Assignment 4

Adrian Bracher (Matr. Nr. 01637180)

21.04.2021

Contents

1 Transforming Data with dplyr

1.1 Selecting columns

We select the columns state, county, population and poverty with a pipe syntax like below.

```
# Select the columns
counties %>%
  select(state, county, population, poverty)
```

```
## # A tibble: 3,138 x 4
##    state   county   population poverty
##    <chr>   <chr>         <dbl>   <dbl>
##  1 Alabama Autauga       55221    12.9
##  2 Alabama Baldwin      195121    13.4
##  3 Alabama Barbour       26932    26.7
##  4 Alabama Bibb          22604    16.8
##  5 Alabama Blount        57710    16.7
##  6 Alabama Bullock       10678    24.6
##  7 Alabama Butler        20354    25.4
##  8 Alabama Calhoun      116648    20.5
##  9 Alabama Chambers      34079    21.6
## 10 Alabama Cherokee      26008    19.2
## # ... with 3,128 more rows
```

1.2 Arranging observations

In this exercise we sort the selected columns in descending order on the column public_work.

```
counties_selected <- counties %>%
  select(state, county, population, private_work, public_work, self_employed)

# Add a verb to sort in descending order of public_work
counties_selected %>%
  arrange(desc(public_work))
```

```
## # A tibble: 3,138 x 6
##    state     county             population private_work public_work self_employed
##    <chr>     <chr>                   <dbl>        <dbl>       <dbl>         <dbl>
##  1 Hawaii    Kalawao                    85         25          64.1          10.9
##  2 Alaska    Yukon-Koyukuk Ce~        5644         33.3        61.7           5.1
##  3 Wisconsin Menominee                4451         36.8        59.1           3.7
##  4 North Da~ Sioux                    4380         32.9        56.8          10.2
##  5 South Da~ Todd                     9942         34.4        55             9.8
##  6 Alaska    Lake and Peninsu~        1474         42.2        51.6           6.1
##  7 Californ~ Lassen                  32645         42.6        50.5           6.8
##  8 South Da~ Buffalo                  2038         48.4        49.5           1.8
##  9 South Da~ Dewey                    5579         34.9        49.2          14.7
## 10 Texas     Kenedy                    565         51.9        48.1           0
## # ... with 3,128 more rows
```

1.3 Filtering for conditions

Filter() can be used like below to filter the data using one (or more) criteria.

```
counties_selected <- counties %>%
  select(state, county, population)
```

```r
# Filter for counties in the state of California that have a population above 1000000
counties_selected %>%
  filter(state == "California", population > 1000000)
```

```
## # A tibble: 9 x 3
##   state      county         population
##   <chr>      <chr>               <dbl>
## 1 California Alameda           1584983
## 2 California Contra Costa      1096068
## 3 California Los Angeles      10038388
## 4 California Orange            3116069
## 5 California Riverside         2298032
## 6 California Sacramento        1465832
## 7 California San Bernardino    2094769
## 8 California San Diego          3223096
## 9 California Santa Clara       1868149
```

1.4 Filtering and arranging

The functions arrange() and filter() can also be used consecutively.

```r
counties_selected <- counties %>%
  select(state, county, population, private_work, public_work, self_employed)

# Filter for Texas and more than 10000 people; sort in descending order of private_work
counties_selected %>%
  arrange(desc(private_work)) %>%
  filter(state == "Texas", population > 10000)
```

```
## # A tibble: 169 x 6
##    state county  population private_work public_work self_employed
##    <chr> <chr>        <dbl>        <dbl>       <dbl>         <dbl>
##  1 Texas Gregg       123178         84.7         9.8           5.4
##  2 Texas Collin      862215         84.1        10             5.8
##  3 Texas Dallas     2485003         83.9         9.5           6.4
##  4 Texas Harris     4356362         83.4        10.1           6.3
##  5 Texas Andrews      16775         83.1         9.6           6.8
##  6 Texas Tarrant    1914526         83.1        11.4           5.4
##  7 Texas Titus        32553         82.5        10             7.4
##  8 Texas Denton      731851         82.2        11.9           5.7
##  9 Texas Ector       149557         82          11.2           6.7
## 10 Texas Moore        22281         82          11.7           5.9
## # ... with 159 more rows
```

1.5 Calculating the number of government employees

In this exercise we learn that mutate can be used to introduce a new column to the data frame and also that it can be filled with a calculation using the data in other columns.

```r
counties_selected <- counties %>%
  select(state, county, population, public_work)

# Sort in descending order of the public_workers column
counties_selected %>%
```

```
  mutate(public_workers = public_work * population / 100) %>%
  arrange(desc(public_workers))
```

```
## # A tibble: 3,138 x 5
##    state      county         population public_work public_workers
##    <chr>      <chr>               <dbl>       <dbl>          <dbl>
##  1 California Los Angeles      10038388        11.5       1154415.
##  2 Illinois   Cook              5236393        11.5        602185.
##  3 California San Diego         3223096        14.8        477018.
##  4 Arizona    Maricopa          4018143        11.7        470123.
##  5 Texas      Harris            4356362        10.1        439993.
##  6 New York   Kings             2595259        14.4        373717.
##  7 California San Bernardino    2094769        16.7        349826.
##  8 California Riverside         2298032        14.9        342407.
##  9 California Sacramento        1465832        21.8        319551.
## 10 California Orange            3116069        10.2        317839.
## # ... with 3,128 more rows
```

