Lab Session 2

Starting Out In Tidyverse

Sarah Moore

Week 2, Day 1

WHAT IS THE TIDYVERSE?

- Mentioned briefly last week that it's this suite of packages in R. We mostly will use verbs associated with `dplyr`.
- Can chain together functions in really elaborate and useful ways so that our code is efficient and easy to debug.
- To get started, let's actually start an R project. As you'll see mine is already set to the repo of this class up in the right-hand corner. All you need to do is click that icon, choose "New Project" and then pick the working directory as the file that you made for this class.
- Once you have done so, we'll explore the `tidyverse`

Pipes and `dplyr`

- `dplyr` is a package meant specifically for data manipulation and transformation.
- We use the `%>%` pipe to bring together verbs in `dplyr` (and sometimes base R) to get outputs that might filter, summarize, or our data (and more).
- Let's see how the pipe works. We can return to the incarceration data from our previous R exercise. It's available in the course GitHub's new "data" folder.

```
# load the tidyverse package
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.0      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# import the data using read_csv() from tidyverse's readr package
incarceration <- read_csv("~/Library/CloudStorage/OneDrive-NorthwesternUniversity/Teaching_RA/2022_Fall_
```

```
## New names:
## Rows: 147533 Columns: 80
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (5): state, county_name, urbanicity, region, division dbl (75): ...1, yfips,
## year, fips, total_pop, total_pop_15to64, female_pop_1...
## 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.
## * `` -> `...1`
```

```
# remember you can also do this via the link (so go to the data download at
#https://github.com/sarah-moore/lousy-graphs/blob/main/data/incarceration_trends.csv)

# let's take a look at our data

#glimpse(incarceration)

# cool, so now let's see what the pipe does; generally look like this:

# df %>%
  # function() %>%
  # other_function() -> new_df

# let's consider the group_by() function in dplyr--
# we'll group by state and see how many observations there are of the urbanicity variable

incarceration %>%
  group_by(state) %>%
  count(urbanicity)
```

```
## # A tibble: 165 x 3
## # Groups:   state [51]
##    state urbanicity     n
##    <chr> <chr>      <int>
##  1 AK    rural       1269
##  2 AK    small/mid    141
##  3 AL    rural       1786
##  4 AL    small/mid   1034
##  5 AL    suburban     282
##  6 AL    urban         47
##  7 AR    rural       2585
##  8 AR    small/mid    893
##  9 AR    suburban      47
## 10 AZ    rural        329
## # ... with 155 more rows
```

```
# what is this telling us
# generally, it's taking the dataframe, asking it to gather the objects from each state
# and count how many times each value of this other variable occur

# let's try another one

# instead let's get the count of NAs on the arson_crime variable, grouped by year
```

```
incarceration %>%
  group_by(year) %>%
  count(is.na(arson_crime)) -> arson_crime_nas


arson_crime_nas<- incarceration %>%
  group_by(year) %>%
  count(is.na(arson_crime))

# can anyone tell me what's going on here?
```

More `dplyr` verbs

We'll get more practice with the pipe operator as we go on. We can also save objects from our piped workflow, as I showed in the general example above. This occurs like you would save any other object in R. However, in a `tidyverse` workflow, it's typical that you'll see the assignment operator flow to the right `->`, as opposed to the typical base R leftwise operator (`<-`). Both achieve the same thing, it's just a convention for this workflow.

`group_by()`

We already visited this function above. Do we get a feel of what it's doing?

```
# how many facilities are categorized along each of the values in the urbanicity variable?

incarceration %>%
  group_by(urbanicity) %>%
  count()
```

```
## # A tibble: 4 x 2
## # Groups:   urbanicity [4]
##   urbanicity     n
##   <chr>      <int>
## 1 rural      92919
## 2 small/mid  34310
## 3 suburban   17296
## 4 urban       3008
```

```
# what are some instances where group_by is not a great option?

incarceration %>%
  group_by(num_facilites) %>%
count()
```

```
## # A tibble: 16 x 2
## # Groups:   num_facilites [16]
##    num_facilites      n
##            <dbl>  <int>
##  1             0 102554
```

```
##  2             1  26790
##  3             2   9494
##  4             3   3619
##  5             4   1833
##  6             5   1269
##  7             6    611
##  8             7    423
##  9             8    235
## 10             9     47
## 11            10    282
## 12            11     94
## 13            13     47
## 14            14     47
## 15            17     47
## 16            NA    141
```

```
# this works okay... but what about this

incarceration %>%
  group_by(latino_jail_pop) %>%
  count()
```

```
## # A tibble: 10,478 x 2
## # Groups:   latino_jail_pop [10,478]
##    latino_jail_pop     n
##              <dbl> <int>
##  1          0      27679
##  2          0.0164     1
##  3          0.0166     1
##  4          0.0168     1
##  5          0.0185     1
##  6          0.0197     1
##  7          0.0205     1
##  8          0.0217     1
##  9          0.0219     1
## 10          0.0219     1
## # ... with 10,468 more rows
```

```
# ... not so great
```

