

Lab #2 - Gapminder Dataset

Econ 224

August 30th, 2018

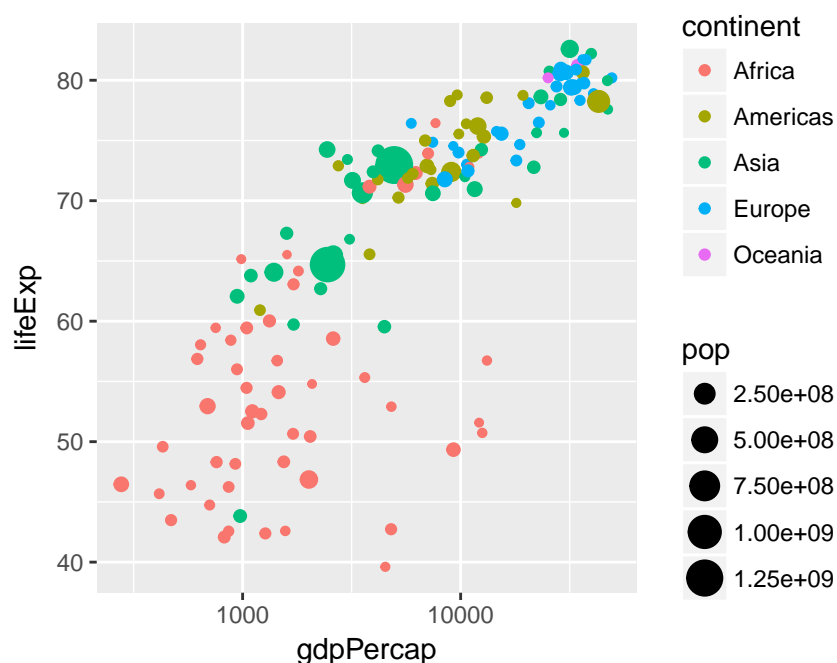
Introduction

Today we'll revisit the `gapminder` dataset and use it to introduce some more advanced features of `dplyr` and `ggplot2`, building on the material from our first lab. Before you begin, make sure that you have loaded the `tidyverse` and `gapminder` packages.

