Homework 7

Sarah Cannon

10/22/2024

```
library(tidyverse)
library(nycflights13)
```

Question 1 (12 points)

a. **Consider the following list of temperatures. Make this list into a factor with a logical ordering, and then sort the list.**

```
temperature_measurements <- c("Cold", "Cold", "Very Cold", "Hot", "Warm", "Cool", "Mediu
m", "Hot", "Very Hot", "Medium", "Warm", "Warm", "Cold", "Cold", "Hot")
```

```
temperature_levels <- c("Very Cold", "Cold", "Cool", "Medium", "Warm", "Hot", "Very Ho
t")
temperature_factor <- factor(temperature_measurements, levels = temperature_levels, orde
red = TRUE)
sorted_temperatures <- sort(temperature_factor)
sorted_temperatures
```

```
##  [1] Very Cold Cold      Cold      Cold      Cold      Cool      Medium    Medium
Warm
## [10] Warm      Warm      Hot       Hot       Hot       Very Hot
## Levels: Very Cold < Cold < Cool < Medium < Warm < Hot < Very Hot
```

b. **Import the temps_cat.csv data set as a tibble. Identify the rows where there are strings that do not fall into one of the seven categories from the previous part. What rows do these values fall on?**

```
temps_cat <- read_csv("temps_cat.csv")
```

```
## Rows: 100 Columns: 2
## ── Column specification ────────────────────────────────────────────────────
──────────────
## Delimiter: ","
## chr (1): Temp
## dbl (1): Index
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
invalid_rows <- temps_cat %>% filter(!Temp %in% temperature_levels)
invalid_rows
```

```
## # A tibble: 6 × 2
##   Index Temp
##   <dbl> <chr>
## 1    28 hot
## 2    29 Medum
## 3    65 cold
## 4    67 Coool
## 5    74 Very cold
## 6    83 Medum
```

c. **Use code to correct the mistakes/typos you found in the previous part. (Since it's been a while, I'll remind you that, for example, to set the seventh value in col1 of tibble t to be 11, you could do: t$col1[7] <- 11)**

```
# Correct the typos in the Temp column
temps_cat$Temp[temps_cat$Index == 28] <- "Hot"
temps_cat$Temp[temps_cat$Index == 29] <- "Medium"
temps_cat$Temp[temps_cat$Index == 65] <- "Cold"
temps_cat$Temp[temps_cat$Index == 67] <- "Cool"
temps_cat$Temp[temps_cat$Index == 74] <- "Very Cold"
temps_cat$Temp[temps_cat$Index == 83] <- "Medium"
```

d. **Now that you've fixed the mistakes/typos, make the Temp column into a factor with a logical ordering, and sort your tibble according to the Temp column.**

```
temps_cat <- temps_cat %>%
  mutate(Temp = factor(Temp, levels = temperature_levels, ordered = TRUE)) %>%
  arrange(Temp)
temps_cat
```

```
## # A tibble: 100 × 2
##    Index Temp
##    <dbl> <ord>
##  1     3 Very Cold
##  2     4 Very Cold
##  3    23 Very Cold
##  4    24 Very Cold
##  5    27 Very Cold
##  6    31 Very Cold
##  7    32 Very Cold
##  8    35 Very Cold
##  9    43 Very Cold
## 10    45 Very Cold
## # ℹ 90 more rows
```

Question 2 (21 points)

a. (3 points) **Import the data set games_cat.csv and turn the Category column into a factor using parse_factor() without specifying any levels.**

```
games_cat <- read_csv("games_cat.csv", skip = 5)
```

