HW 4

Lance Ding

March 2023

Instructions

Write this homework acting as if I don't know what I asked you. For example, don't simply list question numbers for the headings. If you gave this document to someone else who didn't know the assignment, they should be able to understand what you did by reading the headings, code, and accompanying text.

Look to my HW1 and RMarkdown Organization examples for how to write good headings and organize your assignment.

This HW is worth 10 total points. It is adapted from Dr. Martin van der Linden at Emory.

1. Change the author and date fields in the header above to your name and the date.
2. Make sure to load any packages you may need right at the start. Do *NOT* include the `learnr` package, ever, unless you are writing an interactive Tutorial (which you won't do in this) - this will cause problems.
3. Ensure that no chunks have the `include = FALSE` or `echo = FALSE` option, as I want to be able to see *all* your code and output.
4. Brief but descriptive headings and document organization (answers under headings, text near relevant code, brief explanatory text as indicated below, etc.) (1 pt)
5. Load the `fivethirtyeight` R package and the data frames `classic_rock_raw_data` and `classic_rock_song_list`. Explore the data frames a bit and figure out what they contain (you don't need to include this code in your homework).

   First, note that there's an error in the `classic_rock_song_list` file - an Elton John song is listed as being released in 1071, when it was actually 1971. Fix that using `mutate()` and `case_when()` (or another method of your choosing that fixes that ONE entry). Then:

   Add a new column to `classic_rock_raw_data` with the year each song was released using a join. Print the resulting data frame to prove to me you've done it (don't use `head()`, just let it print the first few default number of observations). Avoid ending up with any columns that end in `.x` or `.y`.

   HINT: Make sure you end up with as many observations in the new joined data frame as in the initial `classic_rock_raw_data`. You may need to execute the join on *two or more columns* to do this. Some songs with the same title have more than one artist, so if you join on `song` only you might add observations by creating two different records for each song play with different release years. There is a single column you can use to make this join work, but you may also use multiple columns that identify a unique observation. (3 pts)

5B. **BONUS 2 pts**. This can replace points you lose elsewhere (even for any questions you skip) but cannot raise your total score over 10/10.

This is an exercise to help you see the value of a join like the above.

Roughly speaking, in the U.S. radio stations with a callsign beginning with "K" are located west of the Mississippi River, while those beginning with "W" are located to its east. I want you to visualize the distribution of the release years of the songs played by all these radio stations, split into two categories: west and east. It's up to you to choose a good visualization type that achieves this goal.

The easiest way to do the east/west splitting involves the substr() (or sub-string) function. To extract the first letter of a variable, you would use substr(<VARIABLE NAME>, 1, 1) - if you check the documentation using ?substr, hopefully you can figure out why this is. This should allow you to then create a new variable for west vs. east of the Mississippi, and then you can apply your existing data visualization skills.

Make sure the graph has a title, human-readable axis labels, human-readable legend labels, and NO legend title.

Describe what you see in about 1-2 sentences. Note what you're seeing is a distribution of the release years of the song plays from radio stations, not the years in which these stations released songs (radio stations don't release songs). If you are still confused about this, do a bit more research on the help page for the data frames and/or read the article linked there that they provide the data for.

Final note: you should *not* be getting a result that suggests western radio stations play way more songs than eastern radio stations. The numbers should be roughly equivalent. If you have a result suggesting this, rethink your plot.

6. For the rest of this homework we will be tidying a data set on migrations from the United Nations (UN), which you can find on Canvas. The data set contains the number of residents of different countries and regions of the world who have been classified as "migrants" in various years. (The actual data set you can download from the UN website is really messy - the version on Canvas is already somewhat trimmed down and cleaned.)

   First, import the data (using read_csv() from readr rather than read.csv()). Then drop the Sort order, Notes, Country code, and Type of data (a) columns from the data frame - we don't need them - and rename Major area, region, country or area of destination to area_dest. Note that if you're in a dplyr verb and want to refer to a column/variable name that has a space in it, you must surround the whole name in backticks ". The resulting data frame should have dimensions 237 x 19. (0.5 pts)

7. Ugh, those remaining variable names have all those spaces and weird capitalization and it's just all TOO MUCH. Let's fix that, using the function (that you may not have quite learned yet) janitor::clean_names(). Look here or in Tutorial 10.1 for an introduction. Remember you might need to install this function's package first!

