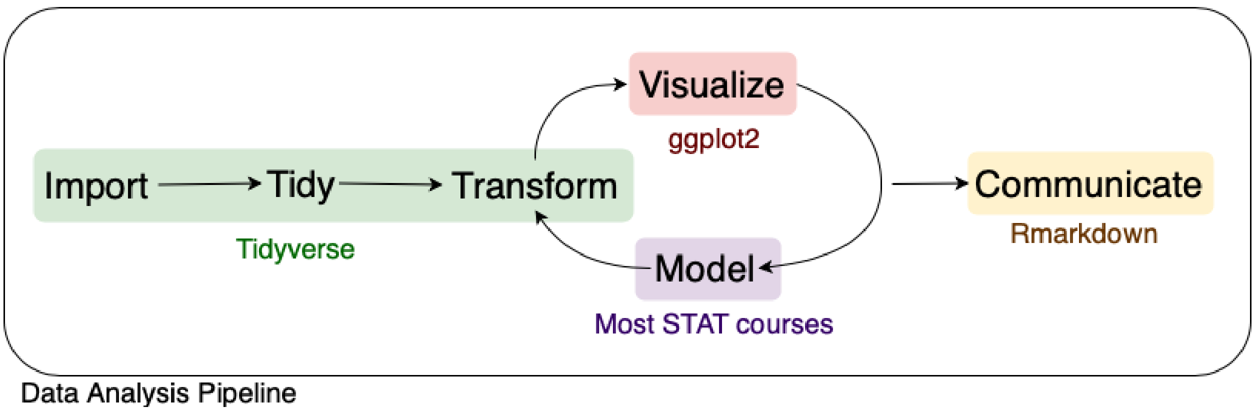
3 tidyverse

The tidyverse is a suite of packages released by RStudio that work very well together ("verse") to make data analysis run smoothly ("tidy"). It's also a package in R that loads all the packages in the tidyverse at once.

```r
library(tidyverse)
```

You actually already know one member of the tidyverse – `ggplot2`! We will highlight three more packages in the tidyverse for data analysis.



Adapted from R for Data Science, Wickham & Grolemund (2017)

3.1 `readr`

The first step in (almost) any data analysis task is reading data into R. Data can take many formats, but we will focus on text files.

> But what about `.xlsx`??

File extensions `.xls` and `.xlsx` are proprietary Excel formats/ These are binary files (meaning if you open one outside of Excel it will not be human readable). An alternable for rectangular data is a `.csv`.

`.csv` is an extension for *comma separated value* files. They are text files – directly readable – where each column is separated by a comma and each row a new line.

```
Rank,Major_code,Major,Total,Men,Women,Major_category,ShareWomen
1,2419,PETROLEUM ENGINEERING,2339,2057,282,Engineering,0.120564344
2,2416,MINING AND MINERAL ENGINEERING,756,679,77,Engineering,0.101851852
```

`.tsv` is an extension for *tab separated value* files. These are also text files, but the columns are separated by tabs instead of commas. Sometimes these will be `.txt` extension files.

```
Rank    Major_code    Major    Total    Men    Women    Major_category    ShareWom
1    2419    PETROLEUM ENGINEERING    2339    2057    282    Engineering    0.1205
2    2416    MINING AND MINERAL ENGINEERING    756    679    77    Engineering
```

The package `readr` provides a fast and friendly way to ready rectangular text data into `R`.

Here is an example csv file from fivethirtyeight.com on how to choose your college major (https://fivethirtyeight.com/features/the-economic-guide-to-picking-a-college-major/).

```
# load readr
library(readr)

# read a csv
recent_grads <- read_csv(file =
  "https://raw.githubusercontent.com/fivethirtyeight/data/master/college-
  majors/recent-grads.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Major = col_character(),
##   Major_category = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

`read_csv()` is just one way to read a file using the `readr` package.

- `read_delim()`: the most generic function. Use the delim argument to read a file with any type of delimiter
- `read_tsv()`: read tab separated files
- `read_lines()`: read a file into a vector that has one element per line of the file

- `read_file()`: read a file into a single character element
- `read_table()`: read a file separated by space

Your Turn

1. Read the NFL salaries dataset from https://raw.githubusercontent.com/ada-lovecraft/ProcessingSketches/master/Bits%20and%20Pieces/Football_Stuff/data/nfl-salaries.tsv into `R`.

2. What is the highest NFL salary in this dataset? Who is the highest paid player?

3. Make a histogram and describe the distribution of NFL salaries.

3.2 dplyr

We almost never will read in data and have it in exactly the right form for visualizing and modeling. Often we need to create variable or summaries.

To facilitate easy transformation of data, we're going to learn how to use the `dplyr` package. `dplyr` uses 6 main verbs, which correspond to some main tasks we may want to perform in an analysis.

We will do this with the `recent_grads` data from fivethiryeight.com we just read into `R` using `readr`.

3.2.1 %>%

Before we get into the verbs in `dplyr`, I want to introduce a new paradigm. All of the functions in the tidyverse are structured such that the first argument is a data frame and they also return a data frame. This allows for efficient use of the pipe operator `%>%` (pronounce this as "then").

