Reshaping and transforming your data

The tidyverse Part 2

Joey Stanley
Doctoral Candidate in Linguistics, University of Georgia
joeystanley.com
orcid.org/0000-0002-9185-0048

Presented at the UGA Willson Center DigiLab
Friday, March 23, 2018

This is the ninth installment of the R workshop series in Spring 2018 and the second workshop that introduces functions from the "tidyverse." This document will cover these introductory topics: (1) loading the familiarizing yourself with the data used in this workshop; (2) various ways of merging datasets together; (3) summarizing your data by group; and (4) reshaping your data from tall to wide and vice versa.

Download this PDF from my website at
joeystanley.com/r2018

1 Introduction

This is the second in a two-part series of Tidyverse workshops. In the last one[1] we looked at three main topics:

- Getting data in and out of R
- Tidying columns (including reordering, renaming, adding, removing, and modifying)
- Filtering

These are very important things you should be relatively comfortable with, regardless of what you use R for. They're also relatively easy topics for you to grasp because, for the most part, you probably know how to do them just as well in Excel. In fact, you might be wondering why bother learning R and tidyverse functions when they can just do what Excel does…

Today's workshop covers some more advanced topics that are a little more difficult to conceptualize and, crucially, are *much* more difficult to do in Excel. In fact, I don't know if Excel can even do some of the stuff that we'll be learning today—and we'll be able to do it in just one line of code! The three main topics are merging, summarizing, and reshaping your data. It took me a while to get the hang of them, but boy once you do they sure are some sweet skills to have up your sleeve.

As always, before we get into anything, let's load the tidyverse package.

```
library(tidyverse)
```

1.1 Data

While I generally never watch sporting events, for some reason I go crazy during the Olympics and watch way more than I should. Given that the Winter Olympics just ended, I thought we could take a look at data from previous Winter Olympics. Today we'll work with data from all the Winter Olympics from the first one in 1924 until the second most recent one in Sochi in 2014. This data was originally downloaded from Kaggle.com[2] and then I modified it a little bit to make it appropriate for this workshop.

The data is stored in three separate spreadsheets, which you can download directly from my website using the links below.

The first is a spreadsheet of all the events that occurred each year. It contains the following columns:

[1] http://joeystanley.com/downloads/180302-tidyverse_part1.html

[2] https://www.kaggle.com/the-guardian/olympic-games/data

1. `Year`: The year the event took place. Some events have been discontinued and others added, so the list of specific events changes from year to year. For example, Military patrol ended in 1948 while mixed doubles curling was added in 2018.

2. `Sport`: The broad terms for the different sports in the games (Biathlon, Bobsleigh, Curling, Ice Hockey, Luge, Skating, and Skiing)

3. `Discipline`: A more narrow term within each sport. For example, Skeleton and Bobsleigh are disciplines within the sport Bobsleigh, or Figure Skating, Short Track Speed Skating, and Speed Skating are disciplines within the sport Skating.

4. `Event`: The name of the specific event that an athlete can medal in. So within the Figure Skating discipline, there are four events: Ice Dancing, Individual, Pairs, and Team.

Unfortunately, with the data that I could find, I don't have information on mixed events, such as Mixed Curling or Mixed Relay Biathlon, which have both men and women on a team. However, this can mostly be inferred by the gender of the athlete and the name of the event, which tend of have the word *mixed* in it (Mixed Relay Biathlon).

Let's go ahead and read that data in so we can get an idea of what we're looking at.

```
events <- read_csv("http://joeystanley.com/data/events.csv")
events

## # A tibble: 716 x 4
##     Year    Sport Discipline Event
##    <int>    <chr>      <chr> <chr>
##  1  2014 Biathlon   Biathlon  10KM
##  2  2010 Biathlon   Biathlon  10KM
##  3  2006 Biathlon   Biathlon  10KM
##  4  2002 Biathlon   Biathlon  10KM
##  5  1998 Biathlon   Biathlon  10KM
##  6  1994 Biathlon   Biathlon  10KM
##  7  1992 Biathlon   Biathlon  10KM
##  8  1988 Biathlon   Biathlon  10KM
##  9  1984 Biathlon   Biathlon  10KM
## 10  1980 Biathlon   Biathlon  10KM
## # ... with 706 more rows
```

The next dataset has information about all the athletes that got at least one medal. This spreadsheet has six straightforward columns containing the person's name, country, gender, discipline, event, year, and medal.

```
athletes <- read_csv("http://joeystanley.com/data/athletes.csv")
athletes

## # A tibble: 5,770 x 7
##                  Athlete Country Gender           Discipline
##                    <chr>   <chr>  <chr>                <chr>
##  1       AAHLBERG, Mats     SWE    Men           Ice Hockey
##  2       AAHLEN, Thomas     SWE    Men           Ice Hockey
##  3     AALAND, Per Knut     NOR    Men Cross Country Skiing
##  4   AALTONEN, Juhamatti    FIN    Men           Ice Hockey
##  5 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  6 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  7 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  8 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  9 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
## 10 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
## # ... with 5,760 more rows, and 3 more variables: Event <chr>, Year <int>,
## #   Medal <chr>
```

Finally, let's read in data about the locations each year. This is a spreadsheet that has columns for the year, city, country, and continent where the games happened. Let's read that in now.

```
years <- read_csv("http://joeystanley.com/data/years.csv")
years

## # A tibble: 24 x 4
##     Year           City       Country     Continent
##    <int>          <chr>         <chr>         <chr>
##  1  2022        Beijing         China          Asia
##  2  2018    Pyeongchang   South Korea          Asia
##  3  2014          Sochi        Russia          Asia
##  4  2010      Vancouver        Canada North America
##  5  2006          Turin         Italy        Europe
##  6  2002 Salt Lake City United States North America
##  7  1998         Nagano         Japan          Asia
##  8  1994    Lillehammer        Norway        Europe
##  9  1992    Albertville        France        Europe
## 10  1988        Calgary        Canada North America
## # ... with 14 more rows
```

Okay, so we have three spreadsheets that are all kinda related. So without further ado, let's get to combining them.

2 Joining datasets

But first, why bother have all this data stored in different spreadsheets in the first place? Why not have a single file that contains all the athletes info, the events, and the location? The reason for it is because splitting them into three reduces the amount of redundant information in each spreadsheet.

Let's say we had a spreadsheet of all the athletes names, years, and city where the Olympics were held. We would have 5,770 rows and three columns. But, in the column with the city would be really redundant. You'd have *Sochi*, *South Korea*, and *Asia* there hundreds of times, right next to a 2014. If we split the file into two, with the athlete name and year in one, and then a smaller one with just the year and city in another, we only need to type *Sochi* once. Yes, we repeat the year, but the number of repeats this way is far fewer than the repeats in a monster spreadsheet. In the end, you can still easily recover the data, and the overall file size of of the spreadsheets is much smaller than the size of a giant one.

Merging datasets might remind you of databases. In fact, they're essentially the same thing! In a database, you have two or more spreadsheets that are linked in some way, usually by some sort of key identifier. With the various `*_join` functions in `dyplr`, you can connect datasets in a way very reminiscent of how you might do it in database software.

2.1 `LEFT_JOIN` AND `RIGHT_JOIN`

So let's start by taking the athlete data and adding the city names. To get an idea of how we might merge them, let's look at them one more time:

```
athletes

## # A tibble: 5,770 x 7
##                  Athlete Country Gender           Discipline
##                    <chr>   <chr>  <chr>                <chr>
##  1       AAHLBERG, Mats     SWE    Men           Ice Hockey
##  2       AAHLEN, Thomas     SWE    Men           Ice Hockey
##  3     AALAND, Per Knut     NOR    Men Cross Country Skiing
##  4  AALTONEN, Juhamatti     FIN    Men           Ice Hockey
##  5 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  6 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  7 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  8 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  9 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
## 10 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
## # ... with 5,760 more rows, and 3 more variables: Event <chr>, Year <int>,
## #   Medal <chr>

years

## # A tibble: 24 x 4
##     Year           City       Country     Continent
##    <int>          <chr>         <chr>         <chr>
##  1  2022        Beijing         China          Asia
##  2  2018    Pyeongchang   South Korea          Asia
##  3  2014          Sochi        Russia          Asia
##  4  2010      Vancouver        Canada North America
##  5  2006          Turin         Italy        Europe
##  6  2002 Salt Lake City United States North America
##  7  1998         Nagano         Japan          Asia
##  8  1994    Lillehammer        Norway        Europe
##  9  1992    Albertville        France        Europe
## 10  1988        Calgary        Canada North America
## # ... with 14 more rows
```

What do these two spreadsheets have in common? Well, they both have a column with the years. Coincidentally, in both spreadsheets, this column is named `Year`. (Identical names across spreadsheets, while not required, do make this kind of work easier.) So, if we want to add the city name to the athlete data, we can use the `left_join` function. I'm going to just `select` the relevant columns right now to make it clearer.

```
athletes %>%
    select(Athlete, Year) %>%
    left_join(years, by = "Year")
```

```
## # A tibble: 5,770 x 5
##                 Athlete  Year           City       Country     Continent
##                   <chr> <int>          <chr>         <chr>         <chr>
##  1       AAHLBERG, Mats  1980    Lake Placid United States North America
##  2       AAHLEN, Thomas  1984       Sarajevo    Yugoslavia        Europe
##  3     AALAND, Per Knut  1980    Lake Placid United States North America
##  4  AALTONEN, Juhamatti  2014          Sochi        Russia          Asia
##  5 AAMODT, Kjetil Andre  1992    Albertville        France        Europe
##  6 AAMODT, Kjetil Andre  1992    Albertville        France        Europe
##  7 AAMODT, Kjetil Andre  1994    Lillehammer        Norway        Europe
##  8 AAMODT, Kjetil Andre  1994    Lillehammer        Norway        Europe
##  9 AAMODT, Kjetil Andre  1994    Lillehammer        Norway        Europe
## 10 AAMODT, Kjetil Andre  2002 Salt Lake City United States North America
## # ... with 5,760 more rows
```

So, what we've done is merged the two datasets. Wherever a year was found in the `Year` column of the `Athlete` dataset, it added the city name from the `years` dataset. That's why we had to specify the argument `by = "Year"`. If you've ever used lookup tables in Excel, this is essentially what we're doing.