   You can make all variable names snake_case (the good and righteous choice) or camelCase (the devil's coding). Print the first few rows of your result. (0.5 pts)

8. Scan through the data a bit. Does there appear to be any data *missing*? If so, what sort of data is it (just tell me generally, no need to list every single value or anything)? How does that "missing" data seem to be stored (that is, what "value" indicates some data is missing)? Answer in brief text. (0.5 pts)

9. The values for the number of migrants by sex are currently spread across a number of columns. Let's make the data frame more tidy by having the number of migrants in a single column; a row for each country, sex (Male, Female, or Both), and year; and 4 columns for the country/region, sex, year, and number of migrants. **HINT**: You'll need some combination of pivot_longer(), pivot_wider(), separate(), and/or unite() to accomplish this (or if you can figure out another way, that's fine, godspeed!).

   You should end up with a data frame that is 4,266 x 4. (2 pts)

10. Great, now I'd like you to sum up the total number of migrants across all countries in each year in our dataset. -Starts coding- This should be eas…oh, hell. There's a problem we still need to solve.

    i) **Identify, describe in text, and solve it.** HINT: Do those values in n_migrants look quite right? (1 pt)

       We'll learn more about this code later, but here's what you would use, along with a `dplyr` verb, to solve this remaining issue: `as.double(str_replace_all(n_migrants, " ", ""))`.

       I think you should be getting to the point where, even though you haven't seen it before, you should be able to puzzle out on your own what `str_replace_all()` does - at least if I tell you "str" is short for "string." Remember the help pages! But as an extra hint, check what `str_replace_all(string = "apples and bananas", pattern = " ", replacement = "")` does.

    ii) What happened to those missing values from question 7? How did they change/how are they stored now? (0.3 pts)

(You don't actually need to give me the total migrants per year once you fix this issue, it was just a plot device to move us along.)

11. Why did we do any of this? Where was the value?

    I'd like you to now create a (facetted) plot of the number of migrants by binary sex (1 line for males, 1 for females, with apologies to our non-binary friends) over time in Japan, China, and South Sudan. See the value now? (1 pt)

12. One last question: what's up with that South Sudan plot? Trace this back to an issue in our data frame and explain why it looks the way it does. (0.2 pts)

To submit this assignment:

Ideally, knit straight to PDF by changing `html_document` to `pdf_document` in line 5 above. This should work as long as you properly installed LaTeX in Tutorial 0.1. Otherwise:

1. Knit to HTML. An HTML document should open automatically in another RStudio window.
2. Click "Open in Browser" in that HTML document. It should open as a webpage in your default browser (e.g. Chrome).
3. Click Ctrl+P/Command+P, but instead of printing a hard copy on your printer click "Save as PDF."
4. Save and upload that document to Canvas.

A note on PDF formatting: you may notice that long lines of code "fly off the side of the page" when you knit to PDF. To fix this:

If you're on a Windows machine:

- Install the `formatR` package
- Change your `opts_chunk$set` code line to the following: `knitr::opts_chunk$set(echo = TRUE, tidy.opts=list(width.cutoff=80), tidy=TRUE)`

That should force your code to always wrap rather than fly off the edge of the page of a PDF. Note this does not fix issues of, say, plot titles that are too long getting cut off. But it should fix all the errors with your code not wrapping. Happy PDFing!

If you're on a Mac: I don't have an easy solution for you. Try and keep your lines of code under about 80 characters. Feel free to use more vertical lines of code to accomplish this. But don't waste large amounts of time formatting. I'll ask you for clarification if something critical is missing.

——BEGIN ANSWER BELOW———

```
pacman::p_load(tidyverse)
library(fivethirtyeight)
```

```
## Some larger datasets need to be installed separately, like senators and
## house_district_forecast. To install these, we recommend you install the
## fivethirtyeightdata package by running:
## install.packages('fivethirtyeightdata', repos =
## 'https://fivethirtyeightdata.github.io/drat/', type = 'source')
```

```
data(classic_rock_song_list, classic_rock_raw_data)
```

