Recitation 8 Data Merging

Seamus Wagner

October 19, 2021

Data merging

We covered basics of this in the previous recitation. Today we will use a synthetic and a practical example of merging. Along with data cleaning, this is likely the most useful practical skill you will learn for coding in graduate school. Think carefully about what exactly you want your data to look like and check it multiple times and hopefully have someone else run and check your code. If you are working on co-authored projects, it is absolutely worth the time to set up a meeting with most/all of the authors so that everyone can walk through the entire coding process. Social science projects tend to have fewer co-authors. This is an ongoing issue for coding reliability, regardless of what anyone tells you or what you may think. If the same people were coding launch procedures for a nuclear power plant, I guarantee that they would want a second set of eyes on their code. The point being, you will make errors in your coding, set up your coding process to try to catch as many as possible and be careful ion every step, but none of that is a substitute for someone else reviewing your code. Also, the more someone's the better. Merging, cleaning/aggregating is often where errors occur and if you ever take part in replication exercises, you are likely to find errors like this.

Merging gets increasingly complex when you are merging across units of analysis or when working with dyadic data, whether it be directed or undirected. A common merging procedure almost any of us will use at some point will be national level data across multiple countries/subnational units. Likewise, the merging of an individual level cross national survey with national level characteristics.

I will be using the tidyverse syntax for merging here. This syntax is largely similar to SQL, which is the most common language used for pulling data stored remotely. SQL is the only skill that has been 100% mentioned in people I talk to looking to hire in data science if you ever plan on going outside of academia. Further, even if you are planning on it, having backup skills in the event of another horrible job market is prudent. SQL itself is not hard to learn (I have heard, I do not know it but looked over the syntax and it looks easy enough). The more important part than the coding of the language is actually understanding what your merges are doing and when to use which.

Merging in tidyverse

There are five basic joins in tidyverse. - Left join: Keep all of A, append B where B matches A and NA otherwise - Right join: Keep A that matches B, append B (and NA otherwise) - Inner join: Keep A that matches B *and* B that matches A (exclude what would be NAs) - Full join: Combines left and right join – keep everything, join where matched and NA otherwise - Anti join: Drop rows where there is a match between A and B

Left joins are the most common in my experience. This is typically because my workflow, and I suspect many others, revolves around the main set of units we want, and we then bring in data based on identification keys in both data sets (country codes, respondent ids, years, etc.). When merging for any of these, the keys that you merge by are vital to successfully merging what you want. When you enter the syntax, whatever

is included in the “by" term will not duplicate, but all other columns will be kept for both data sets and populated with NAs for rows where there is no match among *all* keys.

Right joins are similar except the second set of data are what you keep and only return matching rows from the first set of data. I have never personally use them, but some people prefer them from how they think about joining (less common). My main reason for restructuring if I had a case where right join is ready, is that when I revisit my code I want to know what I did quickly and easily, and right joins are not how I think about merging. The only other use case would be the order of your columns, but you can re-order those too.

Inner joins return matching sets of data by your keys and drops others. Be careful with this, you will lose a lot of data if not done correctly, particularly with dyadic panel data.

Full joins just merge everything together. You will keep all rows and all columns from both data sets. Useful sometimes, but often less useful than it first sounds.

Anti joins are more useful in a business sense or a data discovery sense to my knowledge. Anti joins are helpful for checking mismatches after joins. What cases from merged data do not match and should for instance. In business sense, it could be rows of data that aren't matched on a key for instance (user didn't engage but is in your dataset, sales that didn't occur if sale completion is a filtering variable).

