, but I rarely find this kind of plot useful or informative. I leave this to the reader!

Before we move on to examining the subjects and comparing to MDS, let's take a look at the bootstrapped confidence ellipses we generated.

```
# We will need to reshape these arrays into a tibble that we can more easily
# feed into `stat_ellipse()`

# I hate converting arrays to tibbles - if anyone has a better way than a loop,
# let me know
boot_tbl <- tibble()

for(i in 1:1000){
  boot_tbl <-
    bind_rows(boot_tbl,
              distatis_boots[, , i] %>%
```

```
                as_tibble(rownames = "sample"))
}

p_splus_boots <-
  distatis_results$res4Splus$F %>%
  as_tibble(rownames = "sample") %>%
  ggplot(aes(x = `Factor 1`, y = `Factor 2`)) +
  geom_vline(xintercept = 0, color = "darkgrey") +
  geom_hline(yintercept = 0, color = "darkgrey") +
  stat_ellipse(data = boot_tbl,
               aes(fill = sample),
               show.legend = FALSE,
               geom = "polygon",
               alpha = 1/4) +
  geom_point(aes(color = sample), show.legend = FALSE) +
  ggrepel::geom_text_repel(aes(label = sample)) +
  coord_equal() +
  theme_bw() +
  labs(subtitle = "DISTATIS compromise plot with bootstrapped ellipses") +
  scale_color_viridis_d() +
  scale_fill_viridis_d()

p_splus_boots
```



With bootstrapping, we can see that there are 4 distinct groups of samples that

don't appear to overlap even when we perturb the original data.

We can now turn our consideration to the subjects: how much did they agree? We use the RV matrix to examine this.

```
distatis_results$res4Cmat$C %>%
  round(3)
```

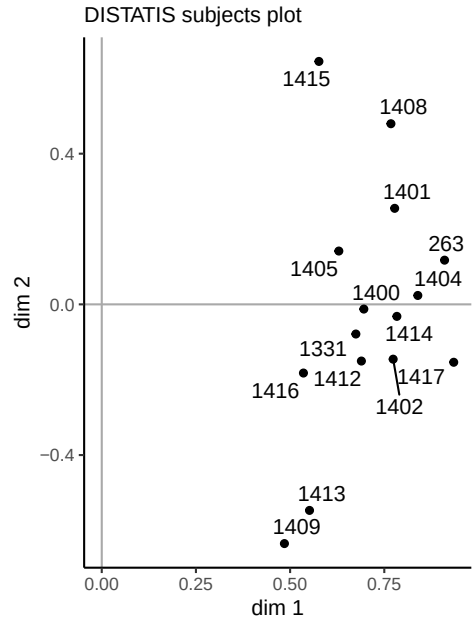
```
##        263  1331  1400  1401  1402  1404  1405  1408  1409  1412  1413  1414
## 263  1.000 0.461 0.564 0.855 0.718 0.778 0.568 0.730 0.333 0.651 0.569 0.614
## 1331 0.461 1.000 0.452 0.333 0.476 0.533 0.511 0.464 0.419 0.322 0.214 0.710
## 1400 0.564 0.452 1.000 0.422 0.452 0.660 0.333 0.464 0.189 0.537 0.261 0.482
## 1401 0.855 0.333 0.422 1.000 0.711 0.627 0.399 0.730 0.181 0.639 0.371 0.438
## 1402 0.718 0.476 0.452 0.711 1.000 0.533 0.466 0.464 0.419 0.645 0.475 0.507
## 1404 0.778 0.533 0.660 0.627 0.533 1.000 0.462 0.581 0.324 0.487 0.493 0.622
## 1405 0.568 0.511 0.333 0.399 0.466 0.462 1.000 0.449 0.142 0.439 0.238 0.462
## 1408 0.730 0.464 0.464 0.730 0.464 0.581 0.449 1.000 0.219 0.432 0.237 0.581
## 1409 0.333 0.419 0.189 0.181 0.419 0.324 0.142 0.219 1.000 0.272 0.601 0.502
## 1412 0.651 0.322 0.537 0.639 0.645 0.487 0.439 0.432 0.272 1.000 0.495 0.372
## 1413 0.569 0.214 0.261 0.371 0.475 0.493 0.238 0.237 0.601 0.495 1.000 0.291
## 1414 0.614 0.710 0.482 0.438 0.507 0.622 0.462 0.581 0.502 0.372 0.291 1.000
## 1415 0.581 0.309 0.309 0.504 0.309 0.493 0.360 0.827 0.068 0.174 0.077 0.493
## 1416 0.377 0.289 0.533 0.254 0.311 0.438 0.212 0.341 0.318 0.238 0.238 0.462
## 1417 0.785 0.700 0.700 0.626 0.700 0.808 0.626 0.597 0.488 0.592 0.494 0.808
##        1415  1416  1417
## 263   0.581 0.377 0.785
## 1331  0.309 0.289 0.700
## 1400  0.309 0.533 0.700
## 1401  0.504 0.254 0.626
## 1402  0.309 0.311 0.700
## 1404  0.493 0.438 0.808
## 1405  0.360 0.212 0.626
## 1408  0.827 0.341 0.597
## 1409  0.068 0.318 0.488
## 1412  0.174 0.238 0.592
## 1413  0.077 0.238 0.494
## 1414  0.493 0.462 0.808
## 1415  1.000 0.216 0.378
## 1416  0.216 1.000 0.626
## 1417  0.378 0.626 1.000
```

RV is a generalization of the concept of (Pearson) correlation to multivariate data. RV coefficients between two matrices can only be positive, and is constrained to the $[0, 1]$ interval, with higher values indicating more agreement.

If we use eigendecomposition on the RV matrix, the coordinates of each subject on the first dimension indicate their maximum agreement. We can use the first

2 dimensions to examine whether there are groups of subjects, as well as how well they agree in general:

```
p_cmat <-
  distatis_results$res4Cmat$G %>%
  as_tibble(rownames = "subject") %>%
  ggplot(aes(x = `dim 1`, y = `dim 2`)) +
  geom_vline(xintercept = 0, color = "darkgrey") +
  geom_hline(yintercept = 0, color = "darkgrey") +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = subject)) +
  coord_equal() +
  theme_classic() +
  labs(subtitle = "DISTATIS subjects plot")

p_cmat
```



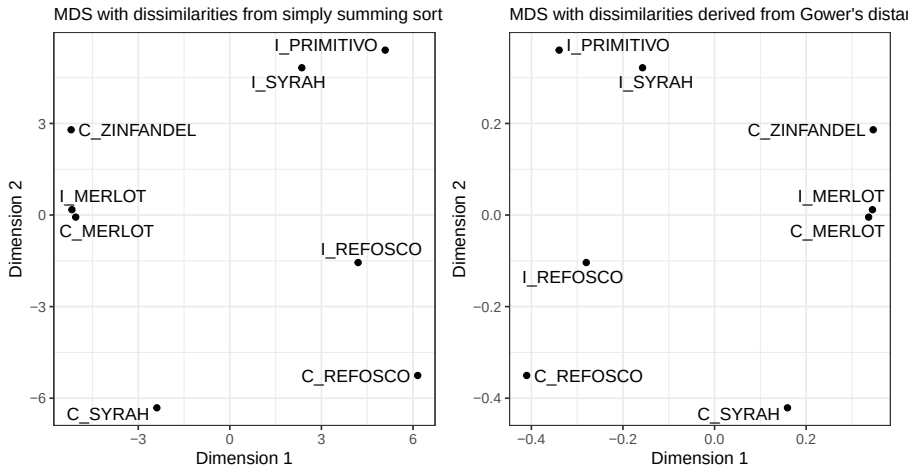
Subjects' positions on the x-axis here are proportional to the weight their sorts receive in the final consensus plot–we can see that overall subjects largely agree, with weights between just less than `0.5` up to about `0.85`. On the second dimension we do see a fair amount of spread, so it is obvious that there is some disagreement among subjects. However, without some grouping variable (like level of experience in winemaking), we can't interpret whether this disagreement is systematic.

9.3 Comparison to MDS

HGH goes on to also analyze these data via metric and non-metric MDS. We will use metric MDS for convenience. Because MDS requires a distance matrix, we have to convert our original sorting data into some kind of consensus. HGH used the `cluster::daisy()` function. Reading the `?daisy` file, it appears that this is a general dissimilarity tool. We'll compare it to simply summing up the observed groups.

```
daisy_data <-
  sorting_data %>%
  column_to_rownames("wine") %>%
  # `daisy()` does not seem to accept string data
  mutate(across(everything(), ~as.factor(.))) %>%
  cluster::daisy()

simple_data <-
  sorting_data %>%
  column_to_rownames("wine") %>%
  DistanceFromSort() %>%
  # Here we summ across the 3rd dimension of the array, "squishing" it
  apply(c(1, 2), sum) %>%
  dist(method = "max")

p_mds_daisy <-
  daisy_data %>%
  cmdscale() %>%
  as_tibble(rownames = "wine") %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = wine)) +
  coord_fixed() +
  labs(x = "Dimension 1", y = "Dimension 2",
       subtitle = "MDS with dissimilarities derived from Gower's distance") +
  theme_bw()

p_mds_simple <-
  simple_data %>%
  cmdscale() %>%
  as_tibble(rownames = "wine") %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = wine)) +
  coord_fixed() +
  labs(x = "Dimension 1", y = "Dimension 2",
```

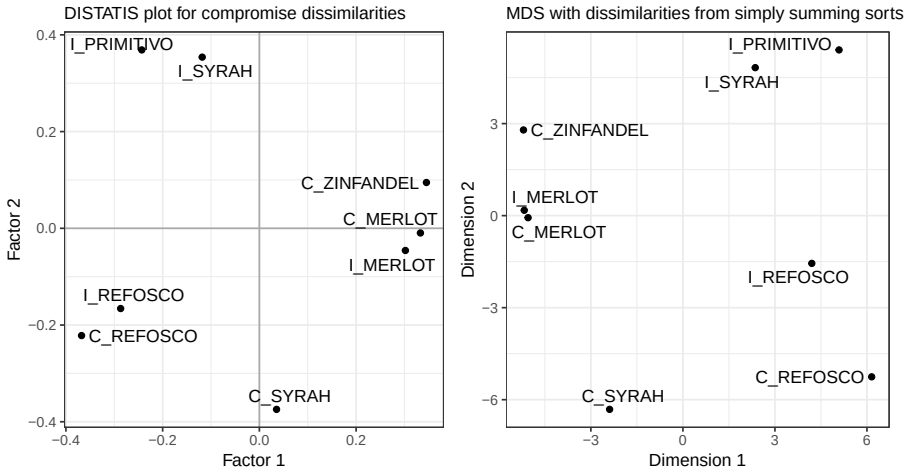
```
       subtitle = "MDS with dissimilarities from simply summing sorts") +
  theme_bw()

library(patchwork)

p_mds_simple + p_mds_daisy
```



We can see that the dissimilarities produced from `daisy()` and from our simple sum of dissimilarities are the same except for a scale constant.

Finally, we can compare this alignment to that from DISTATIS:

```
p_splus + p_mds_simple
```



We can see that overall our results are extremely similar, although there are some changes (the incerased spread between the Merlot wines in the DISTATIS plot, and the decreased spread between the Refosco wines).

9.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
```

```
##  [1] patchwork_1.1.2 DistatisR_1.1.1 here_1.0.1      lubridate_1.9.2
##  [5] forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1
##  [9] readr_2.1.4     tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4       xfun_0.39          ggrepel_0.9.3      lattice_0.21-8
##  [5] tzdb_0.4.0         vctrs_0.6.3        tools_4.3.1        generics_0.1.3
##  [9] parallel_4.3.1     fansi_1.0.4        cluster_2.1.4      highr_0.10
## [13] janeaustenr_1.0.0  pkgconfig_2.0.3    prettyGraphs_2.1.6 tokenizers_0.3.0
## [17] Matrix_1.6-0       readxl_1.4.3       lifecycle_1.0.3    compiler_4.3.1
## [21] farver_2.1.1       munsell_0.5.0      carData_3.0-5      htmltools_0.5.6
## [25] SnowballC_0.7.1    yaml_2.3.7         tidytext_0.4.1     pillar_1.9.0
## [29] car_3.1-2          crayon_1.5.2       MASS_7.3-60        abind_1.4-5
## [33] tidyselect_1.2.0   digest_0.6.33      stringi_1.7.12     bookdown_0.37
## [37] labeling_0.4.3     rprojroot_2.0.3    fastmap_1.1.1      grid_4.3.1
## [41] colorspace_2.1-0   cli_3.6.1          magrittr_2.0.3     utf8_1.2.3
## [45] withr_2.5.0        scales_1.2.1       bit64_4.0.5        timechange_0.2.0
## [49] rmarkdown_2.23     bit_4.0.5          cellranger_1.1.0   hms_1.1.3
## [53] evaluate_0.21      knitr_1.43         viridisLite_0.4.2  rlang_1.1.1
## [57] Rcpp_1.0.11        glue_1.6.2         rstudioapi_0.15.0  vroom_1.6.3
## [61] R6_2.5.1
```

Chapter 10

Correspondence Analysis (CA)

Correspondence Analysis (CA) is a close relative of PCA that can be applied to count data. While PCA is appropriate for application when all observations are interval or ratio data, CA is built for the case when cell counts are positive integers. CA in its modern was developed for application to survey data by French quantitative sociologists, and the most comprehensive introduction to it is probably Michael Greenacre's *Correspondence Analysis in Practice* [Greenacre, 2017]. In sensory evaluation, we often apply CA to check-all-that-apply (CATA) data or to data derived from free-comment/text analysis, because these data involve positive counts. Sometimes these data meet the theoretical requirements for CA, and sometimes they do not. We won't dig into those details here, but I also recommend checking out Abdi and Bera's more succinct articles on the topic [Abdi and Béra, 2015].

None of the data we've been using for is exactly suited for CA (although *Multiple* Correspondence Analysis can be effectively used to analyze sorting data), so HGH used the `author` dataset that is included in the `ca` package (i.e., `ca::author`) to demonstrate the approach. We'll follow suit.

```
library(tidyverse)
library(here)
library(ca)

# We have to load the `author` dataset ourselves
data(author)

author
```

```
##                              a   b   c   d   e   f   g   h   i   j   k   l   m
```

```
## three daughters (buck)       550 116 147 374 1015 131 131 493 442  2 52 302 159
## drifters (michener)          515 109 172 311  827 167 136 376 432  8 61 280 146
## lost world (clark)           590 112 181 265  940 137 119 419 514  6 46 335 176
## east wind (buck)             557 129 128 343  996 158 129 571 555  4 76 291 247
## farewell to arms (hemingway) 589  72 129 339  866 108 159 449 472  7 59 264 158
## sound and fury 7 (faulkner)  541 109 136 228  763 126 129 401 520  5 72 280 209
## sound and fury 6 (faulkner)  517  96 127 356  771 115 189 478 558  6 80 322 163
## profiles of future (clark)   592 151 251 238  985 168 152 381 544  7 39 416 236
## islands (hemingway)          576 120 136 404  873 122 156 593 406  3 90 281 142
## pendorric 3 (holt)           557  97 145 354  909  97 121 479 431 10 94 240 154
## asia (michener)              554 108 206 243  797 164 100 328 471  4 34 293 149
## pendorric 2 (holt)           541  93 149 390  887 133 154 463 518  4 65 265 194
##                                n   o   p q   r   s   t   u   v   w  x   y  z
## three daughters (buck)       534 516 115 4 409 467 632 174  66 155  5 150  3
## drifters (michener)          470 561 140 4 368 387 632 195  60 156 14 137  5
## lost world (clark)           403 505 147 8 395 464 670 224 113 146 13 162 10
## east wind (buck)             479 509  92 3 413 533 632 181  68 187 10 184  4
## farewell to arms (hemingway) 504 542  95 0 416 314 691 197  64 225  1 155  2
## sound and fury 7 (faulkner)  471 589  84 2 324 454 672 247  71 160 11 280  1
## sound and fury 6 (faulkner)  483 617  82 8 294 358 685 225  37 216 12 171  5
## profiles of future (clark)   526 524 107 9 418 508 655 226  89 106 15 142 20
## islands (hemingway)          516 488  91 3 339 349 640 194  40 250  3 104  5
## pendorric 3 (holt)           417 477 100 3 305 415 597 237  64 194  9 140  4
## asia (michener)              482 532 145 8 361 402 630 196  66 149  2  80  6
## pendorric 2 (holt)           484 545  70 4 299 423 644 193  66 218  2 127  2
```

The `author` dataset is a `matrix` of counts for each letter of the English alphabet in several published works by various authors. As such, it's suitable for analysis by CA. We are, in effect, asking which authors (rows) are most associated with which letters (columns) and vice versa.

The `ca::ca()` function has various options, but HGH used it with default settings, which we will do as well:

```
author_ca <- ca(author)

author_ca
```

```
##
##  Principal inertias (eigenvalues):
##            1        2        3        4        5        6        7
## Value      0.007664 0.003688 0.002411 0.001383 0.001002 0.000723 0.000659
## Percentage 40.91%   19.69%   12.87%   7.38%    5.35%    3.86%    3.52%
##            8        9        10       11
## Value      0.000455 0.000374 0.000263 0.000113
## Percentage 2.43%    2%       1.4%     0.6%
```

```
##
##
##  Rows:
##         three daughters (buck) drifters (michener) lost world (clark)
## Mass                 0.085407            0.079728           0.084881
## ChiDist              0.097831            0.094815           0.128432
## Inertia              0.000817            0.000717           0.001400
## Dim. 1              -0.095388            0.405697           1.157803
## Dim. 2              -0.794999           -0.405560          -0.023114
##         east wind (buck) farewell to arms (hemingway)
## Mass            0.089411                     0.082215
## ChiDist         0.118655                     0.122889
## Inertia         0.001259                     0.001242
## Dim. 1         -0.173901                    -0.831886
## Dim. 2          0.434443                    -0.136485
##         sound and fury 7 (faulkner) sound and fury 6 (faulkner)
## Mass                       0.082310                    0.083338
## ChiDist                    0.172918                    0.141937
## Inertia                    0.002461                    0.001679
## Dim. 1                     0.302025                   -0.925572
## Dim. 2                     2.707599                    0.966944
##         profiles of future (clark) islands (hemingway) pendorric 3 (holt)
## Mass                      0.089722            0.082776           0.079501
## ChiDist                   0.187358            0.165529           0.113174
## Inertia                   0.003150            0.002268           0.001018
## Dim. 1                    1.924060           -1.566481          -0.724758
## Dim. 2                   -0.249310           -1.185338          -0.106349
##         asia (michener) pendorric 2 (holt)
## Mass           0.077827           0.082884
## ChiDist        0.155115           0.101369
## Inertia        0.001873           0.000852
## Dim. 1         1.179548          -0.764937
## Dim. 2        -1.186934          -0.091188
##
##
##  Columns:
##                 a         b         c         d         e         f         g
## Mass     0.079847  0.015685  0.022798  0.045967  0.127070  0.019439  0.020025
## ChiDist  0.048441  0.148142  0.222783  0.189938  0.070788  0.165442  0.156640
## Inertia  0.000187  0.000344  0.001132  0.001658  0.000637  0.000532  0.000491
## Dim. 1   0.017623  0.984463  2.115029 -1.925632  0.086722  1.276526 -1.020713
## Dim. 2  -0.320271 -0.398032 -1.373448 -1.135362 -0.684785 -0.732952  0.353017
##                 h        i        j         k         l        m         n
## Mass     0.064928 0.070092 0.000789  0.009181  0.042667 0.025500  0.068968
## ChiDist  0.154745 0.086328 0.412075  0.296727  0.120397 0.159747  0.075706
## Inertia  0.001555 0.000522 0.000134  0.000808  0.000618 0.000651  0.000395
```

