Session 08

%>% Operator

`tidyverse` packages often make use of the pipe operator `%>%` (though it's not required). The main advantage of `the pipeline` is the ability to string multiple functions together by incorporating `%>%`.

In other words, this operator forwards a value/result into the next function call/expression.

For example, a function in `R` such as

`f(x)` can be rewritten as `x %>% f`

This means that functions that take one argument, function(argument), can be rewritten as follows:

`argument %>% function()`.

For instance, a function to filter data can be written as:

`filter(data, variable == numeric_value)` or `data %>% filter(variable == numeric_value)`

The benefit of using `%>%` is not evident here as both functions complete the same task. However, when you perform multiple functions its advantage becomes obvious. We'll see more examples when we start using `tidyr` and `dplyr` packages more.

`tidyr` and `dplyr` packages

First let's install and call up the `tidyr` package and the `dplyr` package.

Using `tidyr`

Let's create a data frame for three participants as column 1, and columns a and b (test version) with some score numbers:

```
##   participant  a  b
## 1         P01 10 50
## 2         P02  8 42
## 3         P03  6 36
```

Pivoting to longer `pivot_longer()` takes four principal arguments:

- the data
- the `names_to` column variable we wish to create from column names
- the `values_to` column variable we wish to create and fill with values
- `cols` are the name of the columns we use to make this pivot (or to drop).

We have three variables (`participant`, `test version` and `result/score`), but only `participant` is currently a proper column. We use `pivot_longer()` to put the `a` and `b` columns into our value pairs of `version` and `score`:

```
## # A tibble: 6 x 3
##   participant version score
##   <chr>       <chr>   <dbl>
## 1 P01         a          10
## 2 P01         b          50
```

```
## 3 P02         a           8
## 4 P02         b          42
## 5 P03         a           6
## 6 P03         b          36
```

Here we have only two columns which we listed individually. Usually the columns to pivot are specified with `dplyr::select()` style notation. `version` and `score` do not exist in our table so we put their names in quotes. In the final tibble the pivoted columns are dropped, and we get new `version` and `score` columns. Otherwise, the relationships between the original variables are preserved.

Pivoting to wider `pivot_wider()` takes three principal arguments:

- the data
- the `names_from` column variable whose values will become new column names
- the `values_from` column variable whose values will fill the new column variables

Let's look at the example from `tidyverse` dataset `us_rent_income`:

```
## # A tibble: 6 x 5
##   GEOID NAME    variable estimate   moe
##   <chr> <chr>   <chr>       <dbl> <dbl>
## 1 01    Alabama income      24476   136
## 2 01    Alabama rent          747     3
## 3 02    Alaska  income      32940   508
## 4 02    Alaska  rent         1200    13
## 5 04    Arizona income      27517   148
## 6 04    Arizona rent          972     4
```

What information can you get from looking at the dataset?

Imagine we want to see the `estimates` for `income` and `rent` as separate columns and `moe` for `income` and `rent` as separate columns. This will be the *wide format*.

```
## # A tibble: 52 x 6
##    GEOID NAME                 estimate_income estimate_rent moe_income moe_rent
##    <chr> <chr>                          <dbl>         <dbl>      <dbl>    <dbl>
##  1 01    Alabama                        24476           747        136        3
##  2 02    Alaska                         32940          1200        508       13
##  3 04    Arizona                        27517           972        148        4
##  4 05    Arkansas                       23789           709        165        5
##  5 06    California                     29454          1358        109        3
##  6 08    Colorado                       32401          1125        109        5
##  7 09    Connecticut                    35326          1123        195        5
##  8 10    Delaware                       31560          1076        247       10
##  9 11    District of Columbia           43198          1424        681       17
## 10 12    Florida                        25952          1077         70        3
## # ... with 42 more rows
```