Exploring and Editing `classic_rock_song_list` and `classic_rock_raw_data`

We are going to explore the `classic_rock_song_list` dataset from the `fivethirtyeight` library. First we can take a look at some summary statistics of the data:

```
dim(classic_rock_song_list)
```

```
## [1] 2229    7
```

```
summary(classic_rock_song_list)
```

```
##      song              artist           release_year    combined
##  Length:2229        Length:2229        Min.   :1071   Length:2229
##  Class :character   Class :character   1st Qu.:1971   Class :character
##  Mode  :character   Mode  :character   Median :1977   Mode  :character
##                                        Mean   :1978
##                                        3rd Qu.:1984
##                                        Max.   :2014
##                                        NA's   :578
##   has_year         playcount      playcount_has_year
##  Mode :logical   Min.   :  0.00   Min.   :  0.00
##  FALSE:577       1st Qu.:  1.00   1st Qu.:  0.00
##  TRUE :1652      Median :  4.00   Median :  3.00
##                  Mean   : 16.88   Mean   : 15.05
##                  3rd Qu.: 21.00   3rd Qu.: 18.00
##                  Max.   :142.00   Max.   :142.00
##
```

From this output we can see that this is a dataset with **2229** rows and **7** columns. Looking at the data dictionary (code not shown), we now know that this `classic_rock_song_list` is a dataset containing metadata of (presumably) classic rock songs. There is an anomaly in the `release_year` column - the minimum, or earliest, year is **1071**, which doesn't make sense because it is way too low. It is likely a typo, and the entry was most likely for **1971**. We can fix that using `mutate()` and `case_when()`:

```r
crsl <- classic_rock_song_list %>%
  mutate(release_year = case_when(
    release_year < 1100 ~ release_year + 900,
    TRUE ~ release_year + 0
    )
  )

crsl %>%
  summary()
```

```
##      song              artist           release_year    combined        
##  Length:2229        Length:2229        Min.   :1955   Length:2229       
##  Class :character   Class :character   1st Qu.:1971   Class :character  
##  Mode  :character   Mode  :character   Median :1977   Mode  :character  
##                                        Mean   :1979                     
##                                        3rd Qu.:1984                     
##                                        Max.   :2014                     
##                                        NA's   :578                      
##   has_year         playcount      playcount_has_year
##  Mode :logical   Min.   :  0.00   Min.   :  0.00    
##  FALSE:577       1st Qu.:  1.00   1st Qu.:  0.00    
##  TRUE :1652      Median :  4.00   Median :  3.00    
##                  Mean   : 16.88   Mean   : 15.05    
##                  3rd Qu.: 21.00   3rd Qu.: 18.00    
##                  Max.   :142.00   Max.   :142.00    
## 
```

```r
# print(crsl) # debug
```

I chose to use `<1100` as a condition to fix all cases that may have a mistyped 9 (if there were any more of those).

Looking at the summary statistics of the new dataframe `crsl`, we can see that the anomaly is no longer there - the minimum value of `release_year` is 1955, which is a much more sensible value.