```
## Dim. 1  -1.501277 0.267473 0.453341 -2.755177  1.018257 0.712695 -0.200364
## Dim. 2  -0.302413 0.889546 0.916032  1.231557 -0.165020 1.400966 -0.417258
##                 o         p         q         r         s         t         u
## Mass     0.076572  0.015159  0.000669  0.051897 0.060660  0.093010 0.029756
## ChiDist  0.088101  0.250617  0.582298  0.111725 0.123217  0.050630 0.119215
## Inertia  0.000594  0.000952  0.000227  0.000648 0.000921  0.000238 0.000423
## Dim. 1  -0.108491  1.610807  4.079786  0.591372 0.860202 -0.100464 0.163295
## Dim. 2   0.637987 -1.837948 -1.914791 -0.734216 0.405610  0.203141 1.017140
##                 v         w         x         y         z
## Mass     0.009612  0.025847 0.001160 0.021902  0.000801
## ChiDist  0.269770  0.232868 0.600831 0.301376  0.833700
## Inertia  0.000700  0.001402 0.000419 0.001989  0.000557
## Dim. 1   2.281333 -2.499232 3.340505 0.001519  6.808100
## Dim. 2   0.177022 -0.284722 4.215355 4.706083 -3.509223
```

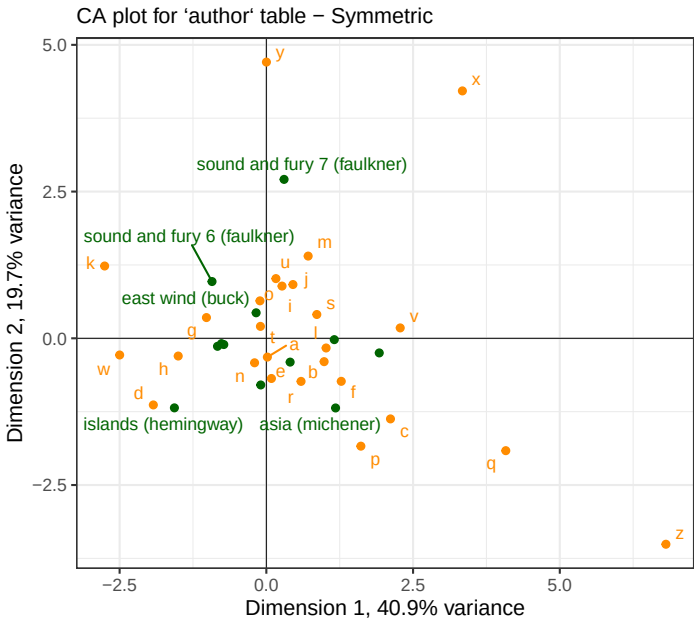
Printing the `author_ca` object gives us details about the analysis, which we are going to follow HGH in not interpreting over-much. We are going to treat CA largely as a visualization method, and not get too into the math: it's not overly complicated, but it involves a lot of saying things like "for rows (correspondingly, columns)" and I just don't see it being worthwhile.

As always, we're going to replace base-`R` plotting with `ggplot2`. We'll do this in a couple of ways. First off, we can get the "standard coordinates" (per the `?ca` documentation) in `$colcoord` and `$rowcoord`. Let's see what happens if we pull them out and plot them directly

```
book_coords <-
  author_ca$rowcoord %>%
  as_tibble(rownames = "name") %>%
  mutate(point_type = "book")

letter_coords <-
  author_ca$colcoord %>%
  as_tibble(rownames = "name") %>%
  mutate(point_type = "letter")

# Now we can bind these together and plot them easily
p_symmetric <-
  bind_rows(book_coords, letter_coords) %>%
  ggplot(aes(x = Dim1, y = Dim2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point(aes(color = point_type)) +
  ggrepel::geom_text_repel(aes(label = name, color = point_type),
                           max.overlaps = 12, size = 3) +
  scale_color_manual(values = c("darkgreen", "darkorange")) +
```

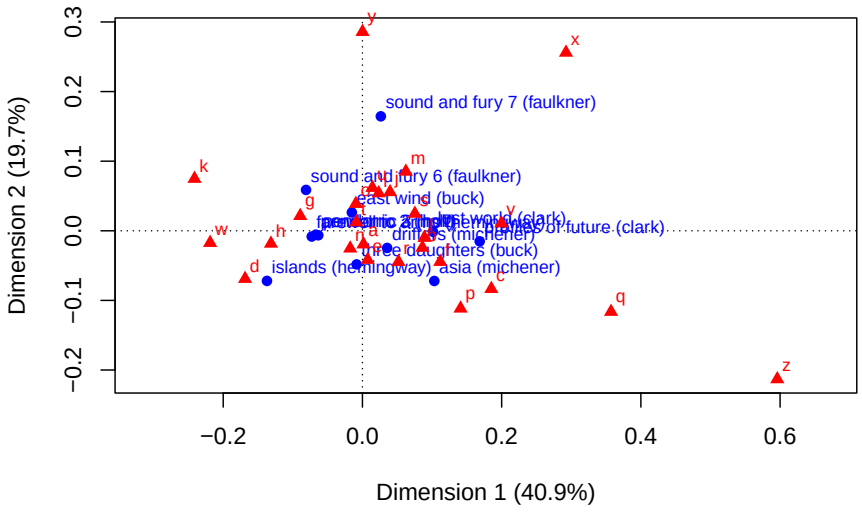
```
  theme_bw() +
  # I am not sure if I mentioned it before, but `coord_equal()` constrains the
  # axes to have equal unit spacing, which is important when we are interested
  # in plotting something, like CA, that spatially imputes similarity.
  coord_equal() +
  theme(legend.position = "none") +
  labs(x = "Dimension 1, 40.9% variance",
       y = "Dimension 2, 19.7% variance",
       subtitle = "CA plot for `author` table - Symmetric")

p_symmetric
```

```
## Warning: ggrepel: 7 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



If we compare this to the base-R plotting version, you can see that ours is a (nicer) version:

```
plot(author_ca)
```



I am not sure why there is a difference in scaling for these two maps - but it appears to be a uniform expansion factor. This "symmetric" map is **not** a biplot: we cannot interpret the proximity of books (rows, in green) to the letter (columns, in orange).

We can also generate various non-"symmetric" plots. In a nutshell, in CA the rows and columns of the contingency table explain the same amount of the variance [because of the sum to marginality/sum to one constraint, Abdi and Béra, 2015], and so in order to actually produce a biplot in which the direct proximity of the row and column points is meaningful, we need to scale appropriately. I'm going to quote Abdi and Béra [2015, 279] directly and extensively, so that I don't get myself reversed:

> In a CA map when two row (respectively, column) points are close to each other, this means that these points have similar profiles, and when two points have the same profile, they will be located exactly at the same place (this is a consequence of the distributional equivalence principle). The proximity between row and column points is more delicate to interpret because of the barycentric principle (see section on "The Transition Formula"): The position of a row (respectively, column) point is determined from its barycenter on the column (respectively, row), and therefore, the proximity between a row point and one column point cannot be interpreted directly

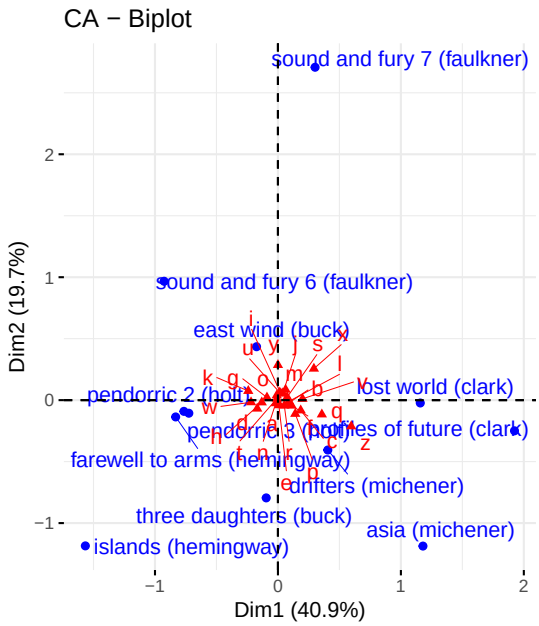
But

> … the plots can reflect [any] asymmetry by normalizing one set [of either rows or columns] such that the variance of its factor scores is equal to 1 for each factor… [i]n the asymmetric plot obtained… [t]he distances between the rows and columns can now be interpreted meaningfully: the distance between a row point and column point reflects their association…

In other words, by construction the standard, symmetric plot tells us that when two row (respectively, column) points are close together, we can say that their profiles are similar, but there is no direct interpretation of the proximity of a row and column point. In the asymmetric plots, the proximity of row and column points becomes meaningful.

The fastest way to do this is to use the `factoextra::fviz_ca()` function, which will show an improved version of the plots HGH noted "look terrible":

```
library(factoextra)

p_biplot <-
  fviz_ca(author_ca,
          map = "colprincipal",
          repel = TRUE) +
  coord_equal()

p_biplot
```



In this plot, the columns are plotted in principal coordinates and the rows are plotted in standard coordinates. We can clean up the plot ourselves. Reading through various help files (`?ca` and `?plot.ca`), I determined that the difference in scaling we observed between our symmetric CA plot and the default is indeed due to differences in scaling: the `ca` package by default gives a symmetric plot in "principal" coordinates: "In this map both the row and column points are scaled to have inertias (weighted variances) equal to the principal inertia (eigenvalue or squared singular value) along the principal axes, that is both rows and columns are in principal coordinates." [`?plot.ca`; Abdi and Béra [2015]]. However, in the output of `ca::ca()`, the `$rowcoord`/`$colcoord` matrices are given in so-called "standard coordinates", and so need to be converted to principal coordinates for plotting [`?ca`; Abdi and Béra [2015]]. Let's see if we can get there.

```
# First off, we see that `plot.ca()` invisibly returns coordinates, so we can
# grab those to check our work.
author_ca_asymmetric_coords <-
  plot.ca(author_ca, map = "colprincipal")
```

```
# Here is where I get puzzled: if we look at the total scaled output for this,
# neither the rows or the colums have inertia equal to the (squared) singular
# values:

author_ca_asymmetric_coords$rows %>% colSums()
```

```
##        Dim1        Dim2
## -0.11379020 -0.07029046
```

```
author_ca_asymmetric_coords$cols %>% colSums()
```

```
##      Dim1      Dim2
## 1.4495184 0.1422949
```

```
author_ca$sv[1:2]
```

```
## [1] 0.08754348 0.06073157
```

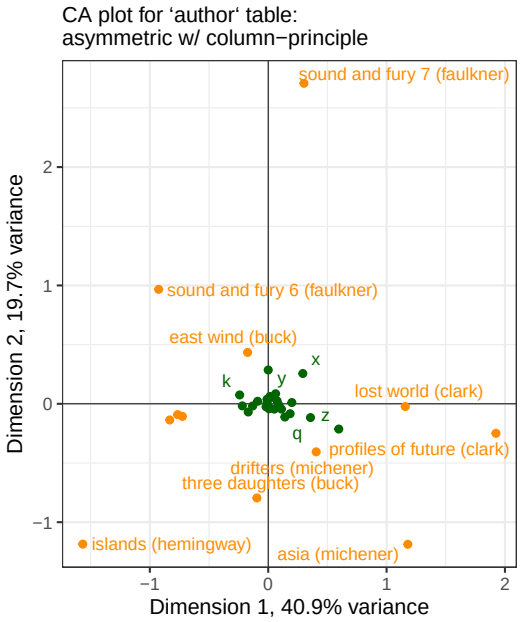
```
author_ca$sv[1:2]^2
```

```
## [1] 0.007663861 0.003688324
```

So you've got me, there! I am not sure what the `ca()` function is doing here. If you understand better than me, please let me know! However, just to demonstrate how we can go on to use these to build prettier versions of our asymmetric maps, we will take the saved results and pipe them into `ggplot2`.

```r
author_column_asymmetric <-
  author_ca_asymmetric_coords %>%
  # `enframe()` takes a list and puts it into a tibble, so we can use `map()` on
  # it!
  enframe() %>%
  # We'll make sure to capture those matrix rownames properly... note the use of
  # ordered arguments passed directly to `map()`, instead of the lambda function
  # ("~") form
  mutate(tbl = map(value, as_tibble, rownames = "label")) %>%
  unnest(tbl)

p_asymmetric <-
  author_column_asymmetric %>%
  ggplot(aes(x = Dim1, y = Dim2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point(aes(color = name)) +
  ggrepel::geom_text_repel(aes(label = label, color = name),
                           max.overlaps = 12, size = 3) +
  scale_color_manual(values = c("darkgreen", "darkorange")) +
  theme_bw() +
  # I am not sure if I mentioned it before, but `coord_equal()` constrains the
  # axes to have equal unit spacing, which is important when we are interested
  # in plotting something, like CA, that spatially imputes similarity.
  coord_equal() +
  theme(legend.position = "none") +
  labs(x = "Dimension 1, 40.9% variance",
       y = "Dimension 2, 19.7% variance",
       subtitle = "CA plot for `author` table:\nasymmetric w/ column-principle")

p_asymmetric
```

```
## Warning: ggrepel: 24 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

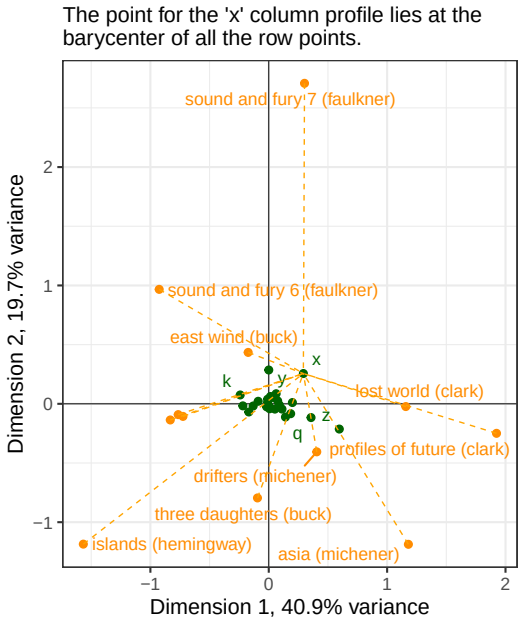


We can see the same kind of difficult-to-parse plot that HGH complained of, although the use of `ggrepel` and some other aesthetic adjustments make it easier to use. We can also use it to illustrate the principle that Abdi and Béra [2015] note in their paper:

> … a row [read column here for our case] is positioned exactly at the barycenter of the columns [read: row]

```
author_ca_x_data <-
  author_ca_asymmetric_coords$rows %>%
  as_tibble(rownames = "label") %>%
  # We will demonstrate with the letter "x", since it appears to be one of the
  # more discriminating column points.
  mutate(xend = author_ca_asymmetric_coords$cols["x", 1],
         yend = author_ca_asymmetric_coords$cols["x", 2])

p_asymmetric +
  geom_segment(data = author_ca_x_data,
               aes(x = Dim1, y = Dim2, xend = xend, yend = yend),
               linetype = 2, linewidth = 1/3, color = "orange") +
  labs(subtitle = "The point for the 'x' column profile lies at the\nbarycenter of all
```

```
## Warning: ggrepel: 24 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Let's look at the actual column profile for "x":

```
(author[, "x"] / sum(author[, "x"])) %>%
  round(3) %>%
  as_tibble(rownames = "book") %>%
  arrange(-value)
```

```
## # A tibble: 12 x 2
##    book                          value
##    <chr>                         <dbl>
##  1 profiles of future (clark)    0.155
##  2 drifters (michener)           0.144
##  3 lost world (clark)            0.134
##  4 sound and fury 6 (faulkner)   0.124
##  5 sound and fury 7 (faulkner)   0.113
##  6 east wind (buck)              0.103
##  7 pendorric 3 (holt)            0.093
##  8 three daughters (buck)        0.052
##  9 islands (hemingway)           0.031
## 10 asia (michener)               0.021
## 11 pendorric 2 (holt)            0.021
## 12 farewell to arms (hemingway) 0.01
```

This tells us what books have the highest count of "x"; looking at the plot, we can see that indeed "x" is closest to those novels… for the most part. Let's keep

digging a little bit. If we look at, for example, `profiles of future (clark)`, we can see that while it is important for "x", "x" is not so important for it...

```
(author["profiles of future (clark)", ] / sum(author["profiles of future (clark)", ])) %>%
  round(5) %>%
  as_tibble(rownames = "letter") %>%
  arrange(value)
```

```
## # A tibble: 26 x 2
##    letter   value
##    <chr>    <dbl>
##  1 j      0.00093
##  2 q      0.0012
##  3 x      0.002
##  4 z      0.00266
##  5 k      0.0052
##  6 v      0.0119
##  7 w      0.0141
##  8 p      0.0143
##  9 y      0.0189
## 10 b      0.0201
## # i 16 more rows
```

But for `drifters (michener)`, maybe the story is different.

```
(author["drifters (michener)", ] / sum(author["drifters (michener)", ])) %>%
  round(5) %>%
  as_tibble(rownames = "letter") %>%
  arrange(value)
```

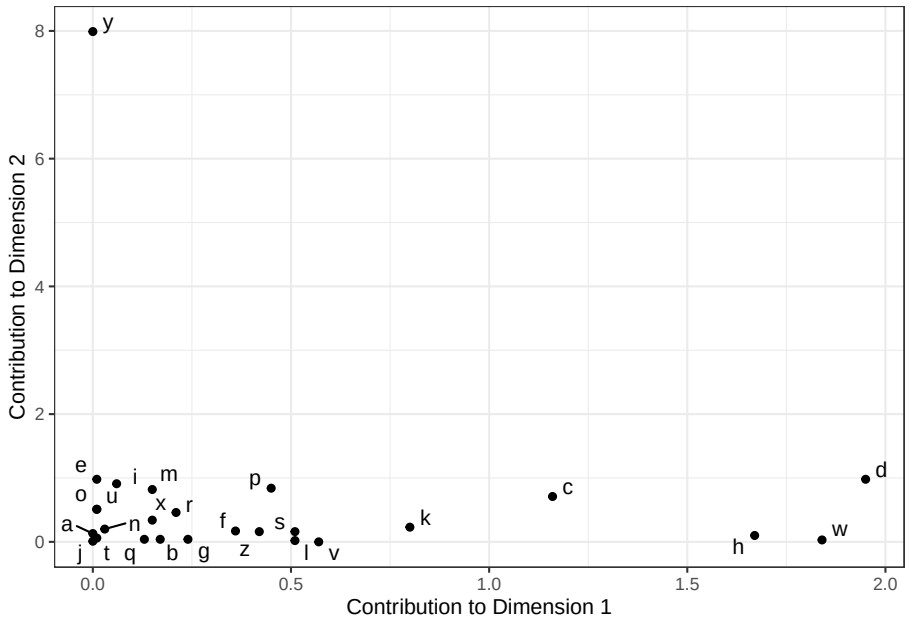
```
## # A tibble: 26 x 2
##    letter   value
##    <chr>    <dbl>
##  1 q      0.0006
##  2 z      0.00075
##  3 j      0.0012
##  4 x      0.0021
##  5 v      0.009
##  6 k      0.00915
##  7 b      0.0163
##  8 g      0.0204
##  9 y      0.0205
## 10 p      0.0210
## # i 16 more rows
```

…nope! Well, sort of. In fact, we have run into a common problem with Correspondence Analysis: the plots are over-influenced by uncommon observations. "x", of course is an uncommon letter of the alphabet, and so minor variations (probably by chance in the `author` sample) have a large effect on the spatial arrangement of the biplot.