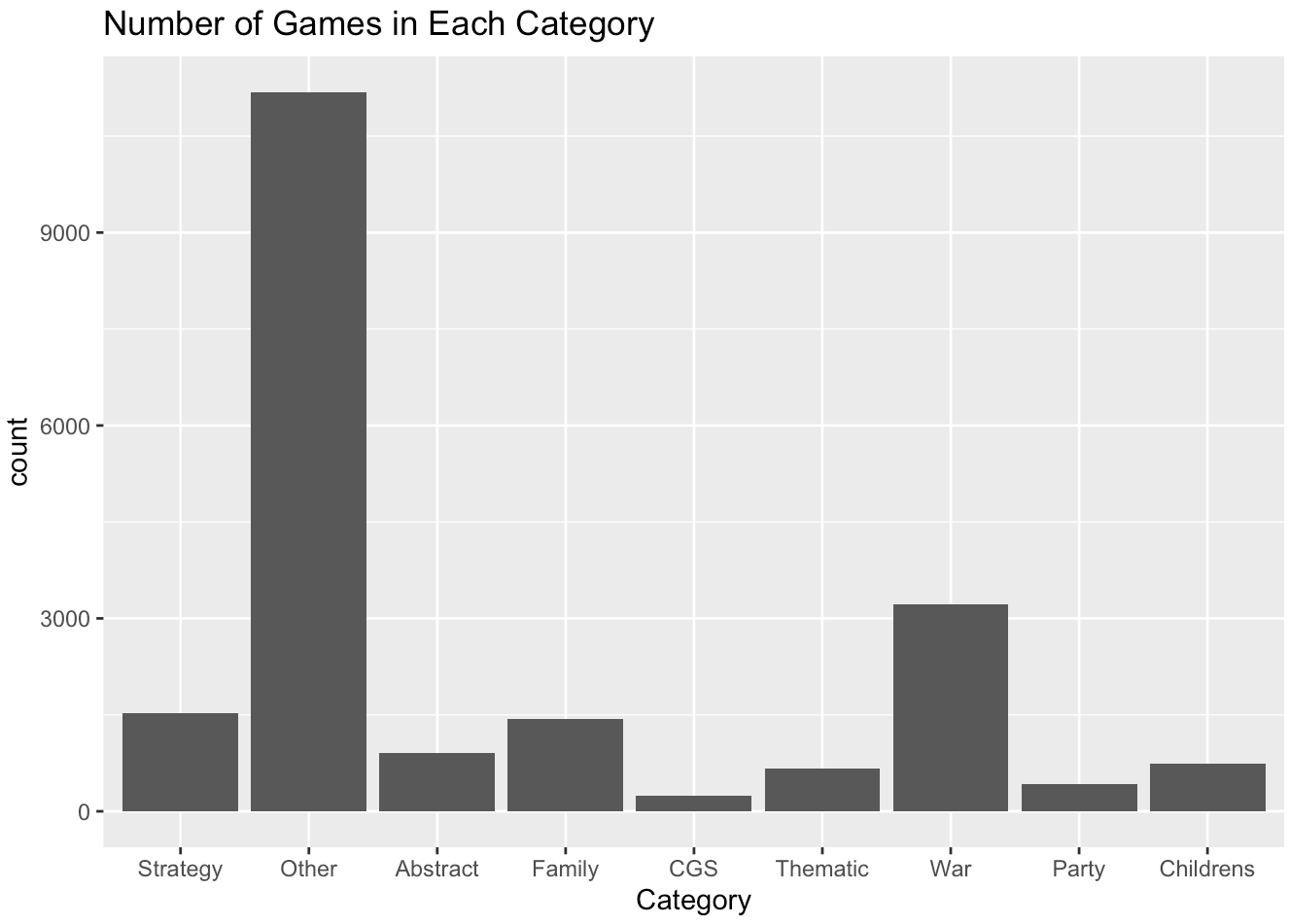
```
## Rows: 20351 Columns: 4
## ── Column specification ────────────────────────────────────────────────────
────────────────
## Delimiter: ","
## chr (2): Name, Category
## dbl (2): BGGId, AvgRating
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
games_cat <- games_cat %>% mutate(Category = parse_factor(Category))
games_cat
```

```
## # A tibble: 20,351 × 4
##    BGGId Name                AvgRating Category
##    <dbl> <chr>                   <dbl> <fct>
##  1     1 Die Macher               7.61 Strategy
##  2     2 Dragonmaster             6.65 Strategy
##  3     3 Samurai                  7.46 Strategy
##  4     4 Tal der Könige           6.60 Other
##  5     5 Acquire                  7.34 Strategy
##  6     6 Mare Mediterraneum       6.55 Other
##  7     7 Cathedral                6.52 Abstract
##  8     8 Lords of Creation        6.11 Other
##  9     9 El Caballero             6.45 Strategy
## 10    10 Elfenland                6.70 Family
## # ℹ 20,341 more rows
```

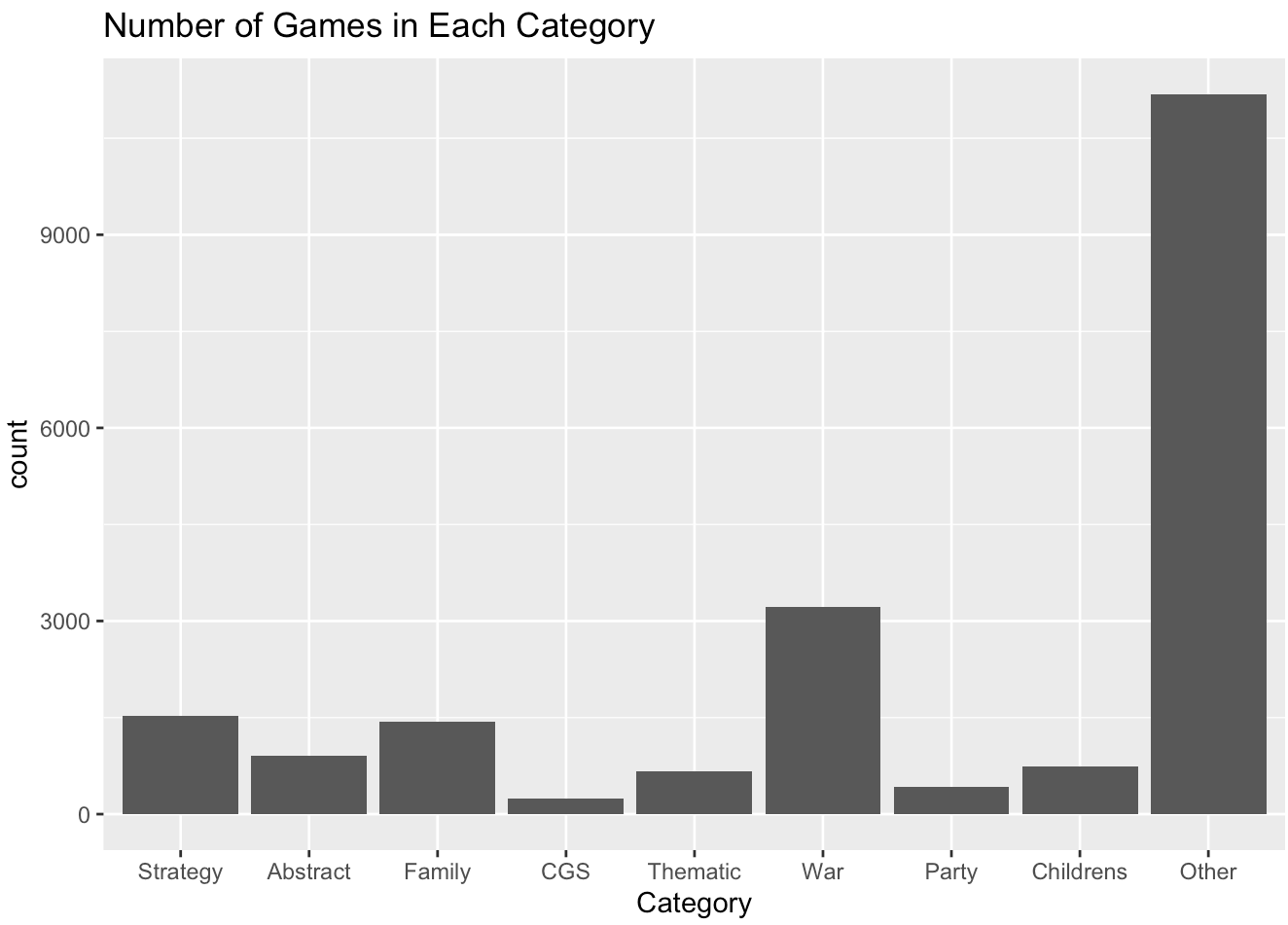
b. (3 points) **Make a bar chart showing the number of games in each of the categories. Make sure you are using the new factor column you made in the previous part. Explain why you see the order of the bars that you do.**

```
ggplot(games_cat) +
  geom_bar(aes(x = Category)) +
  labs(title = "Number of Games in Each Category")
```