1.6 Calculating the percentage of women in a county

```
# Select the columns state, county, population, men, and women
counties_selected <- counties %>%
    select(state, county, population, men, women)


# Calculate proportion_women as the fraction of the population made up of women
counties_selected %>%
    mutate(proportion_women = women/population)
```

```
## # A tibble: 3,138 x 6
##    state   county   population   men women proportion_women
##    <chr>   <chr>         <dbl> <dbl> <dbl>            <dbl>
##  1 Alabama Autauga       55221 26745 28476            0.516
##  2 Alabama Baldwin      195121 95314 99807            0.512
##  3 Alabama Barbour       26932 14497 12435            0.462
##  4 Alabama Bibb          22604 12073 10531            0.466
##  5 Alabama Blount        57710 28512 29198            0.506
##  6 Alabama Bullock       10678  5660  5018            0.470
##  7 Alabama Butler        20354  9502 10852            0.533
##  8 Alabama Calhoun      116648 56274 60374            0.518
##  9 Alabama Chambers      34079 16258 17821            0.523
## 10 Alabama Cherokee      26008 12975 13033            0.501
## # ... with 3,128 more rows
```

1.7 Select, mutate, filter and arrange

In this exercise we combine everything from this chapter.

```
counties %>%
  # Select the five columns
  select(state, county, population, men, women) %>%
  # Add the proportion_men variable
  mutate(proportion_men = men/population) %>%
  # Filter for population of at least 10,000
```

```
  filter(population > 10000) %>%
  # Arrange proportion of men in descending order
  arrange(desc(proportion_men))
```

```
## # A tibble: 2,437 x 6
##    state      county          population   men women proportion_men
##    <chr>      <chr>                <dbl> <dbl> <dbl>          <dbl>
##  1 Virginia   Sussex               11864  8130  3734          0.685
##  2 California Lassen               32645 21818 10827          0.668
##  3 Georgia    Chattahoochee        11914  7940  3974          0.666
##  4 Louisiana  West Feliciana       15415 10228  5187          0.664
##  5 Florida    Union                15191  9830  5361          0.647
##  6 Texas      Jones                19978 12652  7326          0.633
##  7 Missouri   DeKalb               12782  8080  4702          0.632
##  8 Texas      Madison              13838  8648  5190          0.625
##  9 Virginia   Greensville          11760  7303  4457          0.621
## 10 Texas      Anderson             57915 35469 22446          0.612
## # ... with 2,427 more rows
```

2 Aggregating Data

2.1 Counting by region

Count() is used here to count the number of regions.

```
# Use count to find the number of counties in each region
counties_selected %>%
  count(region, sort=TRUE)
```

2.2 Counting citizens by state

A weight can be applied too by specifying the parameter wt of the function count.

```
counties_selected <- counties
```

```
# Find number of counties per state, weighted by citizens
counties_selected %>%
  count(state, wt=citizens, sort = TRUE)
```

```
## # A tibble: 50 x 2
##    state                 n
##    <chr>             <dbl>
##  1 California     24280349
##  2 Texas          16864864
##  3 Florida        13933052
##  4 New York       13531404
##  5 Pennsylvania    9710416
##  6 Illinois        8979999
##  7 Ohio            8709050
##  8 Michigan        7380136
##  9 North Carolina  7107998
## 10 Georgia         6978660
## # ... with 40 more rows
```

2.3 Mutating and counting

Mutate and count can be combined for advanced counting with weights.

```
counties_selected %>%
  # Add population_walk containing the total number of people who walk to work
  mutate(population_walk = population*walk/100) %>%
  count(state, wt=population_walk, sort=TRUE)
```

```
## # A tibble: 50 x 2
##    state                n
##    <chr>            <dbl>
##  1 New York      1237938.
##  2 California    1017964.
##  3 Pennsylvania   505397.
##  4 Texas          430783.
##  5 Illinois       400346.
##  6 Massachusetts  316765.
##  7 Florida        284723.
##  8 New Jersey     273047.
##  9 Ohio           266911.
## 10 Washington     239764.
## # ... with 40 more rows
```

2.4 Summarizing

Aggregations can be made with the function summarizing, for example calculating the min, max and mean like below.

```
# Summarize to find minimum population, maximum unemployment, and average income
counties_selected %>%
  summarize(min_population = min(population), max_unemployment=max(unemployment), average_income=mean(i
```

```
## # A tibble: 1 x 3
##   min_population max_unemployment average_income
##            <dbl>            <dbl>          <dbl>
## 1             85             29.4         46832.
```

2.5 Summarizing by state

In the exercise below we calculate the population density for each state.

```
# Add a density column, then sort in descending order
counties_selected %>%
  group_by(state) %>%
  summarize(total_area = sum(land_area),
            total_population = sum(population)) %>%
  mutate(density = total_population/total_area) %>%
    arrange(desc(density))
```

```
## # A tibble: 50 x 4
##    state         total_area total_population density
##    <chr>              <dbl>            <dbl>   <dbl>
##  1 New Jersey         7354.          8904413   1211.
##  2 Rhode Island       1034.          1053661   1019.
##  3 Massachusetts      7800.          6705586    860.
##  4 Connecticut        4842.          3593222    742.
```

```
##   5 Maryland           9707.          5930538    611.
##   6 Delaware           1949.           926454    475.
##   7 New York          47126.         19673174    417.
##   8 Florida           53625.         19645772    366.
##   9 Pennsylvania      44743.         12779559    286.
##  10 Ohio              40861.         11575977    283.
## # ... with 40 more rows
```

2.6 Summarizing by state and region

In this exercise we learn that we can group by multiple columns at once.

```
# Calculate the average_pop and median_pop columns
counties_selected %>%
  group_by(region, state) %>%
  summarize(total_pop = sum(population)) %>%
  summarize(average_pop = mean(total_pop),
    median_pop = median(total_pop))
```

```
## `summarise()` has grouped output by 'region'. You can override using the `.groups` argument.
```

```
## # A tibble: 4 x 3
##   region        average_pop median_pop
##   <chr>               <dbl>      <dbl>
## 1 North Central    5627687.    5580644
## 2 Northeast        6221058.    3593222
## 3 South            7370486     4804098
## 4 West             5722755.    2798636
```