One way to combat this problem is to look at some other measures of configuration quality. A common quality to examine is the *contribution* of a variable (row or column) to each dimension. We can calculate conrtibutions from the output as the squared factor score for the row, multiplied by the row mass, and divided by the eigenvalue for the corresponding dimension, e.g., for row i and dimension/component/factor l:

$$ctr_i = \frac{f_{i,l}^2 r_i}{\lambda_l}$$

```
author_ctrs <-
  tibble(
  letter = author_ca$colnames,
  ctr_1 = (author_ca$colmass * (author_ca$colcoord[, 1])^2 / author_ca$sv[1]),
  ctr_2 = (author_ca$colmass * (author_ca$colcoord[, 2])^2 / author_ca$sv[2])
) %>%
  mutate(across(where(is.numeric), ~round(., 2)))

author_ctrs %>%
  ggplot(aes(x = ctr_1, y = ctr_2)) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = letter)) +
  theme_bw() +
  labs(x = "Contribution to Dimension 1",
       y = "Contribution to Dimension 2")
```

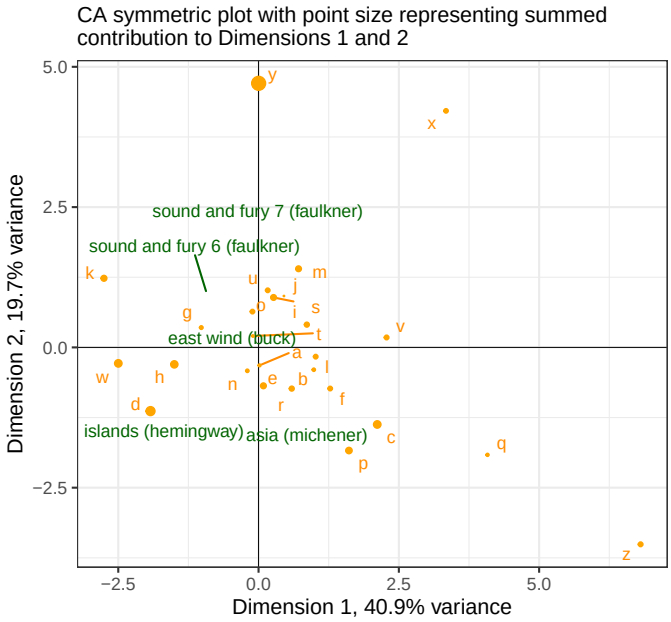


We can see that "y" contributes disproportionately to Dimension 2, and that "d", "w", and "h" are the largest contributors to Dimension 1. We can take this information to help build a potentially more useful visualization:

```
# Neat functionality: remove a layer from a compiled ggplot2 to avoid more work.  You
ggedit::remove_geom(p_symmetric, geom = "point") +
  geom_point(data = . %>%
               filter(point_type == "letter") %>%
               left_join(author_ctrs, by = c("name" = "letter")),
             mapping = aes(size = map2_dbl(ctr_1, ctr_2, sum)),
             color = "orange") +
  scale_size_continuous(range = c(0, 3)) +
  labs(subtitle = "CA symmetric plot with point size representing summed\ncontribution
```

```
## Registered S3 method overwritten by 'ggedit':
##   method from
##   +.gg   ggplot2
```

```
## Warning: ggrepel: 7 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Now even though we cannot directly interpret proximity of books to letters, we can say that the fact that `sound and fury 7 (faulkner)` is strongly separated by Dimension 2 is probably due to an unusual proportion of "y" in the text: and we'd be right! It is the 11th most common letter in that row, as compared to many of the other examples we looked at above (scroll up and check to see if I'm right!).

```
(author["sound and fury 7 (faulkner)", ] / sum(author["sound and fury 7 (faulkner)", ])) %>%
  round(5) %>%
  as_tibble(rownames = "letter") %>%
  arrange(-value) %>%
  print(n = 12)
```

```
## # A tibble: 26 x 2
##    letter  value
##    <chr>   <dbl>
##  1 e      0.111
##  2 t      0.0976
##  3 o      0.0856
##  4 a      0.0786
##  5 i      0.0755
##  6 n      0.0684
##  7 s      0.0659
##  8 h      0.0582
##  9 r      0.0471
```

```
## 10 l      0.0407
## 11 y      0.0407
## 12 u      0.0359
## # i 14 more rows
```

Thus, we can use some careful exploration of data combined with some additional quantities to help us use CA to really explore a data set.

10.1 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] factoextra_1.0.7 ca_0.71.1        here_1.0.1       lubridate_1.9.2
##  [5] forcats_1.0.0    stringr_1.5.0    dplyr_1.1.2      purrr_1.0.1
##  [9] readr_2.1.4      tidyr_1.3.0      tibble_3.2.1     ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4      xfun_0.39         ggrepel_0.9.3     rstatix_0.7.2
##  [5] tzdb_0.4.0        vctrs_0.6.3       tools_4.3.1       generics_0.1.3
##  [9] fansi_1.0.4       highr_0.10        pkgconfig_2.0.3   lifecycle_1.0.3
## [13] compiler_4.3.1    farver_2.1.1      munsell_0.5.0     carData_3.0-5
## [17] httpuv_1.6.11     htmltools_0.5.6   yaml_2.3.7        pillar_1.9.0
## [21] later_1.3.1       car_3.1-2         ggpubr_0.6.0      ellipsis_0.3.2
```

```
## [25] abind_1.4-5       mime_0.12         tidyselect_1.2.0  digest_0.6.33
## [29] stringi_1.7.12    bookdown_0.37     labeling_0.4.3    rprojroot_2.0.3
## [33] fastmap_1.1.1     grid_4.3.1        colorspace_2.1-0  cli_3.6.1
## [37] magrittr_2.0.3    utf8_1.2.3        broom_1.0.5       withr_2.5.0
## [41] shinyBS_0.61.1    scales_1.2.1      promises_1.2.1    backports_1.4.1
## [45] timechange_0.2.0  rmarkdown_2.23    ggsignif_0.6.4    hms_1.1.3
## [49] ggedit_0.3.1      shiny_1.7.5       evaluate_0.21     knitr_1.43
## [53] miniUI_0.1.1.1    rlang_1.1.1       Rcpp_1.0.11       shinyAce_0.4.2
## [57] xtable_1.8-4      glue_1.6.2        jsonlite_1.8.7    rstudioapi_0.15.0
## [61] R6_2.5.1          plyr_1.8.8
```

Chapter 11

Preference Mapping

In general, preference mapping is the name for a broad set of methods that attempts to produce cartesian representations of consumer preferences for a set of products. **External** preference mapping uses some sort of other data (the "external" data)–chemical data, DA data, even product-ingredient data–to explain patterns in consumer liking. **Internal** preference mapping treats consumers themselves as variables explaining the products, and uses simple approaches like PCA to produce a low-dimensional but hopefully informative insight into the patterns of consumer liking.

One of the challenges of preference mapping is the seemingly endless set of options for performing it. As reviewed by Yenket et al. [2011], there are at least 10 distinct approaches to external preference mapping, possibly even more. They each solve slightly different optimization problems, and so produce slightly different results. Therefore, there is a real danger of an analyst going method-by-method until they find one that supports their prior intuition.

On top of that, I personally find that many of the approaches are technically and analytically convoluted and difficult to follow, producing outputs that are prone to misinterpretation. Therefore, in this section we're going to depart pretty strongly from HGH's approach: we're going to entirely ignore `SensoMineR::carto()` and the related workflow, because I find the method itself opaque and the outputs less-than-useful. Instead, we're going to focus on (what I believe to be) the simplest tractable tools for these use cases.

11.1 Data

To do preference mapping, we of course need some consumer data. We have data on consumer responses to the wines profiled in our DA example in the

`torriconsFinal.csv` file. Let's load our standard packages and import the data.

```
library(tidyverse)
library(here)

consumer_data <-read_csv(here("data/torriconsFinal.csv"))

# We'll also need the DA data
descriptive_data <- read_csv(here("data/torriDAFinal.csv")) %>%
  mutate_at(.vars = vars(1:3), ~as.factor(.))

# And finally let's take a look at our data
skimr::skim(consumer_data) %>%
  # This is purely to allow the skimr::skim() to be rendered in PDF, ignore otherwise
  knitr::kable()
```

skim_type	skim_variable	n_missing	complete_rate	numeric.mean	numeric.sd	numeric
numeric	Judge	0	1	53.500000	30.7435630	
numeric	Wine Frequency	0	1	2.018868	0.8942263	
numeric	IT frequency	0	1	2.745283	0.7179511	
numeric	Gender	0	1	1.481132	0.5020175	
numeric	Age	0	1	38.915094	14.4684339	
numeric	C_MERLOT	0	1	5.773585	1.8009681	
numeric	C_SYRAH	0	1	5.537736	2.0150601	
numeric	C_ZINFANDEL	0	1	6.056604	1.7611154	
numeric	C_REFOSCO	0	1	5.443396	2.0427814	
numeric	I_MERLOT	0	1	4.547170	1.9475605	
numeric	I_SYRAH	0	1	5.226415	1.9187447	
numeric	I_PRIMITIVO	0	1	5.009434	2.0023571	
numeric	I_REFOSCO	0	1	4.820755	2.2116452	

So we have a lot of rows, each one of which represents a judge. In fact, we have two kinds of data in this wide data frame: data about consumer preferences (titled with wine names, like `I_SYRAH`) and data about the judges themselves (`Wine Frequency`, `Age`, etc). Let's break these up for ease of use.

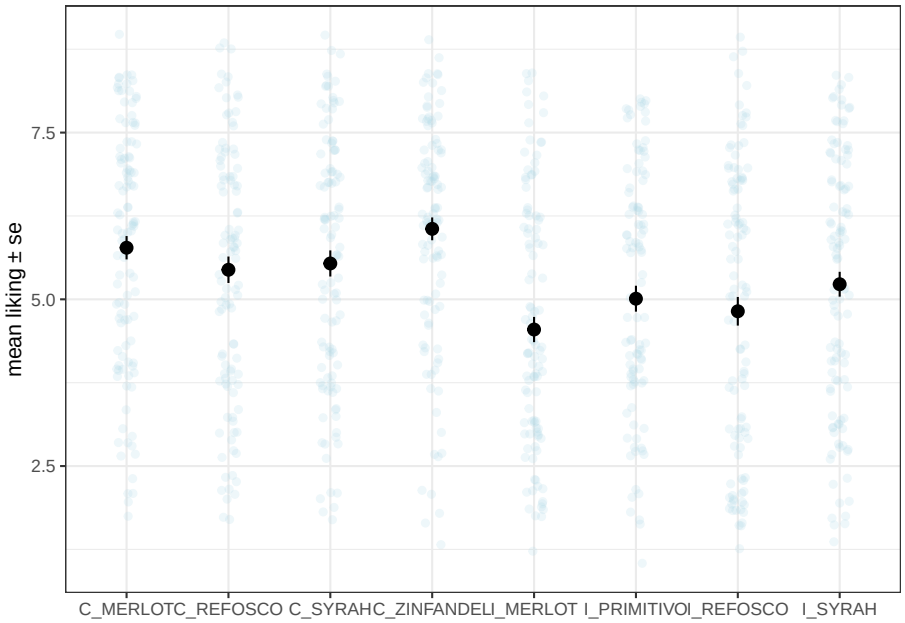
```
consumer_demo <-
  consumer_data %>%
  select(Judge:Age) %>%
  # Everything but Age is a factor anyway, let's fix it
  mutate(across(-Age, ~as.factor(.)))

consumer_data <-
  consumer_data %>%
```

```
  select(Judge, C_MERLOT:I_REFOSCO) %>%
  mutate(Judge = as.factor(Judge))
```

OK, now we're ready to start working! Before we do, let's do a little summary work to get some insight into the data. We might be curious, for example, to get some kind of *mean* $\pm$ *se* estimate for our liking for each wine. Easy enough to do with what we've learned so far:

```
consumer_data %>%
  pivot_longer(-Judge) %>%
  ggplot(aes(x = name, y = value)) +
  geom_jitter(alpha = 1/5, width = 0.1, color = "lightblue") +
  stat_summary() +
  theme_bw() +
  labs(x = NULL, y = "mean liking ± se") +
  scale_y_continuous(limits = c(1, 9))
```

```
## Warning: Removed 21 rows containing missing values (`geom_point()`).
```



See how I scaled that to include the entire range of possible responses on the 9-pt scale? It shows how little variability there actually is in our data.

Anyway, we can see that on average the Californian wines seem to be rated a bit higher in terms of overall liking. If we were interested we might consider

pairing the wine types across region (in this case, we would assume Zinfandel and Primitivo are chosen to be matches for each other) and treat this as a 2-factor design. But that is outside our remit at the moment.

11.2 Internal preference mapping

We'll start out nice and easy with an **internal preference map**. In this approach, we treat each consumer as a variable measured on the wine, and we look for combinations of variables that best explain the variation in the wines. Sound familiar? We're just describing a (covariance) PCA on the consumer data! Let's dive in.

First, we need to transpose our table so that judges are the variables and wines are the observations:

```
internal_map <-
  consumer_data %>%
  column_to_rownames("Judge") %>%
  # Pretty easy - `t()` is the base-`R` transpose function
  t() %>%
  FactoMineR::PCA(graph = FALSE, scale.unit = FALSE)
```

Notice that I just used `t()` to transpose; HGH demonstrated a workflow using the `reshape` package, which we could replicate with some combination of `pivot_*()` functions, but this seemed easier for the application.

Let's take a look at what we got.

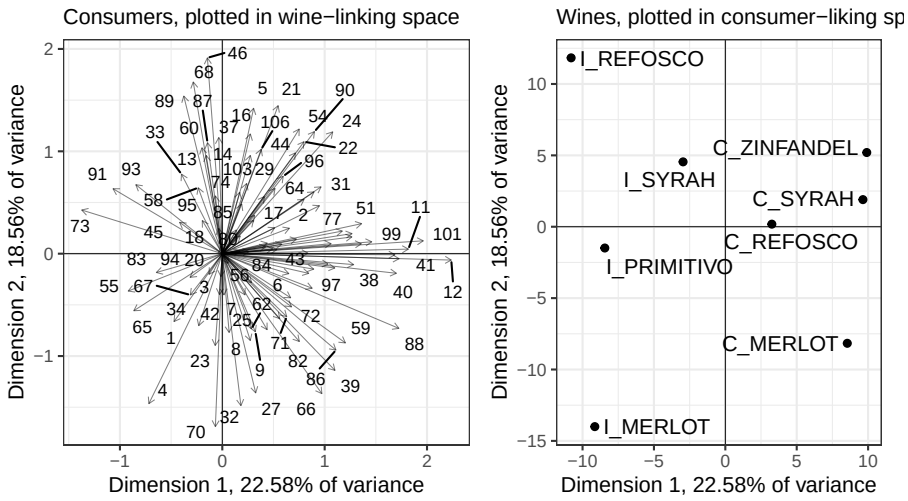
```
internal_coordinates <-
  tibble(point_type = c("subject", "product"),
         data = list(internal_map$var$coord,
                     internal_map$ind$coord)) %>%
  mutate(data = map(data, as_tibble, rownames = "label")) %>%
  unnest(everything())

# The scales for our factor scores are very different, so we will want to plot
# separately and join them up via `patchwork`.

internal_coordinates %>%
  group_by(point_type) %>%
  summarize(across(where(is.numeric), ~mean(abs(.))))
```

```
## # A tibble: 2 x 6
##   point_type Dim.1 Dim.2 Dim.3 Dim.4 Dim.5
```

```
##   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 product    7.84  5.91  6.34  5.47  4.72
## 2 subject    0.639 0.573 0.542 0.496 0.477
```

```
# This is just to avoid re-typing

axis_labels <-
  labs(x = paste0("Dimension 1, ", round(internal_map$eig[1, 2], 2), "% of variance"),
       y = paste0("Dimension 2, ", round(internal_map$eig[2, 2], 2), "% of variance"))

# First the product map

p_internal_products <-
  internal_coordinates %>%
  filter(point_type == "product") %>%
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = label)) +
  coord_equal() +
  labs(subtitle = "Wines, plotted in consumer-liking space") +
  axis_labels +
  theme_bw()

# Then the consumer map

p_internal_consumers <-
  internal_coordinates %>%
  filter(point_type == "subject") %>%
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.05, units = "in"), ends = "first"),
               linewidth = 1/4, alpha = 1/2) +
  ggrepel::geom_text_repel(aes(label = label), size = 3) +
  coord_equal() +
  axis_labels +
  labs(subtitle = "Consumers, plotted in wine-linking space") +
  theme_bw()

library(patchwork)

p_internal_consumers + p_internal_products
```