So why is it called `left_join`? There's a more technical explanation in §13.4.4 of R for data Science[3], but it essentially means we're adding information to the dataframe on the *left* (or, in this case, the data frame that's being piped into the function). If there's a year in the left dataframe (`athletes`) that is not found in the right one (`years`), you'll see an `NA` in the new `City`, `Country`, and `Continent` columns when they're combined. However, if there's a year in the `years` dataset that is not in the `athletes` dataframe, no harm done and it'll be excluded from the combined dataset.

You can think of the `years` dataset as a dictionary, and the `Year` column in the `athletes` dataset as the stuff we're looking up. The cool part is that rather than just retrieving one piece of

[3] http://r4ds.had.co.nz/relational-data.html#mutating-joins

information, if the `year` dataset had more columns, all of them would be appended on to the combined one.

Since there's a `left_join`, there is also a `right_join`, which does the logical opposite. We could get the exact same spreadsheet using `right_join` by moving some things around:

```
years %>%
    right_join(athletes, by = "Year")
```

```
## # A tibble: 5,770 x 10
##     Year           City     Country.x     Continent              Athlete
##    <int>          <chr>         <chr>         <chr>                <chr>
##  1  1980    Lake Placid United States North America       AAHLBERG, Mats
##  2  1984       Sarajevo    Yugoslavia        Europe       AAHLEN, Thomas
##  3  1980    Lake Placid United States North America     AALAND, Per Knut
##  4  2014          Sochi        Russia          Asia  AALTONEN, Juhamatti
##  5  1992    Albertville        France        Europe AAMODT, Kjetil Andre
##  6  1992    Albertville        France        Europe AAMODT, Kjetil Andre
##  7  1994    Lillehammer        Norway        Europe AAMODT, Kjetil Andre
##  8  1994    Lillehammer        Norway        Europe AAMODT, Kjetil Andre
##  9  1994    Lillehammer        Norway        Europe AAMODT, Kjetil Andre
## 10  2002 Salt Lake City United States North America AAMODT, Kjetil Andre
## # ... with 5,760 more rows, and 5 more variables: Country.y <chr>,
## #   Gender <chr>, Discipline <chr>, Event <chr>, Medal <chr>
```

The order of the columns are a little bit different, but the data is all there. In my own code, I don't use `right_join` as much, I guess because conceptually I like `left_join` better, but it's completely up to you.

So what happens if you do the opposite? Let's do a `left_join` with `years` on the left and `athletes` on the right.

```
years %>%
    left_join(athletes, by = "Year")
```

```
## # A tibble: 5,772 x 10
##     Year        City   Country.x Continent             Athlete Country.y
##    <int>       <chr>       <chr>     <chr>               <chr>     <chr>
##  1  2022     Beijing       China      Asia                <NA>      <NA>
##  2  2018 Pyeongchang South Korea      Asia                <NA>      <NA>
##  3  2014       Sochi      Russia      Asia AALTONEN, Juhamatti       FIN
##  4  2014       Sochi      Russia      Asia      ABBOTT, Jeremy       USA
##  5  2014       Sochi      Russia      Asia        ADAMS, Vicki       GBR
##  6  2014       Sochi      Russia      Asia      AGOSTA, Meghan       CAN
##  7  2014       Sochi      Russia      Asia       ALDER, Janine       SUI
##  8  2014       Sochi      Russia      Asia   ALFREDSSON, Daniel       SWE
##  9  2014       Sochi      Russia      Asia       ALTMANN, Livia       SUI
## 10  2014       Sochi      Russia      Asia     ALVAREZ, Eduardo       USA
## # ... with 5,762 more rows, and 4 more variables: Gender <chr>,
## #   Discipline <chr>, Event <chr>, Medal <chr>
```

Turns out, we get essentially the same thing! Just with a couple rows with `NA`s at the top. The reason for why they're very similar is because the years in both spreadsheets nearly perfectly match each other. They're not perfect (the `years` data has 2018 and 2022 in them, but we have no information about those years in the `athletes` data).

So what makes `left_join` and `right_join` different is how they treat missing data. To illustrate, let's create subsets that are more drastically different. First, let's say we only remember the cities for the last seven Winter Olympics:

```
last_7_cities <- years %>%
    filter(Year >= 1998)
last_7_cities

## # A tibble: 7 x 4
##    Year            City       Country     Continent
##   <int>           <chr>         <chr>         <chr>
## 1  2022         Beijing         China          Asia
## 2  2018     Pyeongchang   South Korea          Asia
## 3  2014           Sochi        Russia          Asia
## 4  2010       Vancouver        Canada North America
## 5  2006           Turin         Italy        Europe
## 6  2002 Salt Lake City United States North America
## 7  1998          Nagano         Japan          Asia
```

And then let's say we only had information on the women who medaled in individual figure skating in the nineties (1992, 1994, 1998):

```
skaters_90s <- athletes %>%
    filter(Year >= 1990, Year <= 2000,
           Discipline == "Figure skating", Event == "Individual",
           Gender == "Women")
skaters_90s

## # A tibble: 9 x 7
##             Athlete Country Gender     Discipline      Event  Year  Medal
##               <chr>   <chr>  <chr>          <chr>      <chr> <int>  <chr>
## 1     BAIUL, Oksana     UKR  Women Figure skating Individual  1994   Gold
## 2          CHEN, Lu     CHN  Women Figure skating Individual  1994 Bronze
## 3          CHEN, Lu     CHN  Women Figure skating Individual  1998 Bronze
## 4       ITO, Midori     JPN  Women Figure skating Individual  1992 Silver
## 5   KERRIGAN, Nancy     USA  Women Figure skating Individual  1992 Bronze
## 6   KERRIGAN, Nancy     USA  Women Figure skating Individual  1994 Silver
## 7    KWAN, Michelle     USA  Women Figure skating Individual  1998 Silver
## 8    LIPINSKI, Tara     USA  Women Figure skating Individual  1998   Gold
## 9 YAMAGUCHI, Kristi     USA  Women Figure skating Individual  1992   Gold
```

So we have two datasets that overlap partially, but not completely. Let's see what happens when we join them. Let's try to add the city name to the athletes' information using `left_join`, where the `skaters_90s` data comes first:

```
left_join(skaters_90s, last_7_cities, by = "Year")

## # A tibble: 9 x 10
##             Athlete Country.x Gender     Discipline      Event  Year
##               <chr>     <chr>  <chr>          <chr>      <chr> <int>
## 1     BAIUL, Oksana       UKR  Women Figure skating Individual  1994
## 2          CHEN, Lu       CHN  Women Figure skating Individual  1994
## 3          CHEN, Lu       CHN  Women Figure skating Individual  1998
## 4       ITO, Midori       JPN  Women Figure skating Individual  1992
## 5   KERRIGAN, Nancy       USA  Women Figure skating Individual  1992
## 6   KERRIGAN, Nancy       USA  Women Figure skating Individual  1994
## 7    KWAN, Michelle       USA  Women Figure skating Individual  1998
## 8    LIPINSKI, Tara       USA  Women Figure skating Individual  1998
## 9 YAMAGUCHI, Kristi       USA  Women Figure skating Individual  1992
## # ... with 4 more variables: Medal <chr>, City <chr>, Country.y <chr>,
## #   Continent <chr>
```

If you look through the `City` column, you'll notice that Nagano is the only one there, and the rest of the rows have NAs. So it kept all the skaters' information, and only added the city if it was in the "dictionary." Now let's try a `right_join`:

```
right_join(skaters_90s, last_7_cities, by = "Year")

## # A tibble: 9 x 10
##          Athlete Country.x Gender     Discipline      Event  Year  Medal
##            <chr>     <chr>  <chr>          <chr>      <chr> <int>  <chr>
## 1           <NA>      <NA>   <NA>           <NA>       <NA>  2022   <NA>
## 2           <NA>      <NA>   <NA>           <NA>       <NA>  2018   <NA>
## 3           <NA>      <NA>   <NA>           <NA>       <NA>  2014   <NA>
## 4           <NA>      <NA>   <NA>           <NA>       <NA>  2010   <NA>
## 5           <NA>      <NA>   <NA>           <NA>       <NA>  2006   <NA>
## 6           <NA>      <NA>   <NA>           <NA>       <NA>  2002   <NA>
## 7       CHEN, Lu       CHN  Women Figure skating Individual  1998 Bronze
## 8 KWAN, Michelle       USA  Women Figure skating Individual  1998 Silver
## 9 LIPINSKI, Tara       USA  Women Figure skating Individual  1998   Gold
## # ... with 3 more variables: City <chr>, Country.y <chr>, Continent <chr>
```

So now things look different. Because the `last_7_cities` dataframe was the "main" one, it kept all the data in it, specifically the year. That includes six years for which there was no athlete data. So, in the first six rows, we have `NA`s in all the columns except `Year`. But then, it has all the athletes for the 1998 year for which we have data because that was the only one that overlapped between the two.