2.7 Selecting a county from each region

top_n is then used to select the n (the first parameter, in this case 1) rows with the most walking-to-work citizens.

```
# Group by region and find the greatest number of citizens who walk to work
counties_selected %>%
  group_by(region) %>%
  top_n(1, walk)
```

```
## # A tibble: 4 x 40
## # Groups:   region [4]
##   census_id state  county   region metro population    men  women hispanic white
##   <chr>     <chr>  <chr>    <chr>  <chr>      <dbl>  <dbl>  <dbl>    <dbl> <dbl>
## 1 2013      Alaska Aleutia~ West   Nonm~       3304   2198   1106     12    15
## 2 36061     New Y~ New York North~ Metro    1629507 769434 860073     25.8  47.1
## 3 38051     North~ McIntosh North~ Nonm~       2759   1341   1418      0.9  95.8
## 4 51678     Virgi~ Lexingt~ South  Nonm~       7071   4372   2699      3.9  75.4
## # ... with 30 more variables: black <dbl>, native <dbl>, asian <dbl>,
## #   pacific <dbl>, citizens <dbl>, income <dbl>, income_err <dbl>,
## #   income_per_cap <dbl>, income_per_cap_err <dbl>, poverty <dbl>,
## #   child_poverty <dbl>, professional <dbl>, service <dbl>, office <dbl>,
## #   construction <dbl>, production <dbl>, drive <dbl>, carpool <dbl>,
## #   transit <dbl>, walk <dbl>, other_transp <dbl>, work_at_home <dbl>,
## #   mean_commute <dbl>, employed <dbl>, private_work <dbl>, public_work <dbl>,
## #   self_employed <dbl>, family_work <dbl>, unemployment <dbl>, land_area <dbl>
```

2.8 Finding the highest-income state in each region

In this exercise we select the highest-income state for each region.

```
counties_selected %>%
  group_by(region, state) %>%
  # Calculate average income
  summarize(average_income = mean(income)) %>%
  # Find the highest income state in each region
  top_n(1, average_income)
```

```
## `summarise()` has grouped output by 'region'. You can override using the `.groups` argument.
```

```
## # A tibble: 4 x 3
## # Groups:   region [4]
##   region        state        average_income
##   <chr>         <chr>                 <dbl>
## 1 North Central North Dakota         55575.
## 2 Northeast     New Jersey           73014.
## 3 South         Maryland             69200.
## 4 West          Alaska               65125.
```

2.9 Using summarize, top_n, and count together

In this exercise we combine everything we've learned in this chapter to count the number of states that have more people living in Metro areas and the ones that have more people living in Nonmetro areas.

```
# Count the states with more people in Metro or Nonmetro areas
counties_selected %>%
  group_by(state, metro) %>%
  summarize(total_pop = sum(population)) %>%
  top_n(1, total_pop)  %>%
  ungroup()  %>%
  count(metro)
```

```
## `summarise()` has grouped output by 'state'. You can override using the `.groups` argument.
```

```
## # A tibble: 2 x 2
##   metro        n
##   <chr>    <int>
## 1 Metro       44
## 2 Nonmetro     6
```