```
a %>% b()
```

Taked the result on the left and passes it to the first argument on the right. This is equivalent to

```
b(a)
```

This is useful when we want to chain together many operations in an analysis.

3.2.2 filter()

`filter()` lets us subset observations based on their values. This is similar to using `[]` to subset a data frame, but simpler.

The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame.

Let's subset the `recent_grad` data set to focus on Statistics majors.

```
recent_grads %>% filter(Major == "STATISTICS AND DECISION SCIENCE")
```

```
## # A tibble: 1 x 21
##    Rank Major_code Major Total   Men Women Major_category ShareWomen
##   <dbl>      <dbl> <chr> <dbl> <dbl> <dbl> <chr>               <dbl>
## 1    47       3702 STAT…  6251  2960  3291 Computers & M…      0.526
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>,
## #   Full_time <dbl>, Part_time <dbl>, Full_time_year_round <dbl>,
## #   Unemployed <dbl>, Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>,
## #   P75th <dbl>, College_jobs <dbl>, Non_college_jobs <dbl>,
## #   Low_wage_jobs <dbl>
```

Alternatively, we could look at all Majors in the same category, "Computers & Mathematics", for comparison.

```
recent_grads %>% filter(Major_category == "Computers & Mathematics")
```

```
## # A tibble: 11 x 21
##     Rank Major_code Major  Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  6    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  7    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  8    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  9    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 10    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
## 11   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>,
## #   Full_time <dbl>, Part_time <dbl>, Full_time_year_round <dbl>,
## #   Unemployed <dbl>, Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>,
## #   P75th <dbl>, College_jobs <dbl>, Non_college_jobs <dbl>,
## #   Low_wage_jobs <dbl>
```

Notice we are using `%>%` to pass the data frame to the first argument in `filter()` and we do not need to use `recent_grads$Colum Name` to subset our data.

`dplyr` functions never modify their inputs, so if we need to save the result, we have to do it using `<-`.

```
math_grads <- recent_grads %>% filter(Major_category == "Computers &
  Mathematics")
```

Everything we've already learned about logicals and comparisons comes in handy here, since the second argument of `filter()` is a comparitor expression telling `dplyr` what rows we care about.

3.2.3 `arrange()`

`arrange()` works similarly to `filter()` except that it changes the order of rows rather than subsetting. Again, the first parameter is a data frame and the additional parameters are a set of column names to order by.

```
math_grads %>% arrange(ShareWomen)
```

```
## # A tibble: 11 x 21
##     Rank Major_code Major  Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  2    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
##  3    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  6    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  7    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
##  8   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
##  9    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
## 10    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
## 11    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>,
## #   Full_time <dbl>, Part_time <dbl>, Full_time_year_round <dbl>,
## #   Unemployed <dbl>, Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>,
## #   P75th <dbl>, College_jobs <dbl>, Non_college_jobs <dbl>,
## #   Low_wage_jobs <dbl>
```

If we provide more than one column name, each additional column will be used to break ties in the values of preceding columns.

We can use `desc()` to re-order by a column in descending order.

```
math_grads %>% arrange(desc(ShareWomen))
```

```
## # A tibble: 11 x 21
##     Rank Major_code Major  Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  4   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
##  5    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
##  6    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  7    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  8    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  9    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
## 10    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 11    53       4005 MATH…    609   500   109 Computers & M…      0.179
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>,
## #   Full_time <dbl>, Part_time <dbl>, Full_time_year_round <dbl>,
## #   Unemployed <dbl>, Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>,
## #   P75th <dbl>, College_jobs <dbl>, Non_college_jobs <dbl>,
## #   Low_wage_jobs <dbl>
```

3.2.4 `select()`

Sometimes we have data sets with a ton of variables and often we want to narrow down the ones that we actually care about. `select()` allows us to do this based on the names of the variables.

