Creating Contingency Tables

Now that we know how to get our raw data into R, we are ready to do the fun stuff - investigating our data!

We discussed the main steps of an EDA and covered the most common data validation and basic manipulations for the data. The next few sets of notes dive into **how to find summarize our data**. Recall, how we summarize our data depends on the type of data we have!

- Categorical (Qualitative) variable - entries are a label or attribute
- Numeric (Quantitative) variable - entries are a numerical value where math can be performed

In either situation, we want to describe each variable's distribution, perhaps comparing across different subgroups!

Let's start with summaries of strictly categorical data (or numeric variables with only a few values).

- Categorical data is usually stored as a `character` or `factor` type

Categorical Data Summaries

To summarize categorical variables numerically, we use contingency tables.

To do so visually, we use bar plots.

First, let's read in the appendicitis data from the previous lecture.

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.0
✔ ggplot2   3.4.4     ✔ tibble    3.2.1
✔ lubridate 1.9.2     ✔ tidyr     1.3.0
✔ purrr     1.0.1
── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```
library(readxl)
app_data <- read_excel("app_data.xlsx", sheet = 1)
app_data <- app_data |>
  mutate(BMI = as.numeric(BMI),
         US_Number = as.character(US_Number),
```

```
         SexF = factor(Sex, levels = c("female", "male"), labels = c("Female
         DiagnosisF = as.factor(Diagnosis),
         SeverityF = as.factor(Severity))
app_data
```

```
# A tibble: 782 × 61
     Age   BMI Sex    Height Weight Length_of_Stay Management   Severity
   <dbl> <dbl> <chr>   <dbl>  <dbl>          <dbl> <chr>        <chr>
 1 12.7   16.9 female    148   37                3 conservative
uncomplicated
 2 14.1   31.9 male      147   69.5              2 conservative
uncomplicated
 3 14.1   23.3 female    163   62                4 conservative
uncomplicated
 4 16.4   20.6 female    165   56                3 conservative
uncomplicated
 5 11.1   16.9 female    163   45                3 conservative
uncomplicated
 6 11.0   30.7 male      121   45                3 conservative
uncomplicated
 7  8.98  19.4 female    140   38.5              3 conservative
uncomplicated
 8  7.06  NA   female     NA   21.5              2 conservative
uncomplicated
 9  7.9   15.7 male      131   26.7              3 conservative
uncomplicated
10 14.3   14.9 male      174   45.5              3 conservative
uncomplicated
# ℹ 772 more rows
# ℹ 53 more variables: Diagnosis_Presumptive <chr>, Diagnosis <chr>,
#   Alvarado_Score <dbl>, Paedriatic_Appendicitis_Score <dbl>,
#   Appendix_on_US <chr>, Appendix_Diameter <dbl>, Migratory_Pain <chr>,
#   Lower_Right_Abd_Pain <chr>, Contralateral_Rebound_Tenderness <chr>,
#   Coughing_Pain <chr>, Nausea <chr>, Loss_of_Appetite <chr>,
#   Body_Temperature <dbl>, WBC_Count <dbl>, Neutrophil_Percentage <dbl>, …
```

Let's go!

Contingency Tables

We can use `BaseR` or the `tidyverse`.

Via `BaseR`

Honestly, the easiest way to make contingency tables is through BaseR's `table()` function.

From the help

```
table(...,
      exclude = if (useNA == "no") c(NA, NaN),
      useNA = c("no", "ifany", "always"),
      dnn = list.names(...), deparse.level = 1)
```

where `...` is

> one or more objects which can be interpreted as factors (including numbers or character strings), or a list (such as a data frame) whose components can be so interpreted.

Ok, so we can just pass it the vectors we want or we could pass it a data frame (which remember, is just a list of equal length vectors!).

Let's create some contingency tables for the `SexF`, `DiagnosisF`, and `SeverityF` variables.

```
table(app_data$SexF)
```

```
Female   Male 
   377    403 
```

We can include `NA` if we want to via the `useNA` argument:

```
table(app_data$SexF, useNA = "always")
```

```
Female   Male   <NA> 
   377    403      2 
```

We can create a two-way table (two-way for two variables) by adding the second variable in:

```
table(app_data$SexF, app_data$DiagnosisF)
```

```
        
         appendicitis no appendicitis
  Female          200             176
  Male            262             141
```

What is returned from when we create a table? An array! (homogenous data structure - 1D array is a vector, 2D is a matrix)

That means we can subset them if want to! Let's return the *conditional* one-way table of Sex based on only those that had appendicitis:

```
two_way_sex_diag <- table(app_data$SexF, app_data$DiagnosisF)
two_way_sex_diag[,1]
```

```
Female   Male 
   200    262 
```

Nice! Things do get more complicated if we add in a third variable as it is tough to display that info compactly.