3 Selecting and Transforming Data

3.1 Selecting columns

In the code below different columns are selected and then sorted by the column service.

```
# Glimpse the counties table
glimpse(counties)
```

```
## Rows: 3,138
## Columns: 40
## $ census_id          <chr> "1001", "1003", "1005", "1007", "1009", "1011", "10~
## $ state              <chr> "Alabama", "Alabama", "Alabama", "Alabama", "Alabam~
## $ county             <chr> "Autauga", "Baldwin", "Barbour", "Bibb", "Blount", ~
## $ region             <chr> "South", "South", "South", "South", "South", "South~
```

```
## $ metro              <chr> "Metro", "Metro", "Nonmetro", "Metro", "Metro", "No~
## $ population         <dbl> 55221, 195121, 26932, 22604, 57710, 10678, 20354, 1~
## $ men                <dbl> 26745, 95314, 14497, 12073, 28512, 5660, 9502, 5627~
## $ women              <dbl> 28476, 99807, 12435, 10531, 29198, 5018, 10852, 603~
## $ hispanic           <dbl> 2.6, 4.5, 4.6, 2.2, 8.6, 4.4, 1.2, 3.5, 0.4, 1.5, 7~
## $ white              <dbl> 75.8, 83.1, 46.2, 74.5, 87.9, 22.2, 53.3, 73.0, 57.~
## $ black              <dbl> 18.5, 9.5, 46.7, 21.4, 1.5, 70.7, 43.8, 20.3, 40.3,~
## $ native             <dbl> 0.4, 0.6, 0.2, 0.4, 0.3, 1.2, 0.1, 0.2, 0.2, 0.6, 0~
## $ asian              <dbl> 1.0, 0.7, 0.4, 0.1, 0.1, 0.2, 0.4, 0.9, 0.8, 0.3, 0~
## $ pacific            <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0~
## $ citizens           <dbl> 40725, 147695, 20714, 17495, 42345, 8057, 15581, 88~
## $ income             <dbl> 51281, 50254, 32964, 38678, 45813, 31938, 32229, 41~
## $ income_err         <dbl> 2391, 1263, 2973, 3995, 3141, 5884, 1793, 925, 2949~
## $ income_per_cap     <dbl> 24974, 27317, 16824, 18431, 20532, 17580, 18390, 21~
## $ income_per_cap_err <dbl> 1080, 711, 798, 1618, 708, 2055, 714, 489, 1366, 15~
## $ poverty            <dbl> 12.9, 13.4, 26.7, 16.8, 16.7, 24.6, 25.4, 20.5, 21.~
## $ child_poverty      <dbl> 18.6, 19.2, 45.3, 27.9, 27.2, 38.4, 39.2, 31.6, 37.~
## $ professional       <dbl> 33.2, 33.1, 26.8, 21.5, 28.5, 18.8, 27.5, 27.3, 23.~
## $ service            <dbl> 17.0, 17.7, 16.1, 17.9, 14.1, 15.0, 16.6, 17.7, 14.~
## $ office             <dbl> 24.2, 27.1, 23.1, 17.8, 23.9, 19.7, 21.9, 24.2, 26.~
## $ construction       <dbl> 8.6, 10.8, 10.8, 19.0, 13.5, 20.1, 10.3, 10.5, 11.5~
## $ production         <dbl> 17.1, 11.2, 23.1, 23.7, 19.9, 26.4, 23.7, 20.4, 24.~
## $ drive              <dbl> 87.5, 84.7, 83.8, 83.2, 84.9, 74.9, 84.5, 85.3, 85.~
## $ carpool            <dbl> 8.8, 8.8, 10.9, 13.5, 11.2, 14.9, 12.4, 9.4, 11.9, ~
## $ transit            <dbl> 0.1, 0.1, 0.4, 0.5, 0.4, 0.7, 0.0, 0.2, 0.2, 0.2, 0~
## $ walk               <dbl> 0.5, 1.0, 1.8, 0.6, 0.9, 5.0, 0.8, 1.2, 0.3, 0.6, 1~
## $ other_transp       <dbl> 1.3, 1.4, 1.5, 1.5, 0.4, 1.7, 0.6, 1.2, 0.4, 0.7, 1~
## $ work_at_home       <dbl> 1.8, 3.9, 1.6, 0.7, 2.3, 2.8, 1.7, 2.7, 2.1, 2.5, 1~
## $ mean_commute       <dbl> 26.5, 26.4, 24.1, 28.8, 34.9, 27.5, 24.6, 24.1, 25.~
## $ employed           <dbl> 23986, 85953, 8597, 8294, 22189, 3865, 7813, 47401,~
## $ private_work       <dbl> 73.6, 81.5, 71.8, 76.8, 82.0, 79.5, 77.4, 74.1, 85.~
## $ public_work        <dbl> 20.9, 12.3, 20.8, 16.1, 13.5, 15.1, 16.2, 20.8, 12.~
## $ self_employed      <dbl> 5.5, 5.8, 7.3, 6.7, 4.2, 5.4, 6.2, 5.0, 2.8, 7.9, 4~
## $ family_work        <dbl> 0.0, 0.4, 0.1, 0.4, 0.4, 0.0, 0.2, 0.1, 0.0, 0.5, 0~
## $ unemployment       <dbl> 7.6, 7.5, 17.6, 8.3, 7.7, 18.0, 10.9, 12.3, 8.9, 7.~
## $ land_area          <dbl> 594.44, 1589.78, 884.88, 622.58, 644.78, 622.81, 77~
```

```r
counties %>%
  # Select state, county, population, and industry-related columns
  select(state, county, population, professional, service, office, construction, production) %>%
  # Arrange service in descending order
  arrange(desc(service))
```

```
## # A tibble: 3,138 x 8
##    state   county population professional service office construction production
##    <chr>   <chr>       <dbl>        <dbl>   <dbl>  <dbl>        <dbl>      <dbl>
##  1 Missis~ Tunica      10477         23.9    36.6   21.5          3.5       14.5
##  2 Texas   Kinney       3577         30      36.5   11.6         20.5        1.3
##  3 Texas   Kenedy        565         24.9    34.1   20.5         20.5        0
##  4 New Yo~ Bronx     1428357         24.3    33.3   24.2          7.1       11
##  5 Texas   Brooks       7221         19.6    32.4   25.3         11.1       11.5
##  6 Colora~ Fremo~      46809         26.6    32.2   22.8         10.7        7.6
##  7 Texas   Culbe~       2296         20.1    32.2   24.2         15.7        7.8
##  8 Califo~ Del N~      27788         33.9    31.5   18.8          8.9        6.8
##  9 Minnes~ Mahno~       5496         26.8    31.5   18.7         13.1        9.9
```

```
## 10 Virgin~ Lanca~      11129         30.3     31.2   22.8           8.1         7.6
## # ... with 3,128 more rows
```

3.2 Select helpers

We learn that columns are not only selectable by their exact name, but also with helpers such as ends_with() and starts_with().

```
counties %>%
  # Select the state, county, population, and those ending with "work"
  select(state, county, population, ends_with("work")) %>%
  # Filter for counties that have at least 50% of people engaged in public work
  filter(public_work >= 50)
```

```
## # A tibble: 7 x 6
##   state      county                population private_work public_work family_work
##   <chr>      <chr>                      <dbl>        <dbl>       <dbl>       <dbl>
## 1 Alaska     Lake and Peninsula~         1474         42.2        51.6         0.2
## 2 Alaska     Yukon-Koyukuk Cens~         5644         33.3        61.7         0
## 3 California Lassen                     32645         42.6        50.5         0.1
## 4 Hawaii     Kalawao                       85         25          64.1         0
## 5 North Dak~ Sioux                       4380         32.9        56.8         0.1
## 6 South Dak~ Todd                        9942         34.4        55           0.8
## 7 Wisconsin  Menominee                   4451         36.8        59.1         0.4
```

