Data Wrangling

Paul Dunn

December, 2018

## Business Information Systems

## Central Michigan University

##Getting started with Data Wrangling in R

Data can be in a multitude of formats and part of our job will be to change it to a format needed for analysis. Officially this is the Transform part of ETL (Extract, Transform, Load). This is also called Data Wrangling. In R, we are going to use a library called dplyr which is part of the tidyverse. The EDAWR library just contains some small datasets we can use to learn.

#Uncomment to install or comment these 3 lines out to knit  
#install.packages("devtools")  
#devtools::install\_github("rstudio/EDAWR")  
  
library(tidyr)  
library(dplyr)  
library(EDAWR)

## Forms of data

Data can be in many different forms. It can be in rows and columns:

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

Or it can be like this

## country 2011 2012 2013  
## 1 FR 7000 6900 7000  
## 2 DE 5800 6000 6200  
## 3 US 15000 14000 13000

Or like this:

## city size amount  
## 1 New York large 23  
## 2 New York small 14  
## 3 London large 22  
## 4 London small 16  
## 5 Beijing large 121  
## 6 Beijing small 56

The problem is that data, however useful, is generally never in the form we need. If the above is the structure of relational database table, this is how the data will come out from a query. During the ETL phase we first *E*xtract it from the DB, then we can *T*ransform it. We transform it to whatever spec we need for analysis. This could be Tidy data in R or it could be pivot tables in Excel or what-have-you. The tools that you use for this part are irrelevant. Use what you have and use the tool closest to the data. That is to say, if you have the data in Hadoop, don’t download it, wrangle it in Python, then re-upload it. If you can do the work where the data lives (for large data sets) it will save time. In the specific case of Hadoop, there are many tools you can use: map-reduce with java, python, R (sparklyr), Hive and Pig to name a few. Generally we want our data in rows with variables in columns (like the first example.) The second set above we could transform:

## country 2011 2012 2013  
## 1 FR 7000 6900 7000  
## 2 DE 5800 6000 6200  
## 3 US 15000 14000 13000

## country year count  
## 1 FR 2011 7000  
## 2 DE 2011 5800  
## 3 US 2011 15000  
## 4 FR 2012 6900  
## 5 DE 2012 6000  
## 6 US 2012 14000  
## 7 FR 2013 7000  
## 8 DE 2013 6200  
## 9 US 2013 13000

If you are familiar with Excel, you might recognize this as a pivot. Or, if this data were in MSSQL, you could pivot the data in a select statement:

select daystomanufacture, avg(standardcost) as AverageCost from product group by daystomanufacture;

then becomes:

select 'AverageCost' as cost\_sorted\_by\_day, [0],[1],[2],[3],[4] from (select daystomanufacture, standardcost from product as sourcetable PIVOT (avg(standardcost) for daystomanufacture IN [0],[1],[2],[3],[4])) as PivotTable;

whew! But what do we do if we are using an RDBMS other than MSSQL, like MySQL? Or what if our data is in a text file (csv) or some other format? We can use R quite easily:

To accomplish this with our data above, it is just one line:

storms

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

gather(cases,"year","count",2:4)

## country year count  
## 1 FR 2011 7000  
## 2 DE 2011 5800  
## 3 US 2011 15000  
## 4 FR 2012 6900  
## 5 DE 2012 6000  
## 6 US 2012 14000  
## 7 FR 2013 7000  
## 8 DE 2013 6200  
## 9 US 2013 13000

Gather() as shown above will collapse multiple columns into 2 columns. In that call, cases is the dataset against which we want the call to work, “year” is the name of the new key column, “count” is the new value column, and 2:4 are which columns we want. We can even connect R directly to MySQL and get the data, we’ll go over how to do this later.

Spread(), on the other hand, is what we need to fix up the last table of data shown above. The problem with that table is that it really should be multiple columns (one for small particulate and its count, and one for large particulate and its count both listed for each city.)

storms

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

spread(pollution, size, amount)

## city large small  
## 1 Beijing 121 56  
## 2 London 22 16  
## 3 New York 23 14

As you can see, manipulating data in R using DPLYR is quite simple.