There are more types but those are beyond the scope of this.

```
rm(list=ls())
set.seed(34567874)
#Create some variables and data
n <- 10000
mu <- 1
sigma <- .5

# Make sure to include an id
fakedata <- data.table(id = 1:n,
                       X1 = rbinom(n, 1, .5),
                       X2 = rnorm(n, mu, sigma))
```

Next let's make some ecological variables, we can use country and region for instance.

```
# Here are the data for country and region.
ecodat <- data.table(individual_id = 1:n,
                     E1 = rep(paste0("region_",1:5), each = n/5),
                     E2 = rep(paste0("country_", 1:10), each = n/10))
```

Now for some merging examples. This is intentionally misleading, as merging would not be an important skill if we had perfectly matched keys and equally matched number of observations for everything.

```
# They all (except anti) produce the same thing when the data are structured similarly and we have ids
mergedat <- fakedata %>%
  left_join(ecodat, by = c("id" = "individual_id"))

mergedat1 <- fakedata %>%
  inner_join(ecodat, by = c("id" = "individual_id"))

mergedat2 <- fakedata %>%
  right_join(ecodat, by = c("id" = "individual_id"))

mergedat3 <- fakedata %>%
  full_join(ecodat, by = c("id" = "individual_id"))
```

```
mergedat4 <- fakedata %>%
  anti_join(ecodat, by = c("id" = "individual_id"))
```

Now, say we care not only about individual covariates or geographic variables, but how one's individual covariate relates to that geography. Maybe, for instance, we think that *relative* income matters more than *absolute* income for some outcome of interest since the cost of living and economic hierarchy varies wildly across different geographies. We can construct aggregate values of our covariates by geography, merge them back to our original data, and create relative covariates.

Importantly, we want to join on *both* E1 and E2 here. If we only join on one, we risk duplicating rows.

```
# aggregate individual metric by ecological values
aggreated_dt <- mergedat %>% group_by(E1, E2) %>%
  dplyr::summarise(country_median_X2 = median(X2))

# merge back and, for fun, assign treatment
mergemerge <- mergedat %>%

  # note that here we're merging by both of our geographic covariates, not by id
  left_join(aggreated_dt, by = c("E1","E2")) %>%

  mutate(relative_X2 = X2 - country_median_X2,

         # in theory, say we now have some interactive treatment effect
         Y0 = rnorm(n),
         Y1 = rnorm(n, X1*relative_X2)) %>%

         # and biased probability of selecting into treatment

  mutate(p = invlogit(relative_X2*country_median_X2))
```

Now, say we want to run a very messy experiment where the probability of selecting into treatment is biased. Moreover, we can't assign everyone into treatment and control – about 10% (relatively small in real-world case) of our population goes unobserved one way or the other. Thankfully, this missingness is completely at random and we can ignore it (not a missing data recitation).

But again, check to make sure that all of these steps work as intended!

```
# assign treatment
treatment <- data.table(id = 1:n,
                        D = rbinom(n, 1, mergemerge$p))

# randomly remove ten percent of observations
toremove <- rbinom(n, 1, .1)
treatment$id[toremove == 1] <- NA

# merge treatment to potential outcomes data WHERE TREATMENT IS OBSERVED
# and since the id variables have different names, specify how they relate
merged_treated <- mergemerge %>%
  inner_join(treatment, by = c("id")) %>%
  mutate(Y = D*Y1 + (1-D)*Y0) %>%
  dplyr::select(X1, X2, E1, E2, country_median_X2, relative_X2, D, Y)
```

```r
# check that the join did what you wanted it to
merged_treated %>% group_by(round(X2, 0)) %>%
  dplyr::summarise(n = n(),
                   n.treated = sum(D),
                   prop.treated = sum(D)/n())
```

```
## # A tibble: 5 x 4
##   `round(X2, 0)`     n n.treated prop.treated
## *          <dbl> <int>     <int>        <dbl>
## 1             -1    15         1       0.0667
## 2              0  1424       421       0.296
## 3              1  6201      3163       0.510
## 4              2  1362       923       0.678
## 5              3    12         8       0.667
```

