Data Science

Data Import and Tidy Data

Carley Dziewicki

February 15, 2019

1. Super Bowl Data

Now that it's been a week since the 53rd annual Super Bowl football game and, for those that watched it, your Super Bowl snacks should have digested, it's now time to digest, clean and plot some Super Bowl data. Thanks to Taylor Ashby for bringing to our attention the Super Bowl data set at "data.world." It should not be necessary to be familiar with American football to carry out the procedures requested here but feel free to ask if you have any questions. In any case, here's a link to an overview of the data set (https://data.world/sports/history-of-the-super-bowl). You should **not** need to register to view the description or access the data (even though it may appear *data.world* wants you to do so). The raw data, a comma-separated data file (CSV), can be accessed from this link "https://query.data.world/s/zg4dna5wbbnonds7ewdv7pznh7ou7p (https://query.data.world/s/zg4dna5wbbnonds7ewdv7pznh7ou7p)". This is the link you should use to read in the data in your analysis.

a. The **tidyverse** packages have been loaded in the chunk above so you can go ahead and read in the data. Write a code chunk below that uses `read_csv()` to read in the data from the URL provided. Read it into a data frame called **super_bowl**. Since you are using `read_csv()`, this data frame will be a tibble.

```
library(readr)
super_bowl <- read_csv("data/Super_Bowl.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   Attendance = col_double(),
##   `Winning Pts` = col_double(),
##   `Losing Pts` = col_double(),
##   `Point Difference` = col_double()
## )
```

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

```
#View(super_bowl)
```

b. You should have seems some red print confirming the column specifications the package **readr** used to import the data. Follow directions and use the `spec(...)` command to see the full columnn specications used to read the data. This output is somewhat lengthy but it clearly shows the formats used as well as indicating which names are “non-syntactic”.

```
spec(super_bowl)
```

```
## cols(
##   Date = col_character(),
##   SB = col_character(),
##   Attendance = col_double(),
##   `QB  Winner` = col_character(),
##   `Coach Winner` = col_character(),
##   Winner = col_character(),
##   `Winning Pts` = col_double(),
##   `QB Loser` = col_character(),
##   `Coach Loser` = col_character(),
##   Loser = col_character(),
##   `Losing Pts` = col_double(),
##   MVP = col_character(),
##   Stadium = col_character(),
##   City = col_character(),
##   State = col_character(),
##   `Point Difference` = col_double(),
##   Referee = col_character(),
##   Umpire = col_character(),
##   `Head Linesman` = col_character(),
##   `Line Judge` = col_character(),
##   `Field Judge` = col_character(),
##   `Back Judge` = col_character(),
##   `Side Judge` = col_character()
## )
```

c. Look at the output of the `spec(...)` command and identify two things. (1) Which variables are automatically imported as numeric? (2) Name the first 3 non-syntactic names.

Answer: The variables that are imported as numeric are attendance, winning and losing points, and point difference. The first tree non-syntactic names are QB Winner, Coach Winner, Winning Points.

d. Let’s first fix up the non-syntactic names. A handy package for this purpose is **janitor** which contains a function called `clean_names()`. Write a chunk below that loads the **janitor** library and uses the function to convert all the non-syntactic names to “snake_case” formate (underscores instead of blanks and all variable names start with lower case). Use the `clean_names()` function to save the tibble with

the cleaned up names back under the names **super_bowl**. Check that none of the names in your data set still have the tick marks used for non-syntactic names. Note also that all variables now start with lower case letters.

```
library(janitor)
super_bowl<-super_bowl %>% clean_names(., "snake")
head(super_bowl)
```

```
## # A tibble: 6 x 23
##   date  sb    attendance qb_winner coach_winner winner winning_pts qb_loser
##   <chr> <chr>      <dbl> <chr>     <chr>        <chr>         <dbl> <chr>
## 1 15-J… I          61946 Bart Sta… Vince Lomba… Green…           35 Len Daw…
## 2 14-J… II         75546 Bart Sta… Vince Lomba… Green…           33 Daryle …
## 3 12-J… III        75389 Joe Nama… Weeb Ewbank  New Y…           16 Earl Mo…
## 4 11-J… IV         80562 Len Daws… Hank Stram   Kansa…           23 Joe Kapp
## 5 17-J… V          79204 Earl Mor… Don McCaffe… Balti…           16 Craig M…
## 6 16-J… VI         81023 Roger St… Tom Landry   Dalla…           24 Bob Gri…
## # ... with 15 more variables: coach_loser <chr>, loser <chr>, losing_pts <dbl>,
## #   mvp <chr>, stadium <chr>, city <chr>, state <chr>, point_difference <dbl>,
## #   referee <chr>, umpire <chr>, head_linesman <chr>, line_judge <chr>,
## #   field_judge <chr>, back_judge <chr>, side_judge <chr>
```