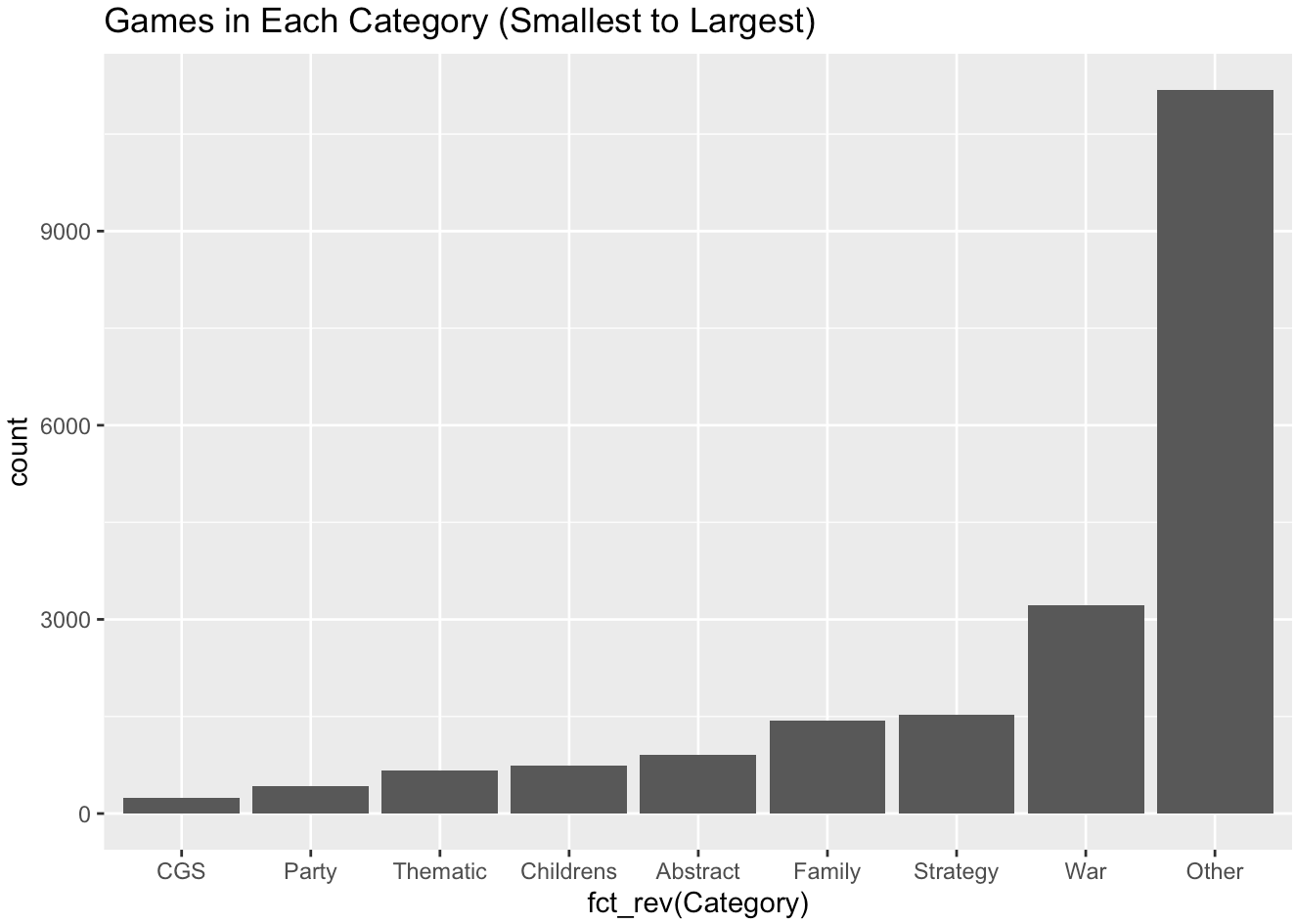
c. (3 points) **The "Other" Category is different from the remaining categories. Write code to produce the same bar chart as the previous part showing the number of games in each of the categories, but the Other category should be either the first bar or the last bar rather than mixed in with the other categories.**

```
games_cat <- games_cat %>%
  mutate(Category = fct_relevel(Category, "Other", after = Inf))
ggplot(games_cat) +
  geom_bar(aes(x = Category)) +
  labs(title = "Number of Games in Each Category")
```



