Tabulated Counts

Table of contents

Summarising data 3
Exercise 5.1 4
Exercise 5.2 4
Exercise 5.3 4
Exercise 5.4 5

Tidying data 6
Exercise 5.5 7

Combining datasets 8
Types of join 8
We will illustrate them using a simple example: 8
Joining data: Example 10
Specifying join keys 11
Exercise 5.6 13
Exercise 5.7 14

case_when() 15
Exercise 5.8 16

Dataset `Toxaemia` 17
Exercise 5.9 17
Exercise 5.10 20

Data `diamonds` 23
Exercise 5.11 23

Summarising data

```
library(tidyverse)
my.tv.data <- read_csv(
  "../data/tv.csv",
  col_types = "nfcc"
  )
```

The argument `col_types = "nfcc"` stands for {`numeric`, `factor`, `character`, `character`}, to match the order of the columns.

```
my.tv.data
```

```
# A tibble: 46 x 4
   TELETIME SEX   SCHOOL STANDARD
      <dbl> <fct> <chr>  <chr>
 1     1482 1     1      4
 2     2018 1     1      4
 3     1849 1     1      4
 4      857 1     1      4
 5     2027 2     1      4
 6     2368 2     1      4
 7     1783 2     1      4
 8     1769 2     1      4
 9     2534 1     1      3
10     2366 1     1      3
# i 36 more rows
```

We often do a summary of a numerical variable for a given categorical variable. For example, we like to see obtain the summary statistics of TV viewing times for various schools. The commands

```
by(my.tv.data$TELETIME, my.tv.data$SCHOOL, summary)
```

We employed the `by()` command above and instead, we may also use `tapply()` `aggregate()` functions:

`tapply(my.data$TELETIME, my.data$SCHOOL, summary)`

`aggregate(my.data$TELETIME, list(my.data$SCHOOL), summary)`

A tabulated summary of categorical data is obtained using the `table()` command. You can also use the tidyverse employing functions like filter, group_by, and summarise.

```r
rangitikei <- read.csv(
  "../data/rangitikei.csv",
  header=TRUE,
  row.names = 1
  )

wind <- rangitikei |> pull(wind)
river <- rangitikei |> pull(river)

table(wind, river)
```

Exercise 5.1

Describe the table generated above. What does it tell you?

Exercise 5.2

Generate a table that summarizes the mean number of vehicles at each level of wind and temperature. What does this table tell you about the data?

```r
# your code goes here
```

Exercise 5.3

Now repeat this for the average people observed. What does this tell you about the data?

```r
# your code goes here
```

Exercise 5.4

Given the two tables you produced describe patterns of people and vehicles at Rangitikei rivers.

Tidying data

Most real analyses will require at least a little tidying. You'll begin by figuring out what the underlying variables and observations are. Sometimes this is easy; other times you'll need to consult with the people who originally generated the data. Next, you'll pivot your data into a tidy form, with variables in the columns and observations in the rows.

The `billboard` dataset records the billboard rank of songs in the year 2000:

```
billboard
```

```
# A tibble: 317 x 79
   artist     track date.entered   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8
   <chr>      <chr> <date>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 2 Pac      Baby~ 2000-02-26      87    82    72    77    87    94    99    NA
 2 2Ge+her    The ~ 2000-09-02      91    87    92    NA    NA    NA    NA    NA
 3 3 Doors D~ Kryp~ 2000-04-08      81    70    68    67    66    57    54    53
 4 3 Doors D~ Loser 2000-10-21      76    76    72    69    67    65    55    59
 5 504 Boyz   Wobb~ 2000-04-15      57    34    25    17    17    31    36    49
 6 98^0       Give~ 2000-08-19      51    39    34    26    26    19     2     2
 7 A*Teens    Danc~ 2000-07-08      97    97    96    95   100    NA    NA    NA
 8 Aaliyah    I Do~ 2000-01-29      84    62    51    41    38    35    35    38
 9 Aaliyah    Try ~ 2000-03-18      59    53    38    28    21    18    16    14
10 Adams, Yo~ Open~ 2000-08-26      76    76    74    69    68    67    61    58
# i 307 more rows
# i 68 more variables: wk9 <dbl>, wk10 <dbl>, wk11 <dbl>, wk12 <dbl>,
#   wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>, wk18 <dbl>,
#   wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>, wk24 <dbl>,
#   wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>, wk30 <dbl>,
#   wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>, wk36 <dbl>,
#   wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>, wk42 <dbl>, ...
```

In this dataset, each observation is a song. The first three columns (artist, track and date.entered) are variables that describe the song. Then we have 76 columns (wk1-wk76) that describe the rank of the song in each week. Here, the column names are one variable (the week) and the cell values are another (the rank).