So this might be kind of intuitive, we don't want to group by anything that has too many values. But particularly we can think of this in terms of how we categorize variables– *discrete or continuous.* A discrete variable will take on a countable value, whereas a continuous is non-countable. What does this mean? Let's think for example of some variables in social science:

1) GDP per capita
2) Satisfaction scores with a politician [1:7]
3) Debt ratios
4) Trust in politicians scored 1 to 4

Based on the definition above which are discrete and which are continuous?

Why does this matter?? In data analysis, and more specifically in data visualization, we need to have an idea of the shape that our data *ought* to take, or at least what is possible. Being able to identify whether your data are countable or uncountable has implications for whether we visualize it one way or another.

The `group_by()` function is illustrative of this point, and the importance of identifying variables as *discrete* or *continuous* will come up outside of `group_by()`.

`count()`

I also threw another `dplyr` verb in here without explanation– `count()`. This one is straightforward, it really just counts the number of observations of the variable specified, or the number of observations along the grouping specified.

If you are familiar with histograms, think of this as the data that generates a histogram. It is just the frequency of each observation.

`select()`

Sometimes we want to create a subset of our data. We can use the `select()` verb to do so. This verb allows us to pick the columns of our data that we want and ditch the rest for now.

We'll get into more complicated motivations for using select later on (like when we want to identify certain strings in our data), but for now let's focus on the simple case of just looking for specific variable names.

```
# let's get rid of everything but the state, county_name, and total_pop
# we'll name this subset population_df

incarceration %>%
  select(state, county_name, total_pop) -> population_df

# glimpse(population_df)


# we can also use select as a way to weed out certain variables that we don't want
incarceration %>%
  select(-c(county_name, total_pop))-> incarceration_no_county

# glimpse(incarceration_no_county)
```

Why we would use select in some instances will become more clear as you continue to move forward with some other `dplyr` functions. It will be really helpful as you try to make intermediate objects and want to make more elegant looking dataframes along the way. An intermediate object in `dplyr` is a dataframe or vector that you create with summarized data from your original dataset, but that is not really a subset either. So for example, our count dataframes (tibbles in `dplyr` language) would be intermediate objects.

`filter()`

Another likely option that you will encounter is `filter()`. Rather than filtering out data based on variable names (as `select()` does), `filter()` will get rid or keep data based on specific values that you specify. To do so requires a little bit more knowledge of the way base R deals with Boolean logic. Specifically, we need to know how to specify the following operations:

1) `==`: equals or is exactly equal to
2) `<=`, `>=`: Less than or equal to, greater than or equal to

