Lab #2 - Gapminder Dataset

Econ 224

August 30th, 2018

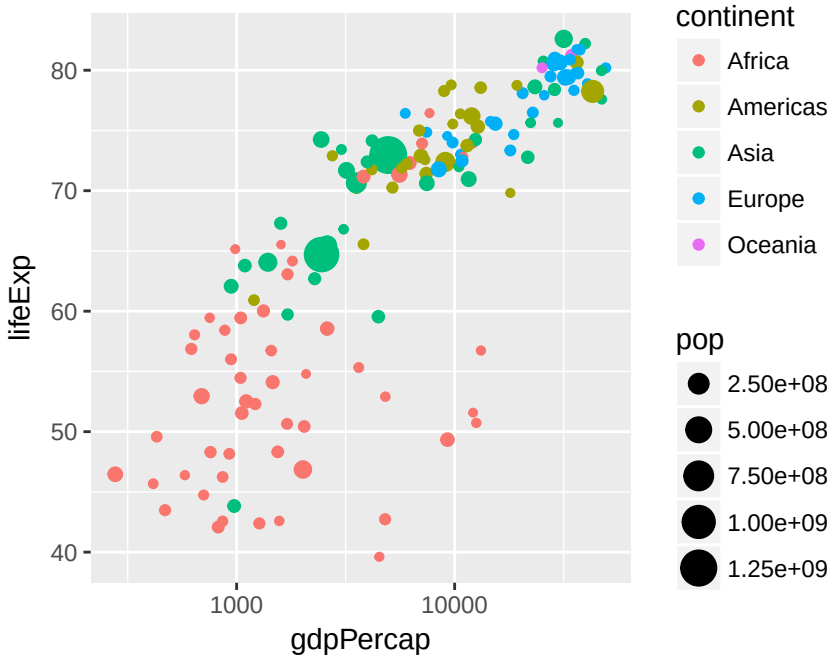
Introduction

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

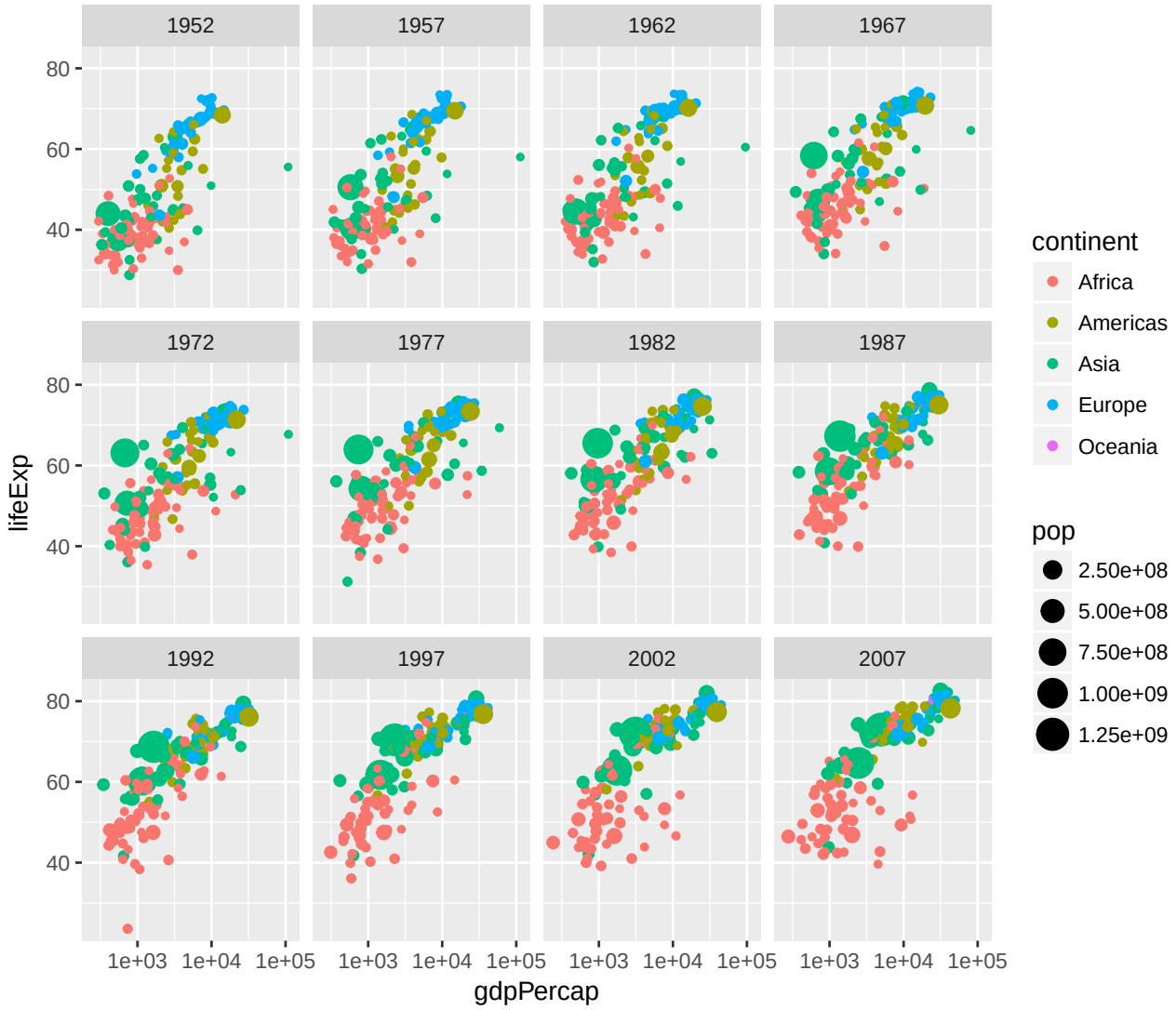
Let's pick up where we left off in lab #1, with a plot of GDP per capita and life expectancy in 2007:

```
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.

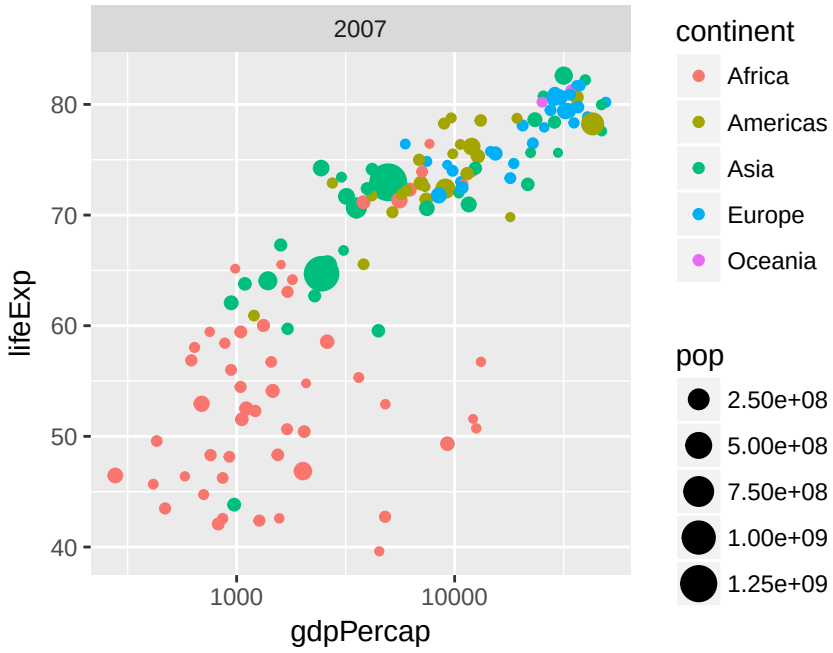
```
ggplot(gapminder_2007) +
  geom_point(aes(x = gdpPercap, y = lifeExp, color = continent, size = pop)) +
  scale_x_log10() +
  facet_wrap(~ year)
```

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

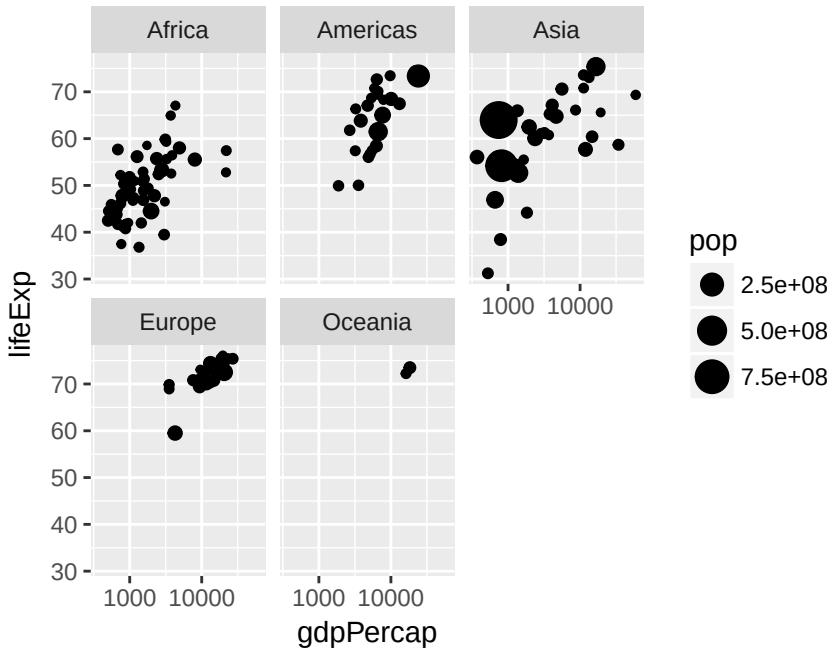
1. We'll only get one facet since the tibble `gapminder_2007` only has data for 2007:

```
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

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

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`. In essence, `group_by` allows us to tell `dplyr` that we don't want to work with the whole dataset at once; rather we want to work with particular *subsets* or groups. The basic idea is similar to what we've done using `filter` in the past. For example, to calculate mean population (in millions) and mean life expectancy in the year 2007, we could use the following code:

```
gapminder %>%
  filter(year == 2007) %>%
  summarize(meanPop = mean(pop) / 1000000, meanLifeExp = mean(lifeExp))
```

```
# A tibble: 1 x 2
  meanPop meanLifeExp
    <dbl>       <dbl>
1    44.0        67.0
```

Using `group_by` we could do the same thing for *all* years in the dataset at once:

```
gapminder %>%
  group_by(year) %>%
  summarize(meanPop = mean(pop) / 1000000, meanLifeExp = mean(lifeExp))
```

```
# A tibble: 12 x 3
    year meanPop meanLifeExp
   <int>   <dbl>       <dbl>
 1  1952    17.0        49.1
 2  1957    18.8        51.5
 3  1962    20.4        53.6
```

```
 4  1967    22.7        55.7
 5  1972    25.2        57.6
 6  1977    27.7        59.6
 7  1982    30.2        61.5
 8  1987    33.0        63.2
 9  1992    36.0        64.2
10  1997    38.8        65.0
11  2002    41.5        65.7
12  2007    44.0        67.0
```

Notice what has changed in the second code block: we replaced `filter(year == 2007)` with `group_by(year)`. This tells `dplyr` that, rather than simply restricting attention to data from 2007, we want to form *subsets* (groups) of the dataset that correspond to the values of the `year` variable. Whatever comes after `group_by` will then be calculated for these subsets.