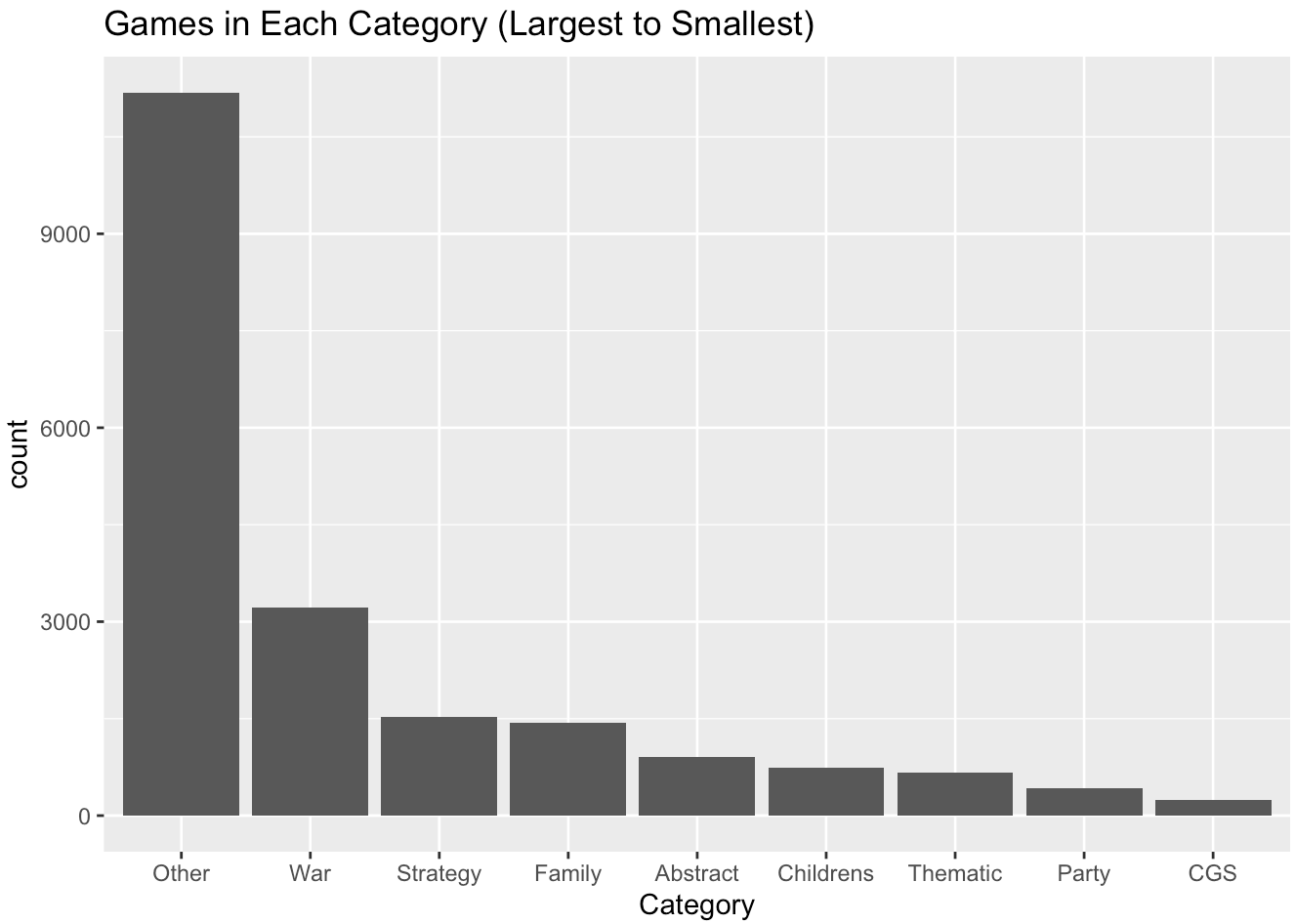
d. (3 points) **Make a new bar chart showing how many games are in each of the different categories where the bars appear in order from smallest to largest.**

```
games_cat <- games_cat %>%
  mutate(Category = fct_infreq(Category))
ggplot(games_cat) +
  geom_bar(aes(x = fct_rev(Category))) +
  labs(title = "Games in Each Category (Smallest to Largest)")
```