e. Now consider the first column of the data frame. It should be clear to you that this is a *date* but, by default, **readr** imports it as a character string. Use `mutate()` and the `parse_date()` function to cast the variable as a variable of type “date”. Hint: You’ll need to provide a format specifications for the date. Resave the revised data under the name **super_bowl**.

```
super_bowl_2<- mutate(super_bowl, date=parse_date(date, format = "%d-%b-%y"))
glimpse(super_bowl_2)
```

```
## Observations: 52
## Variables: 23
## $ date             <date> 2067-01-15, 2068-01-14, 1969-01-12, 1970-01-11, 1...
## $ sb               <chr> "I", "II", "III", "IV", "V", "VI", "VII", "VIII", ...
## $ attendance       <dbl> 61946, 75546, 75389, 80562, 79204, 81023, 90182, 7...
## $ qb_winner        <chr> "Bart Starr", "Bart Starr", "Joe Namath", "Len Daw...
## $ coach_winner     <chr> "Vince Lombardi", "Vince Lombardi", "Weeb Ewbank",...
## $ winner           <chr> "Green Bay Packers", "Green Bay Packers", "New Yor...
## $ winning_pts      <dbl> 35, 33, 16, 23, 16, 24, 14, 24, 16, 21, 32, 27, 35...
## $ qb_loser         <chr> "Len Dawson", "Daryle Lamonica", "Earl Morrall, Jo...
## $ coach_loser      <chr> "Hank Stram", "John Rauch", "Don Shula", "Bud Gran...
## $ loser            <chr> "Kansas City Chiefs", "Oakland Raiders", "Baltimor...
## $ losing_pts       <dbl> 10, 14, 7, 7, 13, 3, 7, 7, 6, 17, 14, 10, 31, 19, ...
## $ mvp              <chr> "Bart Starr", "Bart Starr", "Joe Namath", "Len Daw...
## $ stadium          <chr> "Memorial Coliseum", "Orange Bowl", "Orange Bowl",...
## $ city             <chr> "Los Angeles", "Miami", "Miami", "New Orleans", "M...
## $ state            <chr> "California", "Florida", "Florida", "Louisiana", "...
## $ point_difference <dbl> 25, 19, 9, 16, 3, 21, 7, 17, 10, 4, 18, 17, 4, 12,...
## $ referee          <chr> NA, NA, NA, NA, "Norm Schachter", "Jim Tunney", "T...
## $ umpire           <chr> NA, NA, NA, NA, "Paul Trepinski", "Joe Connell", "...
## $ head_linesman    <chr> NA, NA, NA, NA, "Ed Marion", "Al Sabato", "Tony Ve...
## $ line_judge       <chr> NA, NA, NA, NA, "Jack Fette", "Art Holst", "Bruce ...
## $ field_judge      <chr> NA, NA, NA, NA, "Fritz Graf", "Bob Wortman", "Tony...
## $ back_judge       <chr> NA, NA, NA, NA, "Hugh Gamber", "Ralph Vandenberg",...
## $ side_judge       <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "D...
```