The moral of the story is that if you've got very clean data where the info from one perfectly matches the other, as far as I can tell there's no substantial difference between `left_join` and `right_join` other than the order is different (which you can quickly change with `arrange`). However, if there is a mismatch, the two functions are very different and you have to think about what you want your result to be like.

2.2 `INNER_JOIN` AND `FULL_JOIN`

Okay, so what if you're aware of mismatches between your spreadsheets. Likely, there will be. And you know there're going to be some `NA`s. Is there a way to remove them?

Sure! That's what `inner_join` is for! Let's do exactly what we did above, but using `inner_join` instead of `right_join`.

```
inner_join(skaters_90s, last_7_cities, by = "Year")

## # A tibble: 3 x 10
##          Athlete Country.x Gender     Discipline      Event  Year  Medal
##            <chr>     <chr>  <chr>          <chr>      <chr> <int>  <chr>
## 1       CHEN, Lu       CHN  Women Figure skating Individual  1998 Bronze
## 2 KWAN, Michelle       USA  Women Figure skating Individual  1998 Silver
## 3 LIPINSKI, Tara       USA  Women Figure skating Individual  1998   Gold
## # ... with 3 more variables: City <chr>, Country.y <chr>, Continent <chr>
```

Here, all we get are the figure skaters only from 1998, because that's the only year that overlaps between the two datasets. If we do `inner_join` with the two datasets reversed, as far as I'm aware the result is the exact same but with a different order to the columns:

```
inner_join(last_7_cities, skaters_90s, by = "Year")

## # A tibble: 3 x 10
##    Year   City Country.x Continent        Athlete Country.y Gender
##   <int>  <chr>     <chr>     <chr>          <chr>     <chr>  <chr>
## 1  1998 Nagano     Japan      Asia       CHEN, Lu       CHN  Women
## 2  1998 Nagano     Japan      Asia KWAN, Michelle       USA  Women
## 3  1998 Nagano     Japan      Asia LIPINSKI, Tara       USA  Women
## # ... with 3 more variables: Discipline <chr>, Event <chr>, Medal <chr>
```

You might expect there to be an `outer_join` function which would keep just the athletes whose years are not in the `years` dataframe *and* the years from the `last_7_cities` data frame with no athletes. However, such a function does not exist as far as I'm aware, and honestly I can't think of a case where this might be useful.

However, there is `full_join`, which will keep everything from both.

```
full_join(skaters_90s, last_7_cities, by = "Year")

## # A tibble: 15 x 10
##              Athlete Country.x Gender     Discipline      Event  Year
##                <chr>     <chr>  <chr>          <chr>      <chr> <int>
##  1     BAIUL, Oksana       UKR  Women Figure skating Individual  1994
##  2          CHEN, Lu       CHN  Women Figure skating Individual  1994
##  3          CHEN, Lu       CHN  Women Figure skating Individual  1998
##  4       ITO, Midori       JPN  Women Figure skating Individual  1992
##  5   KERRIGAN, Nancy       USA  Women Figure skating Individual  1992
##  6   KERRIGAN, Nancy       USA  Women Figure skating Individual  1994
##  7    KWAN, Michelle       USA  Women Figure skating Individual  1998
##  8    LIPINSKI, Tara       USA  Women Figure skating Individual  1998
##  9 YAMAGUCHI, Kristi       USA  Women Figure skating Individual  1998
## 10              <NA>      <NA>   <NA>           <NA>        <NA>  2022
## 11              <NA>      <NA>   <NA>           <NA>        <NA>  2018
## 12              <NA>      <NA>   <NA>           <NA>        <NA>  2014
## 13              <NA>      <NA>   <NA>           <NA>        <NA>  2010
## 14              <NA>      <NA>   <NA>           <NA>        <NA>  2006
## 15              <NA>      <NA>   <NA>           <NA>        <NA>  2002
## # ... with 4 more variables: Medal <chr>, City <chr>, Country.y <chr>,
## #   Continent <chr>
```

This dataframe has 15 rows because it keeps all the athletes from `skaters_90s` (and puts `NA`s for those whose years are not in the `last_7_cities` data) *and* the remaining six years that have no representation in the `skaters_90s` dataframe. I have not needed to use this particular function, but you might find it useful.

2.3 Using joins to filter data

There are two more join functions that are pretty slick. They take a sec to wrap your mind around, but once you get them they're really nice to be aware of.

The first is `semi_join`. This filters the data such that only rows in `skaters_90s` that have a match in `last_7_cities` are kept.

```
semi_join(skaters_90s, last_7_cities, by = "Year")
```

```
## # A tibble: 3 x 7
##          Athlete Country Gender     Discipline      Event  Year  Medal
##            <chr>   <chr>  <chr>          <chr>      <chr> <int>  <chr>
## 1       CHEN, Lu     CHN  Women Figure skating Individual  1998 Bronze
## 2 KWAN, Michelle     USA  Women Figure skating Individual  1998 Silver
## 3 LIPINSKI, Tara     USA  Women Figure skating Individual  1998   Gold
```

This is slightly different from `inner_join` because the `City`, `Country`, and `Continent` columns are not in the resulting dataframe. So it really is just a filter. Another way to do this is with `%in%`, if that makes more sense conceptually.

```
skaters_90s %>%
    filter(Year %in% last_7_cities$Year)
```

```
## # A tibble: 3 x 7
##          Athlete Country Gender     Discipline      Event  Year  Medal
##            <chr>   <chr>  <chr>          <chr>      <chr> <int>  <chr>
## 1       CHEN, Lu     CHN  Women Figure skating Individual  1998 Bronze
## 2 KWAN, Michelle     USA  Women Figure skating Individual  1998 Silver
## 3 LIPINSKI, Tara     USA  Women Figure skating Individual  1998   Gold
```

The opposite of this is `anti_join`. This returns all the rows of the first dataframe that do *not* have a match in the second. In other words, this shows us the data in the `skaters_90s` dataframe from 1994 and 1992 because `last_7_cities` does not have data for 1994 and 1992.

```
anti_join(skaters_90s, last_7_cities, by = "Year")

## # A tibble: 6 x 7
##             Athlete Country Gender      Discipline      Event  Year  Medal
##               <chr>   <chr>  <chr>           <chr>      <chr> <int>  <chr>
## 1     BAIUL, Oksana     UKR  Women Figure skating Individual  1994   Gold
## 2          CHEN, Lu     CHN  Women Figure skating Individual  1994 Bronze
## 3       ITO, Midori     JPN  Women Figure skating Individual  1992 Silver
## 4   KERRIGAN, Nancy     USA  Women Figure skating Individual  1992 Bronze
## 5   KERRIGAN, Nancy     USA  Women Figure skating Individual  1994 Silver
## 6 YAMAGUCHI, Kristi     USA  Women Figure skating Individual  1992   Gold
```

This `anti_join` is *super* handy. There have been many times where I've needed to compare two similar datasets that were each pretty big, but I knew there were a few discrepancies. With `anti_join` I could isolate those with just a single line of code whereas some other way would have been a lot more work.

3 Summarizing

Working with large datasets is great, but sometimes we want to summarize what's going on. In this section, we look at the `summarize` function, especially in conjunction with `group_by`, which will allow us to create some new summarized versions of your data.

Let's start with our `athletes` dataframe. If we just peek at the first twelve lines, we can already see that some athletes compete in multiple years and/or across different events.

```
athletes

## # A tibble: 5,770 x 7
##                 Athlete Country Gender           Discipline
##                   <chr>   <chr>  <chr>                <chr>
##  1       AAHLBERG, Mats     SWE    Men          Ice Hockey
##  2       AAHLEN, Thomas     SWE    Men          Ice Hockey
##  3     AALAND, Per Knut     NOR    Men Cross Country Skiing
##  4  AALTONEN, Juhamatti     FIN    Men          Ice Hockey
##  5 AAMODT, Kjetil Andre     NOR    Men       Alpine Skiing
##  6 AAMODT, Kjetil Andre     NOR    Men       Alpine Skiing
##  7 AAMODT, Kjetil Andre     NOR    Men       Alpine Skiing
##  8 AAMODT, Kjetil Andre     NOR    Men       Alpine Skiing
##  9 AAMODT, Kjetil Andre     NOR    Men       Alpine Skiing
## 10 AAMODT, Kjetil Andre     NOR    Men       Alpine Skiing
## # ... with 5,760 more rows, and 3 more variables: Event <chr>, Year <int>,
## #   Medal <chr>
```

So for example, Kyetil Andre Aamodt of Norway medaled eight times in the Giant Slalom, Super-G, Alpine Combined, and Downhill in 1992, 1994, 2002, and 2006. That's a lot. Are there people that did more than that? And if so, how can we tell?