Now we can check our naive difference in means.

```r
# calculate naive difference in means
merged_treated %>% group_by(E1, E2) %>%
  dplyr::summarise(n = n(),
                   naive_diff_means = mean(Y[D == 1]) - mean(Y[D == 0]))
```

```
## # A tibble: 10 x 4
## # Groups:   E1 [5]
##    E1       E2             n naive_diff_means
##    <chr>    <chr>      <int>            <dbl>
##  1 region_1 country_1    897           0.0672
##  2 region_1 country_2    904           0.102
##  3 region_2 country_3    900           0.134
##  4 region_2 country_4    896           0.150
##  5 region_3 country_5    902           0.106
##  6 region_3 country_6    903           0.0612
##  7 region_4 country_7    893           0.0728
##  8 region_4 country_8    905           0.0545
##  9 region_5 country_10   909           0.0945
## 10 region_5 country_9    905           0.104
```

Practical example

So the above was just some synthetic data to mess around with. Moving forward let's move into merging real world data sets. This is an example from a term paper I did looking at maritime travel time and colonial expansion. The base data are dyadic maritime travel times between ports for some of the globe from a Pascali paper from 2017. What we want to merge with this are COLDAT, a data set of colonial dyads and the start and end time of colonialism (mostly EU colonialism). There is a lot of code for cleaning data, which I include to show how I do this as a practice, but we will skip if no time.

```r
library(haven)
library(stringr)
library(dplyr)
library(countrycode)
library(data.table)
```

```
rm(list=ls())
#country_d and country_o are backwards according to value labels
distances <- read_dta("BILATERAL_DISTANCES_PUBLIC.dta")
distances$colonizer <- NA
distances$country <- NA

#The country_d variable is the origin variable, and in our case will be colonizing countries
#See country_o for potential colonized countries
distances$colonizer[distances$country_d == "Aden"] <- "Yemen"
distances$colonizer[distances$country_d == "Algeria"] <- "France"
distances$colonizer[distances$country_d == "Angola"] <- "France"
distances$colonizer[distances$country_d == "Argentina"] <- "Spain"
distances$colonizer[distances$country_d == "Australia"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Australia - Queensland"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Australia - South Australia"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Australia - Tasmania"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Australia - Victoria"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Australia - Western Australia"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Austria-Hungary"] <- "Croatia"
distances$colonizer[distances$country_d == "Bahamas"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Barbados"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Bermuda"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Bolivia"] <- "Spain"
distances$colonizer[distances$country_d == "Brazil"] <- "Portugal"
distances$colonizer[distances$country_d == "British Honduras"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Canada"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Cameroon"] <- "France"
distances$colonizer[distances$country_d == "Cape of Good Hope"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Ceylon"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Chile"] <- "Spain"
distances$colonizer[distances$country_d == "China"] <- "China"
distances$colonizer[distances$country_d == "Colombia"] <- "Spain"
distances$colonizer[distances$country_d == "Belgian Congo"] <- "Belgium"
distances$colonizer[distances$country_d == "Costa Rica"] <- "Spain"
distances$colonizer[distances$country_d == "Cuba"] <- "Spain"
distances$colonizer[distances$country_d == "Dominican Republic"] <- "Spain"
distances$colonizer[distances$country_d == "Dutch East Indie"] <- "Netherlands"
distances$colonizer[distances$country_d == "Dutch East Indie - Dutch Borneo"] <- "Netherlands"
distances$colonizer[distances$country_d == "Dutch Guyana"] <- "Netherlands"
distances$colonizer[distances$country_d == "Ecuador"] <- "Spain"
distances$colonizer[distances$country_d == "Egypt"] <- "Egypt"
distances$colonizer[distances$country_d == "El Salvador"] <- "Spain"
distances$colonizer[distances$country_d == "Falklands islands"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Finland"] <- "Russia"
distances$colonizer[distances$country_d == "French Guyana"] <- "France"
distances$colonizer[distances$country_d == "French Indo-China"] <- "France"
distances$colonizer[distances$country_d == "French West Africa"] <- "France"
distances$colonizer[distances$country_d == "German New Guinea"] <- "Germany"
distances$colonizer[distances$country_d == "German South Africa"] <- "Germany"
distances$colonizer[distances$country_d == "Gold Coast"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Greece"] <- "Greece"
distances$colonizer[distances$country_d == "Guadaloupe"] <- "France"
distances$colonizer[distances$country_d == "Guatemala"] <- "Spain"