Let’s look at a few other functions we can use: separate() will split a single column into multiples based on a given separator. this is good for splitting a date into day, month and year

storms

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

storms2<-separate(storms,date,c("year","month","day"))  
storms3<-unite(storms2,combodate,year,month,day,sep = "/")

Unite() is the opposite of separate() and will combine multiple columns down to a single column.

Now let’s go over some ways to access different parts of our data. We have 4 topics to cover: \* Extract existing variables: select() \* Extract existing observations: filter() \* Derive new variables (from existing): mutate() \* Change unit of analysis: summarise() *note different spelling*

Let’s start with select(). This is what we use when we only want certain columns of our data, but all rows.

storms

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

select(storms, storm,date)

## storm date  
## 1 Alberto 2000-08-03  
## 2 Alex 1998-07-27  
## 3 Allison 1995-06-03  
## 4 Ana 1997-06-30  
## 5 Arlene 1999-06-11  
## 6 Arthur 1996-06-17

Read the R statement above as: select storm and date from storms.

If we only want storms where the wind speed was above 60, then we need to use filter(). This is because we are now changing which rows we want, not just which columns.

storms

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

filter(storms,wind>60)

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Allison 65 1005 1995-06-03

filter(storms,wind>35 | pressure>1010)

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

We can combine columns on which we want to filter, by separating them with commas. R will treat the comma as a logical AND. If you want to use OR, then you need a | (pipe) to do that

storms

## storm wind pressure date  
## 1 Alberto 110 1007 2000-08-03  
## 2 Alex 45 1009 1998-07-27  
## 3 Allison 65 1005 1995-06-03  
## 4 Ana 40 1013 1997-06-30  
## 5 Arlene 50 1010 1999-06-11  
## 6 Arthur 45 1010 1996-06-17

filter(storms,wind>35 , pressure>1010)

## storm wind pressure date  
## 1 Ana 40 1013 1997-06-30

Read the above filter line as Filter Storms and show me all rows where wind>50 OR pressure >1010

When we use DPLYR we can ‘chain’ together our commands instead of having to nest them. The operator we use to do this is %>% If you are familiar with \*nix this is similar to the pipe command | If we want to select and filter, we can send the output of one to the input of another. This makes writing and following code much easier.

In the rates data set, which contains tuberculosis rates, I want to see the name of the country and the year in which the rate was higher than 60. If you think of this in English If rate is > 60 show me the name of the country and the year

rates

## # A tibble: 3,484 × 5  
## country year cases population rate  
## <chr> <int> <dbl> <int> <dbl>  
## 1 Afghanistan 1997 128 19021226 0.07  
## 2 Afghanistan 1998 1778 19496836 0.91  
## 3 Afghanistan 1999 745 19987071 0.37  
## 4 Afghanistan 2000 2666 20595360 1.29  
## 5 Afghanistan 2001 4639 21347782 2.17  
## 6 Afghanistan 2002 6509 22202806 2.93  
## 7 Afghanistan 2003 6528 23116142 2.82  
## 8 Afghanistan 2004 8245 24018682 3.43  
## 9 Afghanistan 2005 9949 24860855 4   
## 10 Afghanistan 2006 12469 25631282 4.86  
## # ℹ 3,474 more rows

filter(rates,rate>60) %>% select(country,year,rate)

## # A tibble: 5 × 3  
## country year rate  
## <chr> <int> <dbl>  
## 1 South Africa 2008 63.7  
## 2 South Africa 2009 75.4  
## 3 South Africa 2010 70.9  
## 4 Swaziland 2006 70.8  
## 5 Swaziland 2010 81.4