This is exactly the sort of situation that the `summarise` is perfect for. Since we want to find the information *per person*, we need to group the data by each person. We can do that first using the `group_by` function. By itself this doesn't do much, but it changes how R treats it under the hood. But if we pipe this into `summarise`, we tell R to perform functions *per group*.

```
athletes %>%
    group_by(Athlete) %>%
    summarize()

## # A tibble: 3,761 x 1
##                Athlete
##                  <chr>
##   1      AAHLBERG, Mats
##   2      AAHLEN, Thomas
##   3    AALAND, Per Knut
##   4  AALTONEN, Juhamatti
##   5 AAMODT, Kjetil Andre
##   6          AAS, Roald
##   7       AASLIN, Peter
##   8      ABBOTT, Jeremy
##   9        ABE, Masashi
## 10  ABEL, Clarence John
## # ... with 3,751 more rows
```

Without any arguments, `summarize` simply lists the groups, meaning a list of all the athletes. But we can add columns to this dataframe, similar to how `mutate` works. So if we want to simply see how many rows in the dataframe belong to each of these groups, we can add a new arbitrarily named column called `num_of_medals` and use the `n()` function to count how many there are.

```
athletes %>%
    group_by(Athlete) %>%
    summarize(num_of_medals = n())

## # A tibble: 3,761 x 2
##                Athlete num_of_medals
##                  <chr>         <int>
##   1      AAHLBERG, Mats             1
##   2      AAHLEN, Thomas             1
##   3    AALAND, Per Knut             1
##   4  AALTONEN, Juhamatti            1
##   5 AAMODT, Kjetil Andre            8
##   6          AAS, Roald             2
##   7       AASLIN, Peter             1
## 8       ABBOTT, Jeremy            1
## 9         ABE, Masashi            1
## 10  ABEL, Clarence John          1
## # ... with 3,751 more rows
```

Okay, so that's already interesting. But I don't want to have to sift through 3,700 rows to find out which has the most. Let's add the `arrange` function to the end to put them in order of `num_of_medals`. Since `arrange` normally sorts numbers from smallest to largest, and we actually want to go from largest to smallest, we'll do a reverse sort by simply adding a negative sign (it's just a hyphen `-`) before the name.

```
athletes %>%
    group_by(Athlete) %>%
    summarize(num_of_medals = n()) %>%
    arrange(-num_of_medals)
```

```
## # A tibble: 3,761 x 2
##                    Athlete num_of_medals
##                      <chr>         <int>
##  1 BJOERNDALEN, Ole Einar            13
##  2          DAEHLIE, Björn            12
##  3     BELMONDO, Stefania            10
##  4       SMETANINA, Raisa            10
##  5        BJOERGEN, Marit             9
##  6            DISL, Uschi             9
##  7        EGOROVA, Ljubov             9
##  8       JERNBERG, Sixten             9
##  9     PECHSTEIN, Claudia             9
## 10    AAMODT, Kjetil Andre             8
## # ... with 3,751 more rows
```

So Mr. Aamodt is indeed exceptional. Of all 3,700 medalists, only 16 have gotten 8 or more medals in their career. But this sorting makes it clear that the person with the most medals is one Ole Einar Bjoerndalen with a whopping 13 career medals.

We can use `summarize` to actually do more. We can look at specific columns within the now-grouped `athletes` dataframe to get how many golds, silvers, and bronzes each person has won. Since we're referring to specific column, we can't use `n` anymore, but we can just use the regular `sum` function to add up how many rows in the `Medal` column are `"Gold"`, etc.

```
athletes %>%
    group_by(Athlete) %>%
    summarize(golds = sum(Medal == "Gold"),
              silvers = sum(Medal == "Silver"),
              bronzes = sum(Medal == "Bronze"),
              num_of_medals = n()) %>%
    arrange(-num_of_medals)
```

```
## # A tibble: 3,761 x 5
##                   Athlete golds silvers bronzes num_of_medals
##                     <chr> <int>   <int>   <int>         <int>
##  1 BJOERNDALEN, Ole Einar     8       4       1            13
##  2          DAEHLIE, Björn     8       4       0            12
##  3      BELMONDO, Stefania     2       3       5            10
##  4        SMETANINA, Raisa     4       5       1            10
##  5          BJOERGEN, Marit     6       2       1             9
##  6              DISL, Uschi     2       4       3             9
##  7          EGOROVA, Ljubov     6       3       0             9
##  8        JERNBERG, Sixten     4       3       2             9
##  9      PECHSTEIN, Claudia     5       2       2             9
## 10    AAMODT, Kjetil Andre     4       2       2             8
## # ... with 3,751 more rows
```

Since this is a regular dataframe, we can then do other modifications to it. For example, the two people that have each won 10 medals have slightly different distributions of what medals they are. Raisa Smetanina has gotten 4 golds, 5 silvers, and 1 bronze while Stefania Belmondo has gotten 2 golds, 3 silvers, and 5 bronzes. Objectively, the golds are harder to get, so raw medal count might not be the most accurate picture of these athletes. Perhaps it might be better to give extra weight to golds and less to bronzes. While the topic of exactly how to weight each medal is a controversial and significantly alters the rankings[4], we'll stick with a simple one: Gold = 3, Silver = 2, and Bronze = 1. So let's use `mutate` on this new dataset to create a weighted score.

```
athletes %>%
    group_by(Athlete) %>%
    summarize(golds = sum(Medal == "Gold"),
              silvers = sum(Medal == "Silver"),
              bronzes = sum(Medal == "Bronze"),
              num_of_medals = n()) %>%
    mutate(medals_weighted = golds * 3 + silvers * 2 + bronzes) %>%
    arrange(-medals_weighted)
```

```
## # A tibble: 3,761 x 6
##                   Athlete golds silvers bronzes num_of_medals
##                     <chr> <int>   <int>   <int>         <int>
##  1 BJOERNDALEN, Ole Einar     8       4       1            13
##  2          DAEHLIE, Björn     8       4       0            12
##  3          EGOROVA, Ljubov     6       3       0             9
##  4          BJOERGEN, Marit     6       2       1             9
##  5        SMETANINA, Raisa     4       5       1            10
##  6      PECHSTEIN, Claudia     5       2       2             9
##  7        JERNBERG, Sixten     4       3       2             9
##  8            GROSS, Ricco     4       3       1             8
```

[4] https://www.nytimes.com/interactive/2018/02/14/upshot/which-country-leads-in-the-olympic-medal-count.html

Yeah, so this changes the order a lot. The top two are the same, but now Ljubov Egorva, who "only" has nine medals is ranked higher than both of the athletes with 10 because they have won more golds. Raisa Metanina is fifth according to this system, but our friend Stefania Belmondo is now 19th.

Regardless of how you feel that the medals should be weighted, this shows that you can work with this summarized table just as you could with any other dataset.

3.1.1 *Your turn!*

The Challenge

1. Instead of looking at how many medals there were per person, try seeing how many there were per country.
2. Modify how the medals are weighted to Gold = 5, Silver = 3, and Bronze = 1 to see what kinds of differences that makes in how the countires are ranked.

The Solution

To look at each country instead of each person, you would just use the `group_by` function on the `Country` column instead of the `Athlete` column.

```
athletes %>%
    group_by(Country) %>%
    summarize(num_of_medals = n()) %>%
    arrange(-num_of_medals)
```

```
## # A tibble: 45 x 2
##    Country num_of_medals
##      <chr>         <int>
##  1     USA           653
##  2     CAN           625
##  3     NOR           457
##  4     URS           440
##  5     FIN           434
##  6     SWE           433
##  7     GER           360
##  8     SUI           285
##  9     AUT           280
## 10     RUS           263
## # ... with 35 more rows
```

We can see how this order changes when we weight medals differently:

```
athletes %>%
    group_by(Country) %>%
    summarize(golds = sum(Medal == "Gold"),
              silvers = sum(Medal == "Silver"),
              bronzes = sum(Medal == "Bronze"),
              num_of_medals = n()) %>%
    mutate(medals_weighted = golds * 5 + silvers * 3 + bronzes) %>%
    arrange(-medals_weighted)

## # A tibble: 45 x 6
##    Country golds silvers bronzes num_of_medals medals_weighted
##      <chr> <int>   <int>   <int>         <int>           <dbl>
##  1     CAN   315     203     107           625            2291
##  2     USA   167     319     167           653            1959
##  3     URS   250      97      93           440            1634
##  4     NOR   159     171     127           457            1435
##  5     SWE   127     129     177           433            1199
##  6     GER   137     126      97           360            1160
##  7     FIN    66     147     221           434             992
##  8     RUS    94      90      79           263             819
##  9     AUT    79      98     103           280             792
## 10     SUI    76      77     132           285             743
## # ... with 35 more rows
```

So even though the USA leads in raw medal count, Canada has won almost twice as many golds. Also, if you were to combine the Soviet Union URS and Russia RUS, they might take the lead.