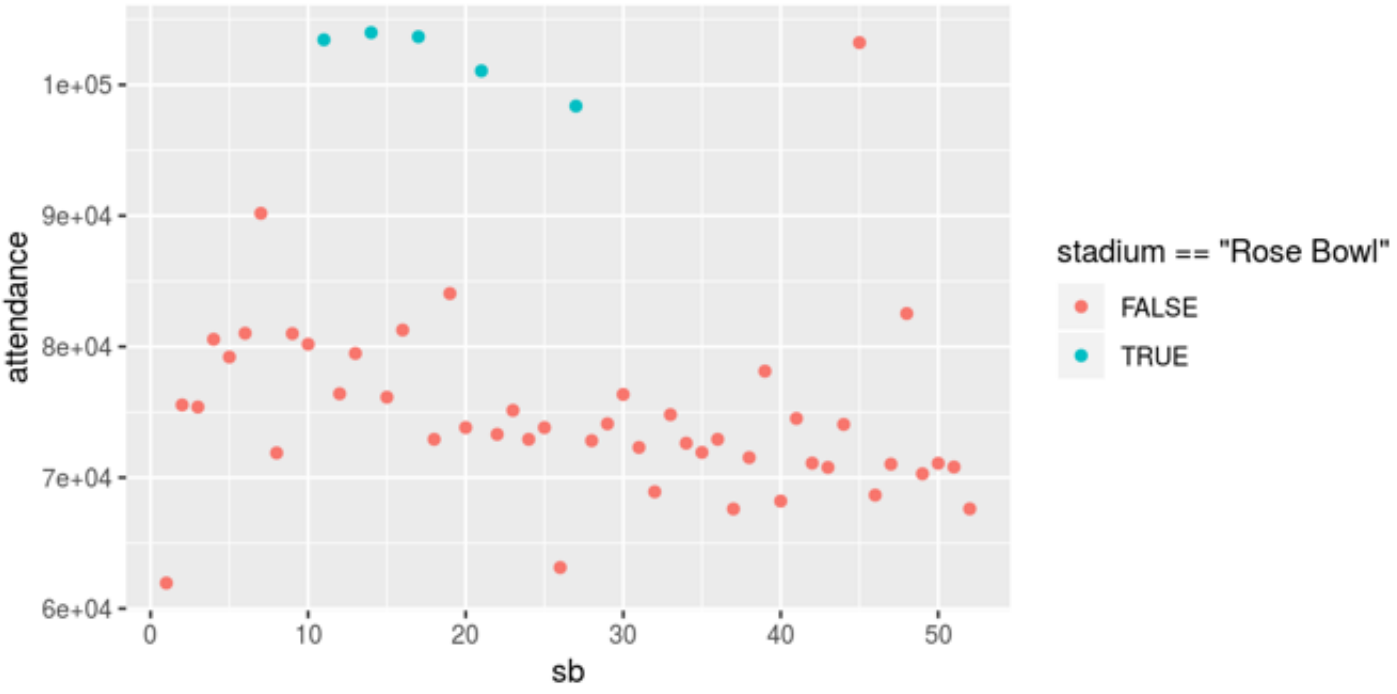
f. The next column contains the variable *sb* indicating a sequence number in Roman numerals representing the Super Bowl in marketing and promition. Use appropriate functions with `mutate()` to convert this variable to a numeric variable *sb* ranging from 1 to 53. This will be a good chance to learn about the functions `as.roman()` and `as.numeric()`.

```
super_bowl<-super_bowl%>% mutate(sb=as.numeric(as.roman(sb)))
super_bowl
```

```
## # A tibble: 52 x 23
##    date     sb attendance qb_winner coach_winner winner winning_pts qb_loser
##    <chr> <dbl>      <dbl> <chr>     <chr>        <chr>         <dbl> <chr>
##  1 15-J…     1      61946 Bart Sta… Vince Lomba… Green…           35 Len Daw…
##  2 14-J…     2      75546 Bart Sta… Vince Lomba… Green…           33 Daryle …
##  3 12-J…     3      75389 Joe Nama… Weeb Ewbank  New Y…           16 Earl Mo…
##  4 11-J…     4      80562 Len Daws… Hank Stram   Kansa…           23 Joe Kapp
##  5 17-J…     5      79204 Earl Mor… Don McCaffe… Balti…           16 Craig M…
##  6 16-J…     6      81023 Roger St… Tom Landry   Dalla…           24 Bob Gri…
##  7 14-J…     7      90182 Bob Grie… Don Shula    Miami…           14 Bill Ki…
##  8 13-J…     8      71882 Bob Grie… Don Shula    Miami…           24 Fran Ta…
##  9 12-J…     9      80997 Terry Br… Chuck Noll   Pitts…           16 Fran Ta…
## 10 18-J…    10      80187 Terry Br… Chuck Noll   Pitts…           21 Roger S…
## # ... with 42 more rows, and 15 more variables: coach_loser <chr>, loser <chr>,
## #   losing_pts <dbl>, mvp <chr>, stadium <chr>, city <chr>, state <chr>,
## #   point_difference <dbl>, referee <chr>, umpire <chr>, head_linesman <chr>,
## #   line_judge <chr>, field_judge <chr>, back_judge <chr>, side_judge <chr>
```

g. Now take a break for a picture. Use **ggplot2** to create a line plot showing *attendance* (y) against the superbowl number (*sb*) on the (x) axis. On top of this lineplot, superimpose a point plot (geom_point()) that colors the points by whether or not the game took place in the “Rose Bowl” *stadium*. What do you notice? Which game now sticks out as out of place? In what stadium was it played? To finish the story, find out in what year the stadium opened.

```
ggplot(super_bowl, mapping=aes(x=sb, y=attendance)) + geom_point(aes(color = stadium
=="Rose Bowl"))
```



h. Now inspect the *qb_winner* variable. You will notice that in some years, there is more than one name listed as the *qb_winner*, the quarterback for the winning team. In this form, the variable is difficult to use. Instead, use the `separate()` command to split this column into two new variables, *qbw_starter* and *qbw_backup* corresponding to the starting quarterback for the winning team and the backup quarterback for the winning team (listed second when more than one winning quarterback is listed). When using the `separate()` command, be sure to specify the separator symbol correctly for these data and use the "fill" option to fill in missing values where needed and avoid unnecessary warning messages. (Check the help on the `separate()` command if necessary). Save the data with the new variables in **super_bowl**.

```
super_bowl2<- super_bowl %>%
  separate(qb_winner, c("qbw_starter", "qbw_backup"), sep=",")
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 50 rows [1, 2, 3,
## 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ...].
```