We can use the `dplyr` function `pivot_longer()` to change this data into `long format`.

The command `pivot_wider()` does the opposite to `pivot_longer()`

Exercise 5.5

Use `pivot_longer()` to tidy this data

```
# your code goes here
```

Combining datasets

Sometimes you may want to analyse information that is in multiple tables.

We will introduce a series of functions for joining tables together.

Two tables can be connected through a pair of keys, within each table.

Every join involves a pair of keys: a primary key and a foreign key. A primary key is a variable or set of variables that uniquely identifies each observation. When more than one variable is needed, the key is called a compound key.

Types of join

There are four types of mutating join, which differ in their behaviour when a match is not found.

- `inner_join(x, y)` only includes observations that match in both x and y.
- `left_join(x, y)` includes all observations in `x`, regardless of whether they match or not. This is the most commonly used join because it ensures that you don't lose observations from your primary table.
- `right_join(x, y)` includes all observations in `y`. It's equivalent to `left_join(y, x)`, but the columns and rows will be ordered differently.
- `full_join()` includes all observations from x and y.

The left, right and full joins are collectively know as outer joins. When a row doesn't match in an outer join, the new variables are filled in with missing values.

We will illustrate them using a simple example:

First lets make some data:

```
df1 <- tibble(x = c(1, 2), y = 2:1)
df1
```

```
# A tibble: 2 x 2
      x     y
  <dbl> <int>
1     1     2
2     2     1
```

```
df2 <- tibble(x = c(3, 1), a = 10, b = "a")
df2
```

```
# A tibble: 2 x 3
      x     a b
  <dbl> <dbl> <chr>
1     3    10 a
2     1    10 a
```

Now join the two datasets using the different join functions:

```
df1 %>% inner_join(df2)
```

```
# A tibble: 1 x 4
      x     y     a b
  <dbl> <int> <dbl> <chr>
1     1     2    10 a
```

```
df1 %>% left_join(df2)
```

```
# A tibble: 2 x 4
      x     y     a b
  <dbl> <int> <dbl> <chr>
1     1     2    10 a
2     2     1    NA <NA>
```

```
df1 %>% right_join(df2)
```

```
# A tibble: 2 x 4
      x     y     a b
  <dbl> <int> <dbl> <chr>
1     1     2    10 a
2     3    NA    10 a
```

```r
df2 %>% left_join(df1)
```

```
# A tibble: 2 x 4
      x     a b         y
  <dbl> <dbl> <chr> <int>
1     3    10 a        NA
2     1    10 a         2
```

```r
df1 %>% full_join(df2)
```

```
# A tibble: 3 x 4
      x     y     a b    
  <dbl> <int> <dbl> <chr>
1     1     2    10 a    
2     2     1    NA <NA> 
3     3    NA    10 a    
```

What are the differences between the join functions?

Joining data: Example

For another example, consider the flights and airlines data from the nycflights13 package.

In one table we have flight information with an abbreviation for carrier, and in another we have a mapping between abbreviations and full names.