3.2 Combining summaries with joins

Something that's slightly frustrating about the output of `summarize` is that you can't see other metadata about the athletes. I've been referring to the athletes by name, but I don't know their gender or their country from the summary table alone. If you go under the assumption that their gender or country was consistent across all their medals, you could get around this by using the `first` function within `summarize`, which simply takes the first gender or country it sees with in each group.

```
athletes %>%
    group_by(Athlete) %>%
    summarize(num_of_medals = n(),
              gender = first(Gender),
              country = first(Country)) %>%
    arrange(-num_of_medals)

## # A tibble: 3,761 x 4
##                   Athlete num_of_medals gender country
##                     <chr>         <int>  <chr>   <chr>
##  1 BJOERNDALEN, Ole Einar            13    Men     NOR
##  2         DAEHLIE, Björn            12    Men     NOR
##  3     BELMONDO, Stefania            10  Women     ITA
##  4       SMETANINA, Raisa            10  Women     URS
##  5        BJOERGEN, Marit             9  Women     NOR
##  6            DISL, Uschi             9  Women     GER
##  7        EGOROVA, Ljubov             9  Women     EUN
##  8       JERNBERG, Sixten             9    Men     SWE
```

This is a workaround, but it's potentially buggy. With this data, I think you can be reasonably confident that there are no gender or country changes, but I don't like going off of that assumption, regardless of the data.

This is where the `*_join` functions really start to shine. Because these summary tables are regular dataframes, we can save them and use them to filter the main dataframe. Let's say we want to look at only the athletes with 10 or more medals. Let's create the summarized table, filter it to just show those with 10 or more, and the save it into a new object called `ten_plus`.

```
ten_plus <- athletes %>%
    group_by(Athlete) %>%
    summarize(num_of_medals = n()) %>%
    filter(num_of_medals >= 10)
```

We can then use this in an `inner_join` with the main `athletes` dataframe.

```
inner_join(athletes, ten_plus, by = "Athlete")

## # A tibble: 45 x 8
##              Athlete Country Gender          Discipline
##                <chr>   <chr>  <chr>               <chr>
##  1 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  2 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  3 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  4 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  5 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  6 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  7 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  8 BELMONDO, Stefania     ITA  Women Cross Country Skiing
##  9 BELMONDO, Stefania     ITA  Women Cross Country Skiing
## 10 BELMONDO, Stefania     ITA  Women Cross Country Skiing
## # ... with 35 more rows, and 4 more variables: Event <chr>, Year <int>,
## #   Medal <chr>, num_of_medals <int>
```

Now we have just the 45 medals that these four athletes have won, showing us what events they competed in. And because we used `inner_join`, we now have a new column showing the total number of medals that person has won.

3.2.1 Your Turn!

The challenge

1. The athletes that won 10 or more medals did so all within one discipline. As it turns out, there have been just 14 athletes that have medaled in two *different* disciplines. Use `summarize` to find out who they were, and create a new spreadsheet showing just their data. Hint, you may need to use `length(unique( ))`.

The solution

Here's how I did this. Grouping the `athletes` data by `Athlete`, I create a column arbitrarily named `num_disciplines` that is the length of the list of unique disciplines each person as competed in. I then filter out the ones with just one and save it into an object called `multis`. Then, I use `semi_join` with the original `athletes` dataframe to show me the 40 medals these 14 people have won.

```
multis <- athletes %>%
    group_by(Athlete) %>%
    summarize(num_disciplines = length(unique(Discipline))) %>%
    filter(num_disciplines > 1)
multis

## # A tibble: 14 x 2
##                    Athlete num_disciplines
##                      <chr>           <int>
##  1         BRODAHL, Sverre               2
##  2      ERDMANN, Susi-Lisa               2
##  3             FLAIM, Eric               2
##  4 GROTTUMSBRAATEN, Johan                2
##  5         HAGEN, Oddbjorn               2
##  6            HASU, Heikki               2
##  7          HAUG, Thorleif               2
##  8        HEATON, Jennison               2
##  9            HEATON, John               2
## 10       HOFFSBAKKEN, Olaf               2
## 11            KÄLIN, Alois               2
## 12        REZTSOVA, Anfisa               2
## 13      STRÖMSTAD, Thoralf               2
## 14  WEISSENSTEINER, Gerda                2

athletes %>%
    semi_join(multis, by = "Athlete")

## # A tibble: 40 x 7
##                    Athlete Country Gender                Discipline
##                      <chr>   <chr>  <chr>                     <chr>
##  1         BRODAHL, Sverre     NOR    Men      Cross Country Skiing
##  2         BRODAHL, Sverre     NOR    Men           Nordic Combined
##  3      ERDMANN, Susi-Lisa     GER  Women                      Luge
##  4      ERDMANN, Susi-Lisa     GER  Women                      Luge
##  5      ERDMANN, Susi-Lisa     GER  Women                 Bobsleigh
##  6             FLAIM, Eric     USA    Men             Speed skating
##  7             FLAIM, Eric     USA    Men Short Track Speed Skating
##  8 GROTTUMSBRAATEN, Johan      NOR    Men      Cross Country Skiing
##  9 GROTTUMSBRAATEN, Johan      NOR    Men      Cross Country Skiing
## 10 GROTTUMSBRAATEN, Johan      NOR    Men           Nordic Combined
## # ... with 30 more rows, and 3 more variables: Event <chr>, Year <int>,
## #   Medal <chr>
```

4 Reshaping your data

The final topic in this workshop has to do with reshaping your data. To explain what this section will be about, let's look at the women who medaled in figure skating in the nineties again. I'll simplify the data a little bit by showing just three columns.

```
skaters_90s %>%
    select(Athlete, Year, Medal)
```

```
## # A tibble: 9 x 3
##              Athlete  Year  Medal
##                <chr> <int>  <chr>
## 1     BAIUL, Oksana  1994   Gold
## 2          CHEN, Lu  1994 Bronze
## 3          CHEN, Lu  1998 Bronze
## 4       ITO, Midori  1992 Silver
## 5   KERRIGAN, Nancy  1992 Bronze
## 6   KERRIGAN, Nancy  1994 Silver
## 7    KWAN, Michelle  1998 Silver
## 8    LIPINSKI, Tara  1998   Gold
## 9 YAMAGUCHI, Kristi  1992   Gold
```

Now compare that to this dataset.

```
## # A tibble: 3 x 4
##    Year              Gold          Silver          Bronze
## * <int>             <chr>           <chr>           <chr>
## 1  1992 YAMAGUCHI, Kristi     ITO, Midori KERRIGAN, Nancy
## 2  1994     BAIUL, Oksana KERRIGAN, Nancy        CHEN, Lu
## 3  1998    LIPINSKI, Tara  KWAN, Michelle        CHEN, Lu
```

Is this the same data? Sure. All the information that can be found in one is also found in the others. But what is different? In the top one, we have nine rows, one for each year and medal, and within each row we have three columns that contain information about the person's name, what year they competed, and what medal they got. In the second one we have just three rows, one for each *year*, and there are columns indicating who got gold, silver, and bronze.

The data is identical, but the spreadsheets are organized differently. The first spreadsheet is known as a "tall" format, because, well, there are 9 rows so it's very tall. The second is known as a "wide" format because there are more columns making it wider. Why would you need to use one over the other? There are lots of reasons, but the main one for me is that certain visualizations and statistical models require the data to be in a certain shape. Unfortunately, it's out of the scope of this workshops to show these applications, but it's important to have these tricks up your sleeve for when you need them.

4.1 Going from tall to wide

So how did I make these changes? The key here is the `spread` function, which transforms a spreadsheet from tall to wide. This takes just two arguments. The first is the `key`, which is the column that contains all the values that you want to spread out into each of their own columns. So in the original, the `Medal` column contains just three unique values `"Gold"`, `"Silver"`, and `"Bronze"`. We want each of these to be its *own* column. The second argument is the `value`, which is the column that contains the information you want to populate these newly created cells with. Since we want the names to fill in those cells, we select the `Athlete` column. So, the code for the second spreadsheet is this.

```
skaters_90s %>%
    select(Athlete, Year, Medal) %>%
    spread(Medal, Athlete) %>%
    select(Year, Gold, Silver, Bronze)

## # A tibble: 3 x 4
##    Year             Gold          Silver         Bronze
## * <int>            <chr>           <chr>          <chr>
## 1  1992 YAMAGUCHI, Kristi     ITO, Midori KERRIGAN, Nancy
## 2  1994     BAIUL, Oksana KERRIGAN, Nancy        CHEN, Lu
## 3  1998    LIPINSKI, Tara  KWAN, Michelle        CHEN, Lu
```

I've included some extra code in there about first selecting just the columns we need and then reordering the columns, but the main focus is the `spread` function.