Mutate() allows us to add new columns based on other columns/calculations. If we wanted to see how the rate changes from year-to-year, we can use mutate along with another function called lag(). Lag() looks 1 row behind the current row to get the value it needs. You can also use lead() to look 1 row ahead. We already have the rate, let’s compare how much it changed to the year prior:

rates

## # A tibble: 3,484 × 5  
## country year cases population rate  
## <chr> <int> <dbl> <int> <dbl>  
## 1 Afghanistan 1997 128 19021226 0.07  
## 2 Afghanistan 1998 1778 19496836 0.91  
## 3 Afghanistan 1999 745 19987071 0.37  
## 4 Afghanistan 2000 2666 20595360 1.29  
## 5 Afghanistan 2001 4639 21347782 2.17  
## 6 Afghanistan 2002 6509 22202806 2.93  
## 7 Afghanistan 2003 6528 23116142 2.82  
## 8 Afghanistan 2004 8245 24018682 3.43  
## 9 Afghanistan 2005 9949 24860855 4   
## 10 Afghanistan 2006 12469 25631282 4.86  
## # ℹ 3,474 more rows

temprates<-mutate(rates,"year\_over\_year" = rate - lag(rate))  
View(temprates)

rates

## # A tibble: 3,484 × 5  
## country year cases population rate  
## <chr> <int> <dbl> <int> <dbl>  
## 1 Afghanistan 1997 128 19021226 0.07  
## 2 Afghanistan 1998 1778 19496836 0.91  
## 3 Afghanistan 1999 745 19987071 0.37  
## 4 Afghanistan 2000 2666 20595360 1.29  
## 5 Afghanistan 2001 4639 21347782 2.17  
## 6 Afghanistan 2002 6509 22202806 2.93  
## 7 Afghanistan 2003 6528 23116142 2.82  
## 8 Afghanistan 2004 8245 24018682 3.43  
## 9 Afghanistan 2005 9949 24860855 4   
## 10 Afghanistan 2006 12469 25631282 4.86  
## # ℹ 3,474 more rows

mutate(rates,"year\_over\_year" = rate - lag(rate))

## # A tibble: 3,484 × 6  
## country year cases population rate year\_over\_year  
## <chr> <int> <dbl> <int> <dbl> <dbl>  
## 1 Afghanistan 1997 128 19021226 0.07 NA   
## 2 Afghanistan 1998 1778 19496836 0.91 0.84   
## 3 Afghanistan 1999 745 19987071 0.37 -0.54   
## 4 Afghanistan 2000 2666 20595360 1.29 0.92   
## 5 Afghanistan 2001 4639 21347782 2.17 0.88   
## 6 Afghanistan 2002 6509 22202806 2.93 0.76   
## 7 Afghanistan 2003 6528 23116142 2.82 -0.110  
## 8 Afghanistan 2004 8245 24018682 3.43 0.61   
## 9 Afghanistan 2005 9949 24860855 4 0.57   
## 10 Afghanistan 2006 12469 25631282 4.86 0.86   
## # ℹ 3,474 more rows

Note that the first row has NA for its value, this is because it’s the first one and lag() can’t go back 1 row for data.

population

## # A tibble: 100 × 20  
## country `1995` `1996` `1997` `1998` `1999` `2000` `2001` `2002` `2003` `2004`  
## \* <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>  
## 1 Afghan… 1.76e7 1.84e7 1.90e7 1.95e7 2.00e7 2.06e7 2.13e7 2.22e7 2.31e7 2.40e7  
## 2 Algeria 2.93e7 2.98e7 3.03e7 3.08e7 3.13e7 3.17e7 3.22e7 3.26e7 3.30e7 3.35e7  
## 3 Angola 1.21e7 1.25e7 1.28e7 1.31e7 1.35e7 1.39e7 1.44e7 1.49e7 1.54e7 1.60e7  
## 4 Argent… 3.48e7 3.53e7 3.57e7 3.61e7 3.65e7 3.69e7 3.73e7 3.76e7 3.80e7 3.83e7  
## 5 Azerba… 7.77e6 7.85e6 7.92e6 7.98e6 8.05e6 8.12e6 8.20e6 8.28e6 8.37e6 8.47e6  
## 6 Bangla… 1.20e8 1.22e8 1.25e8 1.27e8 1.30e8 1.32e8 1.35e8 1.37e8 1.39e8 1.41e8  
## 7 Belarus 1.02e7 1.02e7 1.01e7 1.01e7 1.00e7 9.98e6 9.92e6 9.85e6 9.78e6 9.72e6  
## 8 Benin 5.99e6 6.18e6 6.36e6 6.55e6 6.74e6 6.95e6 7.17e6 7.41e6 7.67e6 7.92e6  
## 9 Bolivi… 7.64e6 7.81e6 7.98e6 8.15e6 8.32e6 8.50e6 8.67e6 8.84e6 9.02e6 9.19e6  
## 10 Botswa… 1.58e6 1.62e6 1.66e6 1.69e6 1.72e6 1.76e6 1.78e6 1.81e6 1.83e6 1.85e6  
## # ℹ 90 more rows  
## # ℹ 9 more variables: `2005` <int>, `2006` <int>, `2007` <int>, `2008` <int>,  
## # `2009` <int>, `2010` <int>, `2011` <int>, `2012` <int>, `2013` <int>

