

# Lab #1 - Gapminder Dataset

*Econ 224*

*August 28th, 2018*

## Installing Required Packages

Welcome to the first lab of Econ 224! Today we'll be giving you a crash course in two R packages that we'll be using throughout the semester: `dplyr` and `ggplot2`. Before we can get started, you'll need to install both of these packages. A quick way to install both of them at once, along with several other packages that may come in handy later, is `install.packages('tidyverse')`. Note that you only need to do this *once*. The dataset we'll work with today is also available as an R package called `gapminder`. Make sure that you have both `tidyverse` and `gapminder` installed before continuing.

## The Gapminder Dataset

Our next step is to load both `tidyverse`, which contains `dplyr` and `ggplot2`, and `gapminder`, which contains the data we'll be analyzing today:

```
library(tidyverse)
library(gapminder)
```

## Exercise #1

Now that you've loaded `gapminder`, use the command `?gapminder` to view the R help file for this dataset and read the documentation you find there and answer the following questions:

- What information does this dataset contain?
- How many rows and columns does it have?
- What are the names of each of the columns, and what information does each contain?
- What is the source of the dataset?

## Solution to Exercise # 1

*Write your answer here.*

## What is a tibble?

Let's see what happens if we display the `gapminder` dataset:

```
gapminder
```

```
## # A tibble: 1,704 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.
## 7 Afghanistan Asia      1982   39.9 12881816    978.
## 8 Afghanistan Asia      1987   40.8 13867957    852.
## 9 Afghanistan Asia      1992   41.7 16317921    649.
## 10 Afghanistan Asia      1997   41.8 22227415    635.
## # ... with 1,694 more rows
```

If you're used to working with dataframes in R, this may surprise you. Rather than filling up the screen with lots of useless information, R shows us a helpful summary of the information contained in **gapminder**. This is because **gapminder** is *not* a dataframe; it's a *tibble*, often abbreviated *tbl*. For the moment, all you need to know about tibbles is that they are souped up versions of R dataframes that are designed to work seamlessly with **dplyr**. (If you want to learn more, see the chapter entitled "Tibbles" in *R for Data Science*) But what exactly is **dplyr**?

## What is dplyr?

The **dplyr** package provides a number of powerful but easy-to-use tools for data manipulation in R. A good reference is the chapter entitled "Data Transformation" in *R for Data Science*. We'll be making heavy use of **dplyr** throughout the semester. Rather than trying to explain everything in advance, let's just dive right in.

## Filter Rows with filter

Let's run the following command in R and see what happens:

```
gapminder %>% filter(year == 2007)
```

```
## # A tibble: 142 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 Algeria     Africa    2007   72.3  33333216   6223.
## 4 Angola      Africa    2007   42.7  12420476   4797.
## 5 Argentina   Americas  2007   75.3  40301927  12779.
## 6 Australia   Oceania   2007   81.2  20434176  34435.
## 7 Austria     Europe    2007   79.8   8199783   36126.
## 8 Bahrain     Asia      2007   75.6   708573    29796.
## 9 Bangladesh  Asia      2007   64.1 150448339   1391.
## 10 Belgium    Europe    2007   79.4  10392226   33693.
## # ... with 132 more rows
```

Compare the results of running this command to what we got when we typed `gapminder` into the console above. Rather than displaying the whole dataset, now R is only showing us the 142 rows for which the column `year` has a value of 2007.

So how does this work? The `%>%` symbol is called a *pipe*. Pipes play very nicely with `dplyr` and make our code very easy to understand. The tibble `gapminder` is being piped into the function `filter()`. The argument `year == 2007` tells `filter()` that it should find all the rows such that the logical condition `year == 2007` is TRUE.

Oh no! Have we accidentally deleted all of the other rows of `gapminder`? Nope: we haven't made any changes to `gapminder` at all. If you don't believe me try entering `gapminder` at the console. All that this command does is *display* a subset of `gapminder`. If we wanted to store the result of running this command, we'd need to assign it to a variable, for example

```
gapminder2007 <- gapminder %>% filter(year == 2007)
gapminder2007
```

```
## # A tibble: 142 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 Algeria    Africa    2007  72.3  33333216   6223.
## 4 Angola     Africa    2007  42.7  12420476   4797.
## 5 Argentina  Americas  2007  75.3  40301927  12779.
## 6 Australia  Oceania   2007  81.2  20434176  34435.
## 7 Austria    Europe    2007  79.8   8199783   36126.
## 8 Bahrain    Asia      2007  75.6    708573   29796.
## 9 Bangladesh Asia      2007  64.1 150448339   1391.
##10 Belgium    Europe    2007  79.4  10392226  33693.
## # ... with 132 more rows
```

## Exercise #2

1. Explain the difference between `x = 3` and `x == 3` in R.
2. Use `filter` to choose the subset of `gapminder` for which `year` is 2002.
3. If you instead try to choose the subset with `year` equal to 2005, something will go wrong. Try it and explain what happens and why.
4. Store the data for Asian countries in a tibble called `gapminder_asia`. Display this tibble.