4.1.1 *Your Turn!*

The challenge

Create a different version of the same data such that each medal is on its own row, and you have columns for each year.

The solution

Here's how I would have done it.

```
skaters_90s %>%
    select(Athlete, Year, Medal) %>%
    spread(Year, Athlete)

## # A tibble: 3 x 4
##    Medal            `1992`          `1994`         `1998`
## *  <chr>             <chr>           <chr>          <chr>
## 1 Bronze   KERRIGAN, Nancy        CHEN, Lu       CHEN, Lu
## 2   Gold YAMAGUCHI, Kristi   BAIUL, Oksana LIPINSKI, Tara
## 3 Silver       ITO, Midori KERRIGAN, Nancy KWAN, Michelle
```

4.2 Spreading with more columns

This simple example was straightforward enough, but what happens if we want to keep all the columns? Here we get a slightly different picture. What does each row represent?

```
skaters_90s %>%
    spread(Medal, Athlete)

## # A tibble: 7 x 8
##   Country Gender     Discipline      Event  Year          Bronze
## *   <chr>  <chr>          <chr>      <chr> <int>           <chr>
## 1     CHN  Women Figure skating Individual  1994        CHEN, Lu
## 2     CHN  Women Figure skating Individual  1998        CHEN, Lu
## 3     JPN  Women Figure skating Individual  1992            <NA>
## 4     UKR  Women Figure skating Individual  1994            <NA>
## 5     USA  Women Figure skating Individual  1992 KERRIGAN, Nancy
## 6     USA  Women Figure skating Individual  1994            <NA>
## 7     USA  Women Figure skating Individual  1998            <NA>
## # ... with 2 more variables: Gold <chr>, Silver <chr>
```

Here, each row represents a unique combination of the countries and years. Since Japan only medaled one year here, it only has one row. Since USA medaled in three different years, it has three rows. There are lots of `NA`s in the data because not all counties got all medals every year. This is not a very useful representation of the data.

The reason why this isn't as useful for the one we saw earlier is because the `Country` column and what was once the `Athelete` column are linked. That is, the the country is simply metadata about the athlete. Since we've removed the `Athlete` column by spreading it out over many columns, the `Country` column is sort of stuck following along how it can. The solution is to make sure that the variable you spread has no other unique metadata in the same spreadsheet. This can be done with `select`, as we've seen before.

But why were `Gender`, `Discipline`, and `Event` okay? Well, for one, they're all identical for this small dataset. But the real reason is because they are all linked to the year, rather than the individual. So the key to a successful `spread` is to think about what you want each row to represent. If there is additional metadata about that thing, it's okay to keep the column. But if there's additional metadata about the variable you want to spread out, remove it before `spreading`.

Let's look at another example. Let's look at just the men who medaled in any of the six events within the snowboarding discipline in 2014.

```
snowboarders <- athletes %>%
    filter(Year == 2014, Discipline == "Snowboard", Gender == "Men")
snowboarders

## # A tibble: 15 x 7
##                 Athlete Country Gender Discipline           Event  Year
##                   <chr>   <chr>  <chr>      <chr>           <chr> <int>
##  1        DEIBOLD, Alex     USA     Men  Snowboard Snowboard Cross  2014
##  2     GALMARINI, Nevin     SUI     Men  Snowboard Giant Parall.S.  2014
##  3        HIRANO, Ayumu     JPN     Men  Snowboard        Half-Pipe  2014
##  4        HIRAOKA, Taku     JPN     Men  Snowboard        Half-Pipe  2014
##  5      KARL, Benjamin     AUT     Men  Snowboard Parallel Slalom  2014
##  6           KOSIR, Zan     SLO     Men  Snowboard Giant Parall.S.  2014
##  7           KOSIR, Zan     SLO     Men  Snowboard Parallel Slalom  2014
##  8     KOTSENBURG, Sage     USA     Men  Snowboard       Slopestyle  2014
##  9       MCMORRIS, Mark     CAN     Men  Snowboard       Slopestyle  2014
## 10     OLYUNIN, Nikolay     RUS     Men  Snowboard Snowboard Cross  2014
## 11 PODLADTCHIKOV, Iouri     SUI     Men  Snowboard        Half-Pipe  2014
## 12      SANDBECH, Staale    NOR     Men  Snowboard       Slopestyle  2014
## 13     VAULTIER, Pierre     FRA     Men  Snowboard Snowboard Cross  2014
## 14            WILD, Vic     RUS     Men  Snowboard Giant Parall.S.  2014
## 15            WILD, Vic     RUS     Men  Snowboard Parallel Slalom  2014
## # ... with 1 more variables: Medal <chr>
```

Let's say we want to make a different version of this spreadsheet with one row per medal, and one column per discipline. How would we do that?

Well, because `Event` is the column that we want to spread across multiple columns, we need to use that as the first argument in the `spread` function. Let's say we're just interested in the countries that took the medal and not names of the individuals themselves. We'll then put the `Country` column as the second argument to `spread`. What is the result?

```
snowboarders %>%
    spread(Event, Country)

## # A tibble: 14 x 10
##                 Athlete Gender Discipline  Year  Medal `Giant Parall.S.`
##  *                <chr>  <chr>      <chr> <int>  <chr>             <chr>
##  1        DEIBOLD, Alex    Men  Snowboard  2014 Bronze              <NA>
##  2     GALMARINI, Nevin    Men  Snowboard  2014 Silver               SUI
##  3        HIRANO, Ayumu    Men  Snowboard  2014 Silver              <NA>
##  4        HIRAOKA, Taku    Men  Snowboard  2014 Bronze              <NA>
##  5       KARL, Benjamin    Men  Snowboard  2014 Bronze              <NA>
##  6           KOSIR, Zan    Men  Snowboard  2014 Bronze               SLO
##  7           KOSIR, Zan    Men  Snowboard  2014 Silver              <NA>
##  8     KOTSENBURG, Sage    Men  Snowboard  2014   Gold              <NA>
##  9       MCMORRIS, Mark    Men  Snowboard  2014 Bronze              <NA>
## 10     OLYUNIN, Nikolay    Men  Snowboard  2014 Silver              <NA>
## 11 PODLADTCHIKOV, Iouri    Men  Snowboard  2014   Gold              <NA>
## 12     SANDBECH, Staale    Men  Snowboard  2014 Silver              <NA>
## 13     VAULTIER, Pierre    Men  Snowboard  2014   Gold              <NA>
## 14            WILD, Vic    Men  Snowboard  2014   Gold               RUS
## # ... with 4 more variables: `Half-Pipe` <chr>, `Parallel Slalom` <chr>,
## #   Slopestyle <chr>, `Snowboard Cross` <chr>
```

Hmm. Not quite what we wanted. What does each row show now? Similar to before, we have one row for every unique combination of medals and athlete. Again, the countries are linked to the athletes, so spreading one out into various columns doesn't actually consolidate anything. We can fix this by removing the `Athlete` column first:

```
snowboarders %>%
    select(-Athlete) %>%
    spread(Event, Country)
```

```
## # A tibble: 3 x 9
##   Gender Discipline  Year  Medal `Giant Parall.S.` `Half-Pipe`
## *  <chr>      <chr> <int>  <chr>             <chr>       <chr>
## 1    Men  Snowboard  2014 Bronze               SLO         JPN
## 2    Men  Snowboard  2014   Gold               RUS         SUI
## 3    Men  Snowboard  2014 Silver               SUI         JPN
## # ... with 3 more variables: `Parallel Slalom` <chr>, Slopestyle <chr>,
## #   `Snowboard Cross` <chr>
```

There we go. That's what we wanted.

So this is is a nice way to reshape your data.

4.2.1 *Your Turn!*

The Challenge

Try reshaping the `snowboarders` data so that each discipline is on its own row still, but the athletes names rather than their countries are displayed.

The Solution

```
snowboarders %>%
    select(-Country) %>%
    spread(Event, Athlete)
```

```
## # A tibble: 3 x 9
##   Gender Discipline  Year  Medal `Giant Parall.S.`          `Half-Pipe`
## *  <chr>      <chr> <int>  <chr>             <chr>                <chr>
## 1    Men  Snowboard  2014 Bronze        KOSIR, Zan        HIRAOKA, Taku
## 2    Men  Snowboard  2014   Gold         WILD, Vic PODLADTCHIKOV, Iouri
## 3    Men  Snowboard  2014 Silver  GALMARINI, Nevin         HIRANO, Ayumu
## # ... with 3 more variables: `Parallel Slalom` <chr>, Slopestyle <chr>,
## #   `Snowboard Cross` <chr>
```

4.3 Going fom wide to tall

The logical opposite of `spread` is the `gather` function, which takes a wide dataframe and converts it into a tall format. This is really useful especially if you've inherited some data that was in a format that wasn't ideal. I've noticed that it's easier for me to work in Excel with very wide data, but it's easier in R to work with tall data. So I often find myself having to convert Excel spreadsheets from their original wide format to a tall format.