3.3 Renaming a column after count

The function rename can be used to rename a column, for example the default column name n of count to num_counties.

```
# Rename the n column to num_counties
counties %>%
  count(state) %>%
  rename(num_counties = n)
```

```
## # A tibble: 50 x 2
##    state       num_counties
##    <chr>              <int>
##  1 Alabama               67
##  2 Alaska                28
##  3 Arizona               15
##  4 Arkansas              75
##  5 California            58
##  6 Colorado              64
##  7 Connecticut            8
##  8 Delaware               3
##  9 Florida               67
## 10 Georgia              159
## # ... with 40 more rows
```

3.4 Renaming a column as part of a select

Renaming can also be done with select, below we rename the poverty to poverty_rate.

```
# Select state, county, and poverty as poverty_rate
counties %>%
  select(state, county, poverty_rate = poverty)
```

```
## # A tibble: 3,138 x 3
##    state   county   poverty_rate
##    <chr>   <chr>           <dbl>
##  1 Alabama Autauga          12.9
##  2 Alabama Baldwin          13.4
##  3 Alabama Barbour          26.7
##  4 Alabama Bibb             16.8
##  5 Alabama Blount           16.7
##  6 Alabama Bullock          24.6
##  7 Alabama Butler           25.4
##  8 Alabama Calhoun          20.5
##  9 Alabama Chambers         21.6
## 10 Alabama Cherokee         19.2
## # ... with 3,128 more rows
```

3.5 Using transmute

The verb transmute can be viewed as a combination of the select and mutate functionality. For an example view the code below.

```
counties %>%
  # Keep the state, county, and populations columns, and add a density column
  transmute(state, county, population, density = population/land_area) %>%
  # Filter for counties with a population greater than one million
  filter(population > 1000000) %>%
  # Sort density in ascending order
  arrange(density)
```

```
## # A tibble: 41 x 4
##    state      county         population density
##    <chr>      <chr>               <dbl>   <dbl>
##  1 California San Bernardino    2094769    104.
##  2 Nevada     Clark             2035572    258.
##  3 California Riverside         2298032    319.
##  4 Arizona    Maricopa          4018143    437.
##  5 Florida    Palm Beach        1378806    700.
##  6 California San Diego         3223096    766.
##  7 Washington King              2045756    967.
##  8 Texas      Travis            1121645   1133.
##  9 Florida    Hillsborough      1302884   1277.
## 10 Florida    Orange            1229039   1360.
## # ... with 31 more rows
```

3.6 Choosing among the four verbs

There are different use cases for rename, select, mutate and transmute: - Rename and mutate leave the columns that you don't mention, select and transmute don't. - Rename and select don't allow calculation, mutate and transmute do.

```
# Change the name of the unemployment column
counties %>%
  rename(unemployment_rate = unemployment)
```

```
## # A tibble: 3,138 x 40
```

```
##    census_id state   county  region metro  population   men women hispanic white
##    <chr>     <chr>   <chr>   <chr>  <chr>       <dbl> <dbl> <dbl>    <dbl> <dbl>
##  1 1001      Alabama Autauga South  Metro       55221 26745 28476      2.6  75.8
##  2 1003      Alabama Baldwin South  Metro      195121 95314 99807      4.5  83.1
##  3 1005      Alabama Barbour South  Nonme~      26932 14497 12435      4.6  46.2
##  4 1007      Alabama Bibb    South  Metro       22604 12073 10531      2.2  74.5
##  5 1009      Alabama Blount  South  Metro       57710 28512 29198      8.6  87.9
##  6 1011      Alabama Bullock South  Nonme~      10678  5660  5018      4.4  22.2
##  7 1013      Alabama Butler  South  Nonme~      20354  9502 10852      1.2  53.3
##  8 1015      Alabama Calhoun South  Metro      116648 56274 60374      3.5  73
##  9 1017      Alabama Chambe~ South  Nonme~      34079 16258 17821      0.4  57.3
## 10 1019      Alabama Cherok~ South  Nonme~      26008 12975 13033      1.5  91.7
## # ... with 3,128 more rows, and 30 more variables: black <dbl>, native <dbl>,
## #   asian <dbl>, pacific <dbl>, citizens <dbl>, income <dbl>, income_err <dbl>,
## #   income_per_cap <dbl>, income_per_cap_err <dbl>, poverty <dbl>,
## #   child_poverty <dbl>, professional <dbl>, service <dbl>, office <dbl>,
## #   construction <dbl>, production <dbl>, drive <dbl>, carpool <dbl>,
## #   transit <dbl>, walk <dbl>, other_transp <dbl>, work_at_home <dbl>,
## #   mean_commute <dbl>, employed <dbl>, private_work <dbl>, public_work <dbl>,
## #   self_employed <dbl>, family_work <dbl>, unemployment_rate <dbl>,
## #   land_area <dbl>
```

```r
# Keep the state and county columns, and the columns containing poverty
counties %>%
  select(state, county, contains("poverty"))
```

```
## # A tibble: 3,138 x 4
##    state   county   poverty child_poverty
##    <chr>   <chr>      <dbl>         <dbl>
##  1 Alabama Autauga     12.9          18.6
##  2 Alabama Baldwin     13.4          19.2
##  3 Alabama Barbour     26.7          45.3
##  4 Alabama Bibb        16.8          27.9
##  5 Alabama Blount      16.7          27.2
##  6 Alabama Bullock     24.6          38.4
##  7 Alabama Butler      25.4          39.2
##  8 Alabama Calhoun     20.5          31.6
##  9 Alabama Chambers    21.6          37.2
## 10 Alabama Cherokee    19.2          30.1
## # ... with 3,128 more rows
```

```r
# Calculate the fraction_women column without dropping the other columns
counties %>%
  mutate(fraction_women = women / population)
```

```
## # A tibble: 3,138 x 41
##    census_id state   county  region metro  population   men women hispanic white
##    <chr>     <chr>   <chr>   <chr>  <chr>       <dbl> <dbl> <dbl>    <dbl> <dbl>
##  1 1001      Alabama Autauga South  Metro       55221 26745 28476      2.6  75.8
##  2 1003      Alabama Baldwin South  Metro      195121 95314 99807      4.5  83.1
##  3 1005      Alabama Barbour South  Nonme~      26932 14497 12435      4.6  46.2
##  4 1007      Alabama Bibb    South  Metro       22604 12073 10531      2.2  74.5
##  5 1009      Alabama Blount  South  Metro       57710 28512 29198      8.6  87.9
##  6 1011      Alabama Bullock South  Nonme~      10678  5660  5018      4.4  22.2
##  7 1013      Alabama Butler  South  Nonme~      20354  9502 10852      1.2  53.3
##  8 1015      Alabama Calhoun South  Metro      116648 56274 60374      3.5  73
```

```
##  9 1017      Alabama Chambe~ South  Nonme~      34079 16258 17821      0.4  57.3
## 10 1019      Alabama Cherok~ South  Nonme~      26008 12975 13033      1.5  91.7
## # ... with 3,128 more rows, and 31 more variables: black <dbl>, native <dbl>,
## #   asian <dbl>, pacific <dbl>, citizens <dbl>, income <dbl>, income_err <dbl>,
## #   income_per_cap <dbl>, income_per_cap_err <dbl>, poverty <dbl>,
## #   child_poverty <dbl>, professional <dbl>, service <dbl>, office <dbl>,
## #   construction <dbl>, production <dbl>, drive <dbl>, carpool <dbl>,
## #   transit <dbl>, walk <dbl>, other_transp <dbl>, work_at_home <dbl>,
## #   mean_commute <dbl>, employed <dbl>, private_work <dbl>, public_work <dbl>,
## #   self_employed <dbl>, family_work <dbl>, unemployment <dbl>,
## #   land_area <dbl>, fraction_women <dbl>
```

```r
# Keep only the state, county, and employment_rate columns
counties %>%
  transmute(state, county, employment_rate = employed / population)
```

```
## # A tibble: 3,138 x 3
##    state   county   employment_rate
##    <chr>   <chr>              <dbl>
##  1 Alabama Autauga            0.434
##  2 Alabama Baldwin            0.441
##  3 Alabama Barbour            0.319
##  4 Alabama Bibb               0.367
##  5 Alabama Blount             0.384
##  6 Alabama Bullock            0.362
##  7 Alabama Butler             0.384
##  8 Alabama Calhoun            0.406
##  9 Alabama Chambers           0.402
## 10 Alabama Cherokee           0.390
## # ... with 3,128 more rows
```