Summarise() let’s us summarize our data by a large list of mathematical functions, here are a few: \* min() max() \* mean() \* median() \* sum() \* var() sd() \* first() last() nth() \* n() n\_distinct() All of these functions take a vector and reduce it to a single value (a scalar). Let’s take a look. What is the mean of pollution amount:

pollution

## city size amount  
## 1 New York large 23  
## 2 New York small 14  
## 3 London large 22  
## 4 London small 16  
## 5 Beijing large 121  
## 6 Beijing small 56

summarise(pollution,mean(amount))

## mean(amount)  
## 1 42

Interesting, but it doesn’t tell us much. We really need to be able to get the mean for each city. for each city, show the mean amount of pollution:

group\_by(pollution, city) %>% summarise(avg=mean(amount))

## # A tibble: 3 × 2  
## city avg  
## <chr> <dbl>  
## 1 Beijing 88.5  
## 2 London 19   
## 3 New York 18.5

Any time you think “for each *thing*” you can use group\_by(*thing*) so that you can do aggregations (summarise)

Arrange() is a handy one here because it will let you change the order in which your data is displayed. For the above statement: for each city, show the mean amount of pollution from highest to lowest:

group\_by(pollution, city) %>% summarise(avg=mean(amount)) %>% arrange(desc(avg))

## # A tibble: 3 × 2  
## city avg  
## <chr> <dbl>  
## 1 Beijing 88.5  
## 2 London 19   
## 3 New York 18.5

If you want to sort descending, you can use desc() along with arrange: arrange(desc(avg))

Let’s look at a more complex dataset. The nutrition dataset comes from the USDA and has 8,463 foods with 28 variables. If you want to look at the structure of the data, use the str() function. Head() gives you the top 10 rows, and tail() gives you the bottom 10 rows:

nutrition

## # A tibble: 8,463 × 28  
## food calories protein carbohydrates total\_fat saturated\_fat caffiene  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Butter, salt… 717 0.85 0.06 81.1 51.4 0  
## 2 Butter, whip… 717 0.85 0.06 81.1 50.5 0  
## 3 Butter oil, … 876 0.28 0 99.5 61.9 0  
## 4 Cheese, blue 353 21.4 2.34 28.7 18.7 0  
## 5 Cheese, brick 371 23.2 2.79 29.7 18.8 0  
## 6 Cheese, brie 334 20.8 0.45 27.7 17.4 0  
## 7 Cheese, came… 300 19.8 0.46 24.3 15.3 0  
## 8 Cheese, cara… 376 25.2 3.06 29.2 18.6 NA  
## 9 Cheese, ched… 403 24.9 1.28 33.1 21.1 0  
## 10 Cheese, ches… 387 23.4 4.78 30.6 19.5 NA  
## # ℹ 8,453 more rows  
## # ℹ 21 more variables: cholesterol <dbl>, fiber <dbl>, folic\_acid <dbl>,  
## # sodium <dbl>, calcium <dbl>, iron <dbl>, magnesium <dbl>, manganese <dbl>,  
## # niacin <dbl>, phosphorus <dbl>, potassium <dbl>, riboflavin <dbl>,  
## # selenium <dbl>, thiamin <dbl>, vitamin\_A <dbl>, vitamin\_B6 <dbl>,  
## # vitamin\_B12 <dbl>, vitamin\_C <dbl>, vitamin\_D <dbl>, zinc <dbl>,  
## # group <chr>

str(nutrition)