You can use a join to add the carrier names to the flight data:

```r
library(nycflights13)
# Drop unimportant variables so it's easier to understand the join results.
flights2 <- flights %>% select(year:day, hour, origin, dest, tailnum, carrier)

flights2 %>%
  left_join(airlines)
```

```
# A tibble: 336,776 x 9
    year month   day  hour origin dest  tailnum carrier name                    
   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <chr>                   
 1  2013     1     1     5 EWR    IAH   N14228  UA      United Air Lines Inc.
 2  2013     1     1     5 LGA    IAH    N24211  UA      United Air Lines Inc.
 3  2013     1     1     5 JFK    MIA    N619AA  AA      American Airlines Inc.
 4  2013     1     1     5 JFK    BQN    N804JB  B6      JetBlue Airways
 5  2013     1     1     6 LGA    ATL    N668DN  DL      Delta Air Lines Inc.
 6  2013     1     1     5 EWR    ORD    N39463  UA      United Air Lines Inc.
 7  2013     1     1     6 EWR    FLL    N516JB  B6      JetBlue Airways
 8  2013     1     1     6 LGA    IAD    N829AS  EV      ExpressJet Airlines Inc.
 9  2013     1     1     6 JFK    MCO    N593JB  B6      JetBlue Airways
10  2013     1     1     6 LGA    ORD    N3ALAA  AA      American Airlines Inc.
# i 336,766 more rows
```

As well as x and y, each mutating join takes an argument `by` that controls which variables are used to match observations in the two tables.

Specifying join keys

By default, left_join() will use all variables that appear in both data frames as the join key, but it doesn't always work.

For example lets look at some airline data:

```r
flights2 <- flights |>
  select(year, time_hour, origin, dest, tailnum, carrier)
flights2
```

```
# A tibble: 336,776 x 6
    year time_hour           origin dest  tailnum carrier
   <int> <dttm>              <chr>  <chr> <chr>   <chr>
 1  2013 2013-01-01 05:00:00 EWR    IAH   N14228  UA
 2  2013 2013-01-01 05:00:00 LGA    IAH   N24211  UA
 3  2013 2013-01-01 05:00:00 JFK    MIA   N619AA  AA
 4  2013 2013-01-01 05:00:00 JFK    BQN   N804JB  B6
 5  2013 2013-01-01 06:00:00 LGA    ATL   N668DN  DL
 6  2013 2013-01-01 05:00:00 EWR    ORD   N39463  UA
 7  2013 2013-01-01 06:00:00 EWR    FLL   N516JB  B6
 8  2013 2013-01-01 06:00:00 LGA    IAD   N829AS  EV
 9  2013 2013-01-01 06:00:00 JFK    MCO   N593JB  B6
10  2013 2013-01-01 06:00:00 LGA    ORD   N3ALAA  AA
# i 336,766 more rows
```

```
planes
```

```
# A tibble: 3,322 x 9
   tailnum  year type               manufacturer model engines seats speed engine
   <chr>   <int> <chr>              <chr>        <chr>   <int> <int> <int> <chr> 
 1 N10156   2004 Fixed wing multi~ EMBRAER      EMB-~       2    55    NA Turbo~
 2 N102UW   1998 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 3 N103US   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 4 N104UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 5 N10575   2002 Fixed wing multi~ EMBRAER      EMB-~       2    55    NA Turbo~
 6 N105UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 7 N107US   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 8 N108UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 9 N109UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
10 N110UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
# i 3,312 more rows
```

We are going to try to going flight data to data about types of planes

```
flights2 |>
  left_join(planes)
```

```
# A tibble: 336,776 x 13
    year time_hour           origin dest  tailnum carrier type  manufacturer
   <int> <dttm>              <chr>  <chr> <chr>   <chr>   <chr> <chr>       
 1  2013 2013-01-01 05:00:00 EWR    IAH   N14228  UA      <NA>  <NA>        
 2  2013 2013-01-01 05:00:00 LGA    IAH   N24211  UA      <NA>  <NA>        
 3  2013 2013-01-01 05:00:00 JFK    MIA   N619AA  AA      <NA>  <NA>        
 4  2013 2013-01-01 05:00:00 JFK    BQN   N804JB  B6      <NA>  <NA>        
 5  2013 2013-01-01 06:00:00 LGA    ATL   N668DN  DL      <NA>  <NA>        
 6  2013 2013-01-01 05:00:00 EWR    ORD   N39463  UA      <NA>  <NA>        
 7  2013 2013-01-01 06:00:00 EWR    FLL   N516JB  B6      <NA>  <NA>        
 8  2013 2013-01-01 06:00:00 LGA    IAD   N829AS  EV      <NA>  <NA>        
 9  2013 2013-01-01 06:00:00 JFK    MCO   N593JB  B6      <NA>  <NA>        
10  2013 2013-01-01 06:00:00 LGA    ORD   N3ALAA  AA      <NA>  <NA>        
# i 336,766 more rows
# i 5 more variables: model <chr>, engines <int>, seats <int>, speed <int>,
#   engine <chr>
```

We get a lot of missing matches because our join is trying to use tailnum and year as a compound key. Both flights and planes have a year column but they mean different things:

`flights$year` is the year the flight occurred and `planes$year` is the year the plane was built.