Separating and Uniting

`separate()` function Imagine we have some measurements of how much time people spend on performing a task on an app (researching UX design), measured at two locations (work and home), in the morning and in the evening. Each participant has been randomly assigned to either test or control group. Let's create a mock dataframe for 4 participants.

```
##   id   group work.morning home.morning work.evening home.evening
## 1  1    test    0.2061312   0.19711734    0.6862566   0.00396822
```

```
## 2  2    test    0.5651030   0.31752099    0.3314321   0.97072354
## 3  3 control    0.7691917   0.01065623    0.4325805   0.57274324
## 4  4 control    0.4518332   0.18889723    0.6215998   0.68727385
```

To tidy this dataset, we first use `pivot_longer()` to turn columns `work.morning`, `home.morning`, `work.evening` and `home.evening` into a key value pair `place-time`:

```
## # A tibble: 8 x 4
##      id group place           time
##   <int> <chr> <chr>          <dbl>
## 1     1 test  work.morning 0.206
## 2     1 test  home.morning 0.197
## 3     1 test  work.evening 0.686
## 4     1 test  home.evening 0.00397
## 5     2 test  work.morning 0.565
## 6     2 test  home.morning 0.318
## 7     2 test  work.evening 0.331
## 8     2 test  home.evening 0.971
```

We see that two variables are joined together in one column. `separate()` allows us to tease them apart. So we use `separate()` to split the `place` into `location` and `daytime`, using a regular expression to describe the character that separates them.

```
## # A tibble: 8 x 5
##      id group location daytime    time
##   <int> <chr> <chr>    <chr>     <dbl>
## 1     1 test  work     morning 0.206
## 2     1 test  home     morning 0.197
## 3     1 test  work     evening 0.686
## 4     1 test  home     evening 0.00397
## 5     2 test  work     morning 0.565
## 6     2 test  home     morning 0.318
## 7     2 test  work     evening 0.331
## 8     2 test  home     evening 0.971
```

`unite( ) function` If we want to merge two variables into one and combine the values of two variables, the `unite()` function can paste together multiple variable values into one. In essence, it combines two variables of a single observation into one variable.

Let's combine `location` and `daytime` back to one variable. Using the `tidy2` dataframe we created above, we can re-unite the `location` and `daytime` variables we created and re-create the original `place` variable we had in the `messy2` dataframe.

```
## # A tibble: 8 x 4
##      id group place           time
##   <int> <chr> <chr>          <dbl>
## 1     1 test  work_morning 0.206
## 2     1 test  home_morning 0.197
## 3     1 test  work_evening 0.686
## 4     1 test  home_evening 0.00397
## 5     2 test  work_morning 0.565
## 6     2 test  home_morning 0.318
## 7     2 test  work_evening 0.331
## 8     2 test  home_evening 0.971
```

Using `dplyr`

Let's download the `gapminder` dataset and explore it with the `head()` function:

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

What variables are included? List them below:

- `country`
- `continent`
- `year`
- `lifeExp` for life expectancy
- `pop` for population
- `gdpperCap` for Gross Domestic Product (GDP) per capita

How many observations (rows) are in the dataset?

You can also type `View(gapminder)` in the console to see the full dataset in a new window.

Include some `R` code below to explore the dataset.

- What is the type of each variable?
- Are there any categorical variables? How many levels do they have?
- What is the range of years?
- Is there data for every year over this period?
- Compute some basic descriptive statistics for the dataset. What is the average life expectancy across time and across all countries?

`dplyr` functions

```
## Help on topic 'filter' was found in the following packages:
##
##   Package               Library
##   dplyr                 /Library/Frameworks/R.framework/Versions/4.2/Resources/library
##   stats                 /Library/Frameworks/R.framework/Versions/4.2/Resources/library
##
##
## Using the first match ...
```

`filter()` rows See if you can use the logical operators to manipulate the code below to show:

- The data for Canada

```
## # A tibble: 6 x 6
##   country continent  year lifeExp      pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Canada  Americas   1952    68.8 14785584    11367.
## 2 Canada  Americas   1957    70.0 17010154    12490.
## 3 Canada  Americas   1962    71.3 18985849    13462.
```

```
## 4 Canada  Americas   1967    72.1 20819767    16077.
## 5 Canada  Americas   1972    72.9 22284500    18971.
## 6 Canada  Americas   1977    74.2 23796400    22091.
```