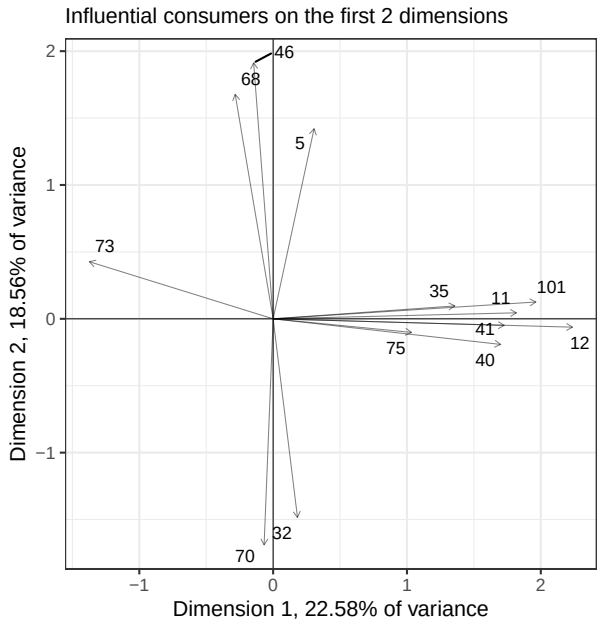
```
## Warning: ggrepel: 29 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



In the original **R Opus**, HGH filtered the consumers to show those with the highest cos^2 values: this measures their effect (in my head I always read "leverage") on the spatial arrangement of the plotted dimensions. She did this using the base-R plotting options for the `carto()` function, and I have no idea how it specifies selection - I have not tried to figure out how that function works in detail. Instead, if we wanted to do something similar we could look into the output of `PCA()` and create a list of influential subjects:

```
influential_subjects <-
  internal_map$var$cos2 %>%
  as_tibble(rownames = "subject") %>%
  filter(Dim.1 > 0.6 | Dim.2 > 0.6) %>%
  pull(subject)

# We can then use these to filter our plot:

internal_coordinates %>%
  filter(point_type == "subject",
         label %in% influential_subjects) %>%
  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
```

```
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.05, units = "in"), ends = "first"),
               linewidth = 1/4, alpha = 1/2) +
  ggrepel::geom_text_repel(aes(label = label), size = 3) +
  coord_equal() +
  axis_labels +
  labs(subtitle = "Influential consumers on the first 2 dimensions") +
  theme_bw()
```



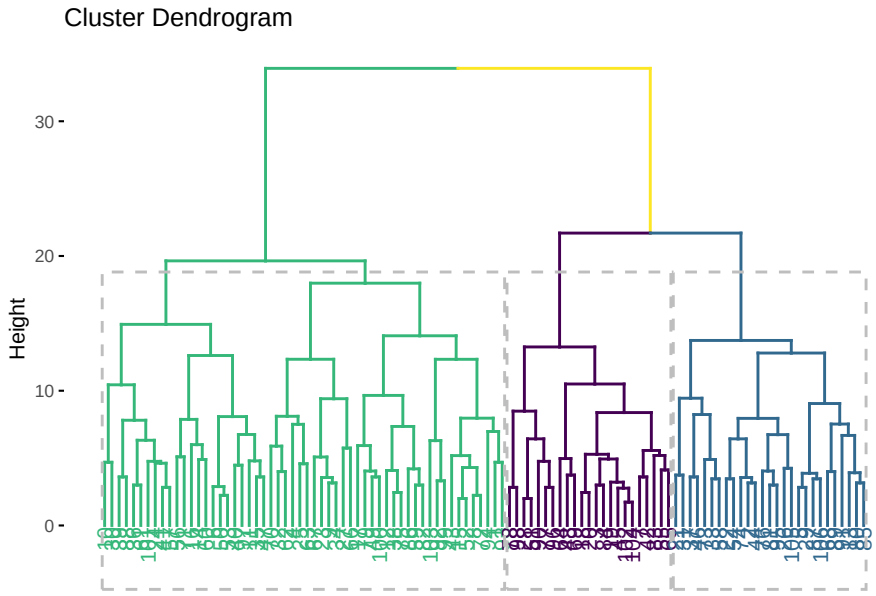
We get fewer points than HGH did; I am assuming that the `SensoMineR` function is filtering its subjects that are influential on ANY of the dimensions of the PCA, not just on the first 2 dimensions, but that seems to be counterindicated (to me) for looking at these 2-dimensional biplots.

One step that HGH didn't take was to cluster the consumers according to their patterns of liking. This is a common end goal of preference mapping in general, and HGH pursued it in the original **R Opus** section on *external* preference mapping. However, there is no reason we can't do it with the results of an internal preference map:

```
internal_consumer_cluster <-
  consumer_data %>%
  column_to_rownames("Judge") %>%
  dist(method = "euclidean") %>%
  hclust(method = "ward.D2")
```

```
internal_consumer_cluster %>%
  factoextra::fviz_dend(k = 3, rect = TRUE) +
  scale_color_viridis_d()
```

```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```



I think there are plausibly 3 clusters of consumers; we could also explore the possibility that there are 2, 4, or even 5. However, 3 seems good to me, because the distance between groups 2 and 3 (blue and green) is quite large, so forcing them to remain in a single group seems less plausible than splitting them.

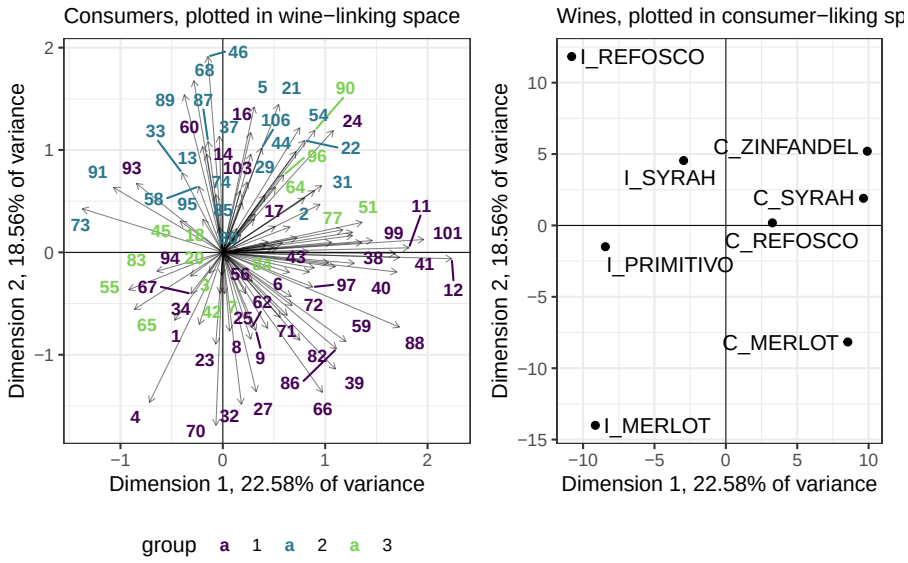
We can take this grouping information and project it back into our preference maps in different ways. I am going to opt for simply replacing the original labels with labels colored by group membership.

```
p_internal_consumers <-
  p_internal_consumers %>%
  ggedit::remove_geom(geom = "text") +
  ggrepel::geom_text_repel(aes(label = label,
                               color = as.factor(cutree(internal_consumer_cluster, k =
                           size = 3, fontface = "bold") +
  scale_color_viridis_d("group", end = 0.8) +
  theme(legend.position = "bottom")
```

```
## Registered S3 method overwritten by 'ggedit':
##   method from
##   +.gg   ggplot2
```

```
p_internal_consumers + p_internal_products
```

```
## Warning: ggrepel: 29 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



We can see that the purple and teal-ish groups are well-separated in this space, whereas the bright green group is not. I suspect (and leave it as an exercise to find out) that this group would be better separated in a third dimension.

We can then use the clusteres identified here to ask questions about, for example, consumer demographic groups. A quick way to do per-group summaries of (often qualitative) variables is using the `skimr::skim()` function with `group_by()`.

```
consumer_demo %>%
  bind_cols(cluster = cutree(internal_consumer_cluster, k = 3)) %>%
  group_by(cluster) %>%
  skimr::skim() %>%
  # This is purely to allow the skimr::skim() to be rendered in PDF, ignore otherwise
  knitr::kable()
```

skim_type	skim_variable	cluster	n_missing	complete_rate	factor.ordered	factor.n_uni
factor	Judge	1	0	1	FALSE	
factor	Judge	2	0	1	FALSE	
factor	Judge	3	0	1	FALSE	
factor	Wine Frequency	1	0	1	FALSE	
factor	Wine Frequency	2	0	1	FALSE	
factor	Wine Frequency	3	0	1	FALSE	
factor	IT frequency	1	0	1	FALSE	
factor	IT frequency	2	0	1	FALSE	
factor	IT frequency	3	0	1	FALSE	
factor	Gender	1	0	1	FALSE	
factor	Gender	2	0	1	FALSE	
factor	Gender	3	0	1	FALSE	
numeric	Age	1	0	1	NA	
numeric	Age	2	0	1	NA	
numeric	Age	3	0	1	NA	

We can see some interesting (if minor!) differences in these groups. Group 1, for example, has more answers for "2" and "1" for wine consumption frequency, proportionally–since I don't have the original data, I have no idea if this is more or less frequent. I am guessing less, though, because group 1 also has a lower mean age and less of a long tale in the Age variable.

11.3 External preference mapping

As I mentioned above, the range of possible approaches to external preference mapping is quite large [Yenket et al., 2011], and I have little interest in exploring all of them. In particular, I find both the interface and the documentation for `SensoMineR` a little hard to grasp, so I am going to avoid its use in favor of approaches that will yield largely the same answers but with more transparencyt (for me).

11.3.1 Partial Least Squares Regression

I believe the most common tool for preference mapping currently used in sensory evaluation is Partial Least Squares Regression (PLS-R). PLS-R is one of a larger family of PLS methods. All PLS methods are concerned with the explanation of two data tables that contain measurements on the same set of observations: in other words, the data tables have the same rows (the observations, say, in our case, *wines*) but different columns (measurements, in our case we have one table with DA ratings and one with consumer ratings). This kind of dual-table data structure is common in food science: we might have a set of chemical measurements and a set of descriptive ratings, or a set of descriptive ratings

and consumer liking measurements (as we do here), or many other possible configurations.

There are other PLS approaches we will not consider here: correlation (in which the two tables are simultaneously explained) and path modeling (which proposes causal links among the variables in multiple tables). We like PLS-Regression in this case because we want to model an *explanatory* relationship: we want to know how the profiles of (mean) DA ratings in our `descriptive_data` table explains the patterns of consumer liking we see in our `consumer_data` table. In addition, to quote Abdi and Williams [2013, 567]:

> As a regression technique, PLSR [sic] is used to predict a whole table of data (by contrast with standard regression which predicts one variable only), and it can also handle the case of multicolinear predictors (i.e., when the predictors are not linearly independent). These features make PLSR [sic] a very versatile tool because it can be used with very large data sets for which standard regression methods fail.

That all sounds pretty good! To paraphrase the same authors, PLS-R finds a *single set* of latent variables from the predictor matrix (`descriptive_data` for us) that best explains the variation in `consumer_data`.

I am not going to try to do further explanation of PLS-R, because it is a rather complex, iterative fitting procedure. The outcome, however, is relatively easy to understand, and we will talk through it once we generate it. For this, we're going to need to load some new packages!

```
library(pls)

descriptive_means <-
  descriptive_data %>%
  select(ProductName, Red_berry:Astringent) %>%
  group_by(ProductName) %>%
  summarize(across(where(is.numeric), mean)) %>%
  column_to_rownames("ProductName") %>%
  as.matrix()

consumer_data_wide <-
  consumer_data %>%
  column_to_rownames("Judge") %>%
  t() %>%
  as.matrix()

pls_data <-
  list(x = descriptive_means,
```

```
       y = consumer_data_wide)

# `plsr()` has an interface that is built to act like `lm()` or `aov()`
pls_fit <-
  plsr(y ~ x,
       data = pls_data,
       scale = TRUE)

summary(pls_fit)
```

```
## Data:    X dimension: 8 20
##  Y dimension: 8 106
## Fit method: kernelpls
## Number of components considered: 7
## TRAINING: % variance explained
##        1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X    3.660e+01  61.8849   74.726   84.556   93.330    98.51      100
## 1    1.793e+00   5.6562   11.742   41.441   90.691    90.85      100
## 2    1.860e+01  43.9844   46.198   62.851   65.184    92.07      100
## 3    7.218e+00  10.7913   14.893   22.859   94.121    95.10      100
....
```

I only printed out the first few lines (as otherwise the `summary()` function will print several hundred lines). Overall we see *almost* the same results as HGH did (good news). However, we are not exactly the same in terms of variance explained. I wonder if this is because HGH lost subject #1 in data wrangling? I can't explain the difference otherwise. We'll see that our solutions do start to diverge as we go on.

HGH decided to only examine 2 components, but I believe that if we investigate cross-validation we would probably choose to retain only a single component based on $PRESS$ values:

```
pls_val <- plsr(y ~ x, data = pls_data, scale = TRUE, validation = "LOO")

# PRESS values for the predictor (DA) variables
pls_val$validation$PRESS[1, ]
```

```
##  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## 43.08863 62.24039 78.94974 62.31154 56.22270 87.64232
```

We can see that $PRESS$ is lowest with only a single component.

...but, I am lazy and more interested in showing *how* to do things than making the multiple plots and analyses we'd need for 3 dimensions. I leave investigating the 3rd component as an exercise for the reader.

Following our typical practice, we will see how to extract elements of the output of `plsr()` to produce nicer looking plots in `ggplot2`. This will take a little digging because the `pls` package makes use of a lot of (what I consider) tricky and obtuse indexing and object formats. Bear with me.

```
# The "scores" are the factor scores for the observations (wines) based on the
# predictor (DA) variables
pls_scores <- scores(pls_fit)

# Note this ugly output object of class "scores".  This is a problem.
pls_scores
```

```
##                  Comp 1     Comp 2     Comp 3      Comp 4      Comp 5
## C_MERLOT     1.48290560 -1.0856932  1.6664093  0.08429563  0.67322911
## C_SYRAH      2.84859866 -2.6602500 -1.6667701  1.55380857 -1.32209708
## C_ZINFANDEL  0.03184129 -0.9422643 -1.6209039 -2.16194022  0.08136815
## C_REFOSCO   -0.43645910 -1.8055096  0.6126194  0.10221889  2.47673917
## I_MERLOT     2.34624636  2.6446633  1.9018586  0.80383273 -0.57175907
## I_SYRAH     -3.51138619 -1.1718605  1.6098890 -1.03261515 -1.84741358
## I_PRIMITIVO  1.58414715  3.3609231 -1.1958901 -1.20118210  0.15460504
## I_REFOSCO   -4.34589378  1.6599913 -1.3072122  1.85158164  0.35532825
##                 Comp 6       Comp 7
## C_MERLOT    -1.1483719 -0.938560264
## C_SYRAH      0.4962983 -0.009963997
## C_ZINFANDEL -1.3552622  0.470925662
## C_REFOSCO    1.1209200  0.409238012
## I_MERLOT    -0.4332254  0.739342596
## I_SYRAH      0.7803878 -0.057086046
## I_PRIMITIVO  1.1672424 -0.509146097
## I_REFOSCO   -0.6279891 -0.104749865
## attr(,"class")
## [1] "scores"
## attr(,"explvar")
##    Comp 1    Comp 2    Comp 3    Comp 4    Comp 5    Comp 6    Comp 7
## 36.598160 25.286782 12.841009  9.829632  8.774663  5.176661  1.493093
```

```
# My solution is old-fashioned class coercion:
class(pls_scores) <- "matrix"

pls_scores <-
  pls_scores %>%
  as_tibble(rownames = "wine")

pls_scores
```

```
## # A tibble: 8 x 8
##   wine        `Comp 1` `Comp 2` `Comp 3` `Comp 4` `Comp 5` `Comp 6` `Comp 7`
##   <chr>          <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 C_MERLOT      1.48     -1.09     1.67    0.0843   0.673    -1.15   -0.939
## 2 C_SYRAH       2.85     -2.66    -1.67    1.55    -1.32      0.496  -0.00996
## 3 C_ZINFANDEL   0.0318   -0.942   -1.62   -2.16     0.0814   -1.36    0.471
## 4 C_REFOSCO    -0.436    -1.81     0.613   0.102    2.48      1.12    0.409
## 5 I_MERLOT      2.35      2.64     1.90    0.804   -0.572    -0.433   0.739
## 6 I_SYRAH      -3.51     -1.17     1.61   -1.03    -1.85      0.780  -0.0571
## 7 I_PRIMITIVO   1.58      3.36    -1.20   -1.20     0.155     1.17   -0.509
## 8 I_REFOSCO    -4.35      1.66    -1.31    1.85     0.355    -0.628  -0.105
```

We have to take the same approach for the loadings:

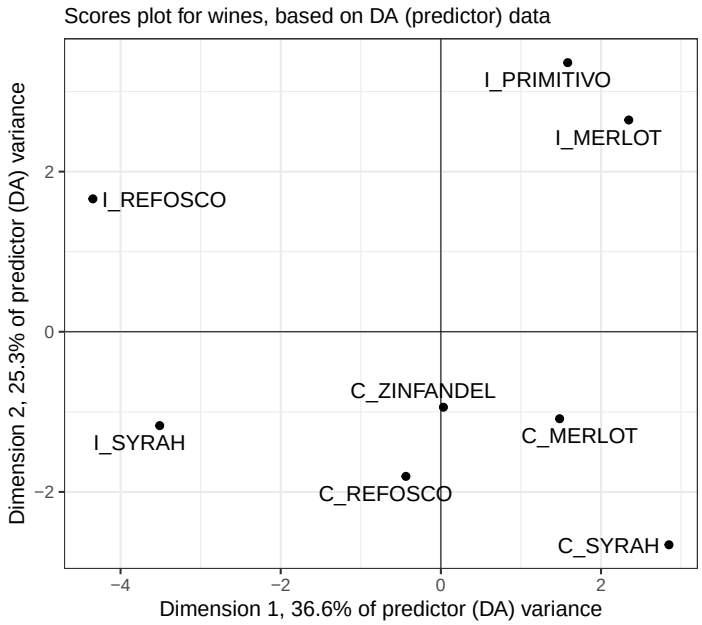
```
# The "scores" are the factor loadings for the predictor (DA) variables that
# explain how the scores for the wines are generated.
pls_loadings <- loadings(pls_fit)

class(pls_loadings) <- "matrix"

pls_loadings <-
  pls_loadings %>%
  as_tibble(rownames = "descriptor")

pls_loadings
```

```
## # A tibble: 20 x 8
##    descriptor     `Comp 1` `Comp 2` `Comp 3` `Comp 4` `Comp 5` `Comp 6` `Comp 7`
##    <chr>             <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 Red_berry      -0.299    -0.0372   0.255     0.0284 -0.161     0.333    0.317
##  2 Dark_berry     -0.331     0.0726  -0.124     0.127   0.135    -0.108   -0.517
##  3 Jam            -0.362    -0.0338   0.0276   -0.0759 -0.102     0.155   -0.174
##  4 Dried_fruit    -0.0884    0.214    0.00334   0.192   0.600    -0.129   -0.0280
##  5 Artificial_fr~ -0.340     0.0425  -0.0836    0.111  -0.215     0.113   -0.337
##  6 Chocolate       0.00735  -0.329   -0.265     0.126  -0.367    -0.0988  -0.133
##  7 Vanilla        -0.227    -0.335   -0.0582    0.0340 -0.0204    0.251    0.0659
##  8 Oak             0.205    -0.243   -0.0606    0.153   0.401     0.248   -0.192
##  9 Burned          0.239    -0.217   -0.245     0.239  -0.214    -0.119   -0.0742
## 10 Leather         0.261     0.244   -0.174    -0.139   0.00743   0.314   -0.103
## 11 Earthy          0.299     0.188   -0.164    -0.115  -0.201    -0.157    0.0185
## 12 Spicy           0.118     0.0269  -0.0704    0.580  -0.0442    0.433    0.313
## 13 Pepper          0.188     0.187    0.359     0.273  -0.00282  -0.286   -0.234
## 14 Grassy         -0.253     0.224    0.199     0.157  -0.203    -0.251    0.178
## 15 Medicinal       0.187     0.281   -0.0652   -0.385  -0.0773    0.214    0.00614
## 16 Band-aid        0.250     0.275   -0.0976   -0.0698 -0.231     0.221   -0.0170
```