```

```
distances$colonizer[distances$country_d == "Guyana"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Hadramaut"] <- "Yemen"
distances$colonizer[distances$country_d == "Haiti"] <- "France"
distances$colonizer[distances$country_d == "Iceland"] <- "Denmark"
distances$colonizer[distances$country_d == "India - British Possessions"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Lagos"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Liberia"] <- "Liberia"
distances$colonizer[distances$country_d == "Libia-Tripoli"] <- "Tunisia"
distances$colonizer[distances$country_d == "Madagascar"] <- "France"
distances$colonizer[distances$country_d == "Malta"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Martinique"] <- "France"
distances$colonizer[distances$country_d == "Mauritius"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Mexico"] <- "Spain"
distances$colonizer[distances$country_d == "Morocco"] <- "France"
distances$colonizer[distances$country_d == "Mozambique"] <- "Portugal"
distances$colonizer[distances$country_d == "New Zealand"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Nicaragua"] <- "Spain"
distances$colonizer[distances$country_d == "Oman"] <- "Oman"
distances$colonizer[distances$country_d == "Panama"] <- "Spain"
distances$colonizer[distances$country_d == "Persia"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Peru"] <- "Spain"
distances$colonizer[distances$country_d == "Philippines"] <- "Spain"
distances$colonizer[distances$country_d == "Puerto Rico"] <- "Spain"
distances$colonizer[distances$country_d == "Rio de Oro"] <- "Spain"
distances$colonizer[distances$country_d == "Arabia"] <- "Yemen"
distances$colonizer[distances$country_d == "Siam"] <- "Thailand"
distances$colonizer[distances$country_d == "Somalia"] <- "Italy"
distances$colonizer[distances$country_d == "Straits Settlements"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Sweeden"] <- "Sweeden"
distances$colonizer[distances$country_d == "Tanzania"] <- "Germany"
distances$colonizer[distances$country_d == "Trinidad and Tobago"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Ottoman Empire"] <- "Turkey"
distances$colonizer[distances$country_d == "Virgin Islands"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Belgium"] <- "Belgium"
distances$colonizer[distances$country_d == "France"] <- "France"
distances$colonizer[distances$country_d == "Germany"] <- "Germany"
distances$colonizer[distances$country_d == "Russia"] <- "Russia"
distances$colonizer[distances$country_d == "Sweden"] <- "Sweden"
distances$colonizer[distances$country_d == "United Kingdom"] <- "United Kingdom"
distances$colonizer[distances$country_d == "United States"] <- "United Kingdom"
distances$colonizer[distances$country_d == "Uruguay"] <- "Spain"
distances$colonizer[distances$country_d == "Venezuela"] <- "Spain"
distances$colonizer[distances$country_d == "Italy"] <- "Italy"
distances$colonizer[distances$country_d == "Denmark"] <- "Denmark"
distances$colonizer[distances$country_d == "Japan"] <- "Japan"
distances$colonizer[distances$country_d == "Netherlands"] <- "Netherlands"
distances$colonizer[distances$country_d == "Norway"] <- "Norway"
distances$colonizer[distances$country_d == "Spain"] <- "Spain"
distances$colonizer[distances$country_d == "Romania"] <- "Romania"
distances$colonizer[distances$country_d == "Portugal"] <- "Portugal"

#Now for the destination countries from Pascali
distances$country[distances$country_o == "Aden"] <- "Yemen"
```

```r
distances$country[distances$country_o == "Algeria"] <- "Algeria"
distances$country[distances$country_o == "Angola"] <- "Angola"
distances$country[distances$country_o == "Argentina"] <- "Argentina"
distances$country[distances$country_o == "Australia"] <- "Australia"
distances$country[distances$country_o == "Australia - Queensland"] <- "Australia"
distances$country[distances$country_o == "Australia - South Australia"] <- "Australia"
distances$country[distances$country_o == "Australia - Tasmania"] <- "Australia"
distances$country[distances$country_o == "Australia - Victoria"] <- "Australia"
distances$country[distances$country_o == "Australia - Western Australia"] <- "Australia"
distances$country[distances$country_o == "Austria-Hungary"] <- "Croatia"
distances$country[distances$country_o == "Bahamas"] <- "Bahamas"
distances$country[distances$country_o == "Barbados"] <- "Barbados"
distances$country[distances$country_o == "Bermuda"] <- "Bermuda"
distances$country[distances$country_o == "Bolivia"] <- "Bolivia"
distances$country[distances$country_o == "Brazil"] <- "Brazil"
distances$country[distances$country_o == "British Honduras"] <- "Belize"
distances$country[distances$country_o == "Canada"] <- "Canada"
distances$country[distances$country_o == "Cameroon"] <- "Cameroon"
distances$country[distances$country_o == "Cape of Good Hope"] <- "South Africa"
distances$country[distances$country_o == "Ceylon"] <- "Sri Lanka"
distances$country[distances$country_o == "Chile"] <- "Chile"
distances$country[distances$country_o == "China"] <- "China"