## Solution to Exercise #2

*Write your answer and code here*

1. The first assigns the value 3 to the variable `x`; the second tests whether `x` is equal to 3 and returns either TRUE or FALSE.
2. Use the following code:

```
gapminder %>% filter(year == 2002)
```

```
## # A tibble: 142 x 6
##   country    continent  year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int> <dbl>    <int>    <dbl>
## 1 Afghanistan Asia      2002  42.1  25268405    727.
## 2 Albania    Europe    2002  75.7   3508512   4604.
## 3 Algeria    Africa    2002  71.0  31287142   5288.
## 4 Angola     Africa    2002  41.0  10866106   2773.
## 5 Argentina  Americas  2002  74.3  38331121   8798.
## 6 Australia  Oceania   2002  80.4  19546792  30688.
## 7 Austria    Europe    2002  79.0   8148312  32418.
## 8 Bahrain    Asia      2002  74.8    656397  23404.
## 9 Bangladesh Asia      2002  62.0 135656790   1136.
## 10 Belgium   Europe    2002  78.3  10311970  30486.
## # ... with 132 more rows
```

3. If you go back to the help file for `gapminder` you'll see that it only contains data for every fifth year. The year 2005 isn't in our dataset so `dplyr` will display an empty tibble:

```
gapminder %>% filter(year == 2005)
```

```
## # A tibble: 0 x 6
## # ... with 6 variables: country <fct>, continent <fct>, year <int>,
## #   lifeExp <dbl>, pop <int>, gdpPercap <dbl>
```

4. Use the following code:

```
gapminder_asia <- gapminder %>% filter(continent == 'Asia')
gapminder_asia
```

```
## # A tibble: 396 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.
## 7 Afghanistan Asia      1982  39.9 12881816    978.
## 8 Afghanistan Asia      1987  40.8 13867957    852.
## 9 Afghanistan Asia      1992  41.7 16317921    649.
## 10 Afghanistan Asia      1997  41.8 22227415    635.
## # ... with 386 more rows
```

## Filtering two variables

We can use `filter` to subset on two or more variables. For example, here we display data for the US in 2007:

```
gapminder %>% filter(year == 2007, country == 'United States')
```

```
## # A tibble: 1 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>
## 1 United States Americas   2007   78.2 301139947  42952.
```

## Exercise #3

1. When I displayed data for the US in 2007, I put quotes around `United States` but not around `year`. Explain why.
2. Which country had the higher life expectancy in 1977: Ireland or Brazil? Which had the higher GDP per capita?

## Solution to Exercise #3

*Write your answer and code here*

1. This is because `year` contains numeric data while `country` contains character data, aka string data.
2. From the results of the following code, we see that Ireland had both a higher life expectancy and GDP per capita.

```
gapminder %>% filter(year == 1977, country == 'Ireland')
```

```
## # A tibble: 1 x 6
##   country continent year lifeExp      pop gdpPercap
##   <fct>    <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Ireland Europe    1977   72.0 3271900  11151.
```

```
gapminder %>% filter(year == 1977, country == 'Brazil')
```

```
## # A tibble: 1 x 6
##   country continent year lifeExp      pop gdpPercap
##   <fct>    <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Brazil  Americas   1977   61.5 114313951  6660.
```

## Sort data with arrange

Suppose we wanted to sort `gapminder` by `gdpPercap`. To do this we can use the `arrange` command along with the pipe `%>%` as follows:

```
gapminder %>% arrange(gdpPercap)
```

```
## # A tibble: 1,704 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>
## 1 Congo, Dem. Rep. Africa    2002   45.0 55379852    241.
## 2 Congo, Dem. Rep. Africa    2007   46.5 64606759    278.
## 3 Lesotho      Africa    1952   42.1  748747     299.
```

```
## 4 Guinea-Bissau Africa 1952 32.5 580653 300.
## 5 Congo, Dem. Rep. Africa 1997 42.6 47798986 312.
## 6 Eritrea Africa 1952 35.9 1438760 329.
## 7 Myanmar Asia 1952 36.3 20092996 331
## 8 Lesotho Africa 1957 45.0 813338 336.
## 9 Burundi Africa 1952 39.0 2445618 339.
## 10 Eritrea Africa 1957 38.0 1542611 344.
## # ... with 1,694 more rows
```