Since these represent different types of data and we might want to keep both, we can be explicit about how to join the two data tables:

```
flights2 |>
  left_join(planes, join_by(tailnum))
```

```
# A tibble: 336,776 x 14
   year.x time_hour           origin dest  tailnum carrier year.y type
    <int> <dttm>              <chr>  <chr> <chr>   <chr>    <int> <chr>
 1   2013 2013-01-01 05:00:00 EWR    IAH   N14228  UA        1999 Fixed wing mu~
 2   2013 2013-01-01 05:00:00 LGA    IAH   N24211  UA        1998 Fixed wing mu~
 3   2013 2013-01-01 05:00:00 JFK    MIA   N619AA  AA        1990 Fixed wing mu~
 4   2013 2013-01-01 05:00:00 JFK    BQN   N804JB  B6        2012 Fixed wing mu~
 5   2013 2013-01-01 06:00:00 LGA    ATL   N668DN  DL        1991 Fixed wing mu~
 6   2013 2013-01-01 05:00:00 EWR    ORD   N39463  UA        2012 Fixed wing mu~
 7   2013 2013-01-01 06:00:00 EWR    FLL   N516JB  B6        2000 Fixed wing mu~
 8   2013 2013-01-01 06:00:00 LGA    IAD   N829AS  EV        1998 Fixed wing mu~
 9   2013 2013-01-01 06:00:00 JFK    MCO   N593JB  B6        2004 Fixed wing mu~
10   2013 2013-01-01 06:00:00 LGA    ORD   N3ALAA  AA          NA <NA>
# i 336,766 more rows
# i 6 more variables: manufacturer <chr>, model <chr>, engines <int>,
#   seats <int>, speed <int>, engine <chr>
```

Now by default there is a year.x and year.y coming from the flights and planes data respectively. You can change this using the `suffix` argument in join or by renaming the columns after the join.

Exercise 5.6

Imagine you've found the top 10 most popular destinations using this code:

```
top_dest <- flights2 |>
  count(dest, sort = TRUE) |>
  head(10)
```

How can you find all flights to those destinations?

```
# your code goes here
```

Exercise 5.7

What do the tail numbers that don't have a matching record in planes have in common? (Hint: one variable explains ~90% of the problems.)

```
# your code goes here
```

case_when()

Sometimes you want to add a new variable to your data based on existing variables.

dplyr's case_when() is inspired by SQL's CASE statement and provides a flexible way of performing different computations for different conditions. It has a special syntax that unfortunately looks like nothing else you'll use in the tidyverse. It takes pairs that look like condition ~ output. condition must be a logical vector; when it's TRUE, output will be used.

This can take the place of if_else() statements.

For example:

```r
flights |>
  mutate(
    status = case_when( # make a new variable status based on the flights arrival delay
      is.na(arr_delay)      ~ "cancelled",
      arr_delay < -30       ~ "very early",
      arr_delay < -15       ~ "early",
      abs(arr_delay) <= 15  ~ "on time",
      arr_delay < 60        ~ "late",
      arr_delay < Inf       ~ "very late",
    ),
    .keep = "used"
  )
```

```
# A tibble: 336,776 x 2
   arr_delay status
       <dbl> <chr>
 1        11 on time
 2        20 late
 3        33 late
 4       -18 early
 5       -25 early
 6        12 on time
 7        19 late
 8       -14 on time
 9        -8 on time
```

```
10         8 on time
# i 336,766 more rows
```

Exercise 5.8

Write a case_when() statement that uses the month and day columns from flights to label a selection of important US holidays (e.g., New Years Day, 4th of July, Thanksgiving, and Christmas). First create a logical column that is either TRUE or FALSE, and then create a character column that either gives the name of the holiday or is NA.

```
# your code goes here
```

Dataset `Toxaemia`

This dataset is from the `vcdExtra` package. Two signs of *toxaemia*, an abnormal condition during pregnancy characterized by high blood pressure (hypertension) and high levels of protein in the urine. If untreated, both the mother and baby are at risk of complications or death. The dataset `Toxaemia` represents 13384 expectant mothers in Bradford, England in their first pregnancy, who were also classified according to social class and the number of cigarettes smoked per day.