distances$country[distances$country_o == "Colombia"] <- "Colombia"
distances$country[distances$country_o == "Belgian Congo"] <- "Democratic Republic of the Congo"
distances$country[distances$country_o == "Costa Rica"] <- "Costa Rica"
distances$country[distances$country_o == "Cuba"] <- "Cuba"
distances$country[distances$country_o == "Dominican Republic"] <- "Dominican Republic"
distances$country[distances$country_o == "Dutch East Indie"] <- "Indonesia"
distances$country[distances$country_o == "Dutch East Indie - Dutch Borneo"] <- "Brunei"
distances$country[distances$country_o == "Dutch Guyana"] <- "Suriname"
distances$country[distances$country_o == "Ecuador"] <- "Ecuador"
distances$country[distances$country_o == "Egypt"] <- "Egypt"
distances$country[distances$country_o == "El Salvador"] <- "El Salvador"
distances$country[distances$country_o == "Falklands islands"] <- "Falkland islands"
distances$country[distances$country_o == "Finland"] <- "Finland"
distances$country[distances$country_o == "French Guyana"] <- "French Guyana"
distances$country[distances$country_o == "French Indo-China"] <- "Vietnam"
distances$country[distances$country_o == "French West Africa"] <- "Senegal"
distances$country[distances$country_o == "German New Guinea"] <- "Papua New Guinea"
distances$country[distances$country_o == "German South Africa"] <- "Namibia"
distances$country[distances$country_o == "Gold Coast"] <- "Ghana"
distances$country[distances$country_o == "Greece"] <- "Greece"
distances$country[distances$country_o == "Guadaloupe"] <- "Guadeloupe"
distances$country[distances$country_o == "Guatemala"] <- "Guatemala"
distances$country[distances$country_o == "Guyana"] <- "Guyana"
distances$country[distances$country_o == "Hadramaut"] <- "Yemen"
distances$country[distances$country_o == "Haiti"] <- "Haiti"
distances$country[distances$country_o == "Iceland"] <- "Iceland"
distances$country[distances$country_o == "India - British Possessions"] <- "India"
distances$country[distances$country_o == "Lagos"] <- "Nigeria"
distances$country[distances$country_o == "Liberia"] <- "Liberia"
distances$country[distances$country_o == "Libia-Tripoli"] <- "Libya"
distances$country[distances$country_o == "Madagascar"] <- "Madagascar"
```

```
distances$country[distances$country_o == "Malta"] <- "Malta"
distances$country[distances$country_o == "Martinique"] <- "Martinique"
distances$country[distances$country_o == "Mauritius"] <- "Mauritius"
distances$country[distances$country_o == "Mexico"] <- "Mexico"
distances$country[distances$country_o == "Morocco"] <- "Morocco"
distances$country[distances$country_o == "Mozambique"] <- "Mozambique"
distances$country[distances$country_o == "New Zealand"] <- "New Zealand"
distances$country[distances$country_o == "Nicaragua"] <- "Nicaragua"
distances$country[distances$country_o == "Oman"] <- "Oman"
distances$country[distances$country_o == "Panama"] <- "Panama"
distances$country[distances$country_o == "Persia"] <- "Iran"
distances$country[distances$country_o == "Peru"] <- "Peru"
distances$country[distances$country_o == "Philippines"] <- "Philippines"
distances$country[distances$country_o == "Puerto Rico"] <- "Puerto Rico"
#Not sure where we use SADR/western Sahara/Morocco, went with western sahara as it has iso codes. Not s
distances$country[distances$country_o == "Rio de Oro"] <- "Western Sahara"
distances$country[distances$country_o == "Arabia"] <- "Yemen"
distances$country[distances$country_o == "Siam"] <- "Thailand"
distances$country[distances$country_o == "Somalia"] <- "Somalia"
distances$country[distances$country_o == "Straits Settlements"] <- "Malaysia"
distances$country[distances$country_o == "Sweden"] <- "Sweden"
distances$country[distances$country_o == "Tanzania"] <- "Tanzania"
distances$country[distances$country_o == "Trinidad and Tobago"] <- "Trindidad and Tobago"
distances$country[distances$country_o == "Ottoman Empire"] <- "Turkey"
distances$country[distances$country_o == "Virgin Islands"] <- "Virgin Islands (British)"
distances$country[distances$country_o == "Belgium"] <- "Belgium"
distances$country[distances$country_o == "France"] <- "France"
distances$country[distances$country_o == "Germany"] <- "Germany"
distances$country[distances$country_o == "Russia"] <- "Russia"
distances$country[distances$country_o == "United Kingdom"] <- "United Kingdom"
distances$country[distances$country_o == "United States"] <- "United States"
distances$country[distances$country_o == "Uruguay"] <- "Uruguay"
distances$country[distances$country_o == "Venezuela"] <- "Venezuela"
distances$country[distances$country_o == "Italy"] <- "Italy"
distances$country[distances$country_o == "Denmark"] <- "Denmark"
distances$country[distances$country_o == "Japan"] <- "Japan"
distances$country[distances$country_o == "Netherlands"] <- "Netherlands"
distances$country[distances$country_o == "Norway"] <- "Norway"
distances$country[distances$country_o == "Spain"] <- "Spain"
distances$country[distances$country_o == "Romania"] <- "Romania"
distances$country[distances$country_o == "Portugal"] <- "Portugal"
```