- All data for countries in Oceania

```
## # A tibble: 6 x 6
##   country   continent  year lifeExp      pop gdpPercap
##   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Australia Oceania    1952    69.1  8691212    10040.
## 2 Australia Oceania    1957    70.3  9712569    10950.
## 3 Australia Oceania    1962    70.9 10794968    12217.
## 4 Australia Oceania    1967    71.1 11872264    14526.
## 5 Australia Oceania    1972    71.9 13177000    16789.
## 6 Australia Oceania    1977    73.5 14074100    18334.
```

- Rows where the life expectancy is greater than 82

```
## # A tibble: 2 x 6
##   country          continent  year lifeExp       pop gdpPercap
##   <fct>            <fct>     <int>   <dbl>     <int>     <dbl>
## 1 Hong Kong, China Asia       2007    82.2   6980412    39725.
## 2 Japan            Asia       2007    82.6 127467972    31656.
```

You can also use Boolean operators to return only the rows that contain:

- United States before 1980

```
## # A tibble: 6 x 6
##   country       continent  year lifeExp       pop gdpPercap
##   <fct>         <fct>     <int>   <dbl>     <int>     <dbl>
## 1 United States Americas   1952    68.4 157553000    13990.
## 2 United States Americas   1957    69.5 171984000    14847.
## 3 United States Americas   1962    70.2 186538000    16173.
## 4 United States Americas   1967    70.8 198712000    19530.
## 5 United States Americas   1972    71.3 209896000    21806.
## 6 United States Americas   1977    73.4 220239000    24073.
```

- Countries where life expectancy in 2007 is below 50 or over 75

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       2007    43.8 31889923      975.
## 2 Albania     Europe     2007    76.4  3600523     5937.
## 3 Angola      Africa     2007    42.7 12420476     4797.
## 4 Argentina   Americas   2007    75.3 40301927    12779.
## 5 Australia   Oceania    2007    81.2 20434176    34435.
## 6 Austria     Europe     2007    79.8  8199783    36126.
```

Sorting with `arrange()` Find the records with the smallest population.

```
## # A tibble: 6 x 6
##   country               continent  year lifeExp   pop gdpPercap
##   <fct>                 <fct>     <int>   <dbl> <int>     <dbl>
## 1 Sao Tome and Principe Africa     1952    46.5 60011      880.
## 2 Sao Tome and Principe Africa     1957    48.9 61325      861.
## 3 Djibouti              Africa     1952    34.8 63149     2670.
## 4 Sao Tome and Principe Africa     1962    51.9 65345     1072.
```

```
## 5 Sao Tome and Principe Africa     1967    54.4 70787     1385.
## 6 Djibouti              Africa     1957    37.3 71851     2865.
```

mutate You can create new variables with `mutate()`.

- Create a new variable with total GDP

```
## # A tibble: 6 x 7
##   country     continent  year lifeExp      pop gdpPercap     totalGDP
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>        <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.  6567086330.
## 2 Afghanistan Asia       1957    30.3  9240934      821.  7585448670.
## 3 Afghanistan Asia       1962    32.0 10267083      853.  8758855797.
## 4 Afghanistan Asia       1967    34.0 11537966      836.  9648014150.
## 5 Afghanistan Asia       1972    36.1 13079460      740.  9678553274.
## 6 Afghanistan Asia       1977    38.4 14880372      786. 11697659231.
```

group_by Create a summary table with the `population` and `GDP` by `continent` for the year `1952`:

```
## # A tibble: 5 x 3
##   continent        pop totalGDP
##   <fct>          <int>    <dbl>
## 1 Africa     237640501  3.12e11
## 2 Americas   345152446  2.94e12
## 3 Asia      1395357351  1.13e12
## 4 Europe     418120846  2.55e12
## 5 Oceania     10686006  1.08e11
```

summarise Use `summarise()` to compute three statistics about the data:

- The first (minimum) year in the dataset
- The last (maximum) year in the dataset
- The number of unique countries

```
## # A tibble: 1 x 3
##   year_min year_max n_countries
##      <int>    <int>       <int>
## 1     1952     2007         142
```