## tibble [8,463 × 28] (S3: tbl\_df/tbl/data.frame)  
## $ food : chr [1:8463] "Butter, salted" "Butter, whipped, with salt" "Butter oil, anhydrous" "Cheese, blue" ...  
## $ calories : num [1:8463] 717 717 876 353 371 334 300 376 403 387 ...  
## $ protein : num [1:8463] 0.85 0.85 0.28 21.4 23.24 ...  
## $ carbohydrates: num [1:8463] 0.06 0.06 0 2.34 2.79 0.45 0.46 3.06 1.28 4.78 ...  
## $ total\_fat : num [1:8463] 81.1 81.1 99.5 28.7 29.7 ...  
## $ saturated\_fat: num [1:8463] 51.4 50.5 61.9 18.7 18.8 ...  
## $ caffiene : num [1:8463] 0 0 0 0 0 0 0 NA 0 NA ...  
## $ cholesterol : num [1:8463] 215 219 256 75 94 100 72 93 105 103 ...  
## $ fiber : num [1:8463] 0 0 0 0 0 0 0 0 0 0 ...  
## $ folic\_acid : num [1:8463] 0 0 0 0 0 0 0 0 0 0 ...  
## $ sodium : num [1:8463] 643 659 2 1146 560 ...  
## $ calcium : num [1:8463] 24 24 4 528 674 184 388 673 721 643 ...  
## $ iron : num [1:8463] 0.02 0.16 0 0.31 0.43 0.5 0.33 0.64 0.68 0.21 ...  
## $ magnesium : num [1:8463] 2 2 0 23 24 20 20 22 28 21 ...  
## $ manganese : num [1:8463] 0 0.004 0 0.009 0.012 0.034 0.038 0.021 0.01 0.012 ...  
## $ niacin : num [1:8463] 0.042 0.042 0.003 1.016 0.118 ...  
## $ phosphorus : num [1:8463] 24 23 3 387 451 188 347 490 512 464 ...  
## $ potassium : num [1:8463] 24 26 5 256 136 152 187 93 98 95 ...  
## $ riboflavin : num [1:8463] 0.034 0.034 0.005 0.382 0.351 0.52 0.488 0.45 0.375 0.293 ...  
## $ selenium : num [1:8463] 1 1 0 14.5 14.5 14.5 14.5 14.5 13.9 14.5 ...  
## $ thiamin : num [1:8463] 0.005 0.005 0.001 0.029 0.014 0.07 0.028 0.031 0.027 0.046 ...  
## $ vitamin\_A : num [1:8463] 2499 2499 3069 721 1080 ...  
## $ vitamin\_B6 : num [1:8463] 0.003 0.003 0.001 0.166 0.065 0.235 0.227 0.074 0.074 0.074 ...  
## $ vitamin\_B12 : num [1:8463] 0.17 0.13 0.01 1.22 1.26 1.65 1.3 0.27 0.83 0.83 ...  
## $ vitamin\_C : num [1:8463] 0 0 0 0 0 0 0 0 0 0 ...  
## $ vitamin\_D : num [1:8463] 60 60 73 21 22 20 18 NA 24 NA ...  
## $ zinc : num [1:8463] 0.09 0.05 0.01 2.66 2.6 2.38 2.38 2.94 3.11 2.79 ...  
## $ group : chr [1:8463] "Dairy and Egg Products" "Dairy and Egg Products" "Dairy and Egg Products" "Dairy and Egg Products" ...

head(nutrition)

## # A tibble: 6 × 28  
## food calories protein carbohydrates total\_fat saturated\_fat caffiene  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Butter, salted 717 0.85 0.06 81.1 51.4 0  
## 2 Butter, whipp… 717 0.85 0.06 81.1 50.5 0  
## 3 Butter oil, a… 876 0.28 0 99.5 61.9 0  
## 4 Cheese, blue 353 21.4 2.34 28.7 18.7 0  
## 5 Cheese, brick 371 23.2 2.79 29.7 18.8 0  
## 6 Cheese, brie 334 20.8 0.45 27.7 17.4 0  
## # ℹ 21 more variables: cholesterol <dbl>, fiber <dbl>, folic\_acid <dbl>,  
## # sodium <dbl>, calcium <dbl>, iron <dbl>, magnesium <dbl>, manganese <dbl>,  
## # niacin <dbl>, phosphorus <dbl>, potassium <dbl>, riboflavin <dbl>,  
## # selenium <dbl>, thiamin <dbl>, vitamin\_A <dbl>, vitamin\_B6 <dbl>,  
## # vitamin\_B12 <dbl>, vitamin\_C <dbl>, vitamin\_D <dbl>, zinc <dbl>,  
## # group <chr>

tail(nutrition)