The logic is very similar to what we saw above for `filter`. Here, we pipe the tibble `gapminder` into the function `arrange()`. The argument `gdpPercap` tells `arrange()` that we want to sort by GDP per capita. Note that by default `arrange()` sorts in *ascending order*. If we want to sort in *descending order*, we use the function `desc()` as follows:

```
gapminder %>% arrange(desc(gdpPercap))
```

```
## # A tibble: 1,704 x 6
##   country    continent year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>   <dbl>   <int>    <dbl>
## 1 Kuwait     Asia      1957    58.0  212846  113523.
## 2 Kuwait     Asia      1972    67.7  841934  109348.
## 3 Kuwait     Asia      1952    55.6  160000  108382.
## 4 Kuwait     Asia      1962    60.5  358266   95458.
## 5 Kuwait     Asia      1967    64.6  575003   80895.
## 6 Kuwait     Asia      1977    69.3 1140357   59265.
## 7 Norway     Europe    2007    80.2 4627926  49357.
## 8 Kuwait     Asia      2007    77.6 2505559   47307.
## 9 Singapore  Asia      2007    80.0 4553009   47143.
## 10 Norway     Europe    2002    79.0 4535591   44684.
## # ... with 1,694 more rows
```

## Exercise #4

1. What is the lowest life expectancy in the `gapminder` dataset? Which country and year does it correspond to?
2. What is the highest life expectancy in the `gapminder` dataset? Which country and year does it correspond to?

## Solution to Exercise #4

Write your code and solutions here

1. The lowest life expectancy was Rwanda in 1992: 23.6 years at birth.

```
gapminder %>% arrange(lifeExp)
```

```
## # A tibble: 1,704 x 6
##   country    continent year lifeExp      pop gdpPercap
##   <fct>      <fct>    <int>   <dbl>   <int>    <dbl>
```

```
## 1 Rwanda Africa 1992 23.6 7290203 737.
## 2 Afghanistan Asia 1952 28.8 8425333 779.
## 3 Gambia Africa 1952 30 284320 485.
## 4 Angola Africa 1952 30.0 4232095 3521.
## 5 Sierra Leone Africa 1952 30.3 2143249 880.
## 6 Afghanistan Asia 1957 30.3 9240934 821.
## 7 Cambodia Asia 1977 31.2 6978607 525.
## 8 Mozambique Africa 1952 31.3 6446316 469.
## 9 Sierra Leone Africa 1957 31.6 2295678 1004.
## 10 Burkina Faso Africa 1952 32.0 4469979 543.
## # ... with 1,694 more rows
```

2. The highest life expectancy was in 2007 in Japan: 82.6 years at birth.

```
gapminder %>% arrange(desc(lifeExp))
```

```
## # A tibble: 1,704 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>    <int>    <dbl>
## 1 Japan        Asia      2007   82.6 127467972 31656.
## 2 Hong Kong, China Asia      2007   82.2  6980412 39725.
## 3 Japan        Asia      2002    82 127065841 28605.
## 4 Iceland      Europe     2007   81.8   301931 36181.
## 5 Switzerland  Europe     2007   81.7  7554661 37506.
## 6 Hong Kong, China Asia      2002   81.5  6762476 30209.
## 7 Australia    Oceania    2007   81.2 20434176 34435.
## 8 Spain         Europe     2007   80.9 40448191 28821.
## 9 Sweden        Europe     2007   80.9  9031088 33860.
## 10 Israel       Asia      2007   80.7  6426679 25523.
## # ... with 1,694 more rows
```

## Understanding the pipe: %>%

Let's revisit the pipe, %>%, that we've used in the code examples above. I told you that the command `gapminder %>% filter(year == 2007)` “pipes” the tibble `gapminder` into the function `filter()`. But what exactly does this mean? Take a look at the R help file for the `dplyr` function `filter`. We see that `filter()` takes something called `.data` as its first argument. Moving on to the “Arguments” section of the help file, we see that `.data` is “A `tbl`” i.e. a tibble. To better understand what this means, let's try using `filter` *without* the pipe:

```
filter(gapminder, year == 2007, country == 'United States')
```

```
## # A tibble: 1 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>    <int>    <dbl>
## 1 United States Americas  2007   78.2 301139947 42952.
```

