Integrating Migration Data

In the last two modules, we built familiarity with spatial data, especially rasters. In this module, we will integrate our knowledge of data cleaning with the tidyverse and spatial information from rasters. We'll also manipulate multiple rows/counties at once.

Objectives

1. Calculate statistics for multiple counties at once using `summarise`.
2. Determine the difference in population size and density from source counties to study area counties.
3. Join land cost data to migration data and calculate the difference in land value, using the `exactextractr` package.
4. Determine the average household size of source counties.

Getting Started

(1) Open RStudio and your R project file. **(2)** Create a new Quarto file.

Migration Data

(3) Load the `tidyverse`, `terra`, `sf`, and `tidycensus` libraries into your `R` session.

(4) Download the `county-to-county-2016-2020-ID-newcolnames.csv` file into your `data` folder.

(5) Load the data into your `R` session.

This data estimates the number of people moving (`Movers.in.County.to.County.Flow_Estimate`) from locations around the world (`State.U.S..Island.Area.Foreign.Region.of.Residence.1.Year.Ago` and `County.of.Residence.1.Year.Ago`) to counties in Idaho (`County.of.Current.Residence`). There is also some population data about the source and destination counties. Any columns with `MOE` are the *margin of error* for the estimate column to its left.

(6) `subset` the migration data to a list of counties you're interested in with the `%in%` operator.

```
migration_filt <- subset(migration,
                         County.of.Current.Residence %in% c("____", "_____"))
```

Population Estimates

Average Percent Difference in Population

The migration data have estimates for the total population of the source (`County.of.Residence.1.Year.Ago_Population.1.Year.and.Over_Estimate`) and destination (`County.of.Current.Residence_Population.1.Year.and.Over_Estimate`) counties. **(7)** Create a new column containing the source population subtracted from the destination population.

(8) Create a new column for the percent difference. Divide the population difference by the population of the source county and multiply by 100.

To get one summary statistic for each Idaho county, we need to take a *weighted average* to fairly represent the overall trend. We will need to scale the percent difference column by the number of movers, then calculate an average for each county.

The general equation we'll work with is:

$$\text{Weighted Population Difference} = \text{Population Difference} \cdot \frac{\text{Movers From Destination County}}{\text{Movers From All Destinations}}$$

We can then **add** all weighted population differences to find the weighted average population distance.

The only piece of information we're missing is the number of movers from all destinations to each county. **(9)** Use `summarise` to create a new `data.frame` with the `sum` of all movers to each Idaho county (see module 4 (30) for a refresher if needed).

```
tot_movers <- summarise(migration_filt,
                        .by = ________,
                        total_movers = sum(_________, na.rm=_____))
print(tot_movers)
```

```
  County.of.Current.Residence total_movers
1                  Ada County        32954
2                Boise County          522
3               Canyon County        18254
4               Elmore County         3073
5                  Gem County          956
6               Owyhee County         1199
7              Payette County         1972
```

Now, we need these numbers to be associated with every row of our migration data. We can do this with a `left_join`, making sure to list migration data first (for a refresher on joins, see module 4 pages 4-6).

```
migration_filt <- left_join(migration_filt,
                            tot_movers,
                            by = "County.of.Current.Residence")
```

We have every piece of information needed for the equation above. **(10)** Create a `weight` column, for this and all future calculations, containing the number of movers from the source county divided by the total number of movers to the destination county.

```
migration_filt$weight <-
  migration_filt$Movers.in.County.to.County.Flow_Estimate /
  migration_filt$total_movers
```

(11) Create a new column for the weighted population difference percentage by multiplying the population difference percentage by the `weight` column.

(12) Find the `sum` of weighted population difference percentages for each county using `summarise`.

```
  County.of.Current.Residence w_avg_pop_diff_perc
1                  Ada County           582.20734
2                Boise County           -81.18719
3               Canyon County           192.95323
4               Elmore County           -49.34806
5                  Gem County           -75.02042
6               Owyhee County           -74.53066
7              Payette County           -34.67595
```

On average, Ada and Canyon counties have a much higher population than their source counties, being 582% and 192% more populated on average respectively. The surrounding counties have lower populations than their source counties, ranging from 34% to 81% less populated.