The dataset is a 5 x 3 x 2 x 2 contingency table, with 60 observations on the following 5 variables:

`class` - Social class of mother, a factor with levels: 1, 2, 3, 4, 5

`smoke` - Cigarettes smoked per day during pregnancy, a factor with levels: 0, 1-19, 20+

`hyper` - Hypertension level, a factor with levels: Low, High

`urea` - Protein urea level, a factor with levels: Low, High

`Freq` - frequency in each cell, a numeric vector

Exercise 5.9

Obtain relevant graphical displays for this dataset.

Bar charts-

```
library(tidyverse)

library(vcdExtra)
data(Toxaemia)

Toxaemia |>
  ggplot() +
  aes(x=smoke, y=Freq, fill=hyper) +
  geom_bar(stat='identity')
```

```r
Toxaemia |>
  ggplot() +
  aes(x=smoke, y=Freq, fill=hyper) +
  geom_bar(stat='identity',
           position = "dodge"
           )
```

```r
Toxaemia |>
  ggplot() +
  aes(x=smoke, y=Freq, fill=hyper) +
  geom_bar(stat ='identity',
           position = "dodge") +
  facet_grid(urea ~ ., scales = "free")
```

Mosaic type charts

```r
tab.data <- xtabs(Freq ~ smoke + hyper + urea, data=Toxaemia)

plot(tab.data)
```

```r
mosaic(tab.data, shade=TRUE, legend=TRUE)
```

```r
assoc(tab.data, shade=TRUE)
```

```r
strucplot(tab.data)
```

```r
sieve(tab.data)
```

The full dataset is a 5 x 3 x 2 x 2 contingency table, with 60 observations on the following 5 variables. For this question we will focus on two categorical variables from this dataset, `hyper` and `urea`. This forms a 2 x 2 contingency table since these variables each have two levels.

```r
# subset the data
tox_2 <- Toxaemia |>
  dplyr::select(hyper, urea, Freq)
```

```r
# the tidyverse way
tox_display <- tox_2 |>
  pivot_wider(names_from = urea,
              values_from = Freq,
              values_fn = sum) |>
  column_to_rownames( var = "hyper") # make values of hyper column row names

tox_display
```

```
     High  Low
High  665 2715
Low   589 9415
```

```
# xtabs()
```

Two signs of *toxaemia*, are high blood pressure (hypertension) and high levels of protein in the urine. We want to ask if in our sample of expectant mothers in Bradford, England, is high blood pressure related to high protein levels? If these two variables are associated this may indicate the presence of toxaemia in the sample, if they are independent toxaemia may not be present.

We can test this question using a Chi-squared test.

The null hypothesis of the chi-squared these is that the two variables are independent and the alternative hypothesis is that the two variables are not independent.

Our null hypothesis is that Hypertension level and the Protein urea level in expectant mothers in Bradford, England are independent.

Our alternative hypothesis that Hypertension level and the Protein urea level in expectant mothers in Bradford, England are *not* independent.

Set our alpha = 0.05

```
chisq.test(tox_display)
```

```

	Pearson's Chi-squared test with Yates' continuity correction

data:  tox_display
X-squared = 563.9, df = 1, p-value < 2.2e-16
```

Since our p-value is less than our alpha level we reject the null hypothesis and conclude that the two variables (hyper & urea) are not independent. We found evidence of an association between hypertension levels and protein in urine levels in our sample of expectant mothers in Bradford, England.