Here's another example. Suppose we wanted to calculate mean life expectancy and total population in each *continent* during the year 2007. To accomplish this, we can chain together the `filter`, `group_by` and `summarize` verbs as follows:

```
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarize(meanPop = mean(pop) / 1000000, meanLifeExp = mean(lifeExp))
```

```
# A tibble: 5 x 3
  continent meanPop meanLifeExp
  <fct>       <dbl>       <dbl>
1 Africa       17.9        54.8
2 Americas     36.0        73.6
3 Asia        116.         70.7
4 Europe       19.5        77.6
5 Oceania      12.3        80.7
```

We can also use `group_by` to subset over multiple variables at once. For example, to calculate mean life expectancy and total population in each continent *separately* for every year, we can use the following code:

```
gapminder %>%
  group_by(year, continent) %>%
  summarize(meanPop = mean(pop) / 1000000, meanLifeExp = mean(lifeExp))
```

```
# A tibble: 60 x 4
# Groups:   year [?]
    year continent meanPop meanLifeExp
   <int> <fct>       <dbl>       <dbl>
 1  1952 Africa       4.57        39.1
 2  1952 Americas    13.8         53.3
 3  1952 Asia        42.3         46.3
 4  1952 Europe      13.9         64.4
 5  1952 Oceania      5.34        69.3
 6  1957 Africa       5.09        41.3
 7  1957 Americas    15.5         56.0
 8  1957 Asia        47.4         49.3
 9  1957 Europe      14.6         66.7
10  1957 Oceania      5.97        70.3
# ... with 50 more rows
```

Exercise #4

1. Why doesn't the following code work as expected?

```
gapminder %>%
  summarize(meanLifeExp = mean(lifeExp)) %>%
  group_by(year)
```

2. Calculate the median GDP per capita in each continent in 1977.
3. Repeat 2. but sort your results in descending order.
4. Calculate the mean and standard deviation of life expectancy for separately for each continent in every year *after* 1977. Sort your results in ascending order by the standard deviation of life expectancy.

Solution to Exercise #4

1. The steps are carried out in the wrong order: we need to form groups *first* and then calculate our desired summaries.
2. Use the following:

```
gapminder %>%
  filter(year == 1977) %>%
  group_by(continent) %>%
  summarize(medGDPc = median(gdpPercap))
```

```
# A tibble: 5 x 2
  continent medGDPc
  <fct>       <dbl>
1 Africa      1400.
2 Americas    6281.
3 Asia        3195.
4 Europe     14226.
5 Oceania    17284.
```

3. Use the following:

```
gapminder %>%
  filter(year == 1977) %>%
  group_by(continent) %>%
  summarize(medGDPc = median(gdpPercap)) %>%
  arrange(desc(medGDPc))
```

```
# A tibble: 5 x 2
  continent medGDPc
  <fct>       <dbl>
1 Oceania    17284.
2 Europe     14226.
3 Americas    6281.
4 Asia        3195.
5 Africa      1400.
```

4. Use the following:

```
gapminder %>%
  filter(year > 1977) %>%
  group_by(continent, year) %>%
  summarize(meanGDPc = mean(gdpPercap), sdGDPc = sd(gdpPercap)) %>%
  arrange(sdGDPc)
```

```
# A tibble: 30 x 4
# Groups:   continent [5]
   continent  year meanGDPc sdGDPc
   <fct>     <int>    <dbl>  <dbl>
 1 Oceania    1982   18555.  1304.
 2 Oceania    1987   20448.  2038.
 3 Africa     1987    2283.  2567.
 4 Africa     1992    2282.  2644.
 5 Africa     1997    2379.  2821.
 6 Africa     2002    2599.  2973.
 7 Africa     1982    2482.  3243.
 8 Oceania    1992   20894.  3579.
 9 Africa     2007    3089.  3618.
10 Oceania    1997   24024.  4206.
# ... with 20 more rows
```

Plotting summarized data

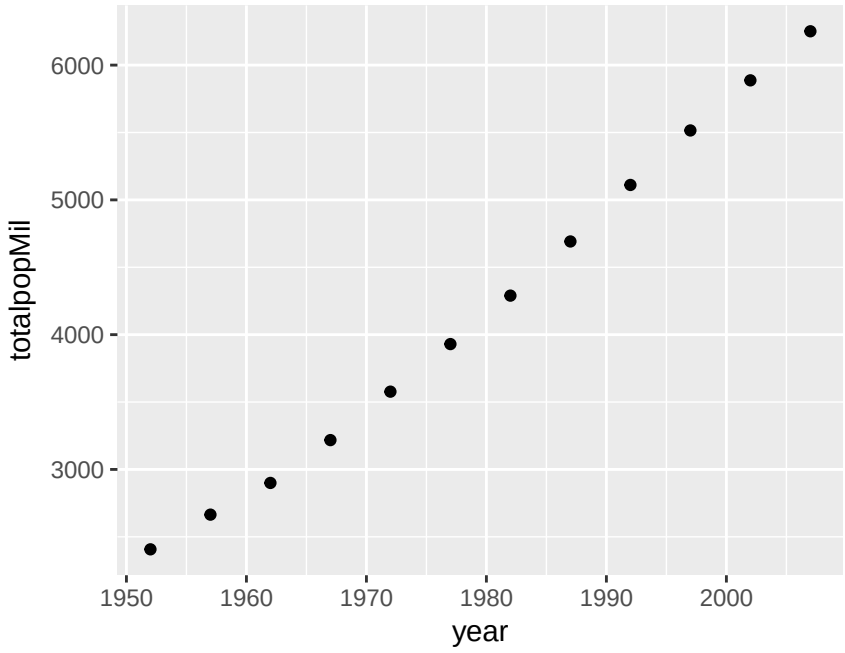
By combining `summarize` and `group_by` with `ggplot`, it's easy to make plots of grouped data. For example, here's how we could plot total world population in millions from 1952 to 2007. First we construct a tibble which I'll name `by_year` containing the desired summary statistic grouped by year and display it:

```
by_year <- gapminder %>%
  mutate(popMil = pop / 1000000) %>%
  group_by(year) %>%
  summarize(totalpopMil = sum(popMil))
by_year
```

```
# A tibble: 12 x 2
    year totalpopMil
   <int>       <dbl>
 1  1952       2407.
 2  1957       2664.
 3  1962       2900.
 4  1967       3217.
 5  1972       3577.
 6  1977       3930.
 7  1982       4289.
 8  1987       4691.
 9  1992       5111.
10  1997       5515.
11  2002       5887.
12  2007       6251.
```

Then we make a scatterplot using `ggplot`:

```
ggplot(by_year) +
  geom_point(aes(x = year, y = totalpopMil))
```