So let's create some very wide data, for the sake of illustration, let's look at a wide version of the women's cross country skiing medalists since the year 2000. First, I apply the appropriate filter. But since there were some team events with multiple people winning medals, each person was on its own row. I'm more interested in the countries than the athletes themselves, so I remove their names, and then use `unique()` to remove any duplicate rows, thus reducing it down to one row per county, per year, per medal. I then apply the `spread` function and put one event in each column with the name of the country in the cells.

```
cc_skiing_wide <- athletes %>%
    filter(Year >= 2000, Discipline == "Cross Country Skiing", Gender == "Wom
en") %>%
    select(-Athlete) %>%
    unique() %>%
    spread(Event, Country)
cc_skiing_wide

## # A tibble: 12 x 13
##    Gender           Discipline  Year  Medal `10KM` `15KM Mass Start`
##  *  <chr>                <chr> <int>  <chr>  <chr>              <chr>
##  1  Women Cross Country Skiing  2002 Bronze    ITA                RUS
##  2  Women Cross Country Skiing  2002   Gold    NOR                ITA
##  3  Women Cross Country Skiing  2002 Silver    RUS                CZE
##  4  Women Cross Country Skiing  2006 Bronze    NOR               <NA>
##  5  Women Cross Country Skiing  2006   Gold    EST               <NA>
##  6  Women Cross Country Skiing  2006 Silver    NOR               <NA>
##  7  Women Cross Country Skiing  2010 Bronze    NOR               <NA>
##  8  Women Cross Country Skiing  2010   Gold    SWE               <NA>
##  9  Women Cross Country Skiing  2010 Silver    EST               <NA>
## 10  Women Cross Country Skiing  2014 Bronze    NOR               <NA>
## 11  Women Cross Country Skiing  2014   Gold    POL               <NA>
## 12  Women Cross Country Skiing  2014 Silver    SWE               <NA>
## # ... with 7 more variables: `30KM` <chr>, `4X5KM Relay` <chr>, `5Km
## #   Pursuit` <chr>, `Combined 7.5 + 7.5Km Mass Start` <chr>,
## #   `Combined7.5+7.5` <chr>, `Sprint 1.5KM` <chr>, `Team Sprint` <chr>
```

So let's say you've been asked to analyze this data, and you were given this spreadsheet as is. It's kind of difficult to work with because there are so many cross country skiing events spread across 9 columns, with lots of `NA`s. The way we can transform this back into a tall format is with the `gather` function.

To think about this conceptually, what `gather` is going to do is combine many columns into just two. One of these columns is going to have the names of the columns you want to gather. In this case, we have 22 columns with various events in them, so we might want to gather them up into a new column called "Events" or something. The other new column you'll need to create is going to have all the values that are currently in the cells of the columns you want to gather together. In this case, we have all the country codes, so we might want to gather them up into a new column called "Country".

So the first two arguments of `gather` are the new arbitrarily-named columns that you're going to create. The first one is the one with the column names and the second one is the one with the cells. Just to show that they are indeed arbitrary, I'm going to use ridiculous names.

```
cc_skiing_wide %>%
    gather(this_column_has_the_events, this_column_has_the_countries)
```

```
## # A tibble: 156 x 2
##    this_column_has_the_events this_column_has_the_countries
##                         <chr>                         <chr>
##  1                     Gender                         Women
##  2                     Gender                         Women
##  3                     Gender                         Women
##  4                     Gender                         Women
##  5                     Gender                         Women
##  6                     Gender                         Women
##  7                     Gender                         Women
##  8                     Gender                         Women
##  9                     Gender                         Women
## 10                     Gender                         Women
## # ... with 146 more rows
```

Wait. Oops. What happened? R didn't know that we only intended to gather just the 9 event columns and just assumed we wanted to gather *everything* into just two columns. So the first column, which was supposed to just have all the column names, did do that, but it also included `Gender`, `Discipline`, `Year`, and `Medal`. If you scroll down to about the 49th entry, you'll see that finally the event columns are there. But this is not what we wanted. We need to tell `gather` that we only wanted to combine certain columns and to leave the rest alone.

The way to do this is by using additional arguments in `gather`. They are, quite simply, the name of the columns you want to gather. This is very similar to the `select` function, so you can type all the columns individually, or, in this case, type the columns you want to leave out, each prefixed with a minus sign/hyphen `-`.

```
cc_skiing_wide %>%
    gather(this_column_has_the_events, this_column_has_the_countries, -Gender
, -Discipline, -Year, -Medal)
```

```
## # A tibble: 108 x 6
##    Gender            Discipline  Year  Medal this_column_has_the_events
##     <chr>                 <chr> <int>  <chr>                      <chr>
##  1  Women Cross Country Skiing  2002 Bronze                       10KM
##  2  Women Cross Country Skiing  2002   Gold                       10KM
##  3  Women Cross Country Skiing  2002 Silver                       10KM
##  4  Women Cross Country Skiing  2006 Bronze                       10KM
##  5  Women Cross Country Skiing  2006   Gold                       10KM
##  6  Women Cross Country Skiing  2006 Silver                       10KM
##  7  Women Cross Country Skiing  2010 Bronze                       10KM
##  8  Women Cross Country Skiing  2010   Gold                       10KM
##  9  Women Cross Country Skiing  2010 Silver                       10KM
## 10  Women Cross Country Skiing  2014 Bronze                       10KM
## # ... with 98 more rows, and 1 more variables:
## #   this_column_has_the_countries <chr>
```

There we go. So by specifying which columns you want to be gathered either by naming them specifically or listing the ones to ignore, we can convert our data back to its original form. In this case though, there's one row per cell of the wide version of the spreadsheet, regardless of whether there was data there or not. So if you scroll through the `this_column_has_the_countries` column, there are lots of `NA`s because some events were not done in some years. We can then just apply a filter to select only those rows that are not `NA` using `!is.na()`. (I'm also going to switch back to sensible column names.)

```
cc_skiing_wide %>%
    gather(Event, Country, -Gender, -Discipline, -Year, -Medal) %>%
    filter(!is.na(Country))
```

```
## # A tibble: 72 x 6
##    Gender           Discipline  Year  Medal Event Country
##     <chr>                <chr> <int>  <chr> <chr>   <chr>
##  1  Women Cross Country Skiing  2002 Bronze  10KM     ITA
##  2  Women Cross Country Skiing  2002   Gold  10KM     NOR
##  3  Women Cross Country Skiing  2002 Silver  10KM     RUS
##  4  Women Cross Country Skiing  2006 Bronze  10KM     NOR
##  5  Women Cross Country Skiing  2006   Gold  10KM     EST
##  6  Women Cross Country Skiing  2006 Silver  10KM     NOR
##  7  Women Cross Country Skiing  2010 Bronze  10KM     NOR
##  8  Women Cross Country Skiing  2010   Gold  10KM     SWE
##  9  Women Cross Country Skiing  2010 Silver  10KM     EST
## 10  Women Cross Country Skiing  2014 Bronze  10KM     NOR
## # ... with 62 more rows
```

Perfect. So that took the original 108 rows with `NA`s to just 72 without any `NA`s.

4.4 DEBUGGING `SPREAD` AND `GATHER` ERRORS

Being able to go from wide to tall and back is a really useful thing to know how to do. However, there are lots of times when things go wrong. In this section, I'll briefly show the two most common problems I face in my own code, what they mean, and how to fix them.

4.4.1 `Error: Duplicate identifiers`

When we were making the wide format for the cross country skiers using `spread`, I used the `unique` function to remove duplicates. Why did I do that? Well, because when I first tried to make the spreadsheet without it, it gave me an error message.

```
athletes %>%
    filter(Year >= 2000, Discipline == "Cross Country Skiing", Gender == "Wom
en") %>%
    select(-Athlete) %>%
    #unique() %>%
    spread(Event, Country) %>%
    print()
```

```
## Error: Duplicate identifiers for rows (1, 35, 61, 85), (4, 33, 53, 95), (1
0, 65, 82, 102), (22, 28, 81, 114), (3, 58, 63, 113), (6, 20, 54, 92), (57, 6
7, 86, 89), (13, 38, 103, 107), (29, 75, 93, 117), (21, 27, 34, 77), (32, 36,
44, 116), (59, 60, 73, 90), (56, 87), (2, 24), (84, 98), (46, 47), (76, 94),
(31, 42), (37, 72), (18, 79), (74, 91)
```

What does this mean? Well, fortunately, `spread` gives me the exact row numbers that caused the error, so I can zoom in and see what the problematic rows are. The way I diagnose this is to create a temporary dataframe, `temp`, that has all the code from the above block up until `spread`.

```
temp <- athletes %>%
    filter(Year >= 2000, Discipline == "Cross Country Skiing", Gender == "Wom
en") %>%
    select(-Athlete)
```

The first set of problematic rows was with rows 1, 35, 61, and 85, so let's take a look at just those.

```
temp[c(1, 35, 61, 85),]
```

```
## # A tibble: 4 x 6
##   Country Gender           Discipline       Event  Year  Medal
##     <chr>  <chr>                <chr>       <chr> <int>  <chr>
## 1     SUI  Women Cross Country Skiing 4X5KM Relay  2002 Bronze
## 2     SUI  Women Cross Country Skiing 4X5KM Relay  2002 Bronze
## 3     SUI  Women Cross Country Skiing 4X5KM Relay  2002 Bronze
## 4     SUI  Women Cross Country Skiing 4X5KM Relay  2002 Bronze
```

Okay, so what we see here are four identical rows. The 2002 Bronze medalists of the 4X5KM Relay from Switzerland. This makes sense. This is a four-person team, so of course there will be four people winning the bronze because they're all part of the winning team. The only reason they're identical here is because we took out the athletes' names.