4 Case Study: The babynames Dataset

4.1 Filtering and arranging for one year

We use filter and arrange to select rows with year = 1990 and sort them by the number column. This results in the most common names for each year.

```r
babynames %>%
  # Filter for the year 1990
  filter(year == 1990) %>%
  # Sort the number column in descending order
  arrange(desc(number))
```

```
## # A tibble: 21,223 x 3
##     year name        number
##    <dbl> <chr>        <int>
##  1  1990 Michael      65560
##  2  1990 Christopher  52520
##  3  1990 Jessica      46615
##  4  1990 Ashley       45797
##  5  1990 Matthew      44925
##  6  1990 Joshua       43382
##  7  1990 Brittany     36650
##  8  1990 Amanda       34504
```

```
## 9  1990 Daniel       33963
## 10  1990 David        33862
## # ... with 21,213 more rows
```

4.2 Using top_n with babynames

The code below results in the most popular name for each year.

```
# Find the most common name in each year
babynames %>%
    group_by(year) %>%
    top_n(1, number)
```

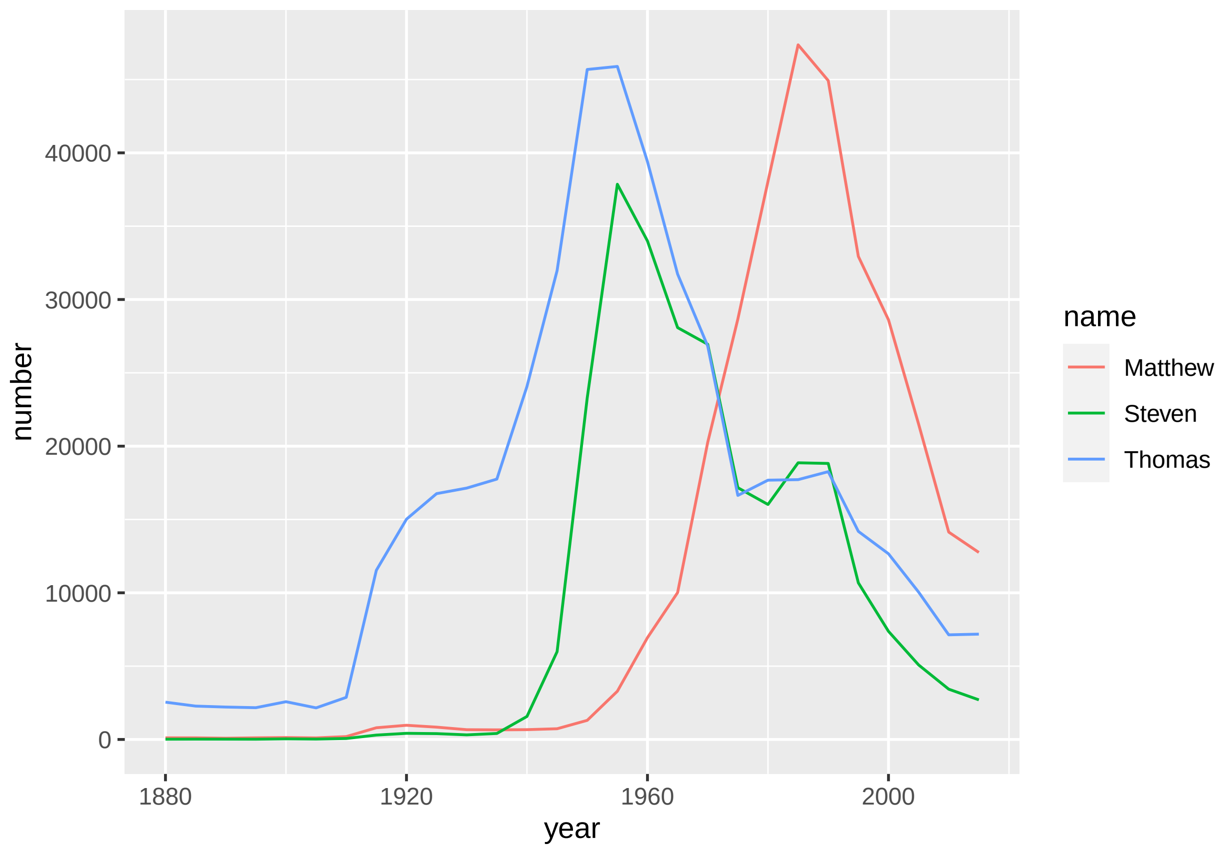
```
## # A tibble: 28 x 3
## # Groups:   year [28]
##     year name  number
##    <dbl> <chr>  <int>
##  1  1880 John    9701
##  2  1885 Mary    9166
##  3  1890 Mary   12113
##  4  1895 Mary   13493
##  5  1900 Mary   16781
##  6  1905 Mary   16135
##  7  1910 Mary   22947
##  8  1915 Mary   58346
##  9  1920 Mary   71175
## 10  1925 Mary   70857
## # ... with 18 more rows
```

4.3 Visualizing names with ggplot2

The %in% keyword checks if a value is found in a vector. Then we plot number over year for the three names selected.

```
# Filter for the names Steven, Thomas, and Matthew
selected_names <- babynames %>%
  filter(name %in% c("Steven", "Thomas", "Matthew"))

# Plot the names using a different color for each name
ggplot(selected_names, aes(x = year, y = number, color = name)) +
  geom_line()
```



4.4 Finding the year each name is most common

We combine everything we've learned to find the most common baby name for each year.

```
# Calculate the fraction of people born each year with the same name
babynames %>%
  group_by(year) %>%
  mutate(year_total = sum(number)) %>%
  ungroup() %>%
  mutate(fraction = number / year_total) %>%
# Find the year each name is most common
  group_by(name) %>%
  top_n(1, fraction)
```

```
## # A tibble: 48,040 x 5
## # Groups:   name [48,040]
##     year name      number year_total  fraction
##    <dbl> <chr>      <int>      <int>     <dbl>
##  1  1880 Abbott         5     201478 0.0000248
##  2  1880 Abe           50     201478 0.000248
##  3  1880 Abner         27     201478 0.000134
##  4  1880 Adelbert      28     201478 0.000139
##  5  1880 Adella        26     201478 0.000129
##  6  1880 Adolf          6     201478 0.0000298
##  7  1880 Adolph        93     201478 0.000462
##  8  1880 Agustus        5     201478 0.0000248
##  9  1880 Albert      1493     201478 0.00741
```

```
## 10  1880 Albertina      7     201478 0.0000347
## # ... with 48,030 more rows
```

4.5 Adding the total and maximum for each name

In this exercise we calculate the total occurrences of one name as name_total and also the max occurrence of one name name_max. Then we add a column fraction_max with mutate.

```
babynames %>%
 group_by(name) %>%
 mutate(name_total = sum(number),
        name_max = max(number)) %>%
 # Ungroup the table
 ungroup()  %>%
 # Add the fraction_max column containing the number by the name maximum
 mutate(fraction_max = number/name_max)
```