```
table(app_data$SexF, app_data$DiagnosisF, app_data$SeverityF)
```

```
, ,  = complicated


        appendicitis no appendicitis
  Female           55               1
  Male             63               0

, ,  = uncomplicated


        appendicitis no appendicitis
  Female          145             175
  Male            199             141
```

If you look at the output you see `, , = complicated`. This is R hinting at how to access this 3D array!

We can return the conditional two-way table of Sex and Diagnosis for only those with an uncomplicated situation:

```r
three_way <- table(app_data$SexF, app_data$DiagnosisF, app_data$SeverityF)
three_way[, , "uncomplicated"]
```

```
        appendicitis no appendicitis
  Female          145             175
  Male            199             141
```

```r
#or
three_way[, , 2]
```

```
        appendicitis no appendicitis
  Female          145             175
  Male            199             141
```

We can also get a one-way table conditional on two of the variables. Here is the one-way table for sex for only those with an uncomplicated situation and no appendicitis:

```r
three_way[, 2, 2]
```

```
Female   Male 
   175    141 
```

Lastly, just note that you can supply a data frame instead of the individual vectors.

```r
table(app_data[, c("SexF", "DiagnosisF")])
```

```
        DiagnosisF
SexF     appendicitis no appendicitis
  Female          200             176
```

```
  Female          [illegible]          [illegible]
  Male            262                  141
```

Via the `tidyverse`

Ok, great. But we might want to stay in the `tidyverse`. We can use the `dplyr::summarize()` function to compute summaries on a `tibble`. This generally outputs a `tibble` with fewer rows than the original (as we are summarizing the variables to view them in a more compact form). We often use `group_by()` to set a grouping variable. **Any summary done will respect the groupings!**

Any of the common summarization functions you can think of are likely permissible in `summarize()`. The one for counting values is simply `n()`. Let's recreate all of our above tables under the `tidyverse` method.

One-way table:

```r
app_data |>
  group_by(SexF) |>
  summarize(count = n())
```

```
# A tibble: 3 × 2
  SexF   count
  <fct>  <int>
1 Female   377
2 Male     403
3 <NA>       2
```

Notice that `NA` values are included by default (probably a good thing). We can remove those with `tidyr::drop_na()`.

```r
app_data |>
  drop_na(SexF) |>
  group_by(SexF) |>
  summarize(count = n())
```

```
# A tibble: 2 × 2
  SexF   count
  <fct>  <int>
1 Female   377
2 Male     403
```

Two-way table: Simply add another grouping variable. The `summarize()` function respects these groups when counting!

```r
app_data |>
  drop_na(SexF, DiagnosisF) |>
  group_by(SexF, DiagnosisF) |>
  summarize(count = n())
```

```
`summarise()` has grouped output by 'SexF'. You can override using the
`.groups` argument.
```

```
# A tibble: 4 × 3
# Groups:   SexF [2]
  SexF   DiagnosisF      count
  <fct>  <fct>           <int>
1 Female appendicitis      200
2 Female no appendicitis   176
3 Male   appendicitis      262
4 Male   no appendicitis   141
```

Nice. But that isn't in the best way for viewing (i.e. a wider format would be more compact for displaying). Let's use `tidyr::pivot_wider()` to fix that!

```
app_data |>
  drop_na(SexF, DiagnosisF) |>
  group_by(SexF, DiagnosisF) |>
  summarize(count = n()) |>
  pivot_wider(names_from = DiagnosisF, values_from = count)
```

```
`summarise()` has grouped output by 'SexF'. You can override using the
`.groups` argument.
```

```
# A tibble: 2 × 3
# Groups:   SexF [2]
  SexF   appendicitis `no appendicitis`
  <fct>         <int>             <int>
1 Female          200               176
2 Male            262               141
```

Three-way table: Again, just add more grouping variables!

```
app_data |>
  drop_na(SexF, DiagnosisF, SeverityF) |>
  group_by(SexF, DiagnosisF, SeverityF) |>
  summarize(count = n())
```

```
`summarise()` has grouped output by 'SexF', 'DiagnosisF'. You can override
using the `.groups` argument.
```

```
# A tibble: 7 × 4
# Groups:   SexF, DiagnosisF [4]
  SexF   DiagnosisF      SeverityF      count
  <fct>  <fct>           <fct>          <int>
1 Female appendicitis    complicated       55
2 Female appendicitis    uncomplicated    145
3 Female no appendicitis complicated        1
4 Female no appendicitis uncomplicated    175
5 Male   appendicitis    complicated       63
6 Male   appendicitis    uncomplicated    199
7 Male   no appendicitis uncomplicated    141
```

We can also pivot this, although there is no great way to get all the info there. We'll just move the severity variable across the top.

```
app_data |>
```

```
  drop_na(SexF, DiagnosisF, SeverityF) |>
  group_by(SexF, DiagnosisF, SeverityF) |>
  summarize(count = n()) |>
  pivot_wider(names_from = SeverityF, values_from = count)
```

```
`summarise()` has grouped output by 'SexF', 'DiagnosisF'. You can override
using the `.groups` argument.

# A tibble: 4 × 4
# Groups:   SexF, DiagnosisF [4]
  SexF   DiagnosisF      complicated uncomplicated
  <fct>  <fct>                 <int>         <int>
1 Female appendicitis             55           145
2 Female no appendicitis           1           175
3 Male   appendicitis             63           199
4 Male   no appendicitis          NA           141
```

Recap!

Contingency tables summarize the distribution of one or more categorical variables. We can create them using

- `table()` - returns an array of counts
- `group_by()` along with `summarize()` and the `n()` function