Next, we should use country codes rather than names. The main reason for this is that multiple names for a country can occur across country data sets, and using a standardized sets of codes allows us to both merge with consistent keys, and check for where keys don't go through. I am choosing to use the iso 3 character codes, you can use whatever you prefer. I like ISO since they are easy to identify in a graph and cover a large number of countries.

```
distances$iso_colonizer <- countrycode(distances$colonizer,
                                       origin = "country.name.en",
                                       destination = "iso3c")
table(is.na(distances$iso_colonizer))
```

```
##
## FALSE
##  8834
```

```
distances$iso_country <- countrycode(distances$country,
                                     origin = "country.name.en",
                                     destination = "iso3c")
table(is.na(distances$iso_country))
```

```
##
## FALSE
##  8834
```

```
coldat <- read.csv("COLDAT_dyads.csv", stringsAsFactors = F)
#Just making the coldat country names capitalized for the first level
coldat$country <- str_to_title(coldat$ï..country)
coldat$colonizer <-  str_to_title(coldat$colonizer)
coldat$iso_colonizer <- countrycode(coldat$colonizer,
                                     origin = "country.name.en",
                                     destination = "iso3c")
table(is.na(coldat$iso_colonizer))
```

```
##
## FALSE
##  1560
```

```
coldat$iso_country <- countrycode(coldat$country,
                                  origin = "country.name.en",
                                  destination = "iso3c")

# Kosovo is NA and does not have an iso3c code, it has a sub code for Serbia I believe, but it is one a
table(is.na(coldat$iso_country))
```

```
##
## FALSE  TRUE
##  1544    16
```