Why is this is a problem? Think about what `spread` is trying to do. It's turning all the events—in this case "4X5KM Relay"—into columns and trying to put each combination of Discipline, Year, and Medal into one row, with the country—in this case "SUI"—filling in that cell. But, in our data, there are four rows for that unique combination of parameters. The country is the same in all four, but `spread` doesn't know that you know that, and it doesn't know what to do. Basically, it's trying to fit four pieces of information—"SUI", "SUI", "SUI", and "SUI"—into one cell.

We can see this in some of the other rows that the error message told us about:

```
temp[c(59, 60, 73, 90),]
```

```
## # A tibble: 4 x 6
##   Country Gender          Discipline       Event  Year  Medal
##     <chr>  <chr>               <chr>       <chr> <int>  <chr>
## 1     FIN  Women Cross Country Skiing 4X5KM Relay  2014 Silver
## 2     FIN  Women Cross Country Skiing 4X5KM Relay  2014 Silver
## 3     FIN  Women Cross Country Skiing 4X5KM Relay  2014 Silver
## 4     FIN  Women Cross Country Skiing 4X5KM Relay  2014 Silver
```

```
temp[c(2, 24),]
```

```
## # A tibble: 2 x 6
##   Country Gender          Discipline       Event  Year Medal
##     <chr>  <chr>               <chr>       <chr> <int> <chr>
## 1     SWE  Women Cross Country Skiing Team Sprint  2006  Gold
## 2     SWE  Women Cross Country Skiing Team Sprint  2006  Gold
```

So one solution to this problem is to do what I did and use the `unique` function to remove these duplicate rows. But this only works if the countries are all unique. What if we wanted to get a spreadsheet that contains not the countries in the cells but the athletes' names.

```
athletes %>%
    filter(Year >= 2000, Discipline == "Cross Country Skiing",
           Gender == "Women") %>%
    spread(Event, Athlete) %>%
    print()
```

```
## Error: Duplicate identifiers for rows (57, 67, 86, 89), (59, 60, 73, 90),
(4, 33, 53, 95), (6, 20, 54, 92), (29, 75, 93, 117), (21, 27, 34, 77), (22, 2
8, 81, 114), (10, 65, 82, 102), (13, 38, 103, 107), (3, 58, 63, 113), (1, 35,
61, 85), (32, 36, 44, 116), (84, 98), (56, 87), (74, 91), (76, 94), (18, 79),
(46, 47), (2, 24), (31, 42), (37, 72)
```

The same error message shows up. Let's create another temporary dataframe and diagnose the problem.

```
temp2 <- athletes %>%
    filter(Year >= 2000, Discipline == "Cross Country Skiing",
           Gender == "Women")

temp2[c(57, 67, 86, 89),]
```

```
## # A tibble: 4 x 7
##                 Athlete Country Gender          Discipline       Event
##                   <chr>   <chr>  <chr>               <chr>       <chr>
## 1       KUITUNEN, Virpi     FIN  Women Cross Country Skiing 4X5KM Relay
## 2        MURANEN, Pirjo     FIN  Women Cross Country Skiing 4X5KM Relay
## 3 ROPONEN, Riitta-Liisa     FIN  Women Cross Country Skiing 4X5KM Relay
## 4  SAARINEN, Aino-Kaisa     FIN  Women Cross Country Skiing 4X5KM Relay
## # ... with 2 more variables: Year <int>, Medal <chr>
```

Yeah, so it's the same problem. It's trying to cram four names into a single cell, and it doesn't know what to do. Here, `unique` wouldn't even work because the rows are all different. A really nice solution would be to simply concatenate their names, maybe separated by commas, and put that one string in the cell. Or, more fancily, combine them in a list, and put that whole list in the cell (which is possible in R). As far as I know, `spread` can't do that, but I'm sure other people have thought of this and have written an R package for it.

The point is that sometimes you just can't make the spreadsheet you want if you're not completely familiar with your data. In this case, `spread` might not be the solution and you may have to do some digging online to find the fix.

4.4.2 Duplicate columns

The other issue I've run into a lot happens when I use one of the `join` functions. To illustrate this, what if I wanted to add the name of the sport to the `athletes` dataframe. We have the discipline and the event, but not the sport itself. This is in the `events` dataframe, so we can just merge them:

```r
left_join(athletes, events, by = "Event")
```

```
## # A tibble: 101,910 x 10
##                Athlete Country Gender         Discipline.x        Event
##                  <chr>   <chr>  <chr>                <chr>        <chr>
##    1       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    2       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    3       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    4       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    5       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    6       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    7       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    8       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##    9       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
##   10       AAHLBERG, Mats     SWE    Men          Ice Hockey   Ice Hockey
## # ... with 1.018e+05 more rows, and 5 more variables: Year.x <int>,
## #   Medal <chr>, Year.y <int>, Sport <chr>, Discipline.y <chr>
```

Whoa, whoa, whoa! We've got several problem here. If you look, we now have a spreadsheet with over 100,000 rows! What happened?? You can get some clues by looking at the column names. We now now `Discipline.x`, `Year.x`, `Year.y`, and `Discipline.y`. What are those about? Turns out there was a column named `Discipline` and another one named `Year` in both the `athletes` and the `events` dataframes. `left_join` didn't know what to do with them, since it has to add both of them, so it just added a little suffix to the end of them. The `*.x` columns refer to the ones from `athletes` and the `*.y` refers to the `events`. The reason why there are so many rows is because it was trying to get unique combinations of all these years and disciplines and events, even though we know they just refer to the same thing.

How can we fix this? As it turns out, you can specify multiple columns that need to match between the columns. So instead of just `Event`, we can have it match `Event`, `Discipline`, and `Year`

```
left_join(athletes, events, by = c("Event", "Discipline", "Year"))
```

```
## # A tibble: 5,770 x 8
##                  Athlete Country Gender           Discipline
##                    <chr>   <chr>  <chr>                <chr>
##  1        AAHLBERG, Mats     SWE    Men          Ice Hockey
##  2        AAHLEN, Thomas     SWE    Men          Ice Hockey
##  3      AALAND, Per Knut     NOR    Men Cross Country Skiing
##  4   AALTONEN, Juhamatti     FIN    Men          Ice Hockey
##  5 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  6 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  7 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  8 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
##  9 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
## 10 AAMODT, Kjetil Andre     NOR    Men        Alpine Skiing
## # ... with 5,760 more rows, and 4 more variables: Event <chr>, Year <int>,
## #   Medal <chr>, Sport <chr>
```

Whew. There we go. We now have the original 5,770 rows that we had before, we don't have any duplicate columns, and the last column is now a new one with the name of the Sport.

This trick of matching multiple columns is actually quite useful for situations like these where you need to make sure things match in multiple ways. For example, there are three different events called `"Individual"` within three different disciplines:

```
events %>%
    filter(Event == "Individual") %>%
    select(-Year) %>%
    unique()
```

```
## # A tibble: 3 x 3
##       Sport      Discipline      Event
##       <chr>           <chr>      <chr>
## 1 Bobsleigh        Skeleton Individual
## 2   Skating  Figure skating Individual
## 3    Skiing Nordic Combined Individual
```

This is actually really important because if you just do a merge based on the event name, you're going to end up with bad data because it could match any one of these three disciplines. So in that case, it does make sense to do a join on multiple column names (`Discipline` and `Event` to be sure, but I would even throw in `Sport` to be safe).

5 Final remarks

The goal for this workshop was to expose you to some more advanced techniques within the `tidyverse`. We looked at how to merge datasets with the various `*_join` functions, how to get summaries of your data with `group_by` and `summarize`, and then how to reshape your data with `spread` and `gather`. I encourage you to learn more about these if you're still confused by looking through and Chapter 13[5], §5.6[6], and §12.3[7], respectfully, on these topics.

When I first read about these topics, I didn't learn them very well. But, what it did for me was make me aware of what is *possible* in R. So a few months later when a problem came along, I was able to think back to these techniques Turns out I was learning the solutions to problems I hadn't even had yet. But once you need them, they're great.

5 http://r4ds.had.co.nz/relational-data.html

6 http://r4ds.had.co.nz/transform.html#grouped-summaries-with-summarise

7 http://r4ds.had.co.nz/tidy-data.html#spreading-and-gathering