- `<`, `>`: less than, greater than

3) `!`: not or `!=` not equal
4) `&`: and
5) `|`: or

So when we use filter we are going to tell R to keep some values or get rid of some values based on some condition. These can range in complexity, here we can start with the basics.

```
# let's say we only want to look at the urban jail population in our dataset

incarceration %>%
  filter(urbanicity == "urban")-> urban_incarceration

# we'll check to see if this worked by using the unique() function

unique(urban_incarceration$urbanicity)
```

```
## [1] "urban"
```

```
# we successfully isolated the data to only include the urban incarceration data

# what if we wanted to include urban *or* suburban

incarceration %>%
  filter(urbanicity == "urban" | urbanicity == "suburban")-> urban_sub_incarceration

unique(urban_sub_incarceration$urbanicity)
```

```
## [1] "suburban" "urban"
```

```
# can also see if we have the right number of observations we would expect
# by checking our dimensions against the counts from the full dataset

dim(urban_sub_incarceration) #20,304
```

```
## [1] 20304    80
```

```
incarceration %>%
  group_by(urbanicity)  %>%
  count()
```

```
## # A tibble: 4 x 2
## # Groups:   urbanicity [4]
##   urbanicity     n
##   <chr>      <int>
## 1 rural      92919
## 2 small/mid  34310
## 3 suburban   17296
## 4 urban       3008
```

```
17296+3008 #20,304
```

```
## [1] 20304
```

```
# why would we not specify *AND* in this operation?

incarceration %>%
  filter(urbanicity == "urban" & urbanicity == "suburban")-> urban_and_sub_incarceration
dim(urban_and_sub_incarceration) # why is it ZERO ?
```

```
## [1]  0 80
```

```
# now let's filter the data that is NOT urban
# two ways
incarceration %>%
  filter(urbanicity == "rural" | urbanicity == "small/mid" | urbanicity == "suburban")-> smalltown_inca

dim(smalltown_incarceration)
```

```
## [1] 144525     80
```

```
incarceration %>%
  filter(urbanicity != "urban")-> smalltown_incarceration_2

dim(smalltown_incarceration_2)
```

```
## [1] 144525     80
```

```
# what if we wanted to filter based on the data that accounts with latinx jail population above the mea

# first, check the mean
summary(incarceration$latino_jail_pop) #32.22
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##     0.00     0.00     1.64    32.22     8.00 11293.00    67645
```

```
incarceration %>%
  filter(latino_jail_pop >= mean(latino_jail_pop, na.rm = T)) -> higher_latino_pop

# we should now expect that value from above to be our minimum
summary(higher_latino_pop$latino_jail_pop)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    32.22    50.00    92.00   269.12   221.43 11293.00
```

mutate() and summarize()

So far we haven't really focused on any of the verbs that actually change the data as much as change what data we are working with, i.e. what's in the dataframe.

Two verbs in `dplyr` are especially helpful for actually carrying out the process of data manipulation and transformation. Both can achieve similar tasks, but what you end up with as the resulting output is a bit different.

First, let's visit the *help file* from both of these verbs.

```
# let's create a variable that is a ratio of the people incarcerated for violent crime relative to prop

incarceration %>%
  mutate(violent_prop_ratio = log((violent_crime+0.00000001)/(property_crime+0.00000001))) -> incarcera

summary(incarceration_plus$violent_prop_ratio)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -24.72   -2.89   -2.36   -2.82   -1.85   21.25   36759
```

```
# now let's summarize that by state and get the all time state-level mean using summarize()

incarceration_plus %>%
  summarize(mean_viol_prop_ratio = mean(violent_prop_ratio, na.rm=T),
            median_viol_prop_ratio = median(violent_prop_ratio, na.rm=T),
            min_viol_prop_ratio = min(violent_prop_ratio, na.rm=T))
```

```
## # A tibble: 1 x 3
##   mean_viol_prop_ratio median_viol_prop_ratio min_viol_prop_ratio
##                  <dbl>                  <dbl>               <dbl>
## 1                -2.82                  -2.36               -24.7
```

Joining together data

`dplyr` also has multiple ways that you will join together data. To do so, you need to find a key variable between the data so that R can bind it together in some way. The easiest way to ensure you know which join you should be doing is to check out the help file using `?join`.

```
?join

# i will use some code to split up the data into two dataframes

incarceration %>%
  select(year, fips, state, county_name, robbery_crime, burglary_crime, larceny_crime)-> crime_stealing

incarceration %>%
  select(-c(robbery_crime, burglary_crime, larceny_crime))-> less_stealing

# now i want to merge the two back together
# i need to pick the variables to do it on (year, fips, state, county_name)
# i'll use inner_join which gives us the rows in x and y
```

```
less_stealing %>%
  inner_join(crime_stealing, by=c('year', 'fips', 'state', 'county_name'))-> new_incarceration

dim(incarceration)
```

```
## [1] 147533     80
```

```
dim(new_incarceration)
```

```
## [1] 147533     80
```

```
# which variables were most consequential?

less_stealing %>%
  left_join(crime_stealing, by=c('year', 'fips', 'state', 'county_name'))-> left_incarceration
dim(left_incarceration)
```

```
## [1] 147533     80
```

```
less_stealing %>%
  right_join(crime_stealing, by=c('year', 'fips', 'state', 'county_name'))-> right_incarceration
dim(right_incarceration)
```

```
## [1] 147533     80
```

```
less_stealing %>%
  full_join(crime_stealing, by=c('year', 'fips', 'state', 'county_name'))-> full_incarceration
dim(full_incarceration)
```

```
## [1] 147533     80
```