```
math_grads %>% select(Major, ShareWomen, Total, Full_time, P75th)
```

```
## # A tibble: 11 x 5
##    Major                                 ShareWomen  Total Full_time P75th
##    <chr>                                      <dbl>  <dbl>     <dbl> <dbl>
##  1 COMPUTER SCIENCE                           0.223 128319     91485 70000
##  2 MATHEMATICS                                0.448  72397     46399 60000
##  3 COMPUTER AND INFORMATION SYSTEMS           0.254  36698     26348 60000
##  4 INFORMATION SCIENCES                       0.244  11913      9105 58000
##  5 STATISTICS AND DECISION SCIENCE            0.526   6251      3190 60000
##  6 APPLIED MATHEMATICS                        0.434   4939      3465 63000
##  7 MATHEMATICS AND COMPUTER SCIENCE           0.179    609       584 78000
##  8 COMPUTER PROGRAMMING AND DATA PROCESS…     0.269   4168      3204 46000
##  9 COMPUTER ADMINISTRATION MANAGEMENT AN…     0.181   8066      6289 50000
## 10 COMPUTER NETWORKING AND TELECOMMUNICA…     0.305   7613      5495 49000
## 11 COMMUNICATION TECHNOLOGIES                 0.366  18035     11981 45000
```

We can also use

- `starts_with("abc")` matches names that begin with "abc"
- `ends_with("xyz")` matches names that end with "xyz"
- `contains("ijk")` matches names that contain "ijk"
- `everything()` mathes all columns

```
math_grads %>% select(Major, College_jobs:Low_wage_jobs)
```

```
## # A tibble: 11 x 4
##    Major                          College_jobs Non_college_jobs Low_wage_jobs
##    <chr>                                 <dbl>            <dbl>         <dbl>
##  1 COMPUTER SCIENCE                      68622            25667          5144
##  2 MATHEMATICS                           34800            14829          4569
##  3 COMPUTER AND INFORMATION SY…          13344            11783          1672
##  4 INFORMATION SCIENCES                   4390             4102           608
##  5 STATISTICS AND DECISION SCI…           2298             1200           343
##  6 APPLIED MATHEMATICS                    2437              803           357
##  7 MATHEMATICS AND COMPUTER SC…            452               67            25
##  8 COMPUTER PROGRAMMING AND DA…           2024             1033           263
##  9 COMPUTER ADMINISTRATION MAN…           2354             3244           308
## 10 COMPUTER NETWORKING AND TEL…           2593             2941           352
## 11 COMMUNICATION TECHNOLOGIES             4545             8794          2495
```

`rename()` is a function that will rename an existing column and select all columns.

```
math_grads %>% rename(Code_major = Major_code)
```

```
## # A tibble: 11 x 21
##     Rank Code_major Major  Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>  <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  6    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  7    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  8    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  9    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 10    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
## 11   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
## # … with 13 more variables: Sample_size <dbl>, Employed <dbl>,
## #   Full_time <dbl>, Part_time <dbl>, Full_time_year_round <dbl>,
## #   Unemployed <dbl>, Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>,
## #   P75th <dbl>, College_jobs <dbl>, Non_college_jobs <dbl>,
## #   Low_wage_jobs <dbl>
```

3.2.5 `mutate()`

Besides selecting sets of existing columns, we can also add new columns that are functions of existing columns with `mutate()`. `mutate()` always adds new columns at the end of the data frame.