Notice that this gives us *exactly* the same result as

```
gapminder %>% filter(year == 2007, country == 'United States')
```

```
## # A tibble: 1 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>    <int>    <dbl>
## 1 United States Americas   2007   78.2 301139947  42952.
```

In other words *The pipe is gives us an alternative way of supplying the first argument to a function*. Let's try this with a more familiar R function: `mean`. The first argument of `mean` is a vector `x`. So let's try using the pipe to compute the mean of some data:

```
x <- c(1, 5, 2, 7, 2)
x %>% mean
```

```
## [1] 3.4
```

The pipe supplies a function with its *first* argument. If we want to specify additional arguments, we need to do so within the function call itself. For example, here's how we could use the pipe to compute the mean after dropping missing observations:

```
y <- c(1, 5, NA, 7, 2)
y %>% mean(na.rm = TRUE)
```

```
## [1] 3.75
```

One important note about the pipe: it's *not* a base R command. Instead it's a command provided by the package `Magrittr`. (If you're familiar with the Belgian painter Magritte, you may realize that the name of this package is quite witty!) This package is installed automatically along with `dplyr`. So if we load the `tidyverse` package, which includes `dplyr`, the pipe is automatically available.

## Exercise #5

1. Write R code that uses the pipe to calculate the sample variance of `z <- c(4, 1, 5, NA, 3)` excluding the missing observation from the calculation.
2. Re-write the code from your solution to Exercise #4 *without* using the pipe.

## Solution to Exercise #5

*Write your code and solutions here*

1. Use the following code:

```
z <- c(4, 1, 5, NA, 3)
z %>% var(na.rm = TRUE)
```

```
## [1] 2.916667
```

2. Use the following code:



```
arrange(gapminder, lifeExp)
```

```
## # A tibble: 1,704 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>   <int>   <dbl>
## 1 Rwanda      Africa    1992   23.6 7290203    737.
## 2 Afghanistan Asia     1952   28.8 8425333    779.
## 3 Gambia      Africa    1952    30  284320    485.
## 4 Angola      Africa    1952   30.0 4232095   3521.
## 5 Sierra Leone Africa    1952   30.3 2143249    880.
## 6 Afghanistan Asia     1957   30.3 9240934    821.
## 7 Cambodia    Asia     1977   31.2 6978607    525.
## 8 Mozambique   Africa    1952   31.3 6446316    469.
## 9 Sierra Leone Africa    1957   31.6 2295678   1004.
## 10 Burkina Faso Africa    1952   32.0 4469979    543.
## # ... with 1,694 more rows
```

```
arrange(gapminder, desc(lifeExp))
```

```
## # A tibble: 1,704 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>   <int>   <dbl>
## 1 Japan      Asia     2007   82.6 127467972  31656.
## 2 Hong Kong, China Asia     2007   82.2  6980412  39725.
## 3 Japan      Asia     2002    82 127065841  28605.
## 4 Iceland    Europe    2007   81.8   301931  36181.
## 5 Switzerland Europe    2007   81.7  7554661  37506.
## 6 Hong Kong, China Asia     2002   81.5  6762476  30209.
## 7 Australia   Oceania   2007   81.2 20434176  34435.
## 8 Spain       Europe    2007   80.9 40448191  28821.
## 9 Sweden      Europe    2007   80.9  9031088  33860.
## 10 Israel     Asia     2007   80.7  6426679  25523.
## # ... with 1,694 more rows
```

## Chaining commands

In the examples we've looked at so far, the pipe doesn't seem all that useful: it's just an alternative way of specifying the first argument to a function. The true power and convenience of the pipe only becomes apparent we need to *chain* a series of commands together. For example, suppose we wanted to display the 1952 data from `gapminder` sorted by `gdpPercap` in descending order. Using the pipe, this is easy:

```
gapminder %>%
  filter(year == 1952) %>%
  arrange(desc(gdpPercap))
```

```
## # A tibble: 142 x 6
##   country      continent year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int>   <dbl>   <int>   <dbl>
## 1 Kuwait      Asia     1952   55.6  160000  108382.
## 2 Switzerland Europe    1952   69.6  4815000  14734.
```

```
## 3 United States Americas 1952 68.4 157553000 13990.
## 4 Canada Americas 1952 68.8 14785584 11367.
## 5 New Zealand Oceania 1952 69.4 1994794 10557.
## 6 Norway Europe 1952 72.7 3327728 10095.
## 7 Australia Oceania 1952 69.1 8691212 10040.
## 8 United Kingdom Europe 1952 69.2 50430000 9980.
## 9 Bahrain Asia 1952 50.9 120447 9867.
## 10 Denmark Europe 1952 70.8 4334000 9692.
## # ... with 132 more rows
```