Faceting - Plotting multiple subsets at once

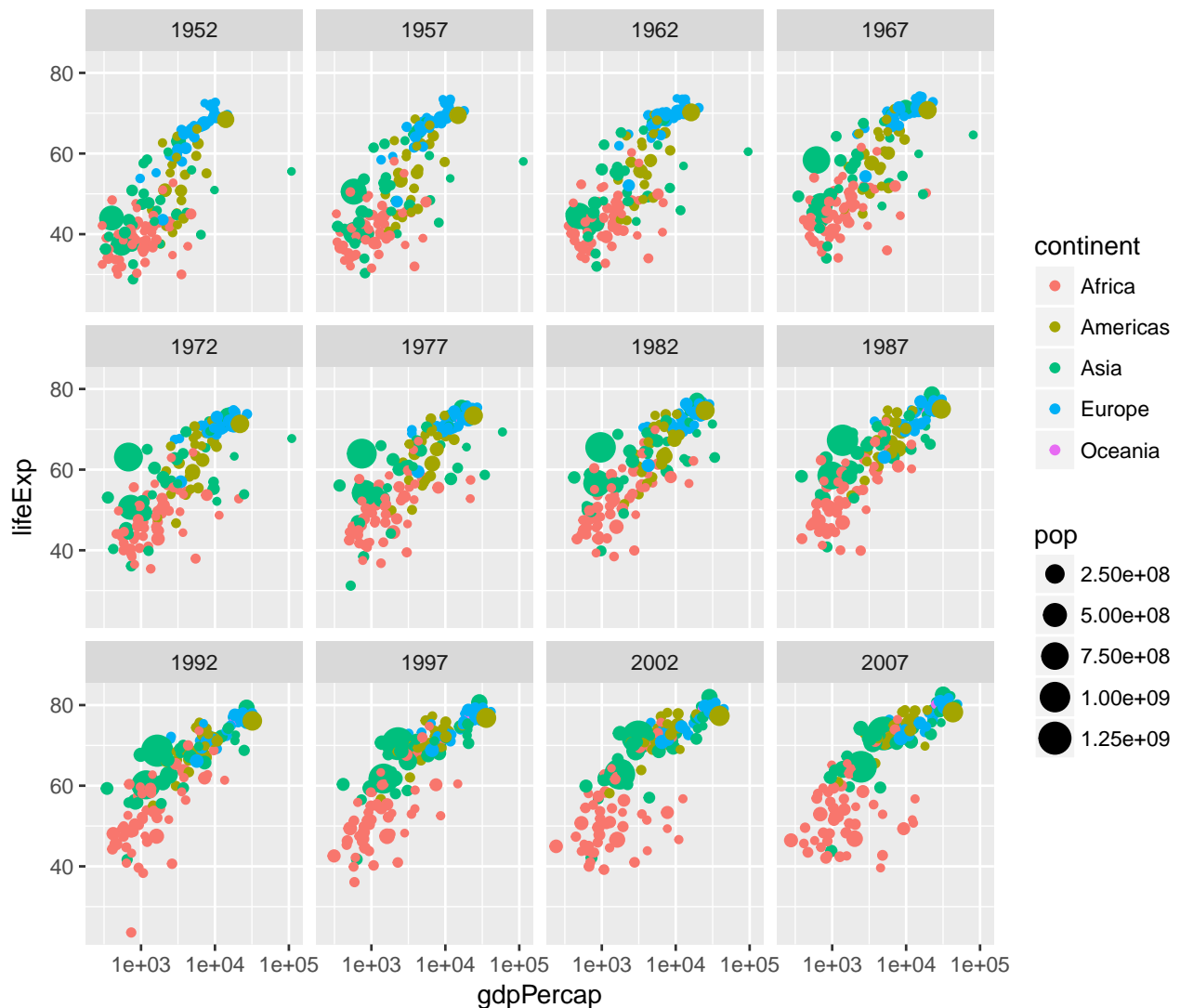
Let's pick up where we left off in lab #1, with a plot of GDP per capita and life expectancy in 2007:

```
gapminder_2007 <- gapminder %>%  
  filter(year == 2007)  
ggplot(gapminder_2007) +  
  geom_point(aes(x = gdpPerCap, y = lifeExp, color = continent, size = pop)) +  
  scale_x_log10()
```



This is an easy way to make a plot for a single year. But what if you wanted to make the same plot for *every year* in the `gapminder` dataset? It would take a lot of copying-and-pasting of the preceding code chunk to accomplish this. Fortunately there's a much easier way: *faceting*. In `ggplot2` a *facet* is a subplot that corresponds to a subset of your dataset, for example the year 2007. We'll now use faceting to reproduce the plot from above for all the years in `gapminder` simultaneously:

```
ggplot(gapminder) +
  geom_point(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  scale_x_log10() +
  facet_wrap(~ year)
```



Note the syntax here: in a similar way to how we added `scale_x_log10()` to plot on the log scale, we add `facet_wrap(~ year)` to facet by `year`. The tilde `~` is important: this has to precede the variable by which you want to facet.

Now that we understand how to produce it, let's take a closer look at this plot. Notice how this plot allows us to visualize five variables *simultaneously*. By looking at how the plots change over time, we see a pattern of increasing GDP per capita and life expectancy throughout the world between 1952 and 2007. Notice in particular the dramatic improvements in both variables in the Asian economies.