Average Difference in Population Density

Another way to measure population is population density, or the average population per square mile. We already have population data for the counties in our migration data, so we need to find the area of each county in order to calculate population density.

The `tigris` library we used in modules 7 and 8 has county area data in square meters. We need to read in `tigris` data for all counties in the US, convert the areas to square miles, and join that data to the migration data.

(13) Use the `tigris` package to read in county data for the entire US.

```r
library(tigris)

us_counties <- counties()
```

(14) Create a new column called `ALAND_sq_mi` and convert the `ALAND` column from square meters to square miles. 1 square mile is 2,589,988.110336 square meters.

(15) Before we join, let's keep our data simple by keeping only the `ALAND_sq_mi`, `STATEFP`, and `COUNTYFP` columns in the counties data, dropping all other columns. (Hint: use `select()`.)

We'll be joining by the STATEFP (FIPS) codes and the COUNTYFP (FIPS) codes in the `tigris` data and the state codes and county FIPS codes in the migration data. **(16)** Take a look at each of those columns in the two `data.frames`. You should see that they are not exact matches – the migration data has different rules for leading zeroes than the `tigris` data. We'll need to fix this before we join since `R` always looks for exact text matches.

First, you will notice that the `migration` state FIPS codes have a leading zero that is absent in the `tigris` county area data. **(17)** Add a zero to the beginning of the county area state codes with the `paste0` function.

```
us_counties$STATEFP <- paste0("0", us_counties$STATEFP)
```

Next, you should see that the county FIPS codes in the migration data don't have leading zeroes like the county area data do. This is a little trickier than the state codes, because we might have to add one *or* two leading zeroes to have three digits total. In this case, `str_pad` from the `stringr` library can help. **(18)** Pad the codes for `Current.Residence.FIPS.County.Code` and `Residence.1.Year.Ago.FIPS.County.Code` to a length of 3 by adding `"0"` to the left side (see `?str_pad` for more information).

```
migration_filt$Current.Residence.FIPS.County.Code <-
  str_pad(string = migration_filt$Current.Residence.FIPS.County.Code,
          width = 3,
          side = "left",
          pad="0")

# do again for previous county codes
```

Now our codes are compatible for two joins: we will want a column for the current county's area and the previous county's area. **(19)** Use a `left_join` to join the migration and county area data by current state and county FIPS codes. You can indicate which columns should match each other even if they have different names.

```
migration_filt_areas <-
  left_join(migration_filt,
            us_counties,
            by = c("Current.Residence.State.Code" = "STATEFP",
                   "Current.Residence.FIPS.County.Code" = "COUNTYFP"))
```

(20) Rename the column `ALAND_sq_mi` to `curr_county_area_sq_mi` with the `rename` function. In this function, the new name is listed first, followed by an equal sign, and then the current column name.

```
migration_filt_areas <- rename(migration_filt_areas,
                               curr_county_area_sq_mi = ALAND_sq_mi)
```

(21) Repeat step (19), this time joining the `migration_filt_areas` data to the `us_counties` data by the columns `Residence.1.Year.Ago.State.U.S..Island.Area.Foreign.Region.Code` and `Residence.1.Year.Ago.FIPS.County.Code`. It's important to note here that we are only using data from the US, so this analysis is limited to domestic migration and we will have some rows with `NA`s after this join for international source areas.

```
migration_filt_areas <-
  left_join(migration_filt_areas,
            ___________,
            by = c("_________" = "________",
                   "_________" = "________"))
```

(22) Rename the `ALAND_sq_mi` column to `prev_county_area_sq_mi`.

(23) Create two new columns with the population density of the current and previous counties by dividing the correct population column by its corresponding area column.