Before continuing, let's consider the structure of the data. We have port level directed dyads for maritime travel. This means we have multiple ports in each country. COLDAT is a country level undirected dyad of colonizing from EU mostly. We have to solve some of the inconsistencies to get an accurate merge. Unfortunately, what an *accurate* merge is changes with what the final product is supposed to be.

For instance, what if the universe is all potential colonial ties from 1500-1980? What if we only want non-landlocked countries and EU as the set of colonizers? Let's take one example of taking the mean of distances/times all of the ports in a given country. This has some implications to think of for somewhere like Australia, where there are thousands of miles and hours difference for some of the sailing times and distances based on ports. I used some data table aggregation procedures since they are faster in this case. You can do a group_by() and mutate() sequence in dplyr to achieve similar results. I then get the distinct direct dyads. In the end, we should get one value for each directed dyad of each pair of countries in the data set.

```
#Merging the distances data into the colonial dyads data.
distances_dt <- data.table(distances)
distances_dt[, geo_dist_m :=  mean(geo_dist), by = list(iso_country, iso_colonizer)]
distances_dt[, TIME_4_1_m :=  mean(TIME_4_1), by = list(iso_country, iso_colonizer)]
distances_dt[, TIME_4_2_m :=  mean(TIME_4_2), by = list(iso_country, iso_colonizer)]
distances_dt[, TIME_5_1_5_m :=  mean(TIME_5_1_5), by = list(iso_country, iso_colonizer)]
distances_dt[, TIME_5_2_5_m :=  mean(TIME_5_2_5), by = list(iso_country, iso_colonizer)]
distances_dt <- distances_dt %>%
  distinct(iso_colonizer, iso_country, geo_dist_m, TIME_4_1_m, TIME_4_2_m, TIME_5_1_5_m, TIME_5_2_5_m, c
```

One thing to note here is that we have positive values for countries with themselves, what should we do about that? This occurs since we take averages. Russia to Russia should be 0 since it should be the same port. We can correct this for each of those cases fairly easily by setting 0's for when iso_country and iso_colonizer equal each other, but then we are using different values for those cases than the rest of the data set. Realistically, we are probably dropping those before the analysis anyway since they theoretically cannot matter, and also empirically can influence our results, a bad combination. We will deal with these cases later, since we may be able to systematically drop them with no additional coding.

Which set of cases would you chose as your base?

```
merged1 <- left_join(coldat, distances_dt, by = c("iso_colonizer", "iso_country"))
merged1 <- merged1 %>%
  dplyr::select(-c("colonizer.y", "country.y")) %>%
  dplyr::rename(colonizer = "colonizer.x",
         country = "country.x")
merged2 <- left_join(distances_dt, coldat, by = c("iso_colonizer", "iso_country"))

# What happens if we use country/colonizer instead of codes?
# Why are these bad?
merged3 <- left_join(coldat, distances_dt, by = c("colonizer", "country"))

merged4 <- left_join(distances_dt, coldat, by = c("colonizer", "country"))
```

Work out what is different about the five joins for yourself if you have time and are interested. It may take some work, but may help you understand joining procedures well. Then check those five joins against a different set of keys.

```
test_join1 <- full_join(coldat, distances, by = c("iso_colonizer", "iso_country"))
test_join2 <- right_join(coldat, distances, by = c("iso_colonizer", "iso_country"))
test_join3 <- inner_join(coldat, distances, by = c("iso_colonizer", "iso_country"))
test_join4 <- left_join(coldat, distances, by = c("iso_colonizer", "iso_country"))
test_join5 <- anti_join(coldat, distances, by = c("iso_colonizer", "iso_country"))

test_join6 <- full_join(coldat, distances, by = c("colonizer", "country"))
test_join7 <- right_join(coldat, distances, by = c("colonizer", "country"))
test_join8 <- inner_join(coldat, distances, by = c("colonizer", "country"))
test_join9 <- left_join(coldat, distances, by = c("colonizer", "country"))
test_join10 <- anti_join(coldat, distances, by = c("colonizer", "country"))
```