```
math_grads %>% mutate(Full_time_rate = Full_time_year_round/Total)
```

```
## # A tibble: 11 x 22
##     Rank Major_code Major   Total   Men Women Major_category ShareWomen
##    <dbl>      <dbl> <chr>   <dbl> <dbl> <dbl> <chr>               <dbl>
##  1    21       2102 COMP… 128319 99743 28576 Computers & M…      0.223
##  2    42       3700 MATH…  72397 39956 32441 Computers & M…      0.448
##  3    43       2100 COMP…  36698 27392  9306 Computers & M…      0.254
##  4    46       2105 INFO…  11913  9005  2908 Computers & M…      0.244
##  5    47       3702 STAT…   6251  2960  3291 Computers & M…      0.526
##  6    48       3701 APPL…   4939  2794  2145 Computers & M…      0.434
##  7    53       4005 MATH…    609   500   109 Computers & M…      0.179
##  8    54       2101 COMP…   4168  3046  1122 Computers & M…      0.269
##  9    82       2106 COMP…   8066  6607  1459 Computers & M…      0.181
## 10    85       2107 COMP…   7613  5291  2322 Computers & M…      0.305
## 11   106       2001 COMM…  18035 11431  6604 Computers & M…      0.366
## # … with 14 more variables: Sample_size <dbl>, Employed <dbl>,
## #   Full_time <dbl>, Part_time <dbl>, Full_time_year_round <dbl>,
## #   Unemployed <dbl>, Unemployment_rate <dbl>, Median <dbl>, P25th <dbl>,
## #   P75th <dbl>, College_jobs <dbl>, Non_college_jobs <dbl>,
## #   Low_wage_jobs <dbl>, Full_time_rate <dbl>
```

```
# we can't see everything
math_grads %>%
  mutate(Full_time_rate = Full_time_year_round/Total) %>%
  select(Major, ShareWomen, Full_time_rate)
```

```
## # A tibble: 11 x 3
##    Major                                              ShareWomen Full_ti-
```

```
me_rate
##    <chr>                                        <dbl>         <dbl>
##  1 COMPUTER SCIENCE                             0.223         0.553
##  2 MATHEMATICS                                  0.448         0.466
##  3 COMPUTER AND INFORMATION SYSTEMS             0.254         0.576
##  4 INFORMATION SCIENCES                         0.244         0.619
##  5 STATISTICS AND DECISION SCIENCE              0.526         0.344
##  6 APPLIED MATHEMATICS                          0.434         0.525
##  7 MATHEMATICS AND COMPUTER SCIENCE             0.179         0.642
##  8 COMPUTER PROGRAMMING AND DATA PROCESSING     0.269         0.589
##  9 COMPUTER ADMINISTRATION MANAGEMENT AND SECURI…   0.181         0.612
## 10 COMPUTER NETWORKING AND TELECOMMUNICATIONS   0.305         0.574
## 11 COMMUNICATION TECHNOLOGIES                   0.366         0.504
```

3.2.6 `summarise()`

The last major verb is summarise(). It collapses a data frame to a single row based on a summary function.

```
math_grads %>% summarise(mean_major_size = mean(Total))
```

```
## # A tibble: 1 x 1
##   mean_major_size
##             <dbl>
## 1          27183.
```

A useful summary function is a count (`n()`), or a count of non-missing values (`sum(!is.na())`).

```
math_grads %>% summarise(mean_major_size = mean(Total), num_majors =
  n())
```

```
## # A tibble: 1 x 2
##   mean_major_size num_majors
##             <dbl>      <int>
## 1          27183.         11
```

3.2.7 `group_by()`

`summarise()` is not super useful unless we pair it with `group_by()`. This changes the unit of analysis from the complete dataset to individual groups. Then, when we use the `dplyr` verbs on a grouped data frame they'll be automatically applied "by group".

```
recent_grads %>%
  group_by(Major_category) %>%
  summarise(mean_major_size = mean(Total, na.rm = TRUE)) %>%
  arrange(desc(mean_major_size))
```