(24) Create a new column with the difference between the current and previous population density.

(25) Multiply the difference in population density by the weight column to get the weighted population density difference.

```
migration_filt_areas$pop_dens_diff_w <-
  migration_filt_areas$pop_dens_diff * migration_filt_areas$weight
```

(26) Find the weighted average population density difference for each Idaho county with `summarise`.

```
  County.of.Current.Residence w_avg_pop_dens_diff
1                  Ada County           -321.5843
2                Boise County           -369.5272
3               Canyon County           -157.0293
4               Elmore County           -419.2276
5                  Gem County           -326.5419
6               Owyhee County           -285.4928
7              Payette County           -221.6339
```

For these west-central Idaho counties, migrants are on average moving in from more densely populated counties.

Land Cost

(27) Download the land cost raster for the entire US (`Nolte_2020_fair_market_value_USA.tif`) into your `data` folder and read it in to your `R` session.

As in module 7, this raster shows an estimate of the fair market value (land cost) in the natural log of US dollars per hectare (more information is available in Nolte, 2020). **(28)** Transform the cost data to $/ha.

```
land_value_trans <- exp(land_value)
```

Just like with the population data, we need the average land value for both the Idaho (destination) counties and the source counties. Rather than cropping to each county individually, we can use the `exactextractr::exact_extract` function to extract average values for many polygons. Additionally, because we did two joins in this module, we already have polygons for both the source (column `geometry.y`) and destination (column `geometry.x`) counties.

(29) First, select the codes for the current state and county of residence, as well as their corresponding geometries.

```
curr_county_polygons <- select(migration_filt_areas,
                               Current.Residence.State.Code,
                               Current.Residence.FIPS.County.Code,
                               geometry.x)
```

(30) Since we have a lot of repeat rows, we can make our calculations faster by getting rid of the duplicate rows. `unique()` keeps only the unique rows.

```
curr_county_polygons <- unique(curr_county_polygons)
```

Since we didn't load in our data with `st_read`, `R` doesn't know that these data are really spatial data. **(31)** Convert the polygons to a simple features (`sf`) spatial type object. Then, project the county polygons to the CRS of the raster in order to ensure a correct overlay.

```
curr_county_polygons <- st_as_sf(curr_county_polygons)

curr_county_polygons <- st_transform(curr_county_polygons,
                                     crs = st_crs(land_value_trans))
```

(32) Use the code in steps 29-31 to create a similar object for previous county polygons called `prev_county_polygons`.

There's a little extra work we need to do here. Some of our polygons are empty (you can check how many with `table(st_is_empty(prev_county_polygons))`). We need to filter out the empty polygons. **(33)** Use `st_is_empty` and `subset` to filter out empty polygons. The `!` means "not", so we are asking for a subset of `prev_county_polygons` where polygons are not empty.

```
prev_county_polygons_fixed <- subset(prev_county_polygons,
                                     !st_is_empty(prev_county_polygons))
```

exactextractr::exact_extract is a method to summarise raster information for a number of polygons at once. terra has a method for this as well, but exactextractr is much faster for large rasters like this one!

(34) In the Console, install the exactextractr package. Then, load it into your Quarto file.

(35) Extract the mean land value for all the current county polygons. The append_cols argument lets us keep the FIPS code columns with our results.

```
curr_county_land_value <-
  exact_extract(land_value_trans,
                curr_county_polygons,
                fun="mean",
                append_cols = c("Current.Residence.State.Code",
                                "Current.Residence.FIPS.County.Code"))
```

(36) Print the structure (str()) of the curr_county_land_value results. It should be a data.frame of with a mean column holding the mean land value and the state and county code columns.