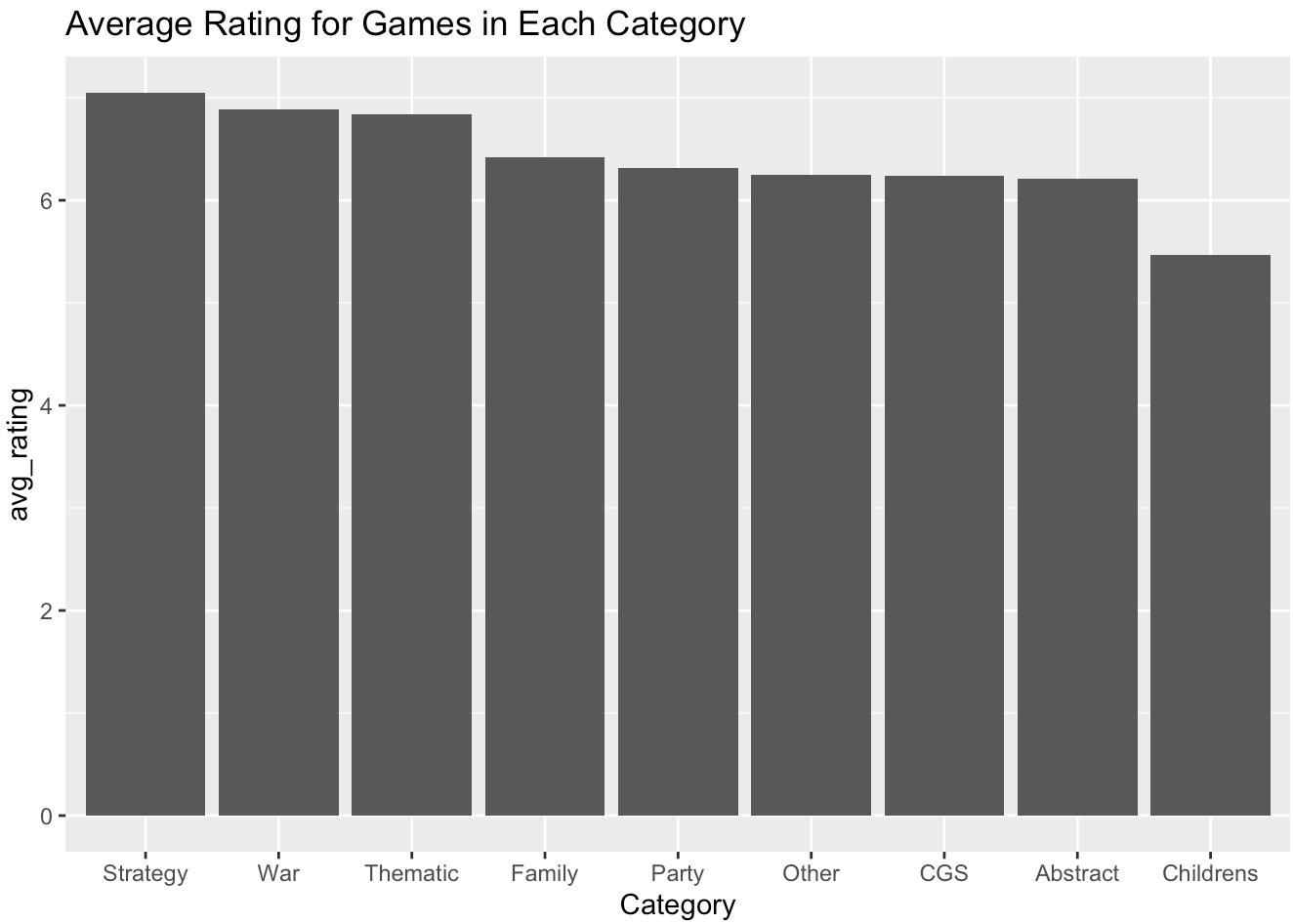
e. (3 points) **Make a new bar chart showing how many games are in each of the different categories where the bars appear in order from largest to smallest (the opposite order from the previous part).**

```
ggplot(games_cat) +
  geom_bar(aes(x = Category)) +
  labs(title = "Games in Each Category (Largest to Smallest)")
```



f. (6 points) **Make a bar chart showing the average rating for games in each category. Your bars should be in order from largest to smallest.**