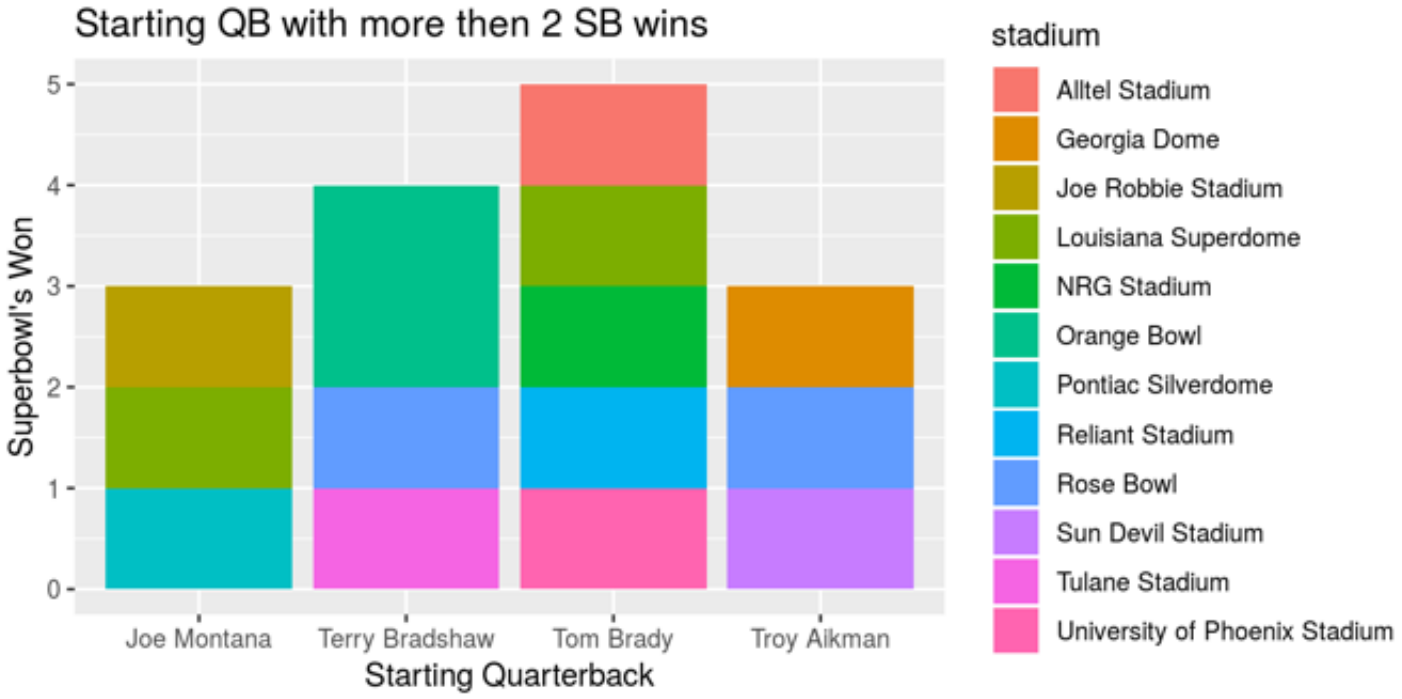
```
super_bowl2
```

```
## # A tibble: 52 x 24
##    date     sb attendance qbw_starter qbw_backup coach_winner winner winning_pts
##    <chr> <dbl>      <dbl> <chr>       <chr>      <chr>         <chr>         <dbl>
##  1 15-J…     1      61946 Bart Starr  <NA>       Vince Lomba… Green…           35
##  2 14-J…     2      75546 Bart Starr  <NA>       Vince Lomba… Green…           33
##  3 12-J…     3      75389 Joe Namath  <NA>       Weeb Ewbank  New Y…           16
##  4 11-J…     4      80562 Len Dawson  " Mike Li… Hank Stram   Kansa…           23
##  5 17-J…     5      79204 "Earl Morr… " Johnny … Don McCaffe… Balti…           16
##  6 16-J…     6      81023 Roger Stau… <NA>       Tom Landry   Dalla…           24
##  7 14-J…     7      90182 Bob Griese  <NA>       Don Shula    Miami…           14
##  8 13-J…     8      71882 Bob Griese  <NA>       Don Shula    Miami…           24
##  9 12-J…     9      80997 Terry Brad… <NA>       Chuck Noll   Pitts…           16
## 10 18-J…    10      80187 Terry Brad… <NA>       Chuck Noll   Pitts…           21
## # ... with 42 more rows, and 16 more variables: qb_loser <chr>,
## #   coach_loser <chr>, loser <chr>, losing_pts <dbl>, mvp <chr>, stadium <chr>,
## #   city <chr>, state <chr>, point_difference <dbl>, referee <chr>,
## #   umpire <chr>, head_linesman <chr>, line_judge <chr>, field_judge <chr>,
## #   back_judge <chr>, side_judge <chr>
```