(37) Rename the mean column to curr_county_land_value. There's many ways to do this, but an easy one is the rename function in the tidyverse.

```
curr_county_land_value <- rename(curr_county_land_value,
                                 curr_county_land_value = mean)
```

(38) Join the current county land values to the migration data with a left_join by the state and county codes. By inputting the migrations data first in a left_join, we retain its structure. Also, left_join can automatically tell which columns to join by in this case because the column names are the same (we didn't change them when we made our subset).

```
migration_filt_lv <- left_join(migration_filt_areas,
                               curr_county_land_value)
```

(39) Repeat steps 35-38 for the previous residence counties, making sure to use the "fixed" data without the empty polygons. Make sure to left_join to the new dataset we created, migration_filt_lv, this time, not the old dataset migration_filt_areas.

When you print the str for the previous county land values, you may notice some NaNs. There are some counties outside the contiguous US that we don't have land value data for (like Alaska).

(40) Create a new column with the difference between the current and previous county's land value.

(41) Create a new column with the difference column multiplied by the `weight` column.

(42) `summarise` the sum of the weighted differences by county.

```
  County.of.Current.Residence w_avg_land_value_diff
1                  Ada County            -116884.07
2                Boise County             -52229.58
3               Canyon County             -65186.75
4               Elmore County             -58226.78
5                  Gem County             -51535.95
6               Owyhee County             -40514.44
7              Payette County             -34604.19
```

On average, land values in these Idaho counties are lower than the counties people are moving from.

Household Size

The US Census hosts household size data, which we can pull directly in `R` with `tidycensus`.

(43) Import the Census API key you received in module 5 with `census_api_key("YOUR API KEY GOES HERE")`. **(44)** Hide your API key in the report with `{r, include=FALSE}`.

(45) Household size has the variable code `B25010_001` in the American Community Survey. We can retrieve the data for this code with `get_acs`, using the `unique` current states and counties in our migration data. Retrieve the 2020 household size data for the current counties of residence:

```r
curr_house_size <- get_acs(geography = "county",
                    state = unique(migration_filt$State.of.Current.Residence),
                    county = unique(migration_filt$County.of.Current.Residence),
                    year = 2020,
                    survey = "acs5",
                    variable = "B25010_001")
```

If you don't supply the state or county codes to `get_acs`, it will retrieve data for all the counties in the US. **(46)** Retrieve 2016 data for household size for all counties in the US.

(47) For both current and previous household size `data.frames`, `select` the `NAME` and `estimate` columns. `rename` the `estimate` columns to `curr_house_size_est` and `prev_house_size_est`, respectively.

In order to join these data, we need matching key columns describing the county in exactly the same text format. There are a number of ways we could do this, but here, I'll show how to join based on the `NAME` column in the `tidycensus` data. We have this information in two separate columns in the migration data, so we need to paste those columns together to match the ‘tidycensus formatting:

```
head(curr_house_size[, "NAME"])
```

```
# A tibble: 6 x 1
  NAME
  <chr>
1 Ada County, Idaho
2 Boise County, Idaho
3 Canyon County, Idaho
4 Elmore County, Idaho
5 Gem County, Idaho
6 Owyhee County, Idaho
```

```
# we need the migration data to match the format above

head(migration_filt[, c("County.of.Current.Residence",
                        "State.of.Current.Residence")])
```

```
  County.of.Current.Residence State.of.Current.Residence
1                  Ada County                      Idaho
2                  Ada County                      Idaho
3                  Ada County                      Idaho
4                  Ada County                      Idaho
5                  Ada County                      Idaho
6                  Ada County                      Idaho
```

(48) Create a column called `curr_NAME` in your filtered migration data with the current state and county names pasted together with a comma and single space between them.

```
migration_filt$curr_NAME <- paste0(migration_filt$County.of.Current.Residence,
                                   ", ",
                                   migration_filt$State.of.Current.Residence)
```

(49) The `setdiff` function can help us check that this worked by showing any entries in the migration data that are not in the household size data. Check that `setdiff` shows no differences between the key columns:

```r
setdiff(migration_filt$curr_NAME, curr_house_size$NAME)
```

```
character(0)
```

(50) Repeating step (48), create a column called `prev_NAME` in your filtered migration data with the previous (`1.Year.Ago`) state and county names pasted together with a comma and single space between them.