## # A tibble: 6 × 28  
## food calories protein carbohydrates total\_fat saturated\_fat caffiene  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Frog legs, raw NA 16.4 0 0.3 0.076 0  
## 2 Fish, mackere… NA 18.5 0 25.1 7.15 0  
## 3 Mollusks, sca… NA 20.5 5.41 0.84 0.218 0  
## 4 Syrup, cane 269 0 73.1 0 0 0  
## 5 Mollusks, sna… NA 16.1 2 1.4 0.361 0  
## 6 Turtle, green… NA 19.8 0 0.5 0.127 0  
## # ℹ 21 more variables: cholesterol <dbl>, fiber <dbl>, folic\_acid <dbl>,  
## # sodium <dbl>, calcium <dbl>, iron <dbl>, magnesium <dbl>, manganese <dbl>,  
## # niacin <dbl>, phosphorus <dbl>, potassium <dbl>, riboflavin <dbl>,  
## # selenium <dbl>, thiamin <dbl>, vitamin\_A <dbl>, vitamin\_B6 <dbl>,  
## # vitamin\_B12 <dbl>, vitamin\_C <dbl>, vitamin\_D <dbl>, zinc <dbl>,  
## # group <chr>

View(nutrition)

Let’s get the mean calorie value from the data:

summarise(nutrition,mean(calories,na.rm=TRUE))

## # A tibble: 1 × 1  
## `mean(calories, na.rm = TRUE)`  
## <dbl>  
## 1 200.

Why are we not getting a value for the mean of calories? Let’s check for missing values. We can use summary() for this. It will tell us the count of missing values for each column

summary(nutrition$calories)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 0.0 77.0 170.0 199.6 290.0 898.0 3802

So we know that calories has 3,802 missing values and that’s messing up our mean calculation. We can use na.rm=TRUE to tell mean() to remove any NAs that it finds.

summarise(nutrition,mean(calories,na.rm=TRUE))

## # A tibble: 1 × 1  
## `mean(calories, na.rm = TRUE)`  
## <dbl>  
## 1 200.

so, the mean calorie for 4,661 (8463-3802) foods is 200. let’s look by food group. To help, let’s start with a statement of what we want to see: for each food group, calculate the mean calories then display the food group and calories in descending order.

group\_by(nutrition,group) %>% summarise(avgcals=mean(calories,na.rm=TRUE)) %>% arrange(desc(avgcals))

## # A tibble: 25 × 2  
## group avgcals  
## <chr> <dbl>  
## 1 Nut and Seed Products 490.  
## 2 Snacks 459.  
## 3 Fats and Oils 440.  
## 4 Baked Products 358.  
## 5 Sweets 317.  
## 6 Cereal Grains and Pasta 284.  
## 7 Spices and Herbs 263.  
## 8 Sausages and Luncheon Meats 252.  
## 9 Dairy and Egg Products 249.  
## 10 Legumes and Legume Products 246.  
## # ℹ 15 more rows

# mean of calories for the entire dataset  
mean\_calories <- mean(nutrition$calories, na.rm = TRUE)  
  
  
  
# the mean calories for each food group  
mean\_calories\_per\_group <- nutrition %>%  
group\_by(group) %>%  
summarise(mean\_calories = mean(calories, na.rm = TRUE))  
  
  
# group with the highest mean calories  
highest\_calories\_group <- mean\_calories\_per\_group %>%  
filter(mean\_calories == max(mean\_calories))  
  
  
# group with the lowest mean calories  
lowest\_calories\_group <- mean\_calories\_per\_group %>%  
filter(mean\_calories == min(mean\_calories))  
  
  
# Output  
mean\_calories

## [1] 199.5973