We can see the expected counts

```r
chisq.test(tox_display)$expected
```

```
         High      Low
High 316.6856 3063.314
Low  937.3144 9066.686
```

```r
# compared to our observed
tox_display
```

```
     High  Low
High  665 2715
Low   589 9415
```

```r
# total counts 13384
```

Exercise 5.10

The genetic information of an organism is stored in its Deoxyribonucleic acid (DNA). DNA is a double stranded helix made up of four different nucleotides. These nucleotides differ in which of the four bases Adenine (A), Guanine (G), Cytosine (C), or Thymine (T) they contain. Nucleotides combine to form amino acids which are the building blocks of proteins. Simply put, three nucleotides form an amino acid and the specific order of a combination dictates what amino acid is formed. A simple pattern that we may want to detect in a DNA sequence is that of the nucleotide at position `i+1` based on the nucleotide at position `i`. The nucleotide positional data collected by a researcher in a particular case is given in the following table:

`i\(i+1)`	A	C	G	T
A	622	316	328	536
C	428	262	204	306
G	354	294	174	266
T	396	330	382	648

Perform a test of association and then obtain the symmetric plot.

```r
tabledata <- data.frame(
  A_1 = c(622, 428, 354, 396),
  C_1 = c(316, 262, 294, 330),
  G_1 = c(328, 204, 174, 382),
  T_1 = c(536, 306, 266, 648),
  row.names = c("A", "C", "G", "T")
  )
```

```r
chisq.test(tabledata)$exp
```

```
       A_1      C_1      G_1      T_1
A 554.8409 370.5104 335.3705 541.2781
C 369.4834 246.7328 223.3322 360.4516
G 334.9983 223.7044 202.4879 326.8094
T 540.6774 361.0523 326.8094 527.4608
```

```r
chisq.test(tabledata)
```

```

    Pearson's Chi-squared test

data:  tabledata
X-squared = 153.21, df = 9, p-value < 2.2e-16
```

```r
chisq.test(tabledata, simulate.p.value = T)
```

```

    Pearson's Chi-squared test with simulated p-value (based on 2000
    replicates)

data:  tabledata
X-squared = 153.21, df = NA, p-value = 0.0004998
```

```r
# if there is an association we can examine patterns
library(MASS)
corresp(tabledata)
```

```
First canonical correlation(s): 0.1443355

 Row scores:
         A          C          G          T
-0.1921802 -0.8894387 -1.0334109  1.4453224
```

```
 Column scores:
       A_1        C_1        G_1        T_1
-1.1304512 -0.6952989  0.8139424  1.1304056
```

```
plot(corresp(tabledata, nf=2))
abline(v=0)
abline(h=0)
```

```
#or
library(FactoMineR)
CA(tabledata)
```

```
**Results of the Correspondence Analysis (CA)**
The row variable has  4  categories; the column variable has 4 categories
The chi square of independence between the two variables is equal to 153.2146 (p-value =  1.
*The results are available in the following objects:

   name              description
1  "$eig"            "eigenvalues"
2  "$col"            "results for the columns"
3  "$col$coord"      "coord. for the columns"
4  "$col$cos2"       "cos2 for the columns"
5  "$col$contrib"    "contributions of the columns"
6  "$row"            "results for the rows"
7  "$row$coord"      "coord. for the rows"
8  "$row$cos2"       "cos2 for the rows"
9  "$row$contrib"    "contributions of the rows"
10 "$call"           "summary called parameters"
11 "$call$marge.col" "weights of the columns"
12 "$call$marge.row" "weights of the rows"
```

Data `diamonds`

The `diamonds` dataset contains the prices and other attributes of almost 54,000 diamonds. Use `?diamonds` to see information for each variable.

```
data("diamonds")
names(diamonds)
```

```
 [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"  
 [8] "x"       "y"       "z"      
```

```
## Some EDA plots
ggplot(diamonds, aes(color))+geom_bar() + facet_wrap(~cut)
```

```
ggplot(diamonds, aes(color))+geom_bar(aes(fill=cut))
```

```
ggplot(diamonds, aes(color))+geom_bar(aes(fill=cut))+ facet_wrap(~clarity)
```

Exercise 5.11

We are interested in whether there is an association between cut and color. Perform a test of association and then obtain the symmetric plot. Write your hypotheses, perform the test, make a decision based on the results of the test, and report a conclusion in context.

```
# your code goes here
```

- More R code examples are here