We will now shift our focus to `classic_rock_raw_data`. We want to append information on the songs' release dates to this dataframe, and we will do so with a `left_join()`. But before we do that, we have to first take a look at `classic_rock_raw_data` to figure out the variables we will be joining on.

```r
summary(classic_rock_raw_data)
```

```
##      song              artist            callsign              time          
##  Length:37673       Length:37673       Length:37673       Min.   :1.403e+09  
##  Class :character   Class :character   Class :character   1st Qu.:1.403e+09  
##  Mode  :character   Mode  :character   Mode  :character   Median :1.403e+09  
##                                                           Mean   :1.403e+09  
##                                                           3rd Qu.:1.403e+09  
##                                                           Max.   :1.403e+09  
##    date_time                       unique_id           combined        
##  Min.   :2014-06-15 20:28:14.00   Length:37673       Length:37673      
##  1st Qu.:2014-06-17 19:54:27.00   Class :character   Class :character  
##  Median :2014-06-19 11:59:14.00   Mode  :character   Mode  :character  
##  Mean   :2014-06-19 10:15:19.71
```

```
##  3rd Qu.:2014-06-21 01:58:29.00
##  Max.   :2014-06-22 19:59:19.00
```

```
print(classic_rock_raw_data)
```

```
## # A tibble: 37,673 x 7
##    song              artist    calls~1   time date_time           uniqu~2 combi~3
##    <chr>             <chr>     <chr>    <int> <dttm>              <chr>   <chr>
##  1 Caught Up in You .38 Spec~ KGLK    1.40e9 2014-06-16 14:28:34 KGLK15~ Caught~
##  2 Caught Up in You .38 Spec~ KGB     1.40e9 2014-06-21 20:58:55 KGB0260 Caught~
##  3 Caught Up in You .38 Spec~ KGB     1.40e9 2014-06-20 01:58:44 KGB0703 Caught~
##  4 Caught Up in You .38 Spec~ KGLK    1.40e9 2014-06-22 16:58:52 KGLK00~ Caught~
##  5 Caught Up in You .38 Spec~ KGLK    1.40e9 2014-06-21 15:58:57 KGLK03~ Caught~
##  6 Caught Up in You .38 Spec~ KGLK    1.40e9 2014-06-18 11:28:20 KGLK11~ Caught~
##  7 Caught Up in You .38 Spec~ KGLK    1.40e9 2014-06-16 22:08:52 KGLK14~ Caught~
##  8 Caught Up in You .38 Spec~ KRFX    1.40e9 2014-06-22 12:58:23 KRFX00~ Caught~
##  9 Caught Up in You .38 Spec~ KRFX    1.40e9 2014-06-17 21:58:17 KRFX11~ Caught~
## 10 Caught Up in You .38 Spec~ KSHE    1.40e9 2014-06-19 07:59:27 KSHE07~ Caught~
## # ... with 37,663 more rows, and abbreviated variable names 1: callsign,
## #   2: unique_id, 3: combined
```

```
dim(classic_rock_raw_data)
```

```
## [1] 37673     7
```

It looks like both `classic_rock_song_list` and `classic_rock_raw_data` have the `combined` column that details the "song and artist name combined", according to their data dictionaries. We will use this common variable to join the `release_year` information from `classic_rock_song_list` to `classic_rock_raw_data`.

```
crrd <- classic_rock_raw_data %>%
  left_join(crsl %>%
              select(combined, release_year),
            by = "combined")
dim(crrd) # debug
```

```
## [1] 37673     8
```

```
print(crrd)
```

```
## # A tibble: 37,673 x 8
##    song         artist calls~1   time date_time           uniqu~2 combi~3 relea~4
##    <chr>        <chr>  <chr>    <int> <dttm>              <chr>   <chr>     <dbl>
##  1 Caught Up ~ .38 S~ KGLK    1.40e9 2014-06-16 14:28:34 KGLK15~ Caught~    1982
##  2 Caught Up ~ .38 S~ KGB     1.40e9 2014-06-21 20:58:55 KGB0260 Caught~    1982
##  3 Caught Up ~ .38 S~ KGB     1.40e9 2014-06-20 01:58:44 KGB0703 Caught~    1982
##  4 Caught Up ~ .38 S~ KGLK    1.40e9 2014-06-22 16:58:52 KGLK00~ Caught~    1982
##  5 Caught Up ~ .38 S~ KGLK    1.40e9 2014-06-21 15:58:57 KGLK03~ Caught~    1982
##  6 Caught Up ~ .38 S~ KGLK    1.40e9 2014-06-18 11:28:20 KGLK11~ Caught~    1982
##  7 Caught Up ~ .38 S~ KGLK    1.40e9 2014-06-16 22:08:52 KGLK14~ Caught~    1982
##  8 Caught Up ~ .38 S~ KRFX    1.40e9 2014-06-22 12:58:23 KRFX00~ Caught~    1982
##  9 Caught Up ~ .38 S~ KRFX    1.40e9 2014-06-17 21:58:17 KRFX11~ Caught~    1982
## 10 Caught Up ~ .38 S~ KSHE    1.40e9 2014-06-19 07:59:27 KSHE07~ Caught~    1982
## # ... with 37,663 more rows, and abbreviated variable names 1: callsign,
## #   2: unique_id, 3: combined, 4: release_year
```