Notice how I split the commands across multiple lines. This is good practice: it makes your code much easier to read. So what's happening when we chain commands in this way? The first step in the chain `gapminder %>% filter(year == 1952)` returns a tibble: the subset of `gapminder` for which `year` is 1952. The next step `%>% arrange(gdpPercap)` pipes this *new* tibble into the function `arrange()`, giving us the desired result. I hope you agree with me that this is pretty intuitive: even if we didn't know anything about `dplyr` we could *almost* figure out what this code is supposed to do. In stark contrast, let's look at the code we'd have to use if we wanted to accomplish the same task *without* using the pipe:

```
arrange(filter(gapminder, year == 1952), desc(gdpPercap))
```

```
## # A tibble: 142 x 6
##   country      continent  year lifeExp      pop gdpPercap
##   <fct>        <fct>    <int> <dbl>    <int>    <dbl>
## 1 Kuwait      Asia      1952  55.6    160000  108382.
## 2 Switzerland Europe    1952  69.6    4815000  14734.
## 3 United States Americas 1952  68.4  157553000 13990.
## 4 Canada      Americas 1952  68.8   14785584 11367.
## 5 New Zealand Oceania  1952  69.4   1994794 10557.
## 6 Norway      Europe   1952  72.7   3327728 10095.
## 7 Australia   Oceania  1952  69.1   8691212 10040.
## 8 United Kingdom Europe   1952  69.2  50430000 9980.
## 9 Bahrain     Asia     1952  50.9    120447 9867.
## 10 Denmark    Europe   1952  70.8   4334000 9692.
## # ... with 132 more rows
```

There are many reasons why this code is harder to read, but the most important one is that the commands `arrange` and `filter` have to appear in the code in the *opposite* of the order in which they are actually being carried out. This is because parentheses are evaluated from *inside to outside*. This is what's great about the pipe: it lets us write our code in a way that accords with the actual order of the steps we want to carry out.

## Exercise #6

1. What was the most populous European country in 1992? Write appropriate `dplyr` code using the pipe to display the information you need to answer this question.
2. Re-write your code from part 1. *without* using the pipe.

## Solution to Exercise #6

Write your code and solutions here

1. The most populous European country in 1992 was Germany.

```
gapminder %>%  
  filter(year == 1992, continent == 'Europe') %>%  
  arrange(desc(pop))
```

```
## # A tibble: 30 x 6  
##   country      continent  year lifeExp      pop gdpPercap  
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>  
## 1 Germany      Europe    1992   76.1 80597764  26505.  
## 2 Turkey        Europe    1992   66.1 58179144   5678.  
## 3 United Kingdom Europe    1992   76.4 57866349  22705.  
## 4 France        Europe    1992   77.5 57374179  24704.  
## 5 Italy          Europe    1992   77.4 56840847  22014.  
## 6 Spain          Europe    1992   77.6 39549438  18603.  
## 7 Poland         Europe    1992   71.0 38370697   7739.  
## 8 Romania        Europe    1992   69.4 22797027   6598.  
## 9 Netherlands    Europe    1992   77.4 15174244  26791.  
## 10 Hungary        Europe    1992   69.2 10348684  10536.  
## # ... with 20 more rows
```

2. Use the following code:

```
arrange(filter(gapminder, year == 1992, continent == 'Europe'), desc(pop))
```

```
## # A tibble: 30 x 6  
##   country      continent  year lifeExp      pop gdpPercap  
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>  
## 1 Germany      Europe    1992   76.1 80597764  26505.  
## 2 Turkey        Europe    1992   66.1 58179144   5678.  
## 3 United Kingdom Europe    1992   76.4 57866349  22705.  
## 4 France        Europe    1992   77.5 57374179  24704.  
## 5 Italy          Europe    1992   77.4 56840847  22014.  
## 6 Spain          Europe    1992   77.6 39549438  18603.  
## 7 Poland         Europe    1992   71.0 38370697   7739.  
## 8 Romania        Europe    1992   69.4 22797027   6598.  
## 9 Netherlands    Europe    1992   77.4 15174244  26791.  
## 10 Hungary        Europe    1992   69.2 10348684  10536.  
## # ... with 20 more rows
```