Exercise #1

1. What would happen if I were to run the following code? Explain briefly.

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  scale_x_log10() +
  facet_wrap(~ year)
```

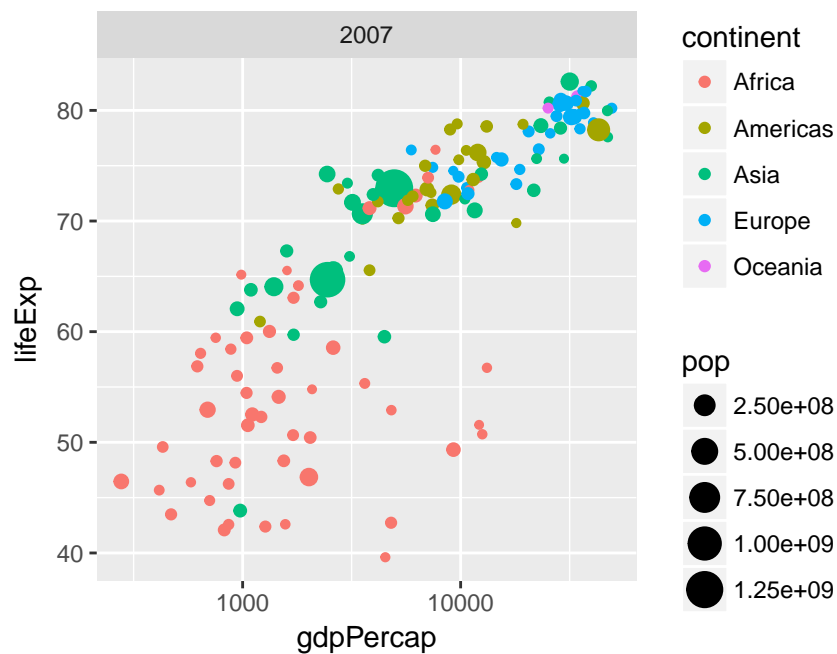
2. Make a scatterplot with data from `gapminder` for the year 1977. Your plot should be faceted by continent with GDP per capita on the log scale on the x-axis, life expectancy on the y-axis, and population indicated by the size of each point.
3. What would happen if you tried to facet by `pop`? Explain briefly.

Solution to Exercise #1

Write your code and solutions here

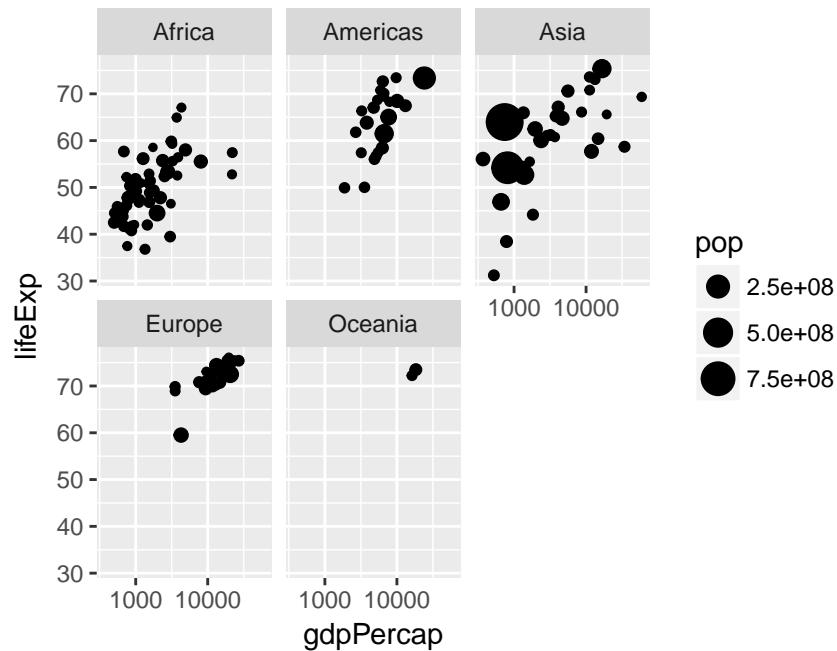
1. We'll only get one facet since the tibble `gapminder_2007` only has data for 2007:

```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  scale_x_log10() +
  facet_wrap(~ year)
```



2. Use the following code:

```
gapminder_1977 <- gapminder %>%
  filter(year == 1977)
ggplot(gapminder_1977) +
  geom_point(aes(x = gdpPercap, y = lifeExp, size = pop)) +
  scale_x_log10() +
  facet_wrap(~ continent)
```



3. You'll get something crazy if you try this. Population is continuous rather than categorical so every country has a different value for this variable. You'll end up with one plot for every country, containing a single point:

```
# Not run: it takes a long time and looks nasty!
gapminder_1977 <- gapminder %>%
  filter(year == 1977)
ggplot(gapminder_1977) +
  geom_point(aes(x = gdpPercap, y = lifeExp, color = continent)) +
  scale_x_log10() +
  facet_wrap(~ pop)
```

dplyr verbs

For the next few sections we'll take a short break from `ggplot2` and turn our attention to `dplyr`. In lab #1 we learned about the pipe, `%>%`, and two `dplyr` functions: `filter()` and `arrange()`. In the parlance of the `dplyr` documentation, these are called “verbs.” In `dplyr` we use `%>%` to combine these verbs in various ways to manipulate a tibble. In this section and the following two, we'll learn three more `dplyr` verbs: `select`, `summarize` and `group_by`.

The select verb

We use the `select` verb to select columns. Using `select` we could do this as follows:

```
gapminder %>% select(pop)
```

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

```

      <int>
1  8425333
2  9240934
3 10267083
4 11537966
5 13079460
6 14880372
7 12881816
8 13867957
9 16317921
10 22227415
# ... with 1,694 more rows

```

To display only `pop`, `country`, and `year`, use the following:

```
gapminder %>% select(pop, country, year)
```

```

# A tibble: 1,704 x 3
      pop country    year
  <int> <fct>    <int>
1  8425333 Afghanistan 1952
2  9240934 Afghanistan 1957
3 10267083 Afghanistan 1962
4 11537966 Afghanistan 1967
5 13079460 Afghanistan 1972
6 14880372 Afghanistan 1977
7 12881816 Afghanistan 1982
8 13867957 Afghanistan 1987
9 16317921 Afghanistan 1992
10 22227415 Afghanistan 1997
# ... with 1,694 more rows

```

Now suppose that we wanted to select every column *except* `pop`. Here's one way to do it:

```
gapminder %>% select(country, continent, year, lifeExp, gdpPercap)
```

```

# A tibble: 1,704 x 5
  country continent year lifeExp gdpPercap
  <fct>    <fct>    <int>   <dbl>   <dbl>
1 Afghanistan Asia      1952    28.8    779.
2 Afghanistan Asia      1957    30.3    821.
3 Afghanistan Asia      1962    32.0    853.
4 Afghanistan Asia      1967    34.0    836.
5 Afghanistan Asia      1972    36.1    740.
6 Afghanistan Asia      1977    38.4    786.
7 Afghanistan Asia      1982    39.9    978.
8 Afghanistan Asia      1987    40.8    852.
9 Afghanistan Asia      1992    41.7    649.
10 Afghanistan Asia      1997    41.8    635.
# ... with 1,694 more rows

```

but that takes a lot of typing! If there were more than a handful of columns in our tibble it would be very difficult to *deselect* a column in this way. Fortunately there's a shortcut: use the minus sign

```
gapminder %>% select(-pop)
```

```
# A tibble: 1,704 x 5
  country      continent  year lifeExp gdpPercap
  <fct>        <fct>    <int>   <dbl>   <dbl>
1 Afghanistan Asia      1952    28.8    779.
2 Afghanistan Asia      1957    30.3    821.
3 Afghanistan Asia      1962    32.0    853.
4 Afghanistan Asia      1967    34.0    836.
5 Afghanistan Asia      1972    36.1    740.
6 Afghanistan Asia      1977    38.4    786.
7 Afghanistan Asia      1982    39.9    978.
8 Afghanistan Asia      1987    40.8    852.
9 Afghanistan Asia      1992    41.7    649.
10 Afghanistan Asia      1997    41.8    635.
# ... with 1,694 more rows
```

Just as we could when *selecting*, we can *deselect* multiple columns by separating their names with a comma:

```
gapminder %>% select(-pop, -year)
```

```
# A tibble: 1,704 x 4
  country      continent lifeExp gdpPercap
  <fct>        <fct>    <dbl>   <dbl>
1 Afghanistan Asia      28.8    779.
2 Afghanistan Asia      30.3    821.
3 Afghanistan Asia      32.0    853.
4 Afghanistan Asia      34.0    836.
5 Afghanistan Asia      36.1    740.
6 Afghanistan Asia      38.4    786.
7 Afghanistan Asia      39.9    978.
8 Afghanistan Asia      40.8    852.
9 Afghanistan Asia      41.7    649.
10 Afghanistan Asia      41.8    635.
# ... with 1,694 more rows
```

It's easy to mix up the `dplyr` verbs `select` and `filter`. Here's a handy mnemonic: `filter` filters Rows while `select` selects Columns. Suppose we wanted to select only the column `pop` from `gapminder`.

Exercise #2

1. Select only the columns `year`, `lifeExp`, and `country` in `gapminder`.
2. Select all the columns *except* `year`, `lifeExp`, and `country` in `gapminder`.

Solution to Exercise #2

Write your code and solutions here

1. Use the following:

```
gapminder %>% select(year, lifeExp, country)
```

```
# A tibble: 1,704 x 3
  year lifeExp country
  <int>   <dbl> <fct>
1  1952    28.8 Afghanistan
2  1957    30.3 Afghanistan
3  1962    32.0 Afghanistan
4  1967    34.0 Afghanistan
5  1972    36.1 Afghanistan
6  1977    38.4 Afghanistan
7  1982    39.9 Afghanistan
8  1987    40.8 Afghanistan
9  1992    41.7 Afghanistan
10 1997    41.8 Afghanistan
# ... with 1,694 more rows
```

2. Use the following:

```
gapminder %>% select(-year, -lifeExp, -country)
```

```
# A tibble: 1,704 x 3
  continent      pop gdpPercap
  <fct>         <int>   <dbl>
1 Asia      8425333    779.
2 Asia      9240934    821.
3 Asia     10267083    853.
4 Asia     11537966    836.
5 Asia     13079460    740.
6 Asia     14880372    786.
7 Asia     12881816    978.
8 Asia     13867957    852.
9 Asia     16317921    649.
10 Asia     22227415    635.
# ... with 1,694 more rows
```

The summarize verb

Suppose we want to calculate the sample mean of the column `lifeExp` in `gapminder`. We can do this using the `summarize` verb as follows:

```
gapminder %>% summarize(mean_lifeExp = mean(lifeExp))
```

```
# A tibble: 1 x 1
  mean_lifeExp
  <dbl>
1      59.5
```

Note the syntax: within `summarize` we have an *assignment statement*. In particular, we assign `mean(lifeExp)` to the variable `mean_lifeExp`. The key thing to know about `summarize` is that it always returns *collapses*

a tibble with many rows into a single row. When we think about computing a sample mean, this makes sense: we want to summarize the column `lifeExp` as a single number. It doesn't actually make much sense to compute the mean of `lifeExp` because this involves averaging over different countries *and* different years. Instead let's compute the mean for a single year: 1952:

```
gapminder %>%
  filter(year == 1952) %>%
  summarize(mean_lifeExp = mean(lifeExp))
```

```
# A tibble: 1 x 1
  mean_lifeExp
    <dbl>
1         49.1
```

We can use `summarize` to compute multiple summary statistics for a single variable, the same summary statistic for multiple variables, or both:

```
gapminder %>%
  filter(year == 1952) %>%
  summarize(mean_lifeExp = mean(lifeExp),
            sd_lifeExp = sd(lifeExp),
            mean_pop = mean(pop))
```

```
# A tibble: 1 x 3
  mean_lifeExp sd_lifeExp mean_pop
    <dbl>      <dbl>      <dbl>
1         49.1         12.2 16950402.
```

Note that if we *don't* explicitly use an assignment statement, R will make up names for us based on the commands that we used:

```
gapminder %>%
  filter(year == 1952) %>%
  summarize(mean(lifeExp), median(lifeExp), max(lifeExp))
```

```
# A tibble: 1 x 3
  `mean(lifeExp)` `median(lifeExp)` `max(lifeExp)`
    <dbl>          <dbl>          <dbl>
1         49.1         45.1         72.7
```

Exercise #3

1. Use `summarize` to compute the 75th percentile of life expectancy in 1977.
2. Use `summarize` to compute the 75th percentile of life expectancy among Asian countries in 1977.

Solution to Exercise #3

Write your code and solutions here

1. The 75th percentile of life expectancy in 1977 was 70.4 years at birth.

```
gapminder %>%  
  filter(year == 1977) %>%  
  summarize(quantile(lifeExp, 0.75))
```

```
# A tibble: 1 x 1  
  `quantile(lifeExp, 0.75)`  
    <dbl>  
1                70.4
```

2. The 75th percentile of life expectancy in 1977 among African countries was

```
gapminder %>%  
  filter(year == 1977, continent == 'Asia') %>%  
  summarize(quantile(lifeExp, 0.75))
```

```
# A tibble: 1 x 1  
  `quantile(lifeExp, 0.75)`  
    <dbl>  
1                65.9
```

The `group_by` verb

The true power of `summarize` is its ability to compute grouped summary statistics in combination with another dplyr verb: `group_by`