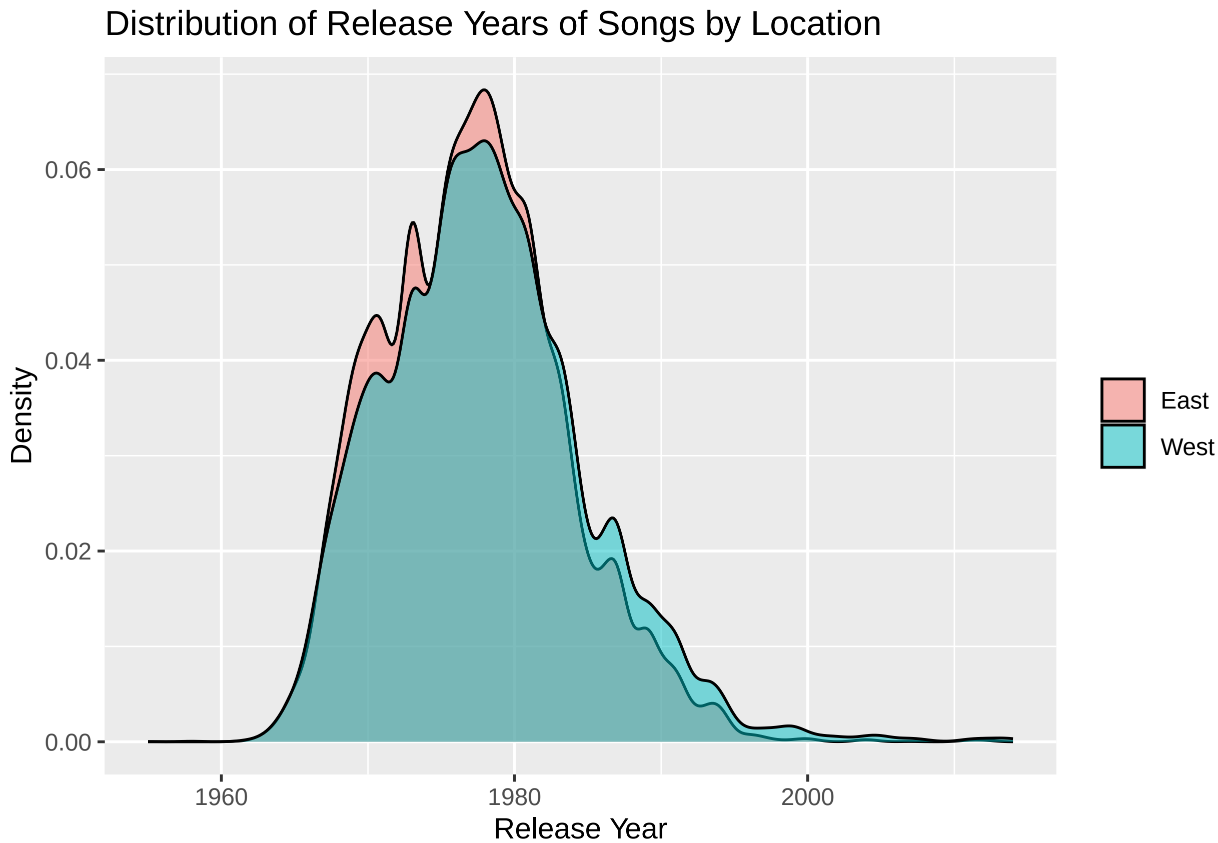
Visualizing `release_year` with respect to radio station location

To put this newly joined dataset to use, we will create a visualization of `crrd`. In specific, we will use the callsign of the radio stations in each entry to roughly determine the location of the radio station with respect to the Mississippi River. We will then create an overlayed density plot of the distribution of `release_year` with respect to location.

```
# crrd %>%
#   mutate(location = case_when(
#     substr(callsign, 1, 1) == "K" ~ "West",
#     substr(callsign, 1, 1) == "W" ~ "East"
#     )
#   ) %>%
#   group_by(location, release_year) %>%
#   summarize(cnt = n()) %>%
#   ggplot(mapping = aes(x = release_year, y = cnt,
#                        fill = location)) +
#   geom_bar(stat = "identity", position = position_dodge())

crrd %>%
  mutate(location = case_when(
    substr(callsign, 1, 1) == "K" ~ "West",
    substr(callsign, 1, 1) == "W" ~ "East"
    )
  ) %>%
  ggplot(mapping = aes(x = release_year, fill = location)) +
  geom_density(alpha = 0.5) +
  labs(title = "Distribution of Release Years of Songs by Location",
       x = "Release Year", y = "Density") +
  theme(legend.title = element_blank())
```

```
## Warning: Removed 4117 rows containing non-finite values (‘stat_density()‘).
```



It seems like the stations on the West of the Mississippi River tended to play more of the newer songs, as indicated by the stronger right skew.