(51) Check the `setdiff` for these key columns. You should see some differences:

```r
setdiff(migration_filt$prev_NAME, prev_house_size$NAME)
```

```
 [1] "Chugach Census Area, Alaska"   "Chesapeake City, Virginia"
 [3] "Virginia Beach City, Virginia" "-, Africa"
 [5] "-, Asia"                       "-, Central America"
 [7] "-, Caribbean"                  "-, Europe"
 [9] "-, U.S. Island Areas"          "-, Oceania and At Sea"
[11] "-, South America"              "-, Northern America"
[13] "Hampton City, Virginia"
```

We can ignore the non-US areas because we don't have data for those. However, there are some US counties that we might be able to resolve. **(52)** Search for `NAME`s in the household size data that might match the `prev_NAME`s in the migration data with `str_starts`.

```r
subset(prev_house_size, str_starts(prev_house_size$NAME, "Chugach"))
```

```
# A tibble: 0 x 2
# i 2 variables: NAME <chr>, prev_house_size_est <dbl>
```

```r
subset(prev_house_size, str_starts(prev_house_size$NAME, "Chesapeake"))
```

```
# A tibble: 1 x 2
  NAME                        prev_house_size_est
  <chr>                                     <dbl>
1 Chesapeake city, Virginia                  2.74
```

```r
subset(prev_house_size, str_starts(prev_house_size$NAME, "Virginia"))
```

```
# A tibble: 1 x 2
  NAME                              prev_house_size_est
  <chr>                                           <dbl>
1 Virginia Beach city, Virginia                    2.62
```

```
subset(prev_house_size, str_starts(prev_house_size$NAME, "Hampton"))
```

```
# A tibble: 2 x 2
  NAME                               prev_house_size_est
  <chr>                                            <dbl>
1 Hampton County, South Carolina                    2.59
2 Hampton city, Virginia                            2.47
```

There is no data for Chugach Census Area, Alaska in the household size data, but it looks like the other differences are caused by "city" being uppercase in the migration data and lowercase in the household size data. **(53)** We can resolve this by replacing "city" with "City" in the household size data.

```
prev_house_size$NAME <- str_replace(string = prev_house_size$NAME,
                                    pattern = "city",
                                    replacement = "City")
```

When we check the `setdiff` again, we can see that we've resolved all the mismatches that we could:

```
setdiff(migration_filt$prev_NAME, prev_house_size$NAME)
```

```
 [1] "Chugach Census Area, Alaska" "-, Africa"
 [3] "-, Asia"                     "-, Central America"
 [5] "-, Caribbean"                "-, Europe"
 [7] "-, U.S. Island Areas"        "-, Oceania and At Sea"
 [9] "-, South America"            "-, Northern America"
```

(54) `left_join` the migration data and current household size data by their matching name columns (Hint: see step 19). **(55)** Do the same for the previous household size data.

(56) Create a new column with the difference between the previous and current county average household sizes.

(57) Multiply the difference column by the `weight` column.

(58) `summarise` the average household size difference for your counties by summing the weighted household size differences.

```
               curr_NAME w_avg_household_size_diff
1      Ada County, Idaho                -0.1934199
2    Boise County, Idaho                -0.3578736
3   Canyon County, Idaho                 0.2215394
4   Elmore County, Idaho                -0.1707062
5      Gem County, Idaho                -0.1979916
6   Owyhee County, Idaho                -0.1633778
7  Payette County, Idaho                -0.1085294
```

On average, people moving to the Treasure Valley are moving from counties with a higher average household size, with the exception of Canyon County.

Finishing up

(59) Go back through your report and add short explanations for what each code chunk does in your own words if you haven't done so already. **(60)** Render your report to a PDF and email it to [INSERT EMAIL HERE].

Statement of original and referenced work:

The entirety of this module is original work authored by Carolyn Koehn.

License