## Change an existing variable or create a new one with mutate

It's a little hard to read the column `pop` in `gapminder` since there are so many digits. Suppose that, instead of raw population, we wanted to display population *in millions*. This requires us to `pop` by 1000000, which we can do using the function `mutate()` from `dplyr` as follows:

```
gapminder %>%  
  mutate(pop = pop / 1000000)
```

```
## # A tibble: 1,704 x 6
```

```
##   country      continent  year lifeExp   pop gdpPercap
##   <fct>        <fct>    <int>  <dbl> <dbl>   <dbl>
##  1 Afghanistan Asia      1952   28.8  8.43    779.
##  2 Afghanistan Asia      1957   30.3  9.24    821.
##  3 Afghanistan Asia      1962   32.0 10.3    853.
##  4 Afghanistan Asia      1967   34.0 11.5    836.
##  5 Afghanistan Asia      1972   36.1 13.1    740.
##  6 Afghanistan Asia      1977   38.4 14.9    786.
##  7 Afghanistan Asia      1982   39.9 12.9    978.
##  8 Afghanistan Asia      1987   40.8 13.9    852.
##  9 Afghanistan Asia      1992   41.7 16.3    649.
## 10 Afghanistan Asia      1997   41.8 22.2    635.
## # ... with 1,694 more rows
```

Note the syntax here: within `mutate()` we have an assignment statement, namely `pop = pop / 1000000`. This tells R to calculate `pop / 1000000` and assign the result to `pop`, in place of the original variable.

We can also use `mutate()` to create a new variable. The `gapminder` dataset doesn't contain overall GDP, only GDP per capita. To calculate GDP, we need to multiply `gdpPercap` by `pop`. But wait! Didn't we just change `pop` so it's expressed in millions? No: we never *stored* the results of our previous command, we simply displayed them. Just as I discussed above, unless you *overwrite* it, the original `gapminder` dataset will be unchanged. With this in mind, we can create the `gdp` variable as follows:

```
gapminder %>% mutate(gdp = pop * gdpPercap)
```

```
## # A tibble: 1,704 x 7
##   country      continent  year lifeExp   pop gdpPercap      gdp
##   <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.
##  7 Afghanistan Asia      1982   39.9 12881816    978. 12598563401.
##  8 Afghanistan Asia      1987   40.8 13867957    852. 11820990309.
##  9 Afghanistan Asia      1992   41.7 16317921    649. 10595901589.
## 10 Afghanistan Asia      1997   41.8 22227415    635. 14121995875.
## # ... with 1,694 more rows
```

## Exercise #7

1. Explain why we used `=` rather than `==` in the `mutate()` examples above.
2. Which country in the Americas had the shortest life expectancy *in months* in the year 1962? Write appropriate dplyr code using the pipe to display the information you need to answer this question.

## Solution to Exercise #7

*Write your code and solutions here*

1. We used `=` because this is the assignment operator. In contrast `==` tests for equality, returning `TRUE` or `FALSE`.
2. Bolivia had the shortest life expectancy: 521 months.

```
gapminder %>%
  mutate(lifeExpMonths = 12 * lifeExp) %>%
  filter(year == 1962, continent == 'Americas') %>%
  arrange(lifeExpMonths)
```

```
## # A tibble: 25 x 7
##   country      continent year lifeExp      pop gdpPercap lifeExpMonths
##   <fct>        <fct>    <int>  <dbl>    <int>    <dbl>        <dbl>
## 1 Bolivia      Americas   1962   43.4  3.59e6   2181.         521.
## 2 Haiti        Americas   1962   43.6  3.88e6   1797.         523.
## 3 Guatemala    Americas   1962   47.0  4.21e6   2750.         563.
## 4 Honduras     Americas   1962   48.0  2.09e6   2291.         576.
## 5 Nicaragua    Americas   1962   48.6  1.59e6   3634.         584.
## 6 Peru         Americas   1962   49.1  1.05e7   4957.         589.
## 7 El Salvador  Americas   1962   52.3  2.75e6   3777.         628.
## 8 Dominican Repu~ Americas   1962   53.5  3.45e6   1662.         642.
## 9 Ecuador      Americas   1962   54.6  4.68e6   4086.         656.
## 10 Brazil      Americas   1962   55.7  7.60e7   3337.         668.
## # ... with 15 more rows
```