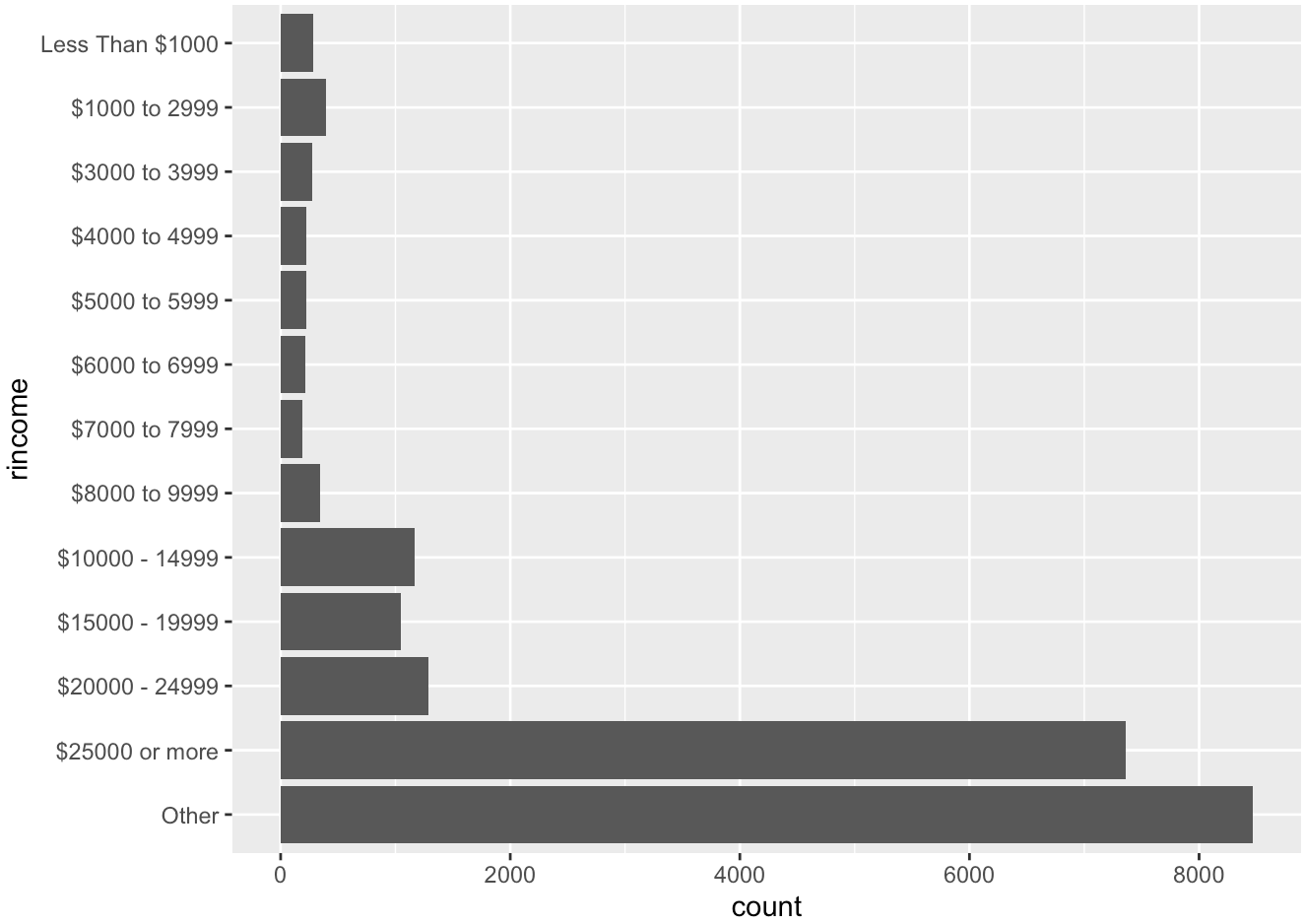
```
games_avg_rating <- games_cat %>%
  group_by(Category) %>%
  summarize(avg_rating = mean(AvgRating, na.rm = TRUE)) %>%
  mutate(Category = fct_reorder(Category, avg_rating, .desc = TRUE))

ggplot(games_avg_rating) +
  geom_bar(aes(x = Category, y = avg_rating), stat = "identity") +
  labs(title = "Average Rating for Games in Each Category")
```



Question 3 (8 points)

Consider the following bar chart showing income levels in the gss_cat data set.

```
gss_cat %>%
  ggplot() +
  geom_bar(mapping = aes(x = rincome)) +
  coord_flip()
```



a. (2 points) **First, explain in your own words what the two functions fct_recode and fct_collapse do, and what the difference is between them.**

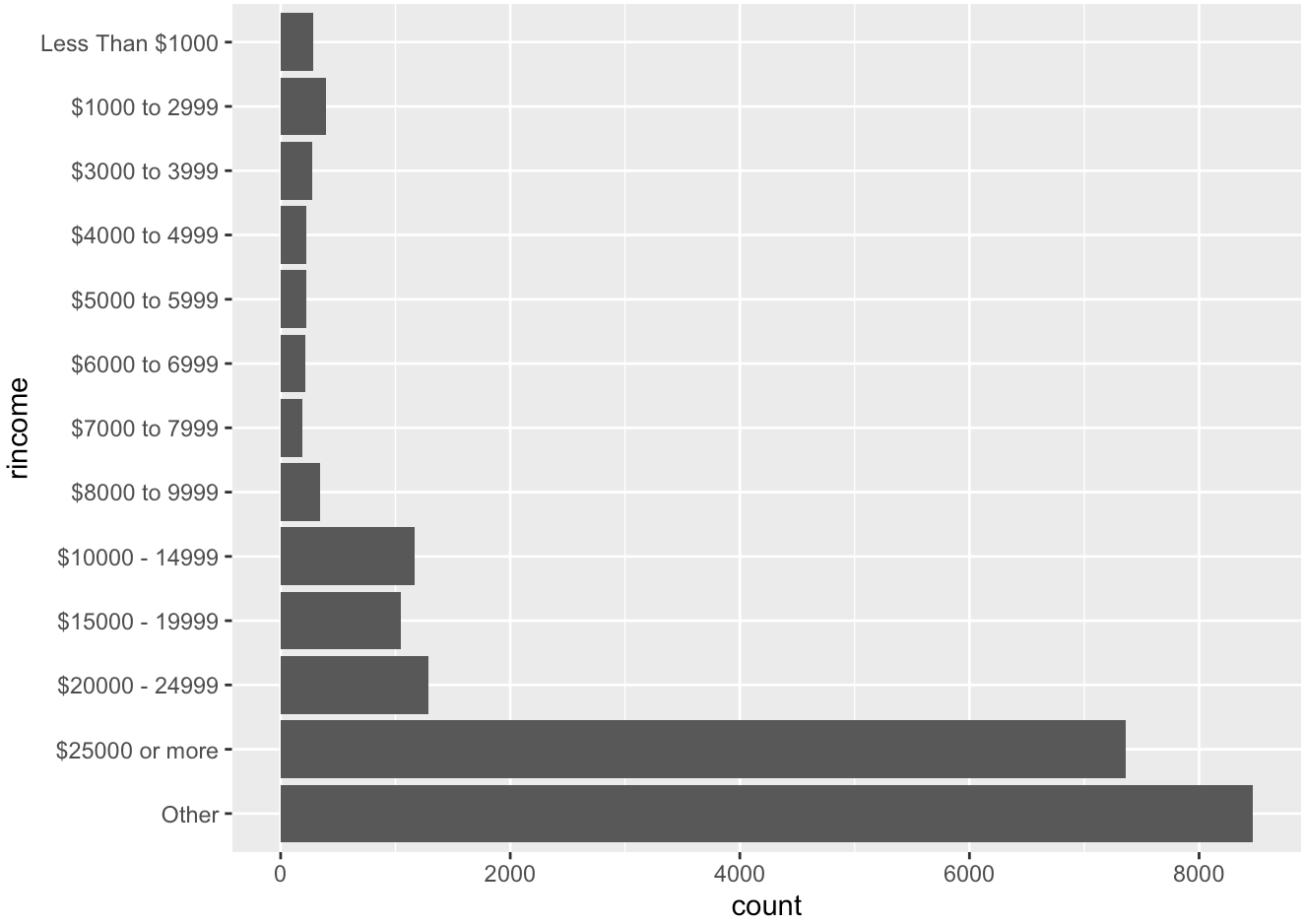
fct_recode allows you to rename levels of a factor, and can also combine multiple levels into one. fct_collapse is specifically for combining multiple levels into one, and is more convenient for this purpose.