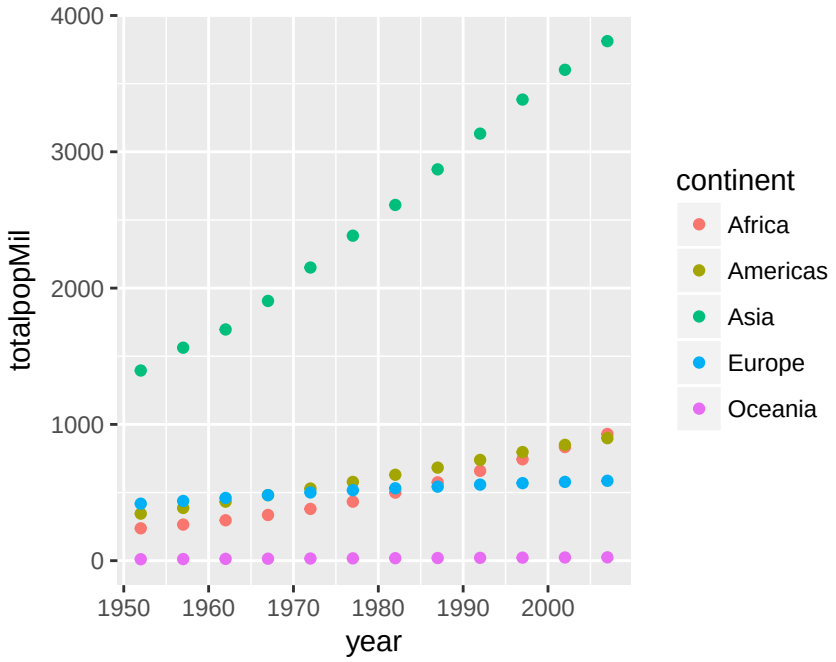


Here's a more complicated example where we additionally use color to plot each continent separately:

```
by_year_continent <- gapminder %>%
  mutate(popMil = pop / 1000000) %>%
  group_by(year, continent) %>%
  summarize(totalpopMil = sum(popMil))
by_year
```

```
# A tibble: 12 x 2
    year totalpopMil
   <int>       <dbl>
 1  1952       2407.
 2  1957       2664.
 3  1962       2900.
 4  1967       3217.
 5  1972       3577.
 6  1977       3930.
 7  1982       4289.
 8  1987       4691.
 9  1992       5111.
10  1997       5515.
11  2002       5887.
12  2007       6251.
```

```
ggplot(by_year_continent) +
  geom_point(aes(x = year, y = totalpopMil, color = continent))
```



Make sure you understand how the preceding example works before attempting the exercise.

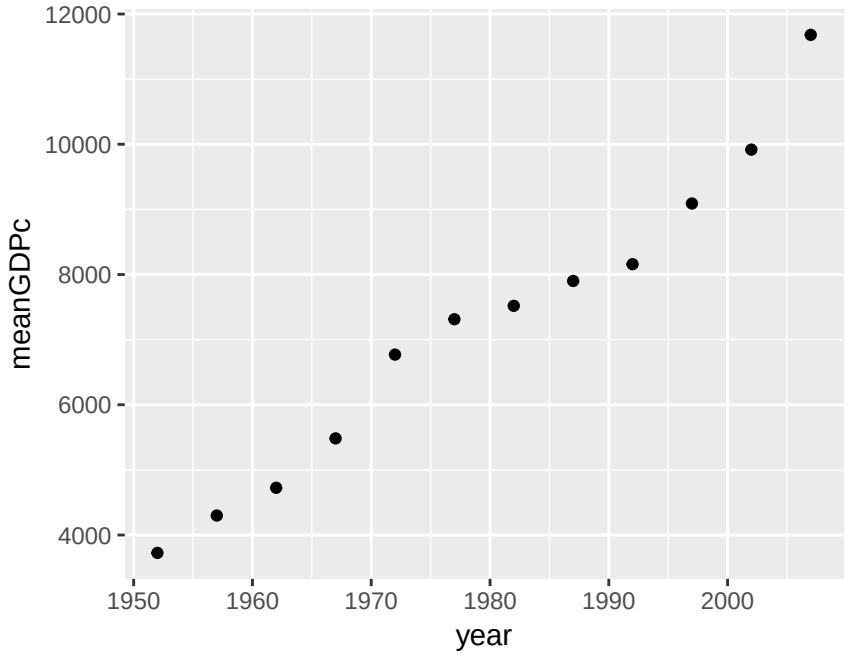
Exercise #5

1. What happens if you append `+ expand_limits(y = 0)` to the preceding `ggplot` code? Why might this be helpful in some cases?
2. Make a scatter with average GDP per capita across all countries in `gapminder` in the y-axis and `year` on the x-axis.
3. Repeat 2. broken down by continent, using color to distinguish the points. Put mean GDP per capita on the log scale.

Solution to Exercise #5

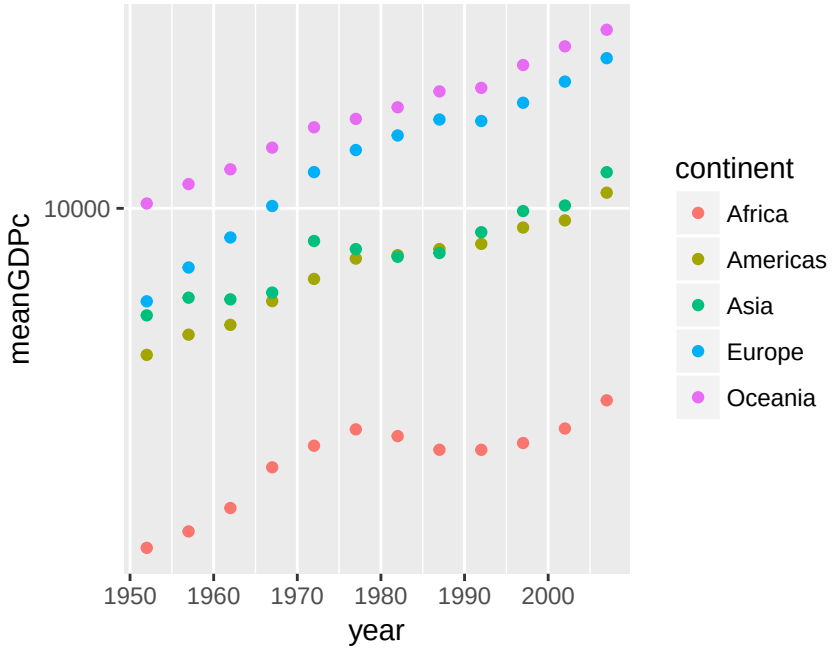
1. The function `expand_limits()` lets us tweak the limits of our x or y-axis in a `ggplot`. In this particular example `expand_limits(y = 0)` ensures that the y-axis begins at zero. Without using this command, `ggplot` will choose the y-axis on its own so that there is no "empty space" in the plot. Sometimes we may want to override this behavior.
2. Use the following:

```
by_year <- gapminder %>%
  group_by(year) %>%
  summarize(meanGDPc = mean(gdpPercap))
ggplot(by_year) +
  geom_point(aes(x = year, y = meanGDPc))
```