```
## # A tibble: 16 x 2
##    Major_category                      mean_major_size
##    <chr>                                         <dbl>
##  1 Business                                    100183.
##  2 Communications & Journalism                  98150.
##  3 Social Science                               58885.
##  4 Psychology & Social Work                     53445.
##  5 Humanities & Liberal Arts                    47565.
##  6 Arts                                         44641.
##  7 Health                                       38602.
##  8 Law & Public Policy                          35821.
##  9 Education                                    34946.
## 10 Industrial Arts & Consumer Services          32827.
## 11 Biology & Life Science                       32419.
## 12 Computers & Mathematics                      27183.
## 13 Physical Sciences                            18548.
## 14 Engineering                                  18537.
## 15 Interdisciplinary                            12296
## 16 Agriculture & Natural Resources               8402.
```

We can group by multiple variables and if we need to remove grouping, and return to operations on ungrouped data, we use `ungroup()`.

Grouping is also useful for `arrange()` and `mutate()` within groups.

Your Turn

Using the NFL salaries from https://raw.githubusercontent.com/ada-lovecraft/ProcessingSketches/master/Bits%20and%20Pieces/Football_Stuff/data/nfl-salaries.tsv that you loaded into `R` in the previous your turn, perform the following.

1. What is the team with the highest paid roster?
2. What are the top 5 paid players?
3. What is the highest paid position on average? the lowest? the most variable?

3.3 `tidyr`

> "Happy families are all alike; every unhappy family is unhappy in its own way." –– Leo Tolstoy

> "Tidy datasets are all alike, but every messy dataset is messy in its own way." –– Hadley Wickham

Tidy data is an organization strategy for data that makes it easier to work with, analyze, and visualize. `tidyr` is a package that can help us tidy our data in a less painful way.

The following all contain the same data, but show different levels of "tidiness".

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```r
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```r
# spread across two data frames
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```r
table4b
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

While these are all representations of the same underlying data, they are not equally easy to use.

There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

`table2` isn't tidy because each variable doesn't have its own column.

`table3` isn't tidy because each value doesn't have its own cell.

`table4a` and `table4b` aren't tidy because each observation doesn't have its own row.

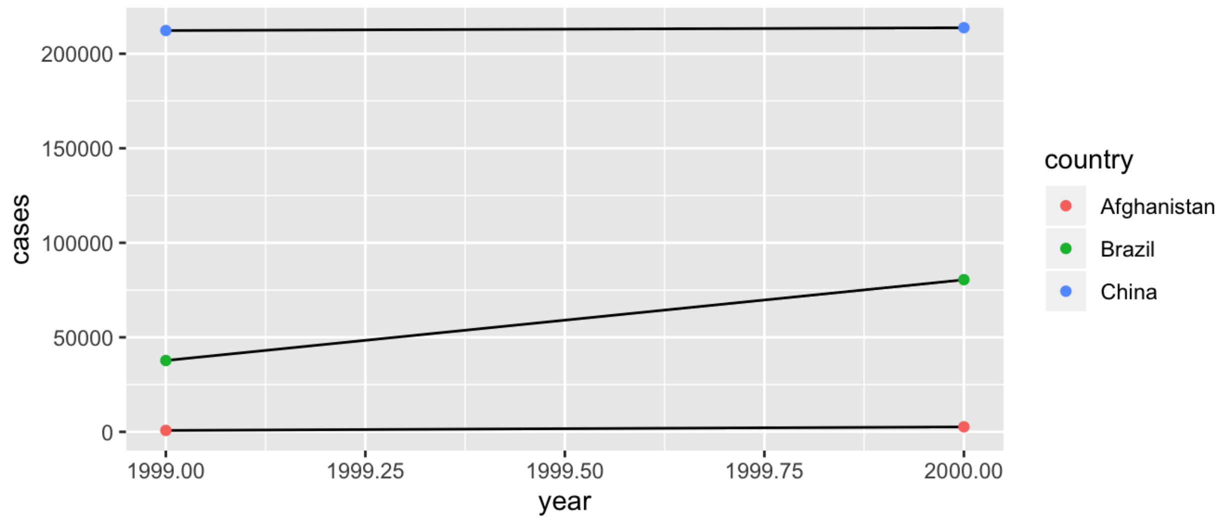
`table1` is tidy!

Being tidy with our data is useful because it's a consistent set of rules to follow for working with data and because it allows `R` to be efficient.

```
# Compute rate per 10,000
table1 %>%
  mutate(rate = cases / population * 10000)
```

```
## # A tibble: 6 x 5
##   country      year  cases population  rate
##   <chr>       <int>  <int>      <int> <dbl>
## 1 Afghanistan  1999    745   19987071 0.373
## 2 Afghanistan  2000   2666   20595360 1.29
## 3 Brazil       1999  37737  172006362 2.19
## 4 Brazil       2000  80488  174504898 4.61
## 5 China        1999 212258 1272915272 1.67
## 6 China        2000 213766 1280428583 1.67
```

```
# Visualize cases over time
library(ggplot2)
ggplot(table1, aes(year, cases)) +
  geom_line(aes(group = country)) +
  geom_point(aes(colour = country))
```



3.3.1 Development Version Pivoting

Note: this section discusses the development functions `pivot_wider()` and `pivot_longer()`. These functions are not yet available in the production version of `tidyr`. See section 3.3.2 (page 62) for currently working code.

Unfortunately, most of the data you will find in the "wild" is not tidy. So, we need tools to help us tidy unruly data.

The main tools in `tidyr` are the ideas of `pivot_longer()` and `pivot_wider()`. As the names imply, `pivot_longer()` "lengthens" our data, increasing the number of rows and decreasing the number of columns. `pivot_wider` does the opposite, increasing the number of columns and decreasing the number of rows.

These two functions resolve one of two common problems:

1. One variable might be spread across multiple columns. (`pivot_longer()`)
2. One observation might be scattered across multiple rows. (`pivot_wider()`)

A common issue with data is when values are used as column names.