b. (3 points) **Rename the level "Lt $1000" to "Less Than $1000", and reproduce the plot at the start of this question with this correction.**

```
gss_cat <- gss_cat %>%
  mutate(rincome = fct_recode(rincome, "Less Than $1000" = "Lt $1000"))
```

```
## Warning: There was 1 warning in `mutate()`.
## ℹ In argument: `rincome = fct_recode(rincome, `Less Than $1000` = "Lt $1000")`.
## Caused by warning:
## ! Unknown levels in `f`: Lt $1000
```

```
gss_cat %>%
  ggplot() +
  geom_bar(mapping = aes(x = rincome)) +
  coord_flip()
```

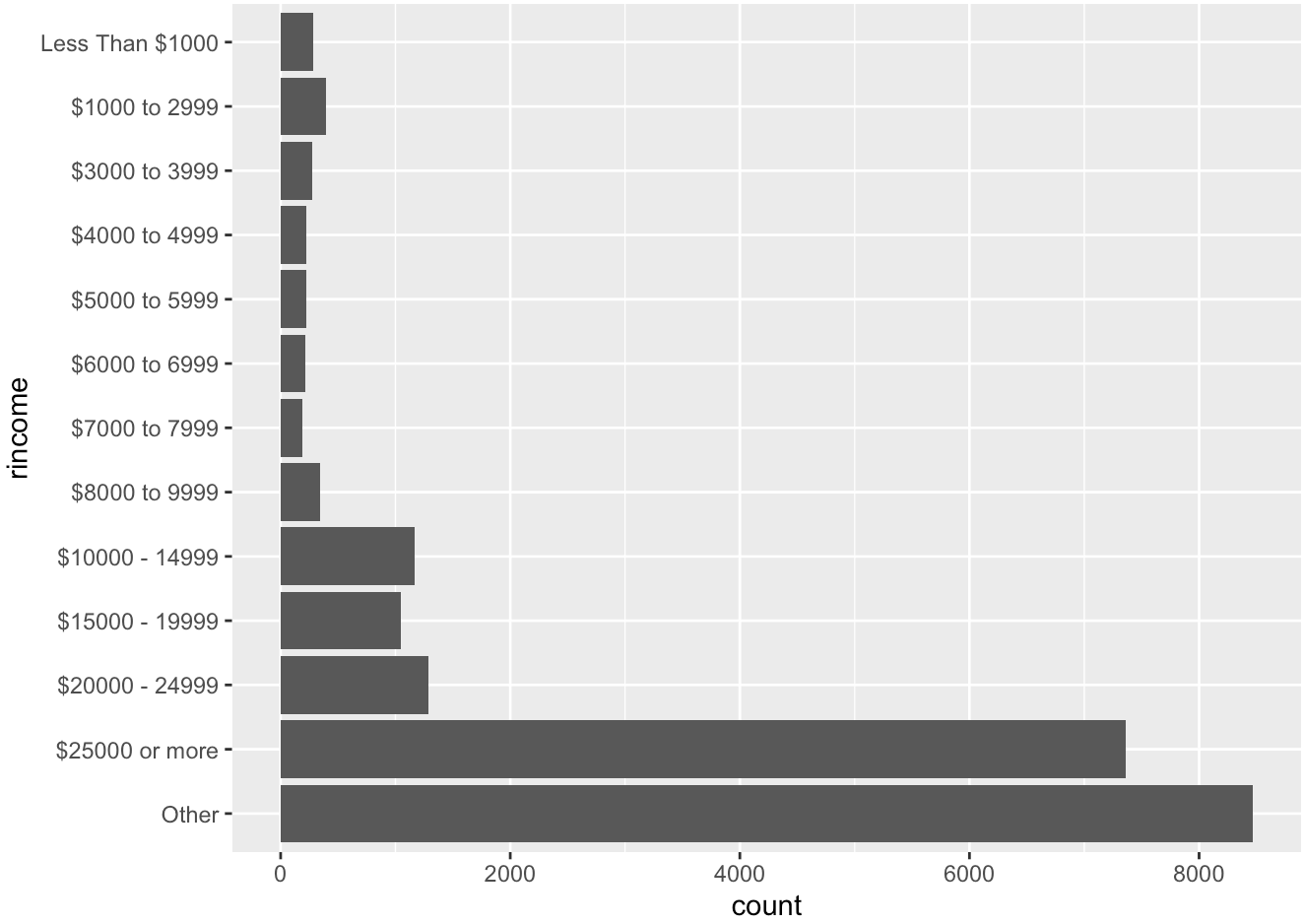


c. (3 points) **Combine the four categories that are not numeric ranges (Not applicable, Refused, Don't know, No answer) into a single level called "Other", and remake the plot with this change.**

```
gss_cat <- gss_cat %>%
  mutate(rincome = fct_collapse(rincome,
                                Other = c("Not applicable", "Refused", "Don't know", "No
answer")))
```

```
## Warning: There was 1 warning in `mutate()`.
## ℹ In argument: `rincome = fct_collapse(...)`.
## Caused by warning:
## ! Unknown levels in `f`: Not applicable, Refused, Don't know, No answer
```

```
gss_cat %>%
  ggplot() +
  geom_bar(mapping = aes(x = rincome)) +
  coord_flip()
```



Question 4 (9 points)

a. (3 points) **Using the flights data set, replace all airlines (in the carrier column) that have fewer than 40000 flights in the data set with a new value "Other Airline".**

```
flights <- flights %>%
  group_by(carrier) %>%
  mutate(carrier = fct_lump_min(carrier, 40000, other_level = "Other Airline"))
```

b. (3 points) **Explain, in your own words, why the following two lines of code result in 8 rows in one case and 9 rows in the other. (Hint: It has nothing do do with the fact that the data type of dest is and the data type of relig is).**

```
flights %>% mutate(dest2 = fct_lump_n(dest, 8)) %>% count(dest2)
```

```
## # A tibble: 45 × 3
## # Groups:   carrier [5]
##    carrier       dest2     n
##    <fct>         <fct> <int>
##  1 Other Airline BOS    6652
##  2 Other Airline CLT   10825
##  3 Other Airline DCA    7984
##  4 Other Airline DFW    7636
##  5 Other Airline LAX    6162
##  6 Other Airline MIA    7234
##  7 Other Airline ORD    9392
##  8 Other Airline RDU    5706
##  9 Other Airline Other 59602
## 10 B6            BOS    4383
## # ℹ 35 more rows
```

```
gss_cat %>% mutate(relig2 = fct_lump_n(relig, 8)) %>% count(relig2)
```

```
## # A tibble: 8 × 2
##   relig2                      n
##   <fct>                   <int>
## 1 Inter-nondenominational   109
## 2 Christian                 689
## 3 Buddhism                  147
## 4 None                     3523
## 5 Jewish                    388
## 6 Catholic                 5124
## 7 Protestant              10846
## 8 Other                     657
```