## Some dplyr terminology

In the examples above, we've learned how to use a number of `dplyr` function: `filter`, `arrange`,

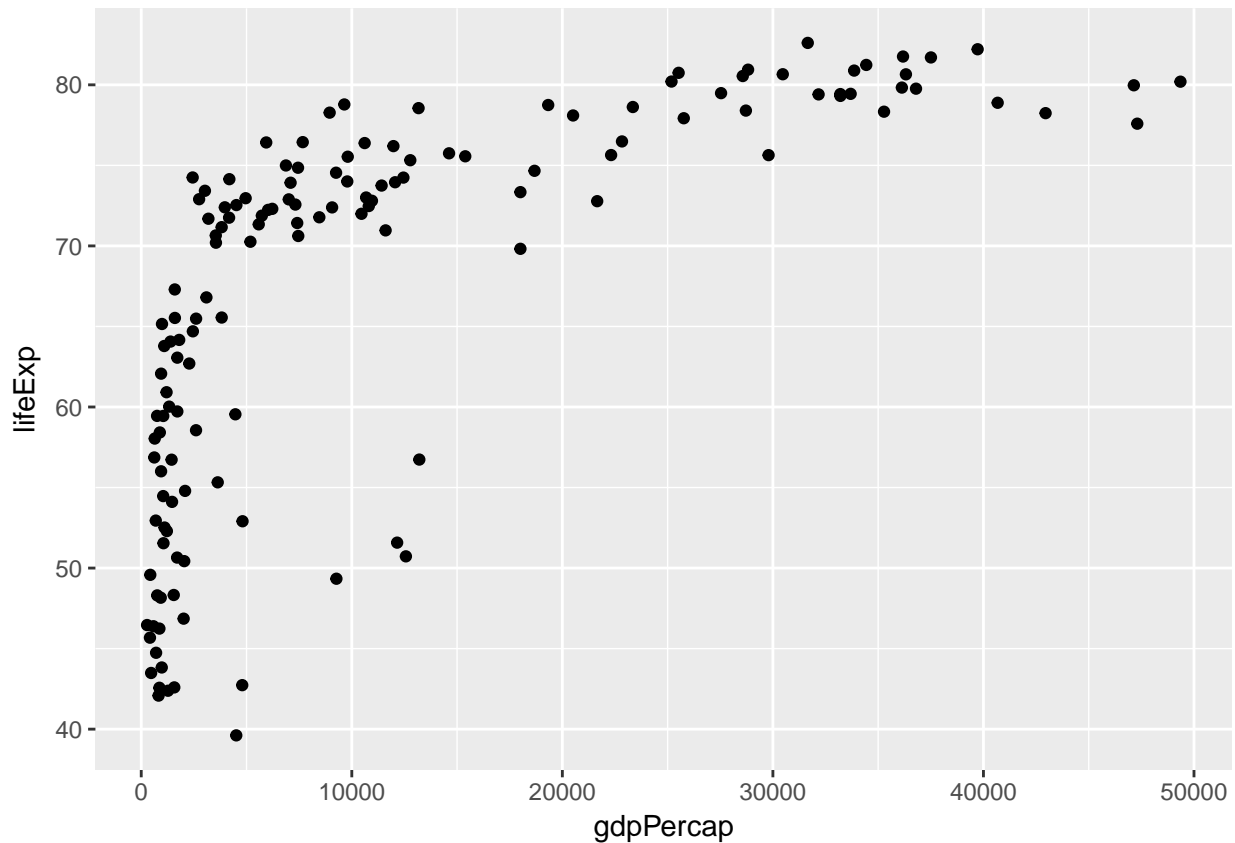
## A simple scatterplot using ggplot2

Now that we know the basics of `dplyr`, we'll turn our attention to graphics. R has many powerful build-in graphics functions that may be familiar to you from Econ 103. In this class, however, we'll use a very powerful package for statistical visualization called `ggplot2`. There's nothing more for you to instead or load, since `ggplot2` is included in the `tidyverse` package, which you've already installed and loaded. We'll start off by constructing a subset of the `gapminder` dataset that contains information from the year 2007 that we'll use for our plots below.

```
gapminder_2007 <- gapminder %>% filter(year == 2007)
```

It takes some time to grow accustomed to `ggplot2` syntax, so rather than giving you a lot of detail, we're going to look at a series of increasingly more complicated examples. Our first example will be a simple scatterplot using `gapminder_2007`. Each point will correspond to a single country in 2007. Its x-coordinate will be GDP per capita and its y-coordinate will be life expectancy. Here's the code:

```
ggplot(gapminder_2007, aes(x = gdpPercap, y = lifeExp)) + geom_point()
```



We see that GDP per capita is a very strong predictor of life expectancy, although the relationship is non-linear.