3. Use the following

```
by_year <- gapminder %>%
  group_by(year, continent) %>%
  summarize(meanGDPc = mean(gdpPercap))
ggplot(by_year) +
  geom_point(aes(x = year, y = meanGDPc, color = continent)) +
  scale_y_log10()
```



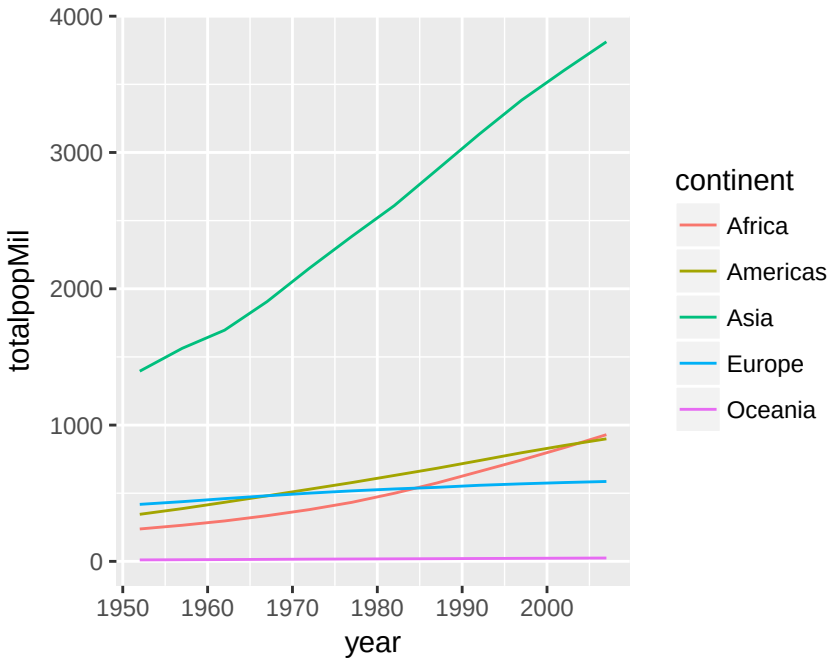
Line plots

Thus far we've only learned how to make one kind of plot with `ggplot`: a scatterplot, which we constructed using `geom_scatter()`. Sometimes we want to *connect* the dots in a scatterplot, for example when we're interested in visualizing a trend over time. The resulting plot is called a *line plot*. To make one, simply replace `geom_scatter()` with `geom_line()`. For example:

```
by_year_continent <- gapminder %>%
  mutate(popMil = pop / 1000000) %>%
  group_by(year, continent) %>%
  summarize(totalpopMil = sum(popMil))
by_year
```

```
# A tibble: 60 x 3
# Groups:   year [?]
    year continent meanGDPc
   <int> <fct>        <dbl>
 1  1952 Africa       1253.
 2  1952 Americas     4079.
 3  1952 Asia         5195.
 4  1952 Europe       5661.
 5  1952 Oceania     10298.
 6  1957 Africa       1385.
 7  1957 Americas     4616.
 8  1957 Asia         5788.
 9  1957 Europe       6963.
10  1957 Oceania     11599.
# ... with 50 more rows
```

```
ggplot(by_year_continent) +
  geom_line(aes(x = year, y = totalpopMil, color = continent))
```

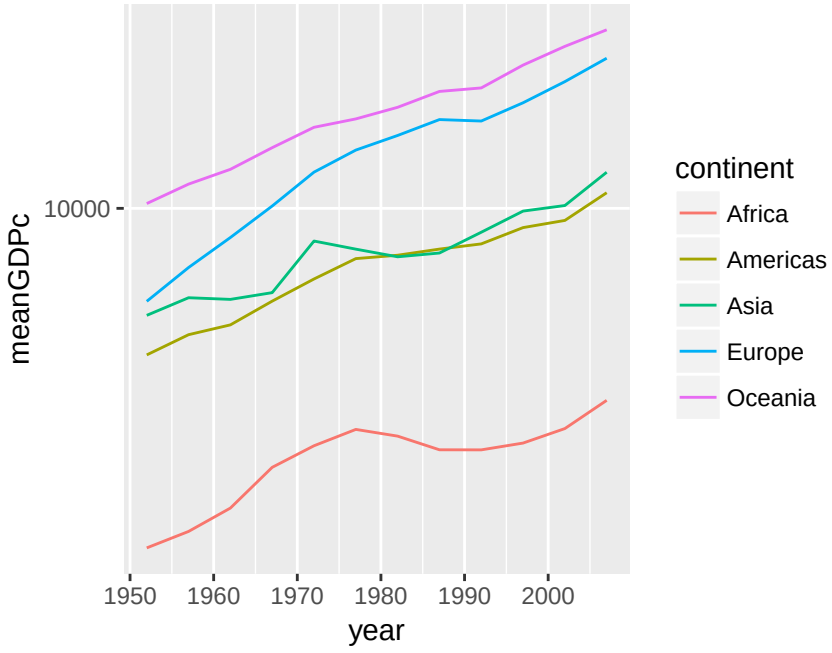


Exercise #6

Repeat exercise 5-3 with a line plot rather than a scatterplot.