i. Create a bar chart of the winning starting quarterbacks (*qbw_starter*) of super bowls for all quarterbacks who have won at least 2 super bowls. Color the bars so that each quarterback is a different color. If you like, use the `reorder()` command so that the bars are ordered from highest to lowest but this is not required. Which quarterbacks have won exactly 3 super bowls?

```
super_bowl2<-super_bowl2 %>%
  group_by(qbw_starter) %>%
  mutate(freq = n()) %>%
  ungroup() %>%
  filter(freq > 2) %>%
  select(-freq)

ggplot(super_bowl2, mapping=aes(x=qbw_starter)) +geom_bar(aes(fill=stadium)) +labs(x= "Starting Quarterback", y= "Superbowl's Won", title="Starting QB with more then 2 SB wins")
```



2. BMI Data for Countries

The file “bmi_data.csv” in the data subdirectory contains annual data on BMI for many countries over time.

a. Use the `read_csv()` function to import the data. Take a look at it using, say, `glimpse()` or `View()` or other tools to ensure you understand how the data are structured and what the variable names mean.

```
library(readr)
bmi_data <- read_csv("data/bmi_data.csv")
```

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

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

```
#glimpse(bmi_data)
```

b. The author of our textbook, Hadley Wickham, would say that data that look like this are not “tidy”. Explain why these data do not fit the definition of a tidy data set.
Answer: In a tidy table, variable are in columns, observations are in rows and values are in cells

c. Use either the `gather()` or `spread()` function to reshape this data set so that it contains the same values but has 3 variables (columns): *Country*, *year* and *bmi*. Call this new data set `bmi_tidy` . What are the variable “types” associated with the three columns?

```
bmi_data_tidy <- bmi_data %>% gather(key = "year", value = "bmi",  `Y1980`:`Y2008`)
bmi_data_tidy
```

```
## # A tibble: 5,771 x 3
##    Country             year    bmi
##    <chr>               <chr> <dbl>
##  1 Afghanistan         Y1980  21.5
##  2 Albania             Y1980  25.2
##  3 Algeria             Y1980  22.3
##  4 Andorra             Y1980  25.7
##  5 Angola              Y1980  20.9
##  6 Antigua and Barbuda Y1980  23.3
##  7 Argentina           Y1980  25.4
##  8 Armenia             Y1980  23.8
##  9 Australia           Y1980  24.9
## 10 Austria             Y1980  24.8
## # ... with 5,761 more rows
```

d. Use an appropriate parsing function combined with mutate to convert the *year* variable from character to numeric. Store the new dataset under the name `bmi_tidy` .