## Exercise #8

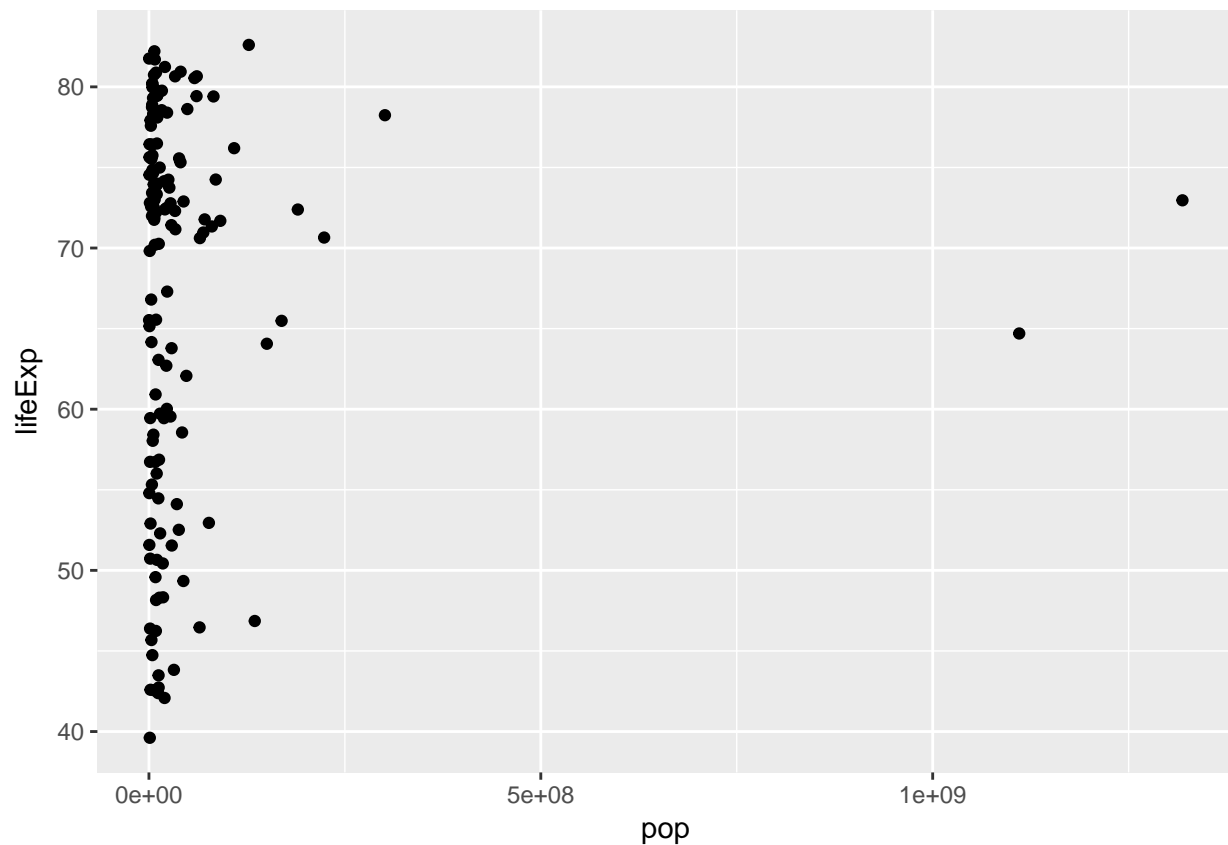
1. Using my code example as a template, make a scatterplot with `pop` on the x-axis and `lifeExp` on the y-axis using `gapminder_2007`. Does there appear to be a relationship between population and life expectancy?
2. Repeat 1. with `gdpPercap` on the y-axis.

## Solution to Exercise #8

*Write your code and solutions here*

1. There is no clear relationship between population and life expectancy based on the 2007 data:

```
ggplot(gapminder_2007, aes(x = pop, y = lifeExp)) + geom_point()
```



2.

There is no clear relationship between population and GDP per capita based on the 2007 data:

```
ggplot(gapminder_2007, aes(x = pop, y = gdpPercap)) + geom_point()
```