The difference in the number of rows is due to the number of unique levels in the 'dest' and 'relig' columns. 'fct_lump_n' keeps the most frequent n levels and lumps the rest into "Other". The number of unique levels in 'dest' and 'relig' after lumping determines the number of rows.

> c. (3 points) **Consider the following summary table that describes how many times different categories appear in a larger data set:**

```
c <- tibble(Category = parse_factor(c("A", "B", "C", "D", "E", "F", "G", "H")),
            count = c(20,67,32,45,65,25,84,24))
c
```

```
## # A tibble: 8 × 2
##   Category count
##   <fct>    <dbl>
## 1 A           20
## 2 B           67
## 3 C           32
## 4 D           45
## 5 E           65
## 6 F           25
## 7 G           84
## 8 H           24
```

You attempt to reorder the categories by how frequently they appear in the larger data set as follows:

```
c %>% mutate(Category2 = fct_infreq(Category))
```

```
## # A tibble: 8 × 3
##   Category count Category2
##   <fct>    <dbl> <fct>
## 1 A           20 A
## 2 B           67 B
## 3 C           32 C
## 4 D           45 D
## 5 E           65 E
## 6 F           25 F
## 7 G           84 G
## 8 H           24 H
```

However, when you then try to sort by this new column, it doesn't sort the categories by count:

```
c %>% mutate(Category2 = fct_infreq(Category)) %>%
  arrange(Category2)
```

```
## # A tibble: 8 × 3
##   Category count Category2
##   <fct>    <dbl> <fct>
## 1 A           20 A
## 2 B           67 B
## 3 C           32 C
## 4 D           45 D
## 5 E           65 E
## 6 F           25 F
## 7 G           84 G
## 8 H           24 H
```

Explain, in your own words, why this didn't work, and how you should make the Category2 column differently so that, if you sort by Category2, the Categories end up in order from smallest count to largest count. (Note: Your solution should not be to do %>%arrange(count); instead, you should modify how Category2 is made so that %>%arrange(Category2) gives the correct order).

The fct_infreq function orders levels by their frequency in the data, not by the 'count' column. To sort by the 'count' column, you should use fct_reorder with the 'count' column as the second argument.

```
c <- c %>% mutate(Category2 = fct_reorder(Category, count))
c %>% arrange(Category2)
```

```
## # A tibble: 8 × 3
##   Category count Category2
##   <fct>    <dbl> <fct>
## 1 A           20 A
## 2 H           24 H
## 3 F           25 F
## 4 C           32 C
## 5 D           45 D
## 6 E           65 E
## 7 B           67 B
## 8 G           84 G
```