Solution to Exercise #6

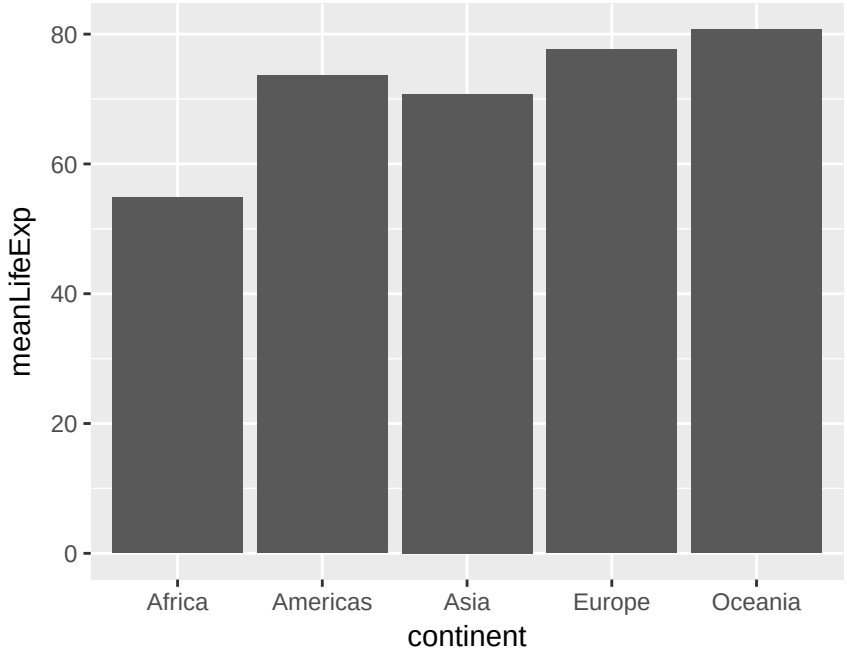
```
by_year <- gapminder %>%
  group_by(year, continent) %>%
  summarize(meanGDPc = mean(gdpPercap))
ggplot(by_year) +
  geom_line(aes(x = year, y = meanGDPc, color = continent)) +
  scale_y_log10()
```



Bar plots

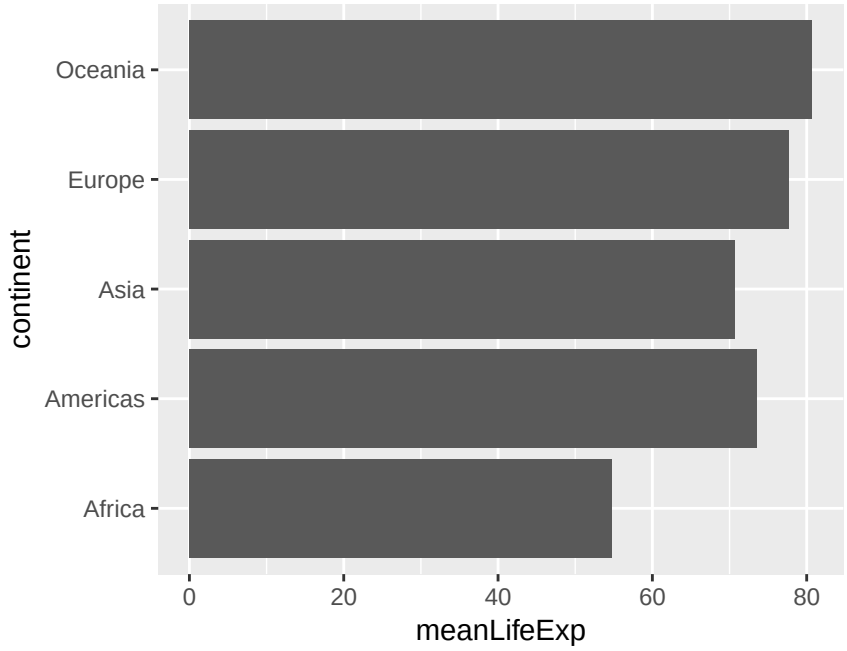
To make a bar plot, we use `geom_col()`. Note that the `x` argument of `aes` needs to be a *categorical variable* for a bar plot to make sense. Here's a simple example:

```
by_continent <- gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  summarize(meanLifeExp = mean(lifeExp))
ggplot(by_continent) +
  geom_col(aes(x = continent, y = meanLifeExp))
```



Sometimes we want to turn a bar plot, or some other kind of plot, on its side. This can be particularly helpful if the x-axis labels are very long. To do this, simply add `+ coord_flip()` to your `ggplot` command, for example:

```
ggplot(by_continent) +
  geom_col(aes(x = continent, y = meanLifeExp)) +
  coord_flip()
```