```
## 17 Sour          -0.0778    0.370  -0.262    -0.129   0.0735    0.226  -0.112
## 18 Bitter        -0.166     0.270  -0.344     0.145  -0.0339   -0.247   0.325
## 19 Alcohol       -0.0999   -0.103  -0.475    -0.287   0.230    -0.126   0.317
## 20 Astringent     0.120     0.282  -0.349     0.307   0.00121  -0.0517 -0.0935
```

Now we can make the first two plots that HGH showed:

```
# We'll reuse this

axis_labels <-
  labs(x = paste0("Dimension 1, ", round(explvar(pls_fit)[1], 1), "% of predictor (DA) variance")
       y = paste0("Dimension 2, ", round(explvar(pls_fit)[2], 1), "% of predictor (DA) variance")

pls_scores %>%
  ggplot(aes(x = `Comp 1`, y = `Comp 2`)) +
  geom_vline(xintercept = 0, linewidth = 1/4) +
  geom_hline(yintercept = 0, linewidth = 1/4) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = wine)) +
  coord_equal() +
  theme_bw() +
  axis_labels +
  labs(subtitle = "Scores plot for wines, based on DA (predictor) data")
```

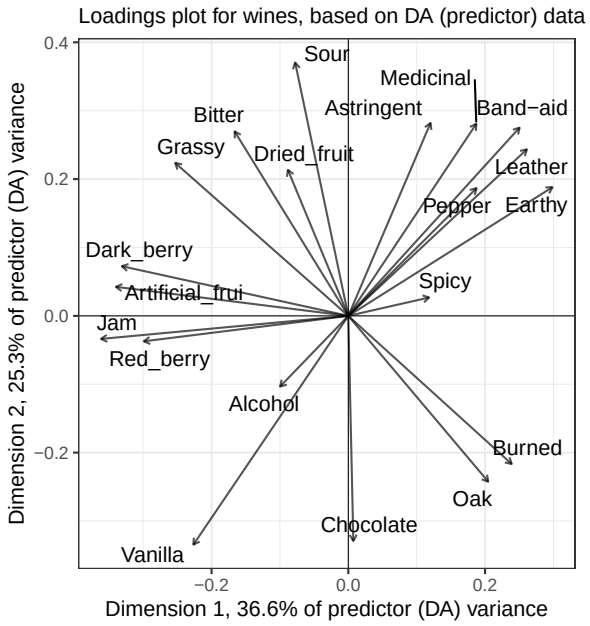


This plot is quite different from HGH's, but I confirmed that even using the

same base-R plotting methods, I got the same plot shown here (but uglier). Is it possible subject #1 made that big a difference?

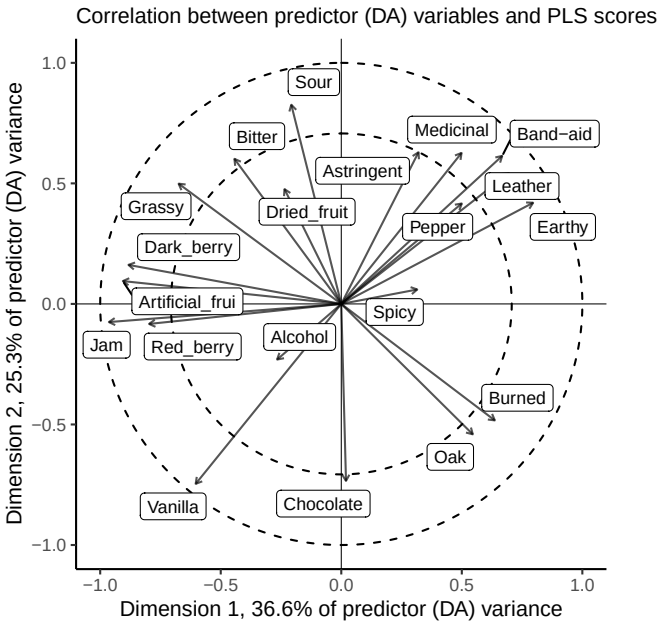
As you might guess, our loadings are also going to end up somewhat different:

```
pls_loadings %>%
  ggplot(aes(x = `Comp 1`, y = `Comp 2`)) +
  geom_vline(xintercept = 0, linewidth = 1/4) +
  geom_hline(yintercept = 0, linewidth = 1/4) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.05, units = "in"), ends = "first"),
               linewidth = 1/2, alpha = 2/3) +
  ggrepel::geom_text_repel(aes(label = descriptor)) +
  coord_equal() +
  axis_labels +
  labs(subtitle = "Loadings plot for wines, based on DA (predictor) data") +
  theme_bw()
```




```
# We start with calculating the correlation betwee the original predictors and
# the factor scores for the wines from the DA in the PLS fit.
cor(pls_data$x, column_to_rownames(pls_scores, "wine")) %>%
  as_tibble(rownames = "descriptor") %>%
  ggplot(aes(x = `Comp 1`, y = `Comp 2`)) +
  geom_vline(xintercept = 0, linewidth = 1/4) +
  geom_hline(yintercept = 0, linewidth = 1/4) +
```

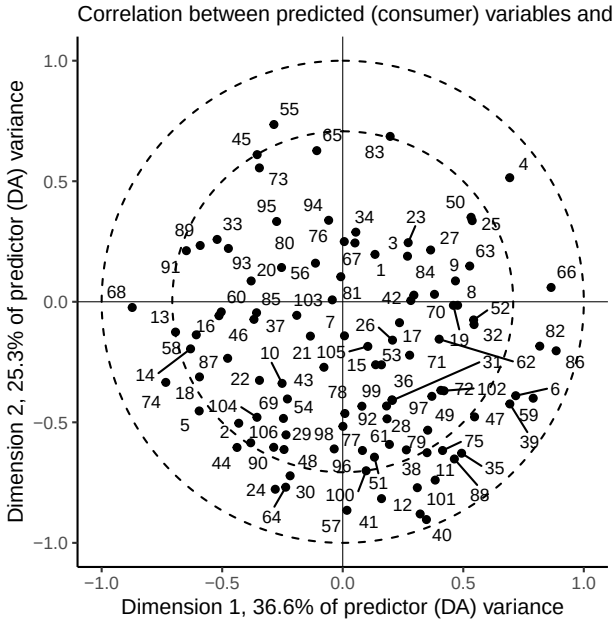
```
  ggforce::geom_circle(aes(x0 = x, y0 = y, r = r),
                       data = crossing(x = 0, y = 0, r = c(sqrt(0.5), 1)),
                       inherit.aes = FALSE,
                       linetype = 2) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.05, units = "in"), ends = "first"),
               linewidth = 1/2,
               alpha = 2/3) +
  ggrepel::geom_label_repel(aes(label = descriptor), size = 3) +
  coord_equal(xlim = c(-1, 1), ylim = c(-1, 1)) +
  axis_labels +
  labs(subtitle = "Correlation between predictor (DA) variables and PLS scores") +
  theme_classic()
```



The correlation plot gives us similar information to the loadings plot, above, but it is scaled so that the squared distance between each descriptor and the origin represents the amount of variance for that variable explained by the two dimensions visualized. The dashed circles are a visual aid for 50% and 100% of variance. For this, it is clear that our 2-dimensional PLS-R fit does a good job of explaining most of the variation in the large majority of descriptors; notable exceptions are `Alcohol`, `Dried_fruit`, and `Spicy`.

We can take the same approach to understand our outcome (consumer) variables:

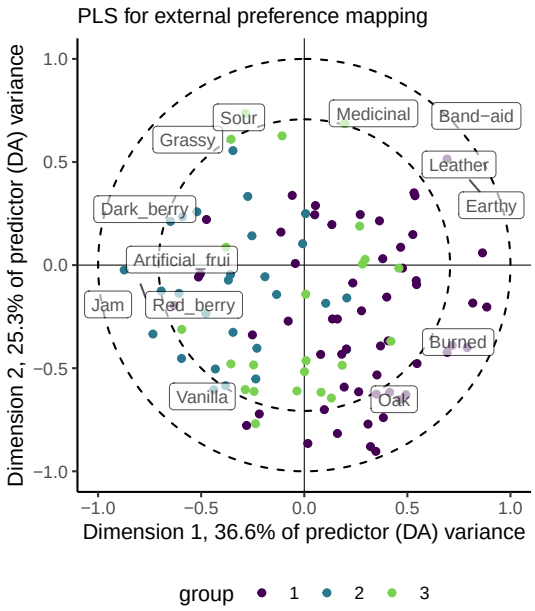
```
cor(pls_data$y, column_to_rownames(pls_scores, "wine")) %>%
  as_tibble(rownames = "consumer") %>%
  ggplot(aes(x = `Comp 1`, y = `Comp 2`)) +
  geom_vline(xintercept = 0, linewidth = 1/4) +
  geom_hline(yintercept = 0, linewidth = 1/4) +
  ggforce::geom_circle(aes(x0 = x, y0 = y, r = r),
                       data = crossing(x = 0, y = 0, r = c(sqrt(0.5), 1)),
                       inherit.aes = FALSE,
                       linetype = 2) +
  geom_point() +
  ggrepel::geom_text_repel(aes(label = consumer), size = 3) +
  coord_equal(xlim = c(-1, 1), ylim = c(-1, 1)) +
  axis_labels +
  labs(subtitle = "Correlation between predicted (consumer) variables and PLS scores") +
  theme_classic()
```



And this is where our external preference map can come in. Let's go ahead and segment these consumers by the clusters we identified previously:

```
cor(pls_data$y, column_to_rownames(pls_scores, "wine")) %>%
  as_tibble(rownames = "consumer") %>%
  # Notice we are relying on positional matching here, generally a bad idea
  mutate(cluster = as.factor(cutree(internal_consumer_cluster, k = 3))) %>%
  ggplot(aes(x = `Comp 1`, y = `Comp 2`)) +
```

```
  geom_vline(xintercept = 0, linewidth = 1/4) +
  geom_hline(yintercept = 0, linewidth = 1/4) +
  ggforce::geom_circle(aes(x0 = x, y0 = y, r = r),
                       data = crossing(x = 0, y = 0, r = c(sqrt(0.5), 1)),
                       inherit.aes = FALSE,
                       linetype = 2) +
  geom_point(aes(color = cluster)) +
  # Here we are adding the correlations of the descriptor data, filtered so that
  # the total variance explained in the 1st 2 dimensions is > 0.75 (pretty
  # arbitrarily)
  ggrepel::geom_label_repel(data = cor(pls_data$x, column_to_rownames(pls_scores, "wine")) %>%
                              as_tibble(rownames = "descriptor") %>%
                              filter(sqrt(`Comp 1`^2 + `Comp 2`^2) > 3/4),
                            mapping = aes(label = descriptor),
                            alpha = 2/3, size = 3) +
coord_equal(xlim = c(-1, 1), ylim = c(-1, 1)) +
  axis_labels +
  labs(subtitle = "PLS for external preference mapping") +
  theme_classic() +
  scale_color_viridis_d("group", end = 0.8) +
  theme(legend.position = "bottom")
```



We're doing a lot in this plot, but hopefully the previous ones have prepared you! This shows us the same consumer plot as above, but we've clustered the consumers by the similarity between their liking scores, as we did in the internal

preference mapping above. We could change this choice–perhaps cluster them based on their loadings in the 1st 2 dimensions of the PLS-R space–but I decided this was the way I wanted to go.

We see a similar pattern of separation of these groups as we did with the internal preference mapping, but now we have the ability to explain the data optimally: it appears the largest cluster (cluster 1) prefers wines that have proportionally higher ratings in attributes like `Oak`, `Burned`, and `Earthy`, while cluster 2 prefers `Sour` wines that are more characterized witht the various fruit flavors. As before, cluster 3 appears to be poorly represented in this space. We'd have to look into the further components to understand their preferences better (and it's not clear to me whether or not we should, based on the $PRESS$ values).

11.4 Other approaches

Beyond PLS-R and the models provided in `SensoMineR`, there are MANY models that can fit the goals of preference mapping: simultaneously explaining 2 data tables that have a variable linking them. One I have used more extensively in the past is Clustering around Latent Variables (CLV), which has its own `R` package and good tutorial here. Hopefully going this extensively through PLS-R gives you enough of a background to tackle this approach (and others!) if you choose to use them.

Whatever approach you choose, though, I would recommend you err on the side of analytical simplicity: preference mapping is one of those areas in sensory evaluation in which apparently unending analytical sophistication is possible, but I have found that the gains in explanatory power or insight is limited. Remember that we are typically working with "small" data that is likely prone to random and sampling error, and so bringing huge power to bear on it is probably overkill. In addition, excessively complex methods are more likely to trip you up (it's happened to me!) and to lose your audience, thus reducing your overall impact.

11.5 Principal Components Regression

To my understanding, Principal Components Regression (PCR) is a type of regression procedure that models a *univariate* outcome (a single $\mathbf{y}$ variable) based on the principal components (estimated through typical SVD) of a set of $\mathbf{X}$ predictors. So I am not entirely sure why HGH chose to apply PCR to the 2-table data (with a set of $\mathbf{Y}$ consumer liking outcomes), or even how it fit properly! PCR is not widely used in sensory evaluation, and so I am a little less familiar with it. The documentation for `?pls::pcr` is not at all detailed.

Inspecting the results that HGH got for PCR in the original **R Opus**, it appears that the results are almost *identical* to those of PLS-R with a rotation around

the y-axis. So I can't imagine this is worth our time, especially for a method that is a) ill-defined in the documentation and b) seems to have been superseded by PLS methods in our field (and superseded by ridge/lasso regression elsewhere).

11.6 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LA
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] pls_2.8-2       patchwork_1.1.2 here_1.0.1      lubridate_1.9.2
##  [5] forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1
##  [9] readr_2.1.4     tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_1.2.0     viridisLite_0.4.2    farver_2.1.1
##  [4] viridis_0.6.4        fastmap_1.1.1        tweenr_2.0.2
##  [7] promises_1.2.1       digest_0.6.33        mime_0.12
## [10] estimability_1.4.1   timechange_0.2.0     lifecycle_1.0.3
## [13] factoextra_1.0.7     cluster_2.1.4        ellipsis_0.3.2
## [16] multcompView_0.1-9   magrittr_2.0.3       compiler_4.3.1
## [19] rlang_1.1.1          tools_4.3.1          utf8_1.2.3
## [22] yaml_2.3.7           knitr_1.43           ggsignif_0.6.4
## [25] skimr_2.1.5          labeling_0.4.3       htmlwidgets_1.6.2
## [28] bit_4.0.5            scatterplot3d_0.3-44 shinyAce_0.4.2
## [31] plyr_1.8.8           repr_1.1.6           abind_1.4-5
```

```
## [34] miniUI_0.1.1.1       withr_2.5.0          polyclip_1.10-4
## [37] grid_4.3.1           fansi_1.0.4          ggpubr_0.6.0
## [40] xtable_1.8-4         colorspace_2.1-0     emmeans_1.8.7
## [43] scales_1.2.1         MASS_7.3-60          flashClust_1.01-2
## [46] cli_3.6.1            mvtnorm_1.2-2        rmarkdown_2.23
## [49] crayon_1.5.2         generics_0.1.3       rstudioapi_0.15.0
## [52] tzdb_0.4.0           ggforce_0.4.1        ggedit_0.3.1
## [55] parallel_4.3.1       base64enc_0.1-3      vctrs_0.6.3
## [58] jsonlite_1.8.7       carData_3.0-5        bookdown_0.37
## [61] car_3.1-2            hms_1.1.3            bit64_4.0.5
## [64] ggrepel_0.9.3        rstatix_0.7.2        FactoMineR_2.8
## [67] dendextend_1.17.1    shinyBS_0.61.1       glue_1.6.2
## [70] DT_0.28              stringi_1.7.12       gtable_0.3.4
## [73] later_1.3.1          munsell_0.5.0        pillar_1.9.0
## [76] htmltools_0.5.6      R6_2.5.1             rprojroot_2.0.3
## [79] shiny_1.7.5          vroom_1.6.3          evaluate_0.21
## [82] lattice_0.21-8       highr_0.10           backports_1.4.1
## [85] leaps_3.1            broom_1.0.5          httpuv_1.6.11
## [88] Rcpp_1.0.11          coda_0.19-4          gridExtra_2.3
## [91] xfun_0.39            pkgconfig_2.0.3
```

Chapter 12

Multiple Factor(ial) Analysis (MFA)

Multiple Factor Analysis MFA; sometimes called Multiple Factor*ial* Analysis, to distinguish from the "Factor Analysis" approaches related to Exploratory and Confirmatory Factor Analysis, see Rencher [2002]] is a surprisingly straightforward extension to PCA that allows the comparison of multiple sets of data collected on the same observations. It is therefore suitable for use on the same sorts of data as PLS and other multi-table approaches. Unlike those approaches, however, MFA is (in its basic forms) a purely *exploratory* data analysis (in the same was as PCA is!)–PLS-R, by contrast, is inferential, claiming that the $\mathbf{Y}$ matrix can be explained by the variables in the $\mathbf{X}$ matrix.

The basic idea of MFA is very simple: given multiple sets of measurements (sets of columns or variables) collected on the same observations (rows), each set can be "normalized" by its first singular value so that all the sets are "commensurate" for a PCA. To put it another way, MFA consists of a sequence of PCAs: first each set of measurements is analyzed by PCA (really, Singular Value Decompositon) to extract its first singular value; then each element of each set of measurements is divided by its respective singular value; finally, the normalized tables are (row-wise) concatenated into a single table, which is then analyzed by a second PCA (SVD). A more complete and detailed explanation is given in Abdi et al. [2013].

In comparison to standard PCA, MFA has several advantages. The primary advantage is that it is possible not only to assess the role of rows (observations or samples) and columns (variables or measurements), but also tables themselves. This allows us to get insight into the agreement and disagreement between different *sets* of measurements: in sensory evaluation, this is particularly helpful when we have multiple (untrained?) subjects, or subjects using different sets of descriptors to assess products.