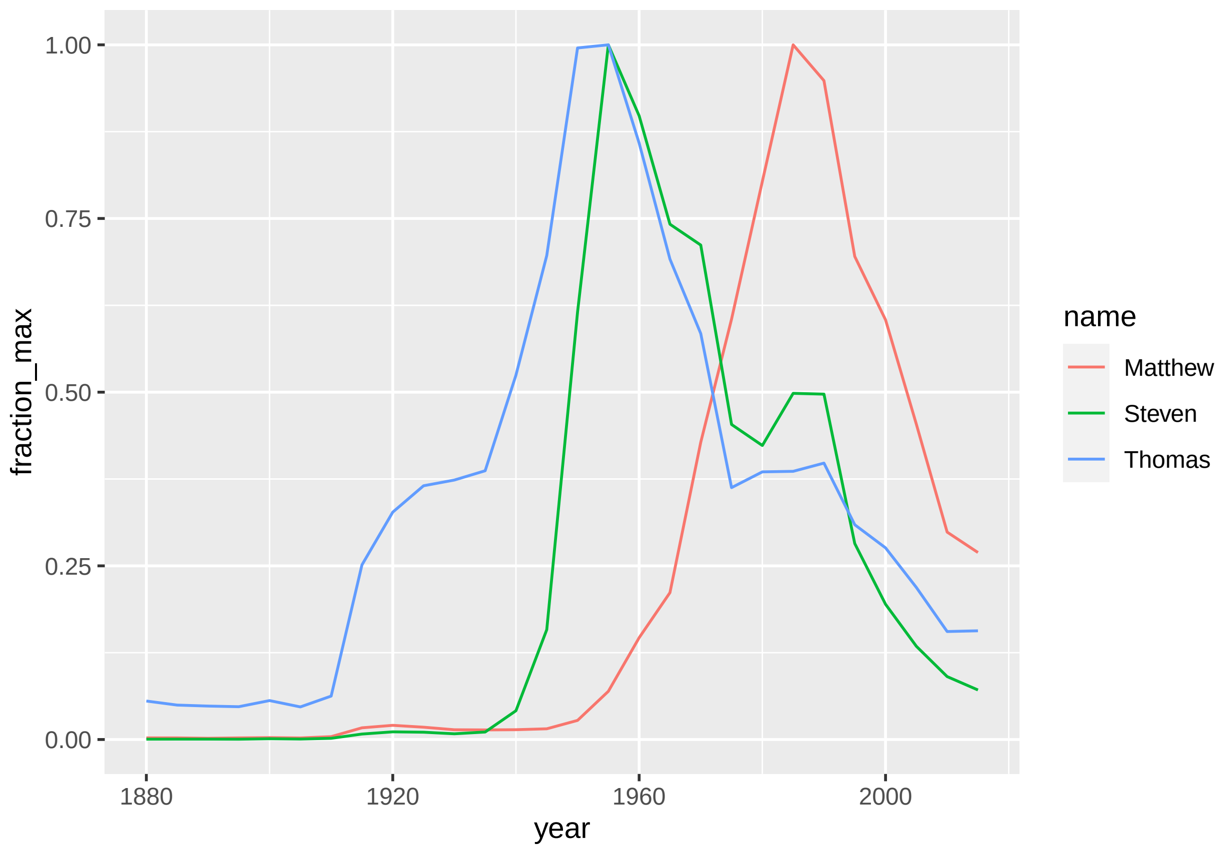
```
## # A tibble: 332,595 x 6
##     year name    number name_total name_max fraction_max
##    <dbl> <chr>    <int>      <int>    <int>        <dbl>
##  1  1880 Aaron      102     114739    14635     0.00697
##  2  1880 Ab           5         77       31     0.161
##  3  1880 Abbie       71       4330      445     0.160
##  4  1880 Abbott       5        217       51     0.0980
##  5  1880 Abby         6      11272     1753     0.00342
##  6  1880 Abe         50       1832      271     0.185
##  7  1880 Abel         9      10565     3245     0.00277
##  8  1880 Abigail     12      72600    15762     0.000761
##  9  1880 Abner       27       1552      199     0.136
## 10  1880 Abraham     81      17882     2449     0.0331
## # ... with 332,585 more rows
```

4.6 Visualizing the normalized change in popularity

In this exercise we filter for three names and then plot the fraction_max column of the previous exercise with ggplot.

```
# Filter for the names Steven, Thomas, and Matthew
names_filtered <- names_normalized %>%
    filter(name %in% c("Steven", "Thomas", "Matthew"))

# Visualize these names over time
ggplot(names_filtered, aes(x=year,y=fraction_max,color=name)) + geom_line()
```



4.7 Using ratios to describe the frequency of a name

In this exercise we learn that the function lag can be used to find the previous value in vector/data frame.

```
babynames_fraction %>%
  # Arrange the data in order of name, then year
  arrange(name, year) %>%
  # Group the data by name
  group_by(name)  %>%
  # Add a ratio column that contains the ratio between each year
  mutate(ratio = fraction / lag(fraction))
```

```
## # A tibble: 332,595 x 6
## # Groups:   name [48,040]
##     year name    number year_total   fraction  ratio
##    <dbl> <chr>    <int>      <int>      <dbl>  <dbl>
##  1  2010 Aaban        9    3672066 0.00000245 NA
##  2  2015 Aaban       15    3648781 0.00000411  1.68
##  3  1995 Aadam        6    3652750 0.00000164 NA
##  4  2000 Aadam        6    3767293 0.00000159  0.970
##  5  2005 Aadam        6    3828460 0.00000157  0.984
##  6  2010 Aadam        7    3672066 0.00000191  1.22
##  7  2015 Aadam       22    3648781 0.00000603  3.16
##  8  2010 Aadan       11    3672066 0.00000300 NA
##  9  2015 Aadan       10    3648781 0.00000274  0.915
## 10  2000 Aadarsh      5    3767293 0.00000133 NA
## # ... with 332,585 more rows
```

4.8 Biggest jumps in a name

In this exercise we evaluate which names had the biggest jumps in popularity in consecutive years.

```
babynames_ratios_filtered %>%
  # Extract the largest ratio from each name
  top_n(1, ratio) %>%
  # Sort the ratio column in descending order
  arrange(desc(ratio))  %>%
  # Filter for fractions greater than or equal to 0.001
  filter(fraction >= 0.001)
```

```
## # A tibble: 291 x 6
## # Groups:   name [291]
##     year name    number year_total fraction ratio
##    <dbl> <chr>    <int>      <int>    <dbl> <dbl>
##  1  1960 Tammy    14365    4152075  0.00346  70.1
##  2  2005 Nevaeh    4610    3828460  0.00120  45.8
##  3  1940 Brenda    5460    2301630  0.00237  37.5
##  4  1885 Grover     774     240822  0.00321  36.0
##  5  1945 Cheryl    8170    2652029  0.00308  24.9
##  6  1955 Lori      4980    4012691  0.00124  23.2
##  7  2010 Khloe     5411    3672066  0.00147  23.2
##  8  1950 Debra     6189    3502592  0.00177  22.6
##  9  2010 Bentley   4001    3672066  0.00109  22.4
## 10  1935 Marlene   4840    2088487  0.00232  16.8
## # ... with 281 more rows
```