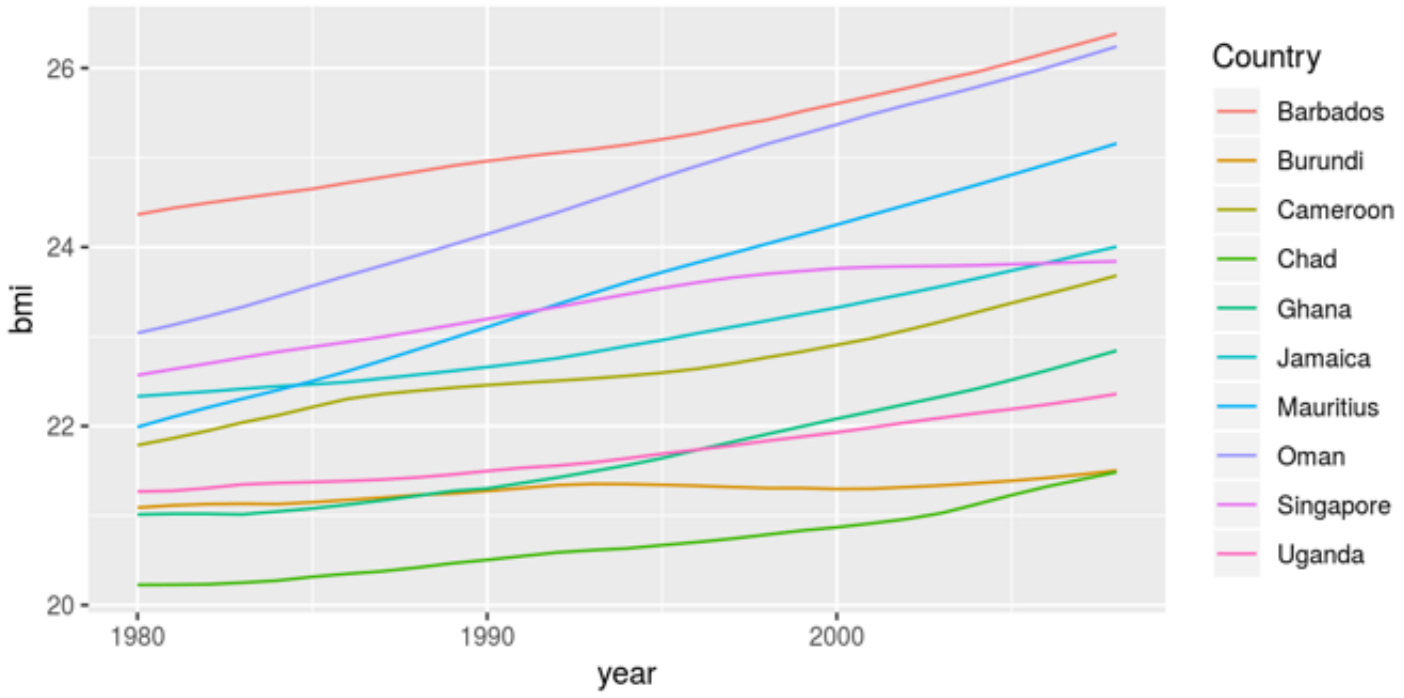
```
bmi_tidy2<-bmi_data_tidy %>% mutate(year = parse_number(year))
bmi_tidy2
```

```
## # A tibble: 5,771 x 3
##    Country              year   bmi
##    <chr>               <dbl> <dbl>
##  1 Afghanistan          1980  21.5
##  2 Albania              1980  25.2
##  3 Algeria              1980  22.3
##  4 Andorra              1980  25.7
##  5 Angola               1980  20.9
##  6 Antigua and Barbuda  1980  23.3
##  7 Argentina            1980  25.4
##  8 Armenia              1980  23.8
##  9 Australia            1980  24.9
## 10 Austria              1980  24.8
## # ... with 5,761 more rows
```

e. Use an appropriate function from the Data Transformation Cheat Sheet to draw a random sample of 10 countries and construct line plots over time of the BMI variable. Rather than sample from the `bmi_tidy` data set, sample the original `bmi_tidy` data set and use the pipe operator to tidy the data and construct the line plot all in one chunk. You may use different colors for each of the 10 countries if you prefer a colorful plot. If you have completed this step correctly, you would get a new random set of 10 countries each time you run this chunk (or knit the document). (If you prefer getting the same results each time, look at documentation for or Google `set.seed()` and then put a line at the beginning of this chunk setting the seed for the random number generator.)

```
linedata<-sample_n(bmi_data,10, replace=FALSE)
linedata2<-linedata  %>% gather(key = "year", value = "bmi",  `Y1980`:`Y2008`) %>% mu
tate(year=parse_number(year))

ggplot(linedata2, mapping=aes(x=year, y=bmi, group=Country))+ geom_line(aes(color=Cou
ntry))
```



3. British Baking Data Set

The *Great British Bake Off* is a very popular long-running British baking competition TV program that has now been running for 9 seasons, airing first in the UK and later gaining popularity in the US. Viewership ratings data and other information about the program are provided in the data set "gbbo_ratings.csv" found in the `data` subdirectory of your workspace.

a. Use data tidying commands to tidy the data into a form that will allow you to construct a plot of 7-day viewership numbers (variables that end with "7day") against the episode number. Note that the variable name "e1_viewers_7day" provides information on the 7day viewership (millions who watched within 7 days of airing) of the 1st episode of the season. Use parsing commands rather than brute force wherever possible. Your plot should have the following characteristics:

- Line plot with episode number on the x axis and number of viewers on the y axis;
- Maps values of the variable *channel* to the color aesthetic;
- Has informative labels.

```
#gather 7 day of airing
gbbo_data <- read_csv("data/gbbo_ratings.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   premiere = col_character(),
##   finale = col_character(),
##   winner = col_character(),
##   day_of_week = col_character(),
##   timeslot = col_time(format = ""),
##   channel = col_character(),
##   runner_up_1 = col_character(),
##   runner_up_2 = col_character(),
##   season_premiere = col_character(),
##   season_finale = col_character(),
##   e1_uk_airdate = col_character(),
##   e2_uk_airdate = col_character(),
##   e3_uk_airdate = col_character(),
##   e4_uk_airdate = col_character(),
##   e5_uk_airdate = col_character(),
##   e6_uk_airdate = col_character(),
##   e7_uk_airdate = col_character(),
##   e8_uk_airdate = col_character(),
##   e9_uk_airdate = col_character(),
##   e10_uk_airdate = col_character()
## )
```

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

```
head(gbbo_data)
```

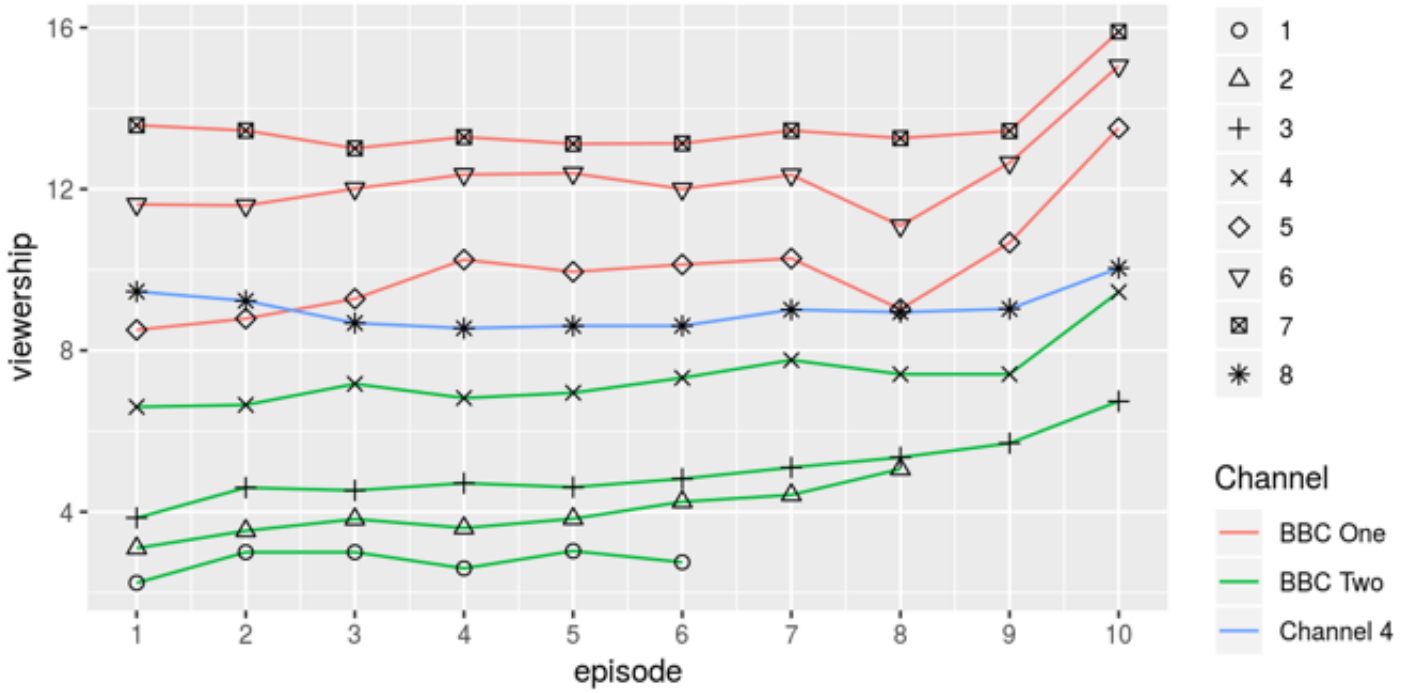
```
## # A tibble: 6 x 44
##   series episodes premiere finale winner avg_uk_viewers day_of_week timeslot
##    <dbl>    <dbl> <chr>    <chr>  <chr>           <dbl> <chr>       <time>
## 1      1        6 17-Aug-… 21-Se… Edd K…           2.77 Tuesday     20:00
## 2      2        8 14-Aug-… 4-Oct… Joann…           4    Tuesday     20:00
## 3      3       10 14-Aug-… 16-Oc… John …           5    Tuesday     20:00
## 4      4       10 20-Aug-… 22-Oc… Franc…           7.35 Tuesday     20:00
## 5      5       10 6-Aug-14 8-Oct… Nancy…          10.0  Wednesday   20:00
## 6      6       10 5-Aug-15 7-Oct… Nadiy…          12.5  Wednesday   20:00
## # ... with 36 more variables: channel <chr>, runner_up_1 <chr>,
## #   runner_up_2 <chr>, season <dbl>, season_premiere <chr>,
## #   season_finale <chr>, e1_viewers_7day <dbl>, e1_viewers_28day <dbl>,
## #   e2_viewers_7day <dbl>, e2_viewers_28day <dbl>, e3_viewers_7day <dbl>,
## #   e3_viewers_28day <dbl>, e4_viewers_7day <dbl>, e4_viewers_28day <dbl>,
## #   e5_viewers_7day <dbl>, e5_viewers_28day <dbl>, e6_viewers_7day <dbl>,
## #   e6_viewers_28day <dbl>, e7_viewers_7day <dbl>, e7_viewers_28day <dbl>,
## #   e8_viewers_7day <dbl>, e8_viewers_28day <dbl>, e9_viewers_7day <dbl>,
## #   e9_viewers_28day <dbl>, e10_viewers_7day <dbl>, e10_viewers_28day <dbl>,
## #   e1_uk_airdate <chr>, e2_uk_airdate <chr>, e3_uk_airdate <chr>,
## #   e4_uk_airdate <chr>, e5_uk_airdate <chr>, e6_uk_airdate <chr>,
## #   e7_uk_airdate <chr>, e8_uk_airdate <chr>, e9_uk_airdate <chr>,
## #   e10_uk_airdate <chr>
```