In the original **R Opus**, HGH analyzed the DA and the consumer data for the wines by MFA. We will follow this, but we will start by first analyzing only the DA data by MFA in order to gain a feel for the methodology (and to avoid boredom with results that will be quite similar to the results from our Preference Mapping section). We will use `FactoMineR` for the `MFA()` function.

```
library(tidyverse)
library(here)
library(FactoMineR)
library(paletteer)

# This is all just standard set up

descriptive_data <-
  read_csv(here("data/torriDAFinal.csv")) %>%
  mutate(across(1:3, ~as.factor(.)))

consumer_data <- read_csv(here("data/torriconsFinal.csv"))

consumer_demo <-
  consumer_data %>%
  select(Judge:Age) %>%
  mutate(across(Judge:Age, ~as.factor(.)))

consumer_data <-
  consumer_data %>%
  select(-(`Wine Frequency`:Age)) %>%
  mutate(Judge = as.factor(Judge))
```

12.1 MFA vs PCA

Typically, we have been conducting PCA on a matrix of mean values for each wine and descriptor: we have averaged across judge and across replication. But we can instead treat these as individual data tables to be analyzed by MFA. Let's contrast the results.

```
descriptive_pca <-
  descriptive_data %>%
  group_by(ProductName) %>%
  summarize(across(where(is.numeric), ~mean(.))) %>%
  column_to_rownames("ProductName") %>%
  # Because MFA typically scales all columns to unit variance, we will do so as
  # well
  PCA(scale.unit = TRUE, graph = FALSE)
```

```
descriptive_very_wide_data <-
  descriptive_data %>%
  pivot_longer(-c(NJ, ProductName, NR)) %>%
  pivot_wider(names_from = c(name, NJ, NR),
              values_from = value,
              names_vary = "fastest",
              names_sep = ".")

glimpse(descriptive_very_wide_data)
```

```
## Rows: 8
## Columns: 841
## $ ProductName            <fct> C_MERLOT, C_SYRAH, C_ZINFANDEL, C_REFOSCO, I_ME~
## $ Red_berry.1331.7       <dbl> 5.1, 5.6, 4.9, 5.0, 3.3, 5.7, 2.9, 3.2
## $ Dark_berry.1331.7      <dbl> 5.8, 1.9, 2.6, 1.9, 7.2, 3.6, 5.1, 6.0
## $ Jam.1331.7             <dbl> 2.1, 3.9, 1.4, 7.8, 0.5, 8.7, 8.7, 4.0
## $ Dried_fruit.1331.7     <dbl> 4.7, 1.2, 5.9, 0.6, 5.8, 1.9, 0.4, 0.7
## $ Artificial_frui.1331.7 <dbl> 1.0, 7.9, 0.8, 6.6, 0.7, 7.4, 6.2, 4.1
## $ Chocolate.1331.7       <dbl> 2.9, 1.0, 2.0, 6.4, 2.1, 3.3, 3.4, 3.6
## $ Vanilla.1331.7         <dbl> 5.0, 8.3, 2.7, 5.5, 1.3, 6.9, 8.1, 4.8
....
```

We have made an *extremely* wide data frame with a set of 20 columns for each unique combination of `NR` (rep) and `NJ` (judge). We can now use that information to create an MFA that will give a best compromise among all these measurements.

```
descriptive_mfa <- descriptive_very_wide_data %>%
  column_to_rownames("ProductName") %>%
  # `group = ` specifies the sets of measurements by length.  We have 42 tables
  # of 20 descriptors.
  MFA(group = rep(20, times = 42), graph = FALSE)

descriptive_mfa
```

```
## **Results of the Multiple Factor Analysis (MFA)**
## The analysis was performed on 8 individuals, described by 840 variables
## *Results are available in the following objects :
##
##   name                 description
## 1 "$eig"               "eigenvalues"
## 2 "$separate.analyses" "separate analyses for each group of variables"
## 3 "$group"             "results for all the groups"
```

```
## 4 "$partial.axes"      "results for the partial axes"
## 5 "$inertia.ratio"     "inertia ratio"
## 6 "$ind"               "results for the individuals"
## 7 "$quanti.var"        "results for the quantitative variables"
## 8 "$summary.quanti"    "summary for the quantitative variables"
## 9 "$global.pca"        "results for the global PCA"
```

We have a structure similar to the output of `PCA()` that we can explore.

```
# Like in PCA, the `$ind$coord matrix stores the compromise factor score for the
# observations
descriptive_mfa$ind$coord
```

```
##                  Dim.1     Dim.2     Dim.3     Dim.4      Dim.5
## C_MERLOT     0.7723305 -4.171193 -2.954031 -5.620380 -0.7324718
## C_SYRAH     -1.4562624 -2.849077  9.074701  1.411735 -4.7204734
## C_ZINFANDEL  0.1367775 -1.711573  2.670878 -5.363997  7.6147229
## C_REFOSCO    2.6219121 -8.087547 -3.874773  5.648048 -0.3530799
## I_MERLOT     6.9880002  4.766397 -3.438988 -1.792783 -4.0954920
## I_SYRAH     -6.6389771  3.068949 -2.518430  5.198407  3.8804125
## I_PRIMITIVO -8.3061000  3.190520 -1.866387 -2.559056 -4.0406986
## I_REFOSCO    5.8823192  5.793524  2.907029  3.078027  2.4470802
```

```
# UNLIKE in PCA, the `$ind$coord.partiel matrix stores the projected coordinates
# for each observation in each table.
descriptive_mfa$ind$coord.partiel
```

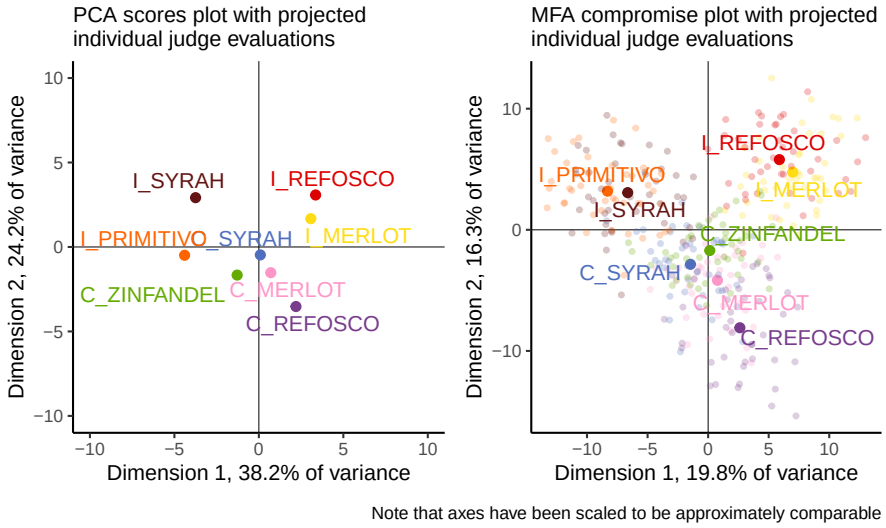
```
##                        Dim.1         Dim.2        Dim.3        Dim.4
## C_MERLOT.Gr1      -0.19053168  -3.629446503  -2.68596672  -4.40043966
## C_MERLOT.Gr2      -4.78396619  -4.926014582  -5.51062842  -7.04473809
## C_MERLOT.Gr3      -0.74993145  -3.018605425  -0.61136476  -2.34024347
## C_MERLOT.Gr4       2.22210511  -2.099845596  -1.81557499  -8.82012771
## C_MERLOT.Gr5      -1.80642414  -7.245935797  -4.44630035  -1.33456791
## C_MERLOT.Gr6       2.82800808  -1.078390329  -7.00711321  -7.38923231
## C_MERLOT.Gr7       4.34894523  -2.337919303  -5.41346962  -7.97502412
## C_MERLOT.Gr8       0.12792494  -9.011485159  -5.29773974  -4.63584711
## C_MERLOT.Gr9      -1.34091440  -5.250368152  -5.08971727  -4.74163321
## C_MERLOT.Gr10      3.57275492 -10.985929526  -7.20226887 -11.18033073
## C_MERLOT.Gr11      5.61137737   0.996081536  -5.33308618  -6.15207138
## C_MERLOT.Gr12      0.84579635  -7.098427362  -0.10136758  -3.26800472
## C_MERLOT.Gr13      0.22303333   1.068272126  -2.40859820  -6.28148245
## C_MERLOT.Gr14      2.98548322  -2.939548397  -3.91949534  -5.52822662
## C_MERLOT.Gr15     -2.42931973  -3.148890253  -2.67932877  -3.21070776
## C_MERLOT.Gr16      1.86168446  -8.336718486  -2.09371088 -15.60286176
```

```
## C_MERLOT.Gr17      2.29846056  -1.483202382   0.50942716  -3.78207470
## C_MERLOT.Gr18      1.87592509  -3.740508610  -0.52836410  -4.69688710
## C_MERLOT.Gr19     -1.75269487  -1.340179706   1.82418023  -1.58732032
....
```

We can use these projected (partial) coordinates to get some ideas about the consensus (or lack thereof) among our judges. Let's wrangle.

```
library(patchwork)

p_descriptive_mfa <-
  descriptive_mfa$ind$coord %>%
  as_tibble(rownames = "wine") %>%
  bind_rows(
    descriptive_mfa$ind$coord.partiel %>%
      as_tibble(rownames = "wine")
  ) %>%
  separate(wine, into = c("wine", "table"), sep = "\\.") %>%
  mutate(table = replace_na(table, "compromise")) %>%
  ggplot(aes(x = Dim.1, y = Dim.2, color = wine)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point(data = . %>% filter(table == "compromise"),
             size = 2) +
  geom_point(data = . %>% filter(table != "compromise"),
             size = 1, alpha = 1/4) +
  ggrepel::geom_text_repel(data = . %>% filter(table == "compromise"),
                           aes(label = wine)) +
  coord_equal() +
  scale_color_paletteer_d("RSkittleBrewer::smarties") +
  labs(subtitle = "MFA compromise plot with projected\nindividual judge evaluations",
       x = paste0("Dimension 1, ", round(descriptive_mfa$eig[1, 2], 1), "% of variance"),
       y = paste0("Dimension 2, ", round(descriptive_mfa$eig[2, 2], 1), "% of variance")) +
  theme_classic() +
  theme(legend.position = "none")

p_descriptive_pca <-
  descriptive_pca$ind$coord %>%
  as_tibble(rownames = "wine") %>%
  ggplot(aes(x = Dim.1, y = Dim.2, color = wine)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point(size = 2) +
  ggrepel::geom_text_repel(aes(label = wine)) +
  coord_equal(xlim = c(-10, 10), ylim = c(-10, 10)) +
```

```
  scale_color_paletteer_d("RSkittleBrewer::smarties") +
  labs(subtitle = "PCA scores plot with projected\nindividual judge evaluations",
       x = paste0("Dimension 1, ", round(descriptive_pca$eig[1, 2], 1), "% of variance
       y = paste0("Dimension 2, ", round(descriptive_pca$eig[2, 2], 1), "% of variance
  theme_classic() +
  theme(legend.position = "none")

p_descriptive_pca + p_descriptive_mfa +
  plot_annotation(caption = "Note that axes have been scaled to be approximately compa
```



We can see some major differences (as well as some similarities!) between these analyses:

- The first two dimensions of the MFA explain far less variance than those of the PCA. Why? Without resorting to talking about dimensionality, we can merely observe that our standard approach to PCA (as above) is to use mean vectors, which means we are literally averaging out variance before we analyze. MFA is perhaps more faithful as a *descriptive* method, which can be seen by…
- …the individual judge's observations, projected into the MFA plot. We can see that while there is definitely *not* complete consensus about each wine based on individual tables, there is actually remarkable agreement: the colored pale points tend to be closer to the their respective compromise (the dark point) than to other points. This is good news for the quality of

our analysis, while also giving us a good picture of how much variability we *actually* have!

- In fact, our spatial configuration is not very different between the two. We see that the separation between the `I_PRIMITIVO` and `I_SYRAH` from the PCA seems to not be supported by the MFA, and ditto for the `I_REFOSCO` and `I_MERLOT`, but otherwise the overall space looks pretty similar.

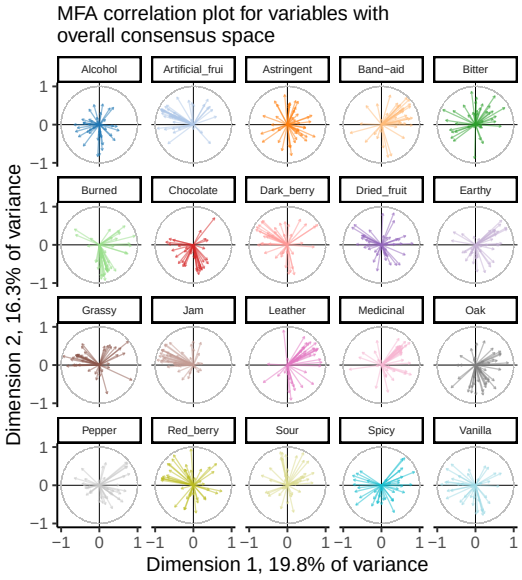
We can take a similar approach to understanding the loadings in the space, but because of the large number of variables to plot and distinguish with aesthetic elements like color, we're going to use `facet_wrap()` to break up the plot into a set, with 1 plot per attribute.

```
# We can see a similar structure for the variables - one per table
descriptive_mfa$quanti.var$coord
```

```
##                               Dim.1         Dim.2        Dim.3         Dim.4
## Red_berry.1331.7       -0.2420588137 -0.6371336481  0.211907444  0.2053230519
## Dark_berry.1331.7       0.3409561657  0.6686617185 -0.406031062 -0.3427279964
## Jam.1331.7             -0.6815368032 -0.0463691326 -0.241509769  0.6173101029
## Dried_fruit.1331.7      0.3612632061  0.0100625165 -0.170846662 -0.6949093381
## Artificial_frui.1331.7 -0.5501233197 -0.1105770048  0.249949163  0.7437361957
## Chocolate.1331.7        0.0582188355 -0.3315085859 -0.598948146  0.5240282618
## Vanilla.1331.7         -0.7391098115 -0.1480763672  0.300182532  0.3593531824
## Oak.1331.7              0.2726983399 -0.5845646815  0.006397835 -0.5679731406
## Burned.1331.7           0.9347565617  0.2337330908 -0.031914519  0.1144208471
....
```

```
descriptive_mfa$global.pca$var$cor %>%
  as_tibble(rownames = "descriptor") %>%
  separate(descriptor,
           into = c("descriptor", "panelist", "rep"),
           sep = "\\.") %>%
  ggplot(aes(x = Dim.1, y = Dim.2, color = descriptor)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_segment(aes(xend = 0, yend = 0),
               arrow = arrow(length = unit(0.02, units = "in"), ends = "first"),
               linewidth = 1/3, alpha = 1/2) +
  ggforce::geom_circle(aes(x0 = 0, y0 = 0, r = 1),
                       linewidth = 1/10, linetype = 3,
                       color = "grey") +
  coord_equal() +
  scale_color_paletteer_d("ggthemes::Classic_20") +
  scale_x_continuous(breaks = c(-1, 0, 1)) +
  scale_y_continuous(breaks = c(-1, 0, 1)) +
```

```
  theme_classic() +
  theme(legend.position = "none",
        strip.text = element_text(size = 6)) +
  facet_wrap(~descriptor) +
  labs(subtitle = "MFA correlation plot for variables with\noverall consensus space",
       x = paste0("Dimension 1, ", round(descriptive_mfa$eig[1, 2], 1), "% of variance",
       y = paste0("Dimension 2, ", round(descriptive_mfa$eig[2, 2], 1), "% of variance",
```

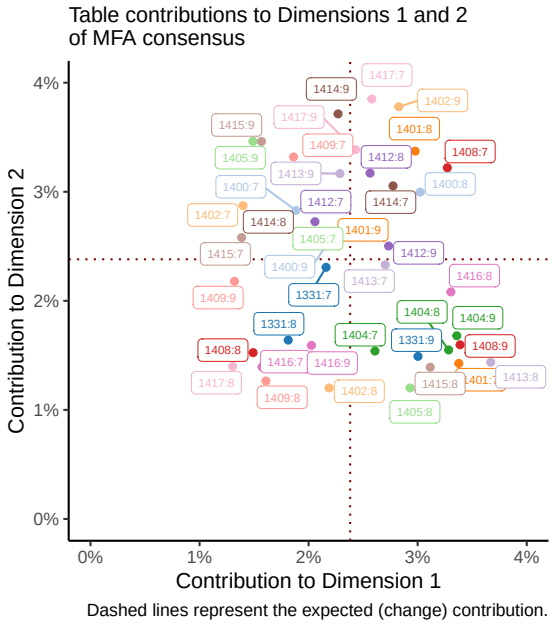


From this plot, we can tell that there is reasonable consensus among judges for some attributes–like `Leather`, `Jam`, `Chocolate`, and `Sour`, and very little for some other attributes, like `Red_berry`, `Oak`, `Alcohol`, and `Dried_fruit`. For some of these, it is quite possible that these two dimensions do not correlate well with the attributes at all–notably `Alcohol` and `Astringent` and possibly `Chocolate`. Again, this gives us the ability, over PCA, to look at some of the ways in which the group coheres around or departs from the consensus.

We could (and would want to!) investigate the contributions of the observations, variables, and sets of variables ("tables") to the consensus solution to understand what is most important. Here, to save space, we will look primarily at the *tables*: which `NR:NJ` combination is most important for the space? We will visualize this as a scatterplot, with contributions to the 1st and 2nd dimensions on the axes.

```
# Some nicer labels
nice_names <-
  crossing(subject = descriptive_data$NJ %>% unique,
```

```
          rep = descriptive_data$NR %>% unique) %>%
  mutate(name = str_c(subject, ":", rep))

descriptive_mfa$group$contrib %>%
  as_tibble(rownames = "group") %>%
  # Once again we're matching by position, which is risky (but fast)
  bind_cols(nice_names) %>%
  ggplot(aes(x = Dim.1, y = Dim.2, color = subject)) +
  geom_vline(xintercept = 1 / 42 * 100, color = "darkred", linetype = 3) +
  geom_hline(yintercept = 1 / 42 * 100, color = "darkred", linetype = 3) +
  geom_point() +
  ggrepel::geom_label_repel(aes(label = name), size = 2, box.padding = 0.1) +
  theme_classic() +
  coord_equal() +
  scale_x_continuous("Contribution to Dimension 1",
                     labels = ~str_c(., "%"),
                     limits = c(0, 4)) +
  scale_y_continuous("Contribution to Dimension 2",
                     labels = ~str_c(., "%"),
                     limits = c(0, 4)) +
  labs(subtitle = "Table contributions to Dimensions 1 and 2\nof MFA consensus",
       caption = "Dashed lines represent the expected (change) contribution.") +
  scale_color_paletteer_d("ggthemes::Classic_20") +
  theme(legend.position = "none")
```



We could do the same thing for per-table variables and observations. The dotted red lines represent the average (expected) contribution, and so points in the lower left quadrant are contributing less than expected to both dimensions. I colored this by subject so that I could see if there were any patterns for subjects who did not ever contribute to the solution, but I don't see anything that obvious. Closer inspection might show some patterns.