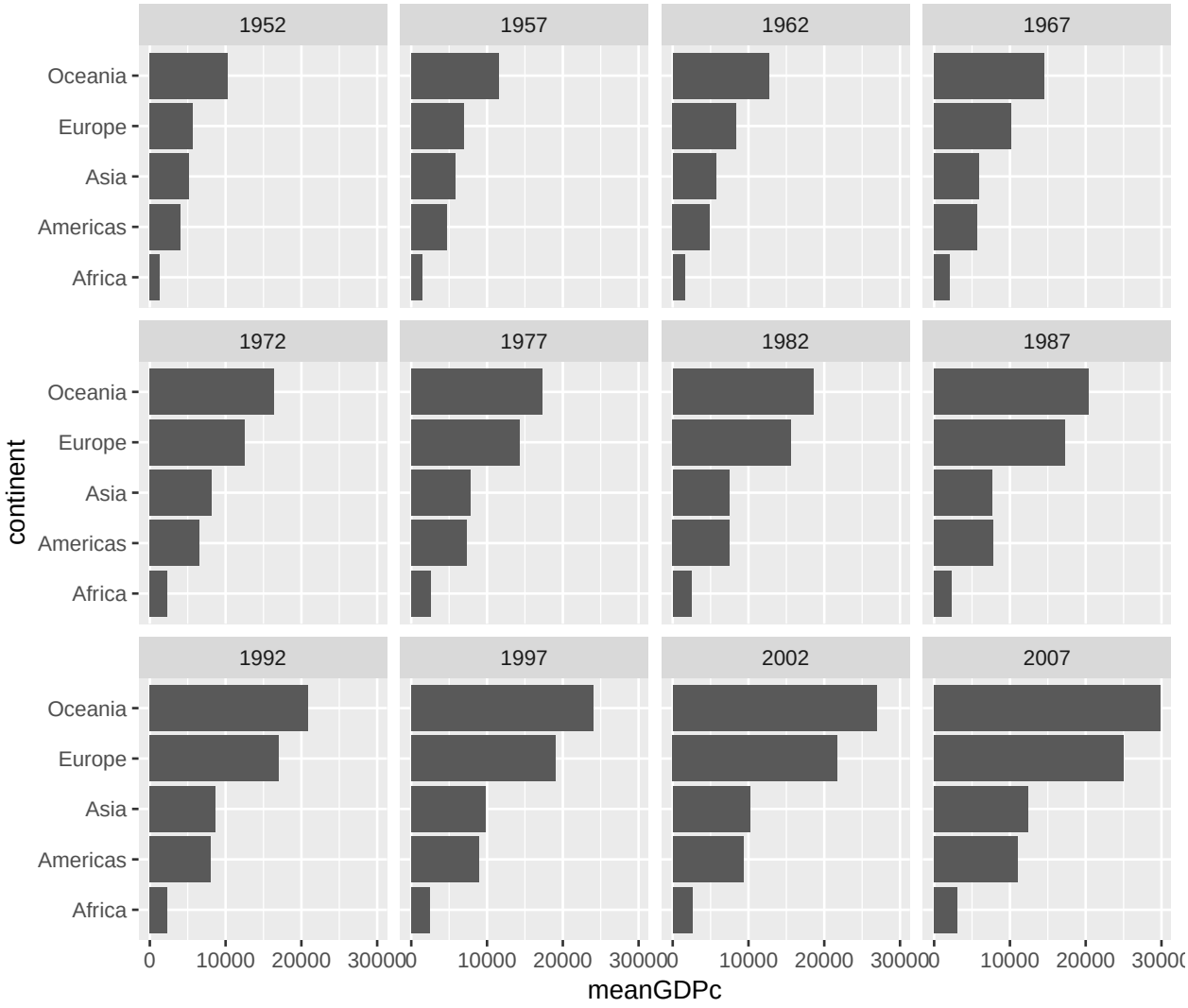


Exercise #7

Make a collection of bar plots faceted by year that compare mean GDP per capita across countries in a given year. Orient your plots so it's easy to read the continent labels.

Solution to Exercise #7

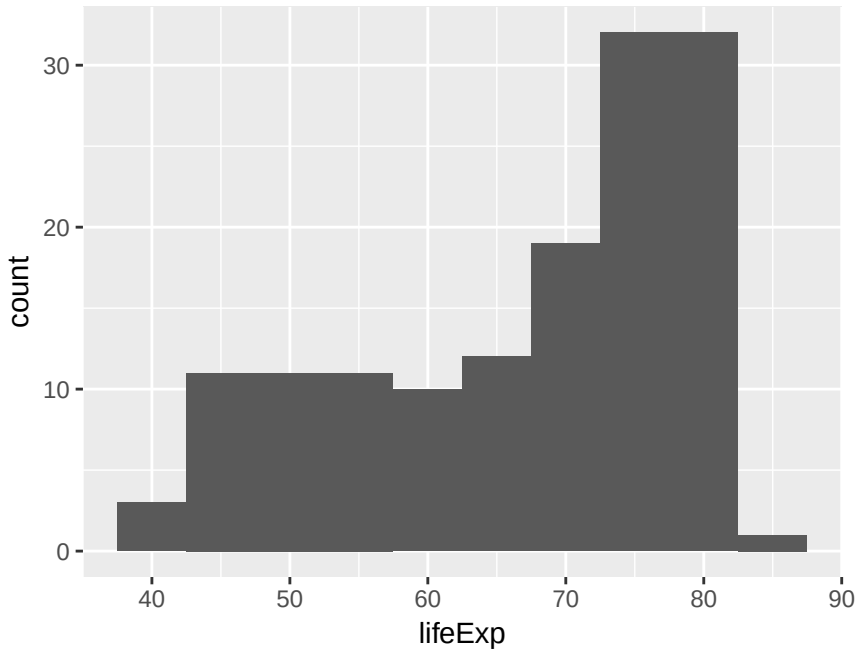
```
by_continent <- gapminder %>%
  group_by(continent) %>%
  summarize(meanGDPc = mean(gdpPercap))
ggplot(by_year) +
  geom_col(aes(x = continent, y = meanGDPc)) +
  facet_wrap(~ year) +
  coord_flip()
```