```
gbbo_tidy <- select(gbbo_data, series, avg_uk_viewers, channel, ends_with("7day"))

head(gbbo_data)
```

```
## # A tibble: 6 x 44
##   series episodes premiere finale winner avg_uk_viewers day_of_week timeslot
##    <dbl>    <dbl> <chr>    <chr>  <chr>           <dbl> <chr>       <time>
## 1      1        6 17-Aug-… 21-Se… Edd K…           2.77 Tuesday     20:00
## 2      2        8 14-Aug-… 4-Oct… Joann…           4    Tuesday     20:00
## 3      3       10 14-Aug-… 16-Oc… John …           5    Tuesday     20:00
## 4      4       10 20-Aug-… 22-Oc… Franc…           7.35 Tuesday     20:00
## 5      5       10 6-Aug-14 8-Oct… Nancy…          10.0  Wednesday   20:00
## 6      6       10 5-Aug-15 7-Oct… Nadiy…          12.5  Wednesday   20:00
## # ... with 36 more variables: channel <chr>, runner_up_1 <chr>,
## #   runner_up_2 <chr>, season <dbl>, season_premiere <chr>,
## #   season_finale <chr>, e1_viewers_7day <dbl>, e1_viewers_28day <dbl>,
## #   e2_viewers_7day <dbl>, e2_viewers_28day <dbl>, e3_viewers_7day <dbl>,
## #   e3_viewers_28day <dbl>, e4_viewers_7day <dbl>, e4_viewers_28day <dbl>,
## #   e5_viewers_7day <dbl>, e5_viewers_28day <dbl>, e6_viewers_7day <dbl>,
## #   e6_viewers_28day <dbl>, e7_viewers_7day <dbl>, e7_viewers_28day <dbl>,
## #   e8_viewers_7day <dbl>, e8_viewers_28day <dbl>, e9_viewers_7day <dbl>,
## #   e9_viewers_28day <dbl>, e10_viewers_7day <dbl>, e10_viewers_28day <dbl>,
## #   e1_uk_airdate <chr>, e2_uk_airdate <chr>, e3_uk_airdate <chr>,
## #   e4_uk_airdate <chr>, e5_uk_airdate <chr>, e6_uk_airdate <chr>,
## #   e7_uk_airdate <chr>, e8_uk_airdate <chr>, e9_uk_airdate <chr>,
## #   e10_uk_airdate <chr>
```

```
gbbo_data %>%
  select(series, channel, ends_with("_7day")) %>%
  gather(key = "key", value = "viewership", ends_with("_7day"), na.rm = TRUE) %>%
  separate(key, into = c("episode"), extra = "drop") %>%
  mutate(episode = parse_number(episode)) %>%
  ggplot(aes(x= episode, y = viewership)) +
  geom_line(aes(group = series, color = channel)) +
  geom_point(aes(shape = factor(series)), size = 2) +
  scale_x_continuous(breaks = 1:10) +
  scale_color_discrete(name = "Channel") +
  scale_shape_manual(name = "Series", values = 1:8)
```




```
# faceting instead of shapes is also fine.
```

b. In the 8th season, the program was moved to the BBC network to a different network. What evidence, if any, do you find in your plot that this move affected the ratings?
Answer: We can see from the plot the the viewership dropped during the eighth season, when the viewership had been increasing.