Tidying and Cleaning `UN_Migrant_2015`

We will be tidying and cleaning up the dataset from `UN_Migrant_2015`, which can be found on canvas. I have the dataset in the local subdirectory `./Datasets`, and will use `read_csv()` to import the dataset into R.

```
UN = read_csv("./Datasets/UN_Migrant_2015.csv")
```

```
## Rows: 237 Columns: 23
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (20): Major area, region, country or area of destination, Type of data (...
## dbl  (3): Sort order, Notes, Country code
##
## 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.
```

```
# knitr::kable(head(UN, 5))
```

Since we will not be needing them, we will use `select` to drop `Sort order`, `Notes`, `Country code` and `Type of data (a)`. Also, we will rename `Major area, region, country or area of destination` to `area_dest` for simplicity's sake.

```r
UN <- UN %>%
  select(-c("Sort order", "Notes", "Country code", "Type of data (a)")) %>%
  rename("area_dest" ="Major area, region, country or area of destination")

dim(UN)
```

```
## [1] 237  19
```

We will now use `janitor::clean_names()` to clean up the names of our columns - I agree that these capital letters are kind dumb.

```r
UN <- UN %>%
  janitor::clean_names()
# knitr::kable(head(UN))
print(UN)
```

```
## # A tibble: 237 x 19
##    area_dest  both_1990 both_1~1 both_~2 both_~3 both_~4 both_~5 male_~6 male_~7
##    <chr>      <chr>     <chr>    <chr>   <chr>   <chr>   <chr>   <chr>   <chr>
##  1 Burundi    333 110   254 853  125 628 172 874 235 259 286 810 163 267 124 165
##  2 Comoros    14 079    13 939   13 799  13 209  12 618  12 555  6 717   6 614
##  3 Djibouti   122 221   99 774   100 507 92 091  101 575 112 351 64 242  52 476
##  4 Eritrea    11 848    12 400   12 952  14 314  15 676  15 941  6 228   6 542
##  5 Ethiopia   1 155 390 806 904  611 384 514 242 567 720 1 072 ~ 607 284 424 117
##  6 Kenya      297 292   618 745  699 139 756 894 926 959 1 084 ~ 160 852 322 189
##  7 Madagascar 23 917    21 177   23 541  26 058  28 905  32 075  13 348  11 901
##  8 Malawi     1 127 724 241 624  232 620 221 661 217 722 215 158 546 520 116 198
##  9 Mauritius  3 613     7 493    15 543  19 647  24 836  28 585  1 763   3 228
## 10 Mayotte    15 229    26 316   45 474  63 176  72 757  76 992  8 780   14 679
## # ... with 227 more rows, 10 more variables: male_2000 <chr>, male_2005 <chr>,
## #   male_2010 <chr>, male_2015 <chr>, female_1990 <chr>, female_1995 <chr>,
## #   female_2000 <chr>, female_2005 <chr>, female_2010 <chr>, female_2015 <chr>,
## #   and abbreviated variable names 1: both_1995, 2: both_2000, 3: both_2005,
## #   4: both_2010, 5: both_2015, 6: male_1990, 7: male_1995
```

After taking a look into the csv file, we can see that there are some values with `...` These are probably the missing values. They appear in the entries for South Sudan - this is probably due to the lack of data for South Sudan in the relevant categories.

Reshaping `UN`

We will now reshape our dataset. The data is currently in a wide format, with its longitudinal data stored in columns. The data is hard to deal with in its default state, so we will split the dataset into 3 - `UN_both`, `UN_male` and `UN_female` and clean each individual sub-dataframe, then combine the three using `rbind()`.