From Abdi et al. [2013, 11-12] we are told that computing bootstrap samples from MFA is as simple as averaging the partial factor scores for a bootstrapped index of tables. We could use bootstrapping in this case to investigate the contribution of our samples to the dimensions. However, that seems like it may be drawing me too far outside the scope of this project! A variation on the bootstrap we built for our PCA would work in this case; I leave it as a (rather involved) exercise for the reader.

With this slightly simpler example in mind, let's turn to an MFA where we have qualitatively *different* variables measured on the same observations.

12.2 MFA with different measurements

```
preference_mfa <-
  # Get the means of the product attributes from the DA
  descriptive_data %>%
  group_by(ProductName) %>%
  summarize(across(where(is.numeric), mean)) %>%

  # Merge these with the (wide: product x consumer) consumer data

  left_join(
    consumer_data %>%
      pivot_longer(-Judge, names_to = "wine") %>%
      pivot_wider(names_from = Judge, values_from = value),
    by = c("ProductName" = "wine")
  ) %>%

  # Cast to row-names for compatibility with `MFA()`

  column_to_rownames("ProductName") %>%

  # Finally, run MFA.  Remember that columns 1-20 are DA, 21-126 are consumer

  MFA(group = c(20, 106), graph = FALSE)
```

```
# What did we get?
preference_mfa
```

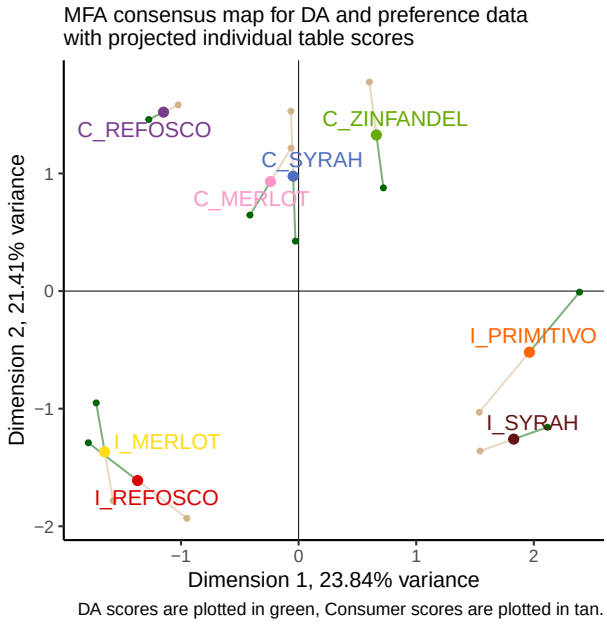
```
## **Results of the Multiple Factor Analysis (MFA)**
## The analysis was performed on 8 individuals, described by 126 variables
## *Results are available in the following objects :
##
##   name                 description
## 1 "$eig"               "eigenvalues"
## 2 "$separate.analyses" "separate analyses for each group of variables"
## 3 "$group"             "results for all the groups"
## 4 "$partial.axes"      "results for the partial axes"
## 5 "$inertia.ratio"     "inertia ratio"
## 6 "$ind"               "results for the individuals"
## 7 "$quanti.var"        "results for the quantitative variables"
## 8 "$summary.quanti"    "summary for the quantitative variables"
## 9 "$global.pca"        "results for the global PCA"
```

We can dive into these results in the same way we did with the DA data, but we are presented with a situation that is simpler in some ways and more complex in others. First, we only have two "groups" or "tables": the DA (means) data and the consumer data. We don't have to, for example, investigate contributions from 42 different tables!

However, we also don't have the same variables measured across both tables, so we need to be a little careful with how we interpret the results. Let's start by looking at the consensus map for the wines, enriched with projections for both how the DA panel and the consumers saw the wines:

```
p_preference_scores <-

  # First we need to get our consensus and partial factor scores into a single
  # tibble.  Specifically, it will be easiest if we have the scores (coordinates)
  # for the partial axes in their own columns.

  left_join(

    # These are our consensus scores

    preference_mfa$ind$coord %>%
      as_tibble(rownames = "product"),

    # And here are our partial scores, manipulated to have per-table dimensional
    # coordinates.
```

```
    preference_mfa$ind$coord.partiel %>%
      as_tibble(rownames = "product") %>%
      separate(product, into = c("product", "group"), sep = "\\.") %>%
      pivot_longer(-c(product, group)) %>%
      pivot_wider(names_from = c(name, group), values_from = value)
  ) %>%

  # And now we plot!

  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point(aes(x = Dim.1_Gr1, y = Dim.2_Gr1),
             color = "darkgreen", size = 1) +
  geom_point(aes(x = Dim.1_Gr2, y = Dim.2_Gr2),
             color = "tan", size = 1) +
  geom_segment(aes(xend = Dim.1_Gr1, yend = Dim.2_Gr1),
               color = "darkgreen", alpha = 1/2) +
  geom_segment(aes(xend = Dim.1_Gr2, yend = Dim.2_Gr2),
               color = "tan", alpha = 1/2) +
  geom_point(aes(color = product),
             size = 2, show.legend = FALSE) +
  ggrepel::geom_text_repel(aes(color = product, label = product), show.legend = FALSE)
  coord_equal() +
  theme_classic() +
  theme(legend.position = "bottom") +
  labs(x = paste0("Dimension 1, ", round(preference_mfa$eig[1, 2], 2), "% variance"),
       y = paste0("Dimension 2, ", round(preference_mfa$eig[2, 2], 2), "% variance"),
       subtitle = "MFA consensus map for DA and preference data\nwith projected indivi
       caption = "DA scores are plotted in green, Consumer scores are plotted in tan."
  scale_color_paletteer_d("RSkittleBrewer::smarties")

p_preference_scores
```



Once again, we find ourselves in slight disagreement with HGH's results for the original **R Opus**: the overall configuration is similar but not identical. As with PLS-R, I wonder if this might have to do with the dropped subject #1? I re-ran the MFA and the plot with that subject dropped (the reader can do this as an exercise: just use `select(-1)` in the appropriate part of the workflow above, and make sure to adjust the group indices for the `MFA()` call), and I got results that were almost identical to HGH, but with slightly different projected points. I am not sure if this is an adjustment to MFA plotting (the `FactoMineR` package has changes signficantly since 2015) or if there is something else I am missing.

One thing that is worth noting is that the disagreement between our two score plots is reasonably small. We can take a look at the RV coefficient between the two projections to get a better picture of this:

```
preference_mfa$group$RV %>%
  round(3)
```

```
##       Gr1   Gr2   MFA
## Gr1 1.000 0.804 0.932
## Gr2 0.804 1.000 0.965
## MFA 0.932 0.965 1.000
```

We can see that with an $RV \approx 0.8$ between the two data sets, we are quite close in agreement. They also both strongly agree with the consensus score plot ($RV > 0.92$ for both). Although the RV is biased upwards for large

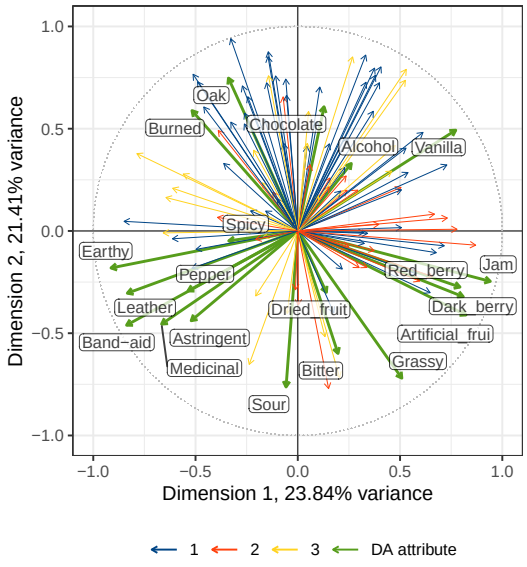
matrices [see Josse and Holmes, 2016], this is still confirmatory information that our perspective is accurate: the two very different data tables are showing us something similar for the same products.

Given this high level of agreement, it will be informative to inspect the loadings from both plots together: while MFA is strictly a descriptive method, we can use these loadings to give us some hypotheses about the relationship between descriptive attributes and "vectors of liking" from the consumers.

```
# For whatever reason, we tend to visualize correlations in MFA. These are the
# same as loadings up to a scale factor, so our interpretations will be the
# same.  We can access them in `$quanti.var$cor`.

preference_mfa$quanti.var$cor %>%
  as_tibble(rownames = "name") %>%

# This one is easy, we just have to keep track of which meausres come from which
# data set--`MFA()` does this sequentially, by the index we set through `group =`

  mutate(type = c(rep("descriptor", 20), rep("liking", 106))) %>%

# But it might be nice if we grabbed the clusters we found previously from our
# internal preference map...

  left_join(
    consumer_data %>%
      column_to_rownames("Judge") %>%
      dist() %>%
      hclust(method = "ward.D2") %>%
      cutree(k = 3) %>%
      as_tibble(rownames = "name") %>%
      transmute(name = as.character(name),
                cluster = as.character(value))
  ) %>%
  mutate(cluster = replace_na(cluster, "DA attribute")) %>%

# And now we plot!

  ggplot(aes(x = Dim.1, y = Dim.2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  ggforce::geom_circle(aes(x0 = 0, y0 = 0, r = 1),
                       linetype = 3, color = "grey", size = 1/4) +
  geom_segment(aes(xend = 0, yend = 0, color = cluster, linewidth = type),
               arrow = arrow(length = unit(0.05, units = "in"), ends = "first")) +
  ggrepel::geom_label_repel(data = . %>% filter(type == "descriptor"),
```

```
                          aes(label = name),
                          size = 3, alpha = 3/4, label.padding = 0.1) +
  scale_linewidth_manual(guide = "none", values = c(3/4, 1/4)) +
  coord_equal() +
  theme_bw() +
  scale_color_paletteer_d("ggthemes::calc") +
  theme(legend.position = "bottom") +
  labs(x = paste0("Dimension 1, ", round(preference_mfa$eig[1, 2], 2), "% variance"),
       y = paste0("Dimension 2, ", round(preference_mfa$eig[2, 2], 2), "% variance"),
       color = NULL)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

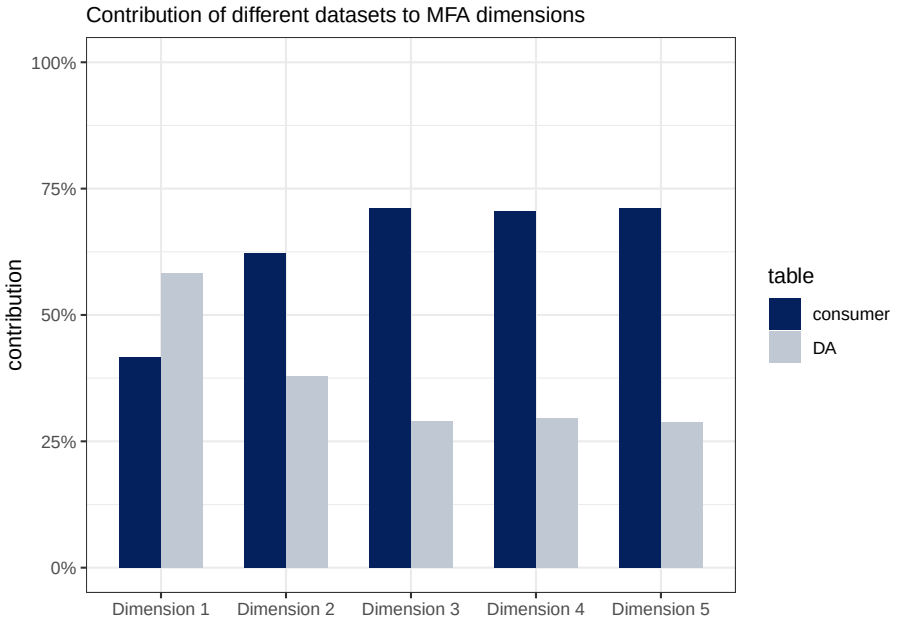


By some careful inspection, we can see that two of the consumer clusters are reasonably well-aligned with some directions in the DA correlation plot: cluster 1 (blue) is aligned with attributes stemming from barrel aging: `Burned`, `Oak`, `Chocolate`, and somewhat with `Vanilla`. Cluster 2 (orange) is aligned more with fruit-related attributes: `Jam`, `Red_berry`, `Dark_berry`, and `Artificial_frui` (sic). Cluster 3, however, does not have any clear alignment with the DA scores. Compare these results to the preference-mapping results from the previous section!

I am not sure that there is any real benefit to pulling the other visualizations that HGH rendered in the original **R Opus**: I think that they were mainly there to demonstrate the functionality of the `MFA()`, which I feel like we have done in sufficient detail, and since she didn't add any commentary I am assuming, like me, that she did not find them to be actually useful for interpreting the data. Given the additional work we put in above, I will not dig into them.

Instead, we will wrap up with a contribution plot to understand which dimensions are being driven by DA and which by consumer data:

```
# Grab the contribution data
preference_mfa$group$contrib %>%
  as_tibble() %>%
  mutate(table = c("DA", "consumer")) %>% # positional matching :(
  pivot_longer(-table) %>%

# And plot!

  ggplot(aes(x = name, y = value)) +
  geom_col(aes(fill = table), color = NA, position = "dodge", width = 2/3) +
  scale_fill_paletteer_d("IslamicArt::ottoman") +
  theme_bw() +
  scale_y_continuous(labels = ~str_c(., "%"), limits = c(0, 100)) +
  scale_x_discrete(labels = str_c("Dimension ", 1:5)) +
  labs(x = NULL,
       y = "contribution",
       color = "dataset",
       subtitle = "Contribution of different datasets to MFA dimensions")
```



We can see from this that the first dimension has more contribution by the DA data, and the second dimension is dominated by the consumer data; all dimensions must of course add up to 100%.

12.3 Wrapping up

We have gone into a fair amount of detail into MFA in this chapter. I think that MFA–which is really a step-wise PCA–is justifiably popular with sensory scientists, and deserves to be used more. However, I *also* feel that the number of outputs from an MFA can be overwhelming, and lead to both over-interpretation by the analyst and a sort of numbness on the part of the audience. Like a lot of these complex methods (and MFA is not the most complicated at all!), it is the analyst's job to justify the analyses: *why* are we looking at particular outputs and visualizations? How do they help us either answer research questions or develop new ones? If we don't do our job, I think we end up providing a sort of "white noise" of analyses, in which it is impossible to determine what is meaningful and what is not.

12.4 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] patchwork_1.1.2 paletteer_1.5.0 FactoMineR_2.8  here_1.0.1
##  [5] lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2
##  [9] purrr_1.0.1     readr_2.1.4     tidyr_1.3.0     tibble_3.2.1
## [13] ggplot2_3.4.3   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4         xfun_0.39            htmlwidgets_1.6.2
##  [4] ggrepel_0.9.3        lattice_0.21-8       tzdb_0.4.0
##  [7] vctrs_0.6.3          tools_4.3.1          generics_0.1.3
## [10] parallel_4.3.1       fansi_1.0.4          highr_0.10
## [13] cluster_2.1.4        pkgconfig_2.0.3      scatterplot3d_0.3-44
## [16] lifecycle_1.0.3      farver_2.1.1         compiler_4.3.1
## [19] munsell_0.5.0        ggforce_0.4.1        leaps_3.1
## [22] htmltools_0.5.6      yaml_2.3.7           pillar_1.9.0
## [25] crayon_1.5.2         MASS_7.3-60          flashClust_1.01-2
## [28] DT_0.28              tidyselect_1.2.0     digest_0.6.33
## [31] mvtnorm_1.2-2        stringi_1.7.12       rematch2_2.1.2
## [34] bookdown_0.37        labeling_0.4.3       polyclip_1.10-4
## [37] rprojroot_2.0.3      fastmap_1.1.1        grid_4.3.1
## [40] colorspace_2.1-0     cli_3.6.1            magrittr_2.0.3
## [43] utf8_1.2.3           withr_2.5.0          scales_1.2.1
## [46] bit64_4.0.5          estimability_1.4.1   timechange_0.2.0
## [49] rmarkdown_2.23       emmeans_1.8.7        bit_4.0.5
## [52] hms_1.1.3            coda_0.19-4          evaluate_0.21
## [55] knitr_1.43           rlang_1.1.1          Rcpp_1.0.11
```

```
## [58] xtable_1.8-4         glue_1.6.2           tweenr_2.0.2
## [61] rstudioapi_0.15.0    vroom_1.6.3          R6_2.5.1
## [64] prismatic_1.1.1      multcompView_0.1-9
```

Chapter 13

Generalized Procrustes Analysis

Generalized Procrustes Analysis (GPA) is the second-to-last topic HGH tackles in the original **R Opus**. As a spoiler, this will be our last (substantive) topic: the coverage that HGH gave to Conjoint Analysis was cursory, and I believe that analysis of these data has moved very far forward since the original **R Opus**. I don't ever use Conjoint Analysis (I leave that to my consumer-economist friends), and so I will not be including this chapter.

GPA is a "multi-table" method like MFA or DISTATIS. GPA involves a set of (I believe) linear transformations of each data table through translation, rotation, and isotropic scaling, with the goal to minimize the difference between the original tables and the consensus table. GPA was an early and popular method of aligning data from multiple subjects on the same sets of samples when those subjects are not trained. Based on my understanding, it has fallen somewhat out of favor as MFA anecdotally and actually can provide very similar results [Tomic et al., 2015]. This can be seen in the less-updated GPA function in the `FactoMineR` function, and by the fact that HGH does not bother to give it much time or interpretation. GPA tends to be applied to (Ultra) Flash Profiling and Napping/Projective Mapping results, and it seems to me that those methods are less popular than they once were.