```
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
```

```
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

We can fix this using `pivot_longer()`.

```
table4a %>%
  pivot_longer(-country, names_to = "year", values_to = "cases")
```

Notice we specified with columns we wanted to consolidate by telling the function the column we *didn't* want to change (`-country`). We can use the `dplyr::select()` syntax here for specifying the columns to pivot.

We can do the same thing with `table4b` and then **join** the databases together by specifying unique identifying attributes.

```
table4a %>%
  pivot_longer(-country, names_to = "year", values_to = "cases") %>%
  left_join(table4b %>% pivot_longer(-country, names_to = "year",
 values_to = "population"))
```

If, instead, variables don't have their own column, we can `pivot_wider()`.

```
table2

table2 %>%
  pivot_wider(names_from = type, values_from = count)
```

3.3.2 Spread and Gather

Note: this section discusses the soon-to-be-deprecated functions `spread()` and `gather()`. These functions will soon be replaced by `pivot_wider()` and `pivot_longer()`. See section 3.3.1 (page 61) for code when this happens.

Unfortunately, most of the data you will find in the "wild" is not tidy. So, we need tools to help us tidy unruly data.

The main tools in `tidyr` are the ideas of `spread()` and `gather()`. `gather()` "lengthens" our data, increasing the number of rows and decreasing the number of columns.

`spread()` does the opposite, increasing the number of columns and decreasing the number of rows.

These two functions resolve one of two common problems:

1. One variable might be spread across multiple columns. (`gather()`)
2. One observation might be scattered across multiple rows. (`spread()`)

A common issue with data is when values are used as column names.

```
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

We can fix this using `gather()`.

```
table4a %>%
  gather(-country, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Brazil      1999   37737
## 3 China       1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil      2000   80488
## 6 China       2000  213766
```

Notice we specified with columns we wanted to consolidate by telling the function the column we *didn't* want to change (`-country`). We can use the `dplyr::select()` syntax here for specifying the columns to pivot.

We can do the same thing with `table4b` and then **join** the databases together by specifying unique identifying attributes.

```
table4a %>%
  gather(-country, key = "year", value = "cases") %>%
  left_join(table4b %>% gather(-country, key = "year", value =
  "population"))
```

```
## Joining, by = c("country", "year")
```

```
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>       <chr>  <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Brazil      1999   37737  172006362
## 3 China       1999  212258 1272915272
## 4 Afghanistan 2000    2666   20595360
## 5 Brazil      2000   80488  174504898
## 6 China       2000  213766 1280428583
```

If, instead, variables don't have their own column, we can `spread()`.

```
table2
```

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```
table2 %>%
  spread(key = type, value = count)
```

```
##   <chr>        <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

3.3.3 Separating and Uniting

So far we have tidied `table2` and `table4a` and `table4b`, but what about `table3`?

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

We need to split the `rate` column into the cases and population columns so that each value has its own cell. The function we will use is `separate()`. We need to specify the column, the value to split on ("/"), and the names of the new coumns.

```
table3 %>%
  separate(rate, into = c("cases", "population"), sep = "/")
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

`unite()` is the opposite of `separate()` – it combines multiple columns into a single column.

Your Turn

1. Is the NFL salaries from https://raw.githubusercontent.com/ada-lovecraft/ProcessingSketches/master/Bits%20and%20Pieces/Football_Stuff/data/nfl-salaries.tsv that you loaded into `R` in a previous your turn tidy? Why or why not?

2. There is a data set in `tidyr` called `world_bank_pop` that contains information about population from the World Bank (https://data.worldbank.org/). Why is this data not tidy? You may want to read more about the data to answer (`?world_bank_pop`).

3. Use functions in `tidyr` to turn this into a tidy form.

3.4 Additional resources

`readr` (https://readr.tidyverse.org)

`dplyr` (https://dplyr.tidyverse.org)

`tidyr` (https://tidyr.tidyverse.org)