I could not figure out how to do this in 1 step - is it possible? If so, how do we do it?

```r
UN_both <- UN %>%
  select(c("area_dest", starts_with("both")))

UN_both <- UN_both %>%
```

```r
  pivot_longer(cols = starts_with("both"),
               names_to = "both_year",
               values_to = "n_migrants") %>%
  separate(col = "both_year",
           into = c("sex", "year"),
           sep = "_"
           )
# print(UN_both) # debug

UN_male <- UN %>%
  select(c("area_dest", starts_with("male")))


UN_male <- UN_male %>%
  pivot_longer(cols = starts_with("male"),
               names_to = "male_year",
               values_to = "n_migrants") %>%
  separate(col = "male_year",
           into = c("sex", "year"),
           sep = "_"
           )
# print(UN_male) # debug

UN_female <- UN %>%
  select(c("area_dest", starts_with("female")))

UN_female <- UN_female %>%
  pivot_longer(cols = starts_with("female"),
               names_to = "female_year",
               values_to = "n_migrants") %>%
  separate(col = "female_year",
           into = c("sex", "year"),
           sep = "_"
           )
# print(UN_female) # debug

# combine
UN <- rbind(UN_both, UN_male, UN_female) %>%
  janitor::clean_names()

print(UN)
```

```
## # A tibble: 4,266 x 4
##    area_dest sex   year  n_migrants
##    <chr>     <chr> <chr> <chr>
##  1 Burundi   both  1990  333 110
##  2 Burundi   both  1995  254 853
##  3 Burundi   both  2000  125 628
##  4 Burundi   both  2005  172 874
##  5 Burundi   both  2010  235 259
##  6 Burundi   both  2015  286 810
##  7 Comoros   both  1990  14 079
##  8 Comoros   both  1995  13 939
```

```
##  9 Comoros   both  2000  13 799
## 10 Comoros   both  2005  13 209
## # ... with 4,256 more rows
```

As the output shows, we have the correct dimensions on our final dataset, and the newly reshaped dataset is much easier to read.

Total Number of Migrants per Year

We will now find the total number of migrants across all countries in each year in `UN`. However, there is a problem - the numbers are strings with a space separating every 3 powers of 10. This makes summing impossible in the current state of the data. We will use `mutate()` in conjunction with `str_replace_all()` and `as_double()` to remove all the dividing whitespace and coerce all the values into the **double** type. This should change all the missing values into the corresponding "missing value value" for numeric types, which is `NA`.

```
UN <- UN %>%
  mutate(n_migrants = as.double(str_replace_all(n_migrants, " ", "")),
         year = as.double(year))
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```
UN %>%
  group_by(year) %>%
  summarize(sum(n_migrants, na.rm = TRUE)) %>%
  ungroup()
```

```
## # A tibble: 6 x 2
##    year `sum(n_migrants, na.rm = TRUE)`
##   <dbl>                           <dbl>
## 1  1990                       323022026
## 2  1995                       341525612
## 3  2000                       367502762
## 4  2005                       407654968
## 5  2010                       472392724
## 6  2015                       519390604
```

```
UN %>%
  filter(area_dest == "South Sudan")
```

```
## # A tibble: 18 x 4
##    area_dest   sex    year n_migrants
##    <chr>       <chr> <dbl>      <dbl>
##  1 South Sudan both   1990         NA
##  2 South Sudan both   1995         NA
##  3 South Sudan both   2000         NA
##  4 South Sudan both   2005         NA
##  5 South Sudan both   2010     257905
##  6 South Sudan both   2015     824122
##  7 South Sudan male   1990         NA
##  8 South Sudan male   1995         NA
```

```
## 9 South Sudan male    2000        NA
## 10 South Sudan male    2005        NA
## 11 South Sudan male    2010    132693
## 12 South Sudan male    2015    420949
## 13 South Sudan female  1990        NA
## 14 South Sudan female  1995        NA
## 15 South Sudan female  2000        NA
## 16 South Sudan female  2005        NA
## 17 South Sudan female  2010    125212
## 18 South Sudan female  2015    403173
```

Although it isn't required, I still went ahead and did it for some practice :D. In this case, where data is missing for only 1 country, should I be dropping data like I did above, or should I interpolate?