However, the same authors point out several advantages of GPA, and so it is worth digging into it a little bit! We're going to be using the same two-table dataset: DA and consumer data.

```
library(tidyverse)
library(here)
library(FactoMineR)
library(paletteer)
```

```
# This is all just standard set up

descriptive_data <-
  read_csv(here("data/torriDAFinal.csv")) %>%
  mutate(across(1:3, ~as.factor(.)))

consumer_data <- read_csv(here("data/torriconsFinal.csv"))

consumer_demo <-
  consumer_data %>%
  select(Judge:Age) %>%
  mutate(across(Judge:Age, ~as.factor(.)))

consumer_data <-
  consumer_data %>%
  select(-(`Wine Frequency`:Age)) %>%
  mutate(Judge = as.factor(Judge))
```

We then follow steps that will seem very familiar from MFA:

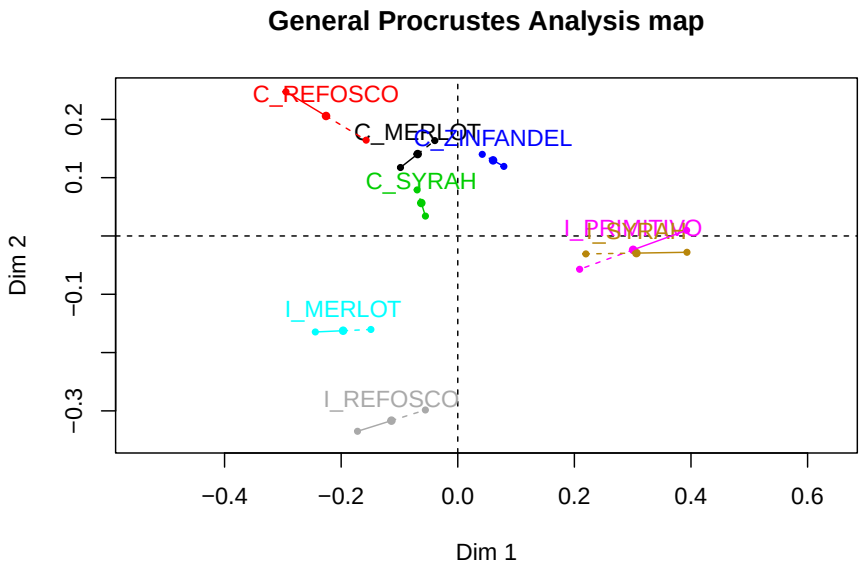
```
preference_gpa <-
  descriptive_data %>%
  group_by(ProductName) %>%
  summarize(across(where(is.numeric), mean)) %>%
  left_join(
    consumer_data %>%
      pivot_longer(-Judge, names_to = "wine") %>%
      pivot_wider(names_from = Judge, values_from = value),
    by = c("ProductName" = "wine")
  ) %>%
  column_to_rownames("ProductName") %>%
  GPA(group = c(20, 106), graph = FALSE)
```

I initially was confused about how it is possible to run GPA with two tables with such different number of measurements: in methods like STATIS we get around the difficulty by in fact analyzing the $\mathbf{X_i X_i^T}$, which will have the same dimensionality because each $\mathbf{X_i}$ has the same number of rows, even if the number of colums is different. According to Gower [1975], GPA solves this problem by instead just adding null columns to the column-wise smaller $\mathbf{X_i}$.

I believe that the GPA alignment is followed by a PCA on the consensus positions, since our points are still in 8-dimensional space ($min(n, k_1, k_2) = 8$ for our data with $n = 8$ wines and $k_1 = 20$ wines and $k_2 = 106$ consumers), but this isn't explicitly clarified by the `?GPA` documentation–let's assume so!

Typically we haven't even wanted to *look* at base-R plots, but it is worthwhile to note that we can tell `GPA()` is less loved, because the function still outputs these, instead of the `factoextra`-flavored plots that other `FactoMineR` functions like `MFA()` and `PCA()` give us:

```
plot(preference_gpa)
```

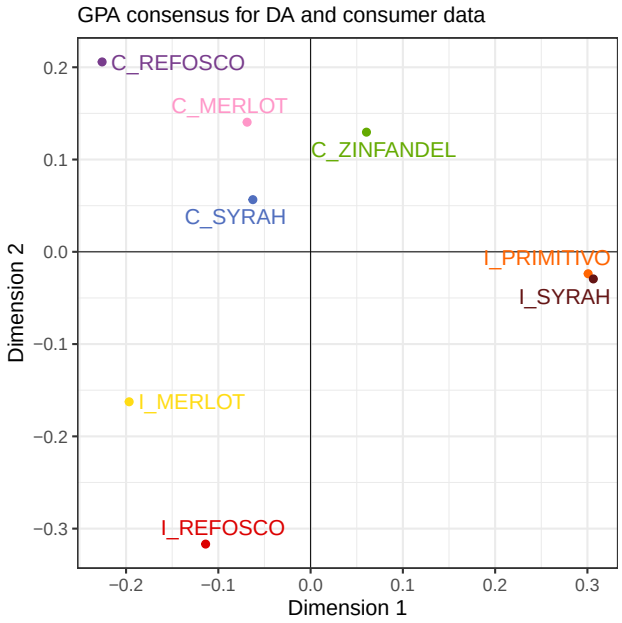


We know how to do better, though!

```
# The `$consensus` table gives us the center points
p_gpa_base <-
  preference_gpa$consensus %>%
  as_tibble(rownames = "wine") %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  geom_point(aes(color = wine)) +
  ggrepel::geom_text_repel(aes(label = wine, color = wine)) +
  coord_equal() +
  scale_color_paletteer_d("RSkittleBrewer::smarties") +
  theme_bw() +
  theme(legend.position = "none") +
  labs(subtitle = "GPA consensus for DA and consumer data",
       x = "Dimension 1",
       y = "Dimension 2")
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
## `.name_repair` is omitted as of tibble 2.0.0.
## i Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

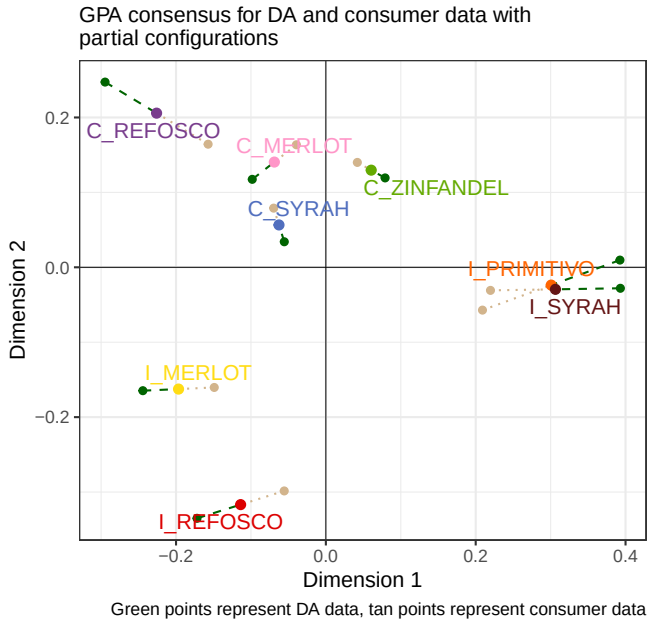
```
p_gpa_base
```



We can then add layers showing the projected position of our two original tables:

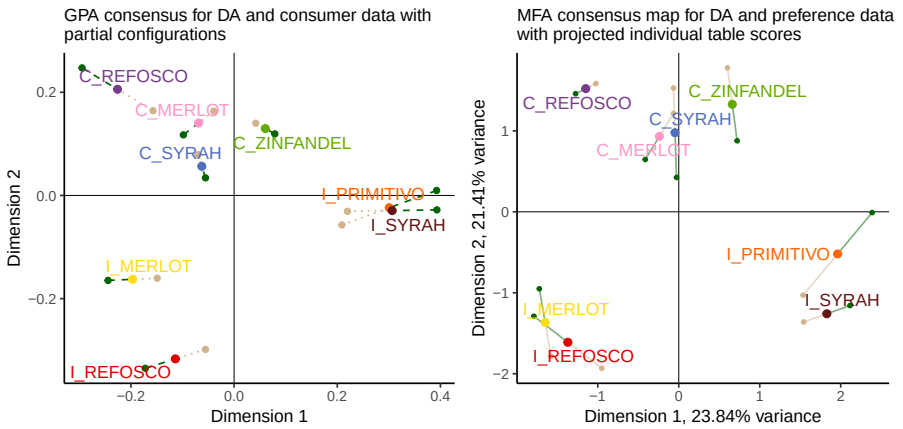
```
p_gpa_partials <-
  tibble(type = c("consensus", "DA", "consumer"),
       coord = list(preference_gpa$consensus,
                    preference_gpa$Xfin[, , 1],
                    preference_gpa$Xfin[, , 2])) %>%
  # Get everything into a tibble
  mutate(coord = map(coord, as_tibble, rownames = "wine")) %>%
  # Give every dimension the same name, since `GPA()` doesn't do this right
  mutate(coord = map(coord, set_names, c("wine", paste0("dim_", 1:7)))) %>%
  unnest(everything()) %>%
  # We're just going to do the first 2 dimensions here
  select(type:dim_2) %>%
  pivot_longer(dim_1:dim_2) %>%
  unite(type, name, col = "dim") %>%
```

```
  pivot_wider(names_from = dim, values_from = value) %>%
  # Plot!
  ggplot(aes(x = consensus_dim_1, y = consensus_dim_2)) +
  geom_vline(xintercept = 0, linewidth = 1/10) +
  geom_hline(yintercept = 0, linewidth = 1/10) +
  # The projected DA data
  geom_point(aes(x = DA_dim_1, y = DA_dim_2),
             inherit.aes = FALSE, color = "darkgreen") +
  geom_segment(aes(x = DA_dim_1, y = DA_dim_2, xend = consensus_dim_1, yend = consensus_dim_2),
               inherit.aes = FALSE, color = "darkgreen", linetype = 2) +
  # The projected consumer data
  geom_point(aes(x = consumer_dim_1, y = consumer_dim_2),
             inherit.aes = FALSE, color = "tan") +
  geom_segment(aes(x = consumer_dim_1, y = consumer_dim_2, xend = consensus_dim_1, yend = consens
               inherit.aes = FALSE, color = "tan", linetype = 3) +
  geom_point(aes(color = wine), size = 2) +
  ggrepel::geom_text_repel(aes(label = wine, color = wine)) +
  coord_equal() +
  scale_color_paletteer_d("RSkittleBrewer::smarties") +
  theme_bw() +
  theme(legend.position = "none") +
  labs(subtitle = "GPA consensus for DA and consumer data with\npartial configurations",
       x = "Dimension 1",
       y = "Dimension 2",
       caption = "Green points represent DA data, tan points represent consumer data")

p_gpa_partials
```



We can compare this to our MFA map for reference (and to see the similarity between the methods.)

```
library(patchwork)

(p_gpa_partials + p_mfa) &
  labs(caption = NULL) &
  theme_classic() &
  theme(legend.position = "none")
```



We can see that we get close but not exactly similar configurations!

Returning to our GPA solution, we can inspect how close our configurations are, using both RV and Procrustean similarity coefficients:

```
preference_gpa$RV
```

```
##           group.1   group.2
## group.1 1.0000000 0.6674211
## group.2 0.6674211 1.0000000
```

```
preference_gpa$simi
```

```
##           group.1   group.2
## group.1 1.0000000 0.8712266
## group.2 0.8712266 1.0000000
```

We've already encountered RV, but to be honest I am not entirely sure what the definition of Procrustes similarity is. A quick search doesn't really get me much more detail, nor does reviewing Gower [1975]. However, from Qannari et al. [1998] I was able to find that Procrustes similarity is defined as

$$s(X,Y) = \frac{trace(\mathbf{X^T Y H})}{\sqrt{trace(\mathbf{X^T X})}\sqrt{trace(\mathbf{Y^T Y})}}$$

This is clearly very close to the definition of the RV coefficient, differing mainly in that the numerator is the Procrustes rotation that adjusts $\mathbf{Y}$ to $\mathbf{X}$, whereas

for RV the numerator instead represents the two double-centered cross-product matrices. I am not sure what the Procrustes similarity emphasizes over the RV, but I believe it is more evenly influenced by dimensions in the data beyond the first principal component based on discussion in Tomic et al. [2015]]. So which should we pay attention to? I don't have a good answer to that–it probably depends on the situation.

13.1 Packages used in this chapter

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] patchwork_1.1.2 paletteer_1.5.0 FactoMineR_2.8  here_1.0.1
##  [5] lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2
##  [9] purrr_1.0.1     readr_2.1.4     tidyr_1.3.0     tibble_3.2.1
## [13] ggplot2_3.4.3   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4         xfun_0.39            htmlwidgets_1.6.2
##  [4] ggrepel_0.9.3        lattice_0.21-8       tzdb_0.4.0
##  [7] vctrs_0.6.3          tools_4.3.1          generics_0.1.3
## [10] parallel_4.3.1       fansi_1.0.4          highr_0.10
## [13] cluster_2.1.4        pkgconfig_2.0.3      scatterplot3d_0.3-44
## [16] lifecycle_1.0.3      farver_2.1.1         compiler_4.3.1
## [19] munsell_0.5.0        leaps_3.1            htmltools_0.5.6
## [22] yaml_2.3.7           pillar_1.9.0         crayon_1.5.2
```

```
## [25] MASS_7.3-60          flashClust_1.01-2    DT_0.28
## [28] tidyselect_1.2.0     digest_0.6.33        mvtnorm_1.2-2
## [31] stringi_1.7.12       rematch2_2.1.2       bookdown_0.37
## [34] labeling_0.4.3       rprojroot_2.0.3      fastmap_1.1.1
## [37] grid_4.3.1           colorspace_2.1-0     cli_3.6.1
## [40] magrittr_2.0.3       utf8_1.2.3           withr_2.5.0
## [43] scales_1.2.1         bit64_4.0.5          estimability_1.4.1
## [46] timechange_0.2.0     rmarkdown_2.23       emmeans_1.8.7
## [49] bit_4.0.5            hms_1.1.3            coda_0.19-4
## [52] evaluate_0.21        knitr_1.43           rlang_1.1.1
## [55] Rcpp_1.0.11          xtable_1.8-4         glue_1.6.2
## [58] rstudioapi_0.15.0    vroom_1.6.3          R6_2.5.1
## [61] prismatic_1.1.1      multcompView_0.1-9
```

Chapter 14

Wrapping up

The analyses commonly used in sensory evaluation (and in other fields that deal with multivariate data) continue to multiply. The **R Opus**–v2 and the original–cover only a fraction of these analyses, although I am glad to say they are probably the most common. Almost all of the methods we've covered here center around eigendecomposition/singular value decomposition. We haven't touched on methods of explanation and prediction like logistic regression, random-forest modeling, or (the increasingly ubiquitous) neural network.

I wanted to talk about all of these! If you look at my manuscript output, I am an inveterate, methodological dilettante, and learning about how these methods operate, how to run them on my own data, and how to present them in replicable form like the scripts in the **R Opus** is a big motivating factor of my work. That's why Bayesian methods intrude into the M/ANOVA chapters, and why I had to delete a rambling and poorly thought out chapter adding a look at Random Forest models into this work.

Everything's got to stop somewhere. If you feel motivated by what I've shown here, or excited to learn how to (for example) fit a Random Forest to predict the consumer liking from the DA data, go do it! There's no better way of learning data-analysis techniques than getting in, making mistakes, and figuring out how to fix them. I hope it'll be fun.

Thanks for making it to the end!

Bibliography

H Abdi and M Béra. Correspondence analysis. In R. Alhajj and J. Rokne, editors, *Encyclopedia of Social Networks and Mining*, pages 275–284. Springer Verlag, New York, 2015.

H. Abdi, D. Valentin, S. Chollet, and C. Chrea. Analyzing assessors and products in sorting tasks: DISTATIS, theory and applications. *Food Quality and Preference*, 18(4):627–640, June 2007. ISSN 09503293. doi: 10.1016/j.foodqual.2006.09.003.

H. Abdi, L.J. Williams, D. Valentin, and M. Bennani-Dosse. STATIS and DISTATIS: Optimum multitable principal component analysis and three way metric multidimensional scaling: STATIS and DISTATIS. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(2):124–167, March 2012. ISSN 19395108. doi: 10.1002/wics.198.

Hervé Abdi and Lynne J. Williams. Partial Least Squares Methods: Partial Least Squares Correlation and Partial Least Square Regression. In Brad Reisfeld and Arthur N. Mayeno, editors, *Computational Toxicology*, volume 930, pages 549–579. Humana Press, Totowa, NJ, 2013. ISBN 978-1-62703-058-8 978-1-62703-059-5. doi: 10.1007/978-1-62703-059-5_23.

Hervé Abdi, Lynne J. Williams, and Domininique Valentin. Multiple factor analysis: Principal component analysis for multitable and multiblock data sets: Multiple factor analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(2):149–179, March 2013. ISSN 19395108. doi: 10.1002/wics.1246.

Per Bruun Brockhoff, Pascal Schlich, and Ib Skovgaard. Taking individual scaling differences into account by analyzing profile data with the Mixed Assessor Model. *Food Quality and Preference*, 39:156–166, January 2015. ISSN 0950-3293. doi: 10.1016/j.foodqual.2014.07.005.

P. Courcoux, P. Faye, and E. M. Qannari. Free sorting as a sensory profiling technique for product development. In J. Delarue and J. Benjamin Lawlor, editors, *Rapid Sensory Profiling Techniques*. Woodhead, 2nd edition, 2023.

Burkhard Dettmar, Caroline Peltier, and Pascal Schlich. Beyond principal component analysis (PCA) of product means: Toward a psychometric view on

sensory profiling data. *Journal of Sensory Studies*, 35(2):e12555, April 2020. ISSN 0887-8250. doi: 10.1111/joss.12555.

John C. Gower. Generalized procrustes analysis. *Psychometrika*, 40:33–51, 1975.

Michael Greenacre. *Correspondence analysis in practice*. CRC press, 2017.

Julie Josse and Susan Holmes. Measuring multivariate association and beyond. *Statistics Surveys*, 10(0):132–167, 2016. ISSN 1935-7516. doi: 10.1214/16-SS116.

John Kruschke. Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. 2014.

A. Solomon Kurz. *Doing Bayesian data analysis in brms and the tidyverse*. Version 1.1.0 edition, 1 2023. URL https://bookdown.org/content/3686/.

C. Peltier, M. Visalli, and P. Schlich. Canonical Variate Analysis of Sensory Profiling Data. *Journal of Sensory Studies*, 30(2015):316–328, 2015.

El Mostafa Qannari, Halliday J.H MacFie, and Philippe Courcoux. Performance indices and isotropic scaling factors in sensory profiling. *Food Quality and Preference*, 10(1):17–21, October 1998. ISSN 0950-3293. doi: 10.1016/S0950-3293(98)00033-0.

Alvin C. Rencher. *Methods of Multivariate Analysis*. Wiley Series in Probability and Mathematical Statistics. J. Wiley, New York, 2nd ed edition, 2002. ISBN 978-0-471-41889-4.

O. Tomic, I. Berget, and T. Næs. A comparison of generalised procrustes analysis and multiple factor analysis for projective mapping data. *Food Quality and Preference*, 43:34–46, July 2015. ISSN 0950-3293. doi: 10.1016/j.foodqual.2015.02.004.

Renoo Yenket, Edgar Chambers IV, and Koushik Adhikari. A COMPARISON OF SEVEN PREFERENCE MAPPING TECHNIQUES USING FOUR SOFTWARE PROGRAMS: COMPARISON OF MAPPING TECHNIQUES. *Journal of Sensory Studies*, 26(2):135–150, April 2011. ISSN 08878250. doi: 10.1111/j.1745-459X.2011.00330.x.