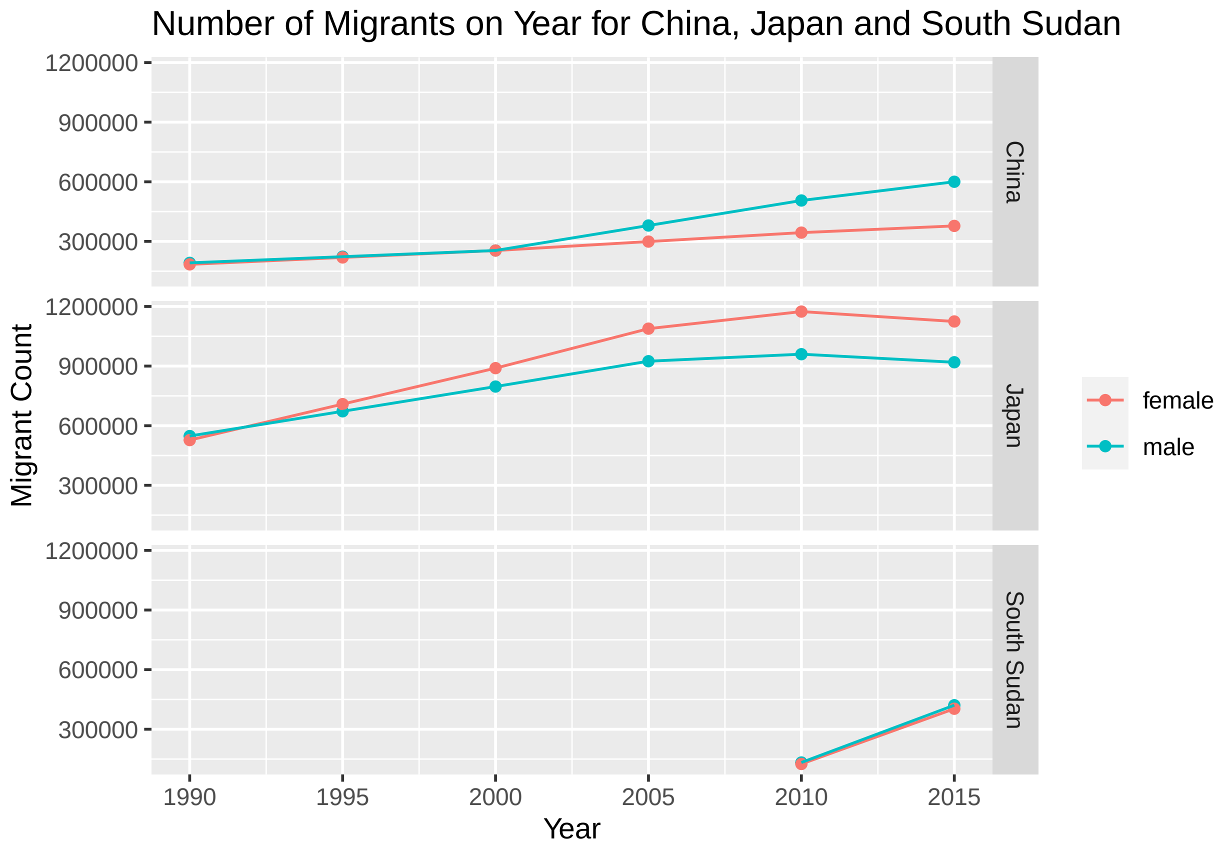
The above output shows a successful sum of `n_migrants` across all of our countries in each year, and the bottom output confirms my suspicion of the treatment of missing values.

Visualizing the cleaned `UN` Dataset

With our cleaned dataset, we can create visualizations quickly and easily. For example, here is a plot of `n_migrants` with respect to time facetted on `area_dest` for Japan, China and South Sudan.

```
UN %>%
  filter((area_dest == "Japan" | area_dest == "China" | area_dest == "South Sudan") & sex != "both") %>%
  ggplot(mapping = aes(x = year, y = n_migrants, color = sex)) +
  geom_point() +
  geom_line() +
  facet_grid(area_dest ~ .) +
  labs(x = "Year",
       y = "Migrant Count",
       title = "Number of Migrants on Year for China, Japan and South Sudan") +
  theme(legend.title = element_blank())
```

```
## Warning: Removed 8 rows containing missing values (`geom_point()`).
```




```
# UN %>% # debug
#   filter((area_dest == "Japan" | area_dest == "China" | area_dest == "South Sudan") & sex != "both")
#   print()
```

I was confused for at least 20 minutes at why I couldn't get the plot to exclude the data for "both" until I checked some example code online and saw that the logical "AND" operator was & and not && like in most other programming languages

The plot for South Sudan is looks like it is missing points for 1990 - 2005. This makes sense, since we saw missing values for South Sudan in the dataset near the start of this analysis. This probably means that `ggplot` treats missing values literally as blank values and ignores them.