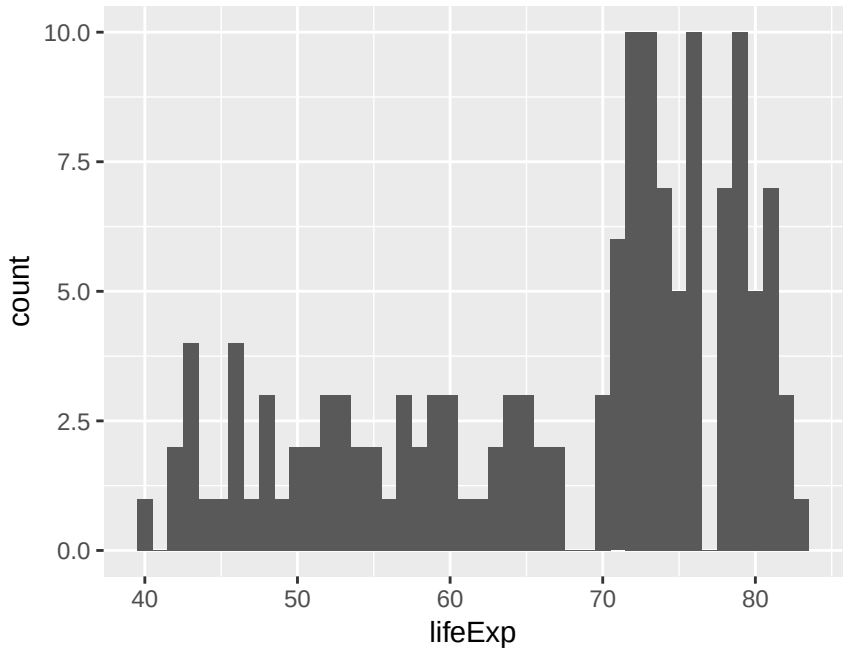
Histograms

To make a `ggplot2` histogram, we use the function `geom_histogram()`. Recall from Econ 103 that a histogram summarizes a *single* variable at a time by forming non-overlapping bins of equal width and calculating the fraction of observations in each bin. If we choose a different width for the bins, we'll get a different histogram. Here's an example of two different bin widths:

```
gapminder_2007 <- gapminder %>%
  filter(year == 2007)
ggplot(gapminder_2007) +
  geom_histogram(aes(x = lifeExp), binwidth = 5)
```




```
ggplot(gapminder_2007) +
  geom_histogram(aes(x = lifeExp), binwidth = 1)
```



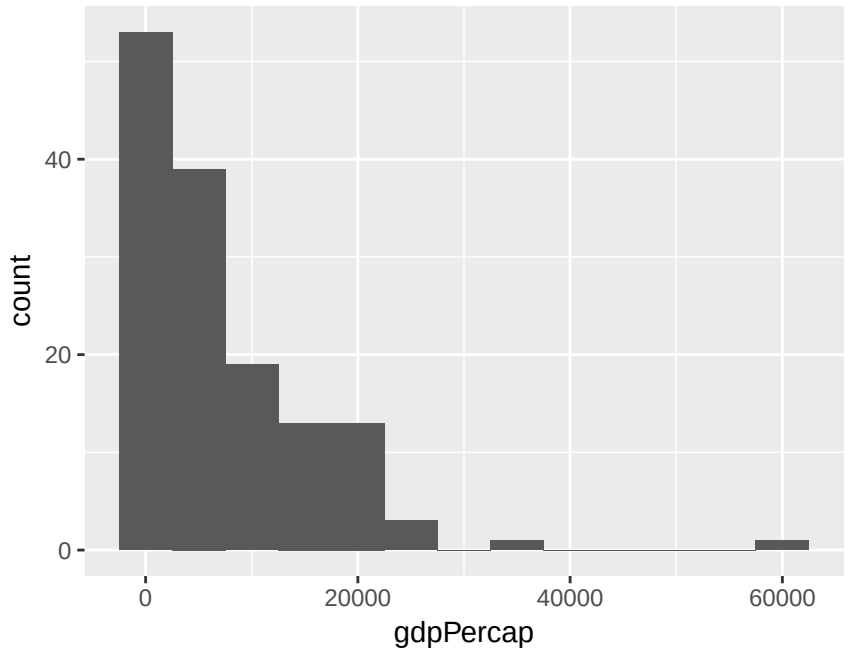
Exercise # 8

1. All of the examples we've seen that use `ggplot` *besides* histograms have involved specifying both `x` and `y` within `aes()`. Why are histograms different?
2. What happens if you don't specify a bin width in either of my two examples?
3. Make a histogram of GDP per capita in 1977. Play around with different bin widths until you find one that gives a good summary of the data.
4. Repeat 3. but put GDP per capita on the log scale.
5. Compare and contrast the two different histograms you've made.

Solution to Exercise #8

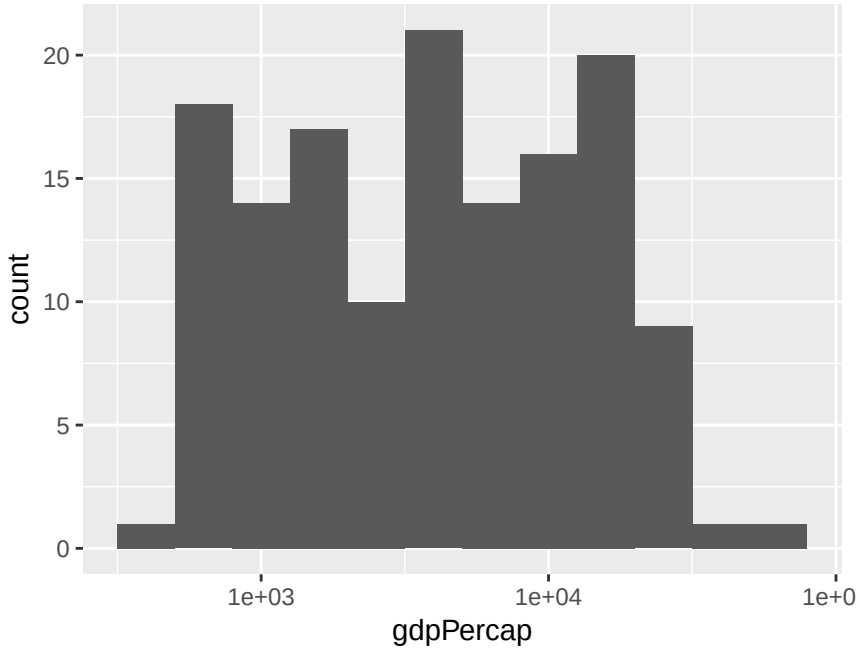
1. This is because histograms only depict a single variable while the other plots we've made show two variables at once.
2. If you don't specify a bin width, `ggplot2` will pick one for you and possibly give you a warning suggesting that you pick a better bin width manually.
3. There's no obvious *right answer* for the bin width, but here's one possibility:

```
gapminder1977 <- gapminder %>%
  filter(year == 1977)
ggplot(gapminder_1977) +
  geom_histogram(aes(x = gdpPercap), binwidth = 5000)
```



4. You'll need a much smaller bin width when using the log scale, for example:

```
ggplot(gapminder_1977) +
  geom_histogram(aes(x = gdpPercap), binwidth = 0.2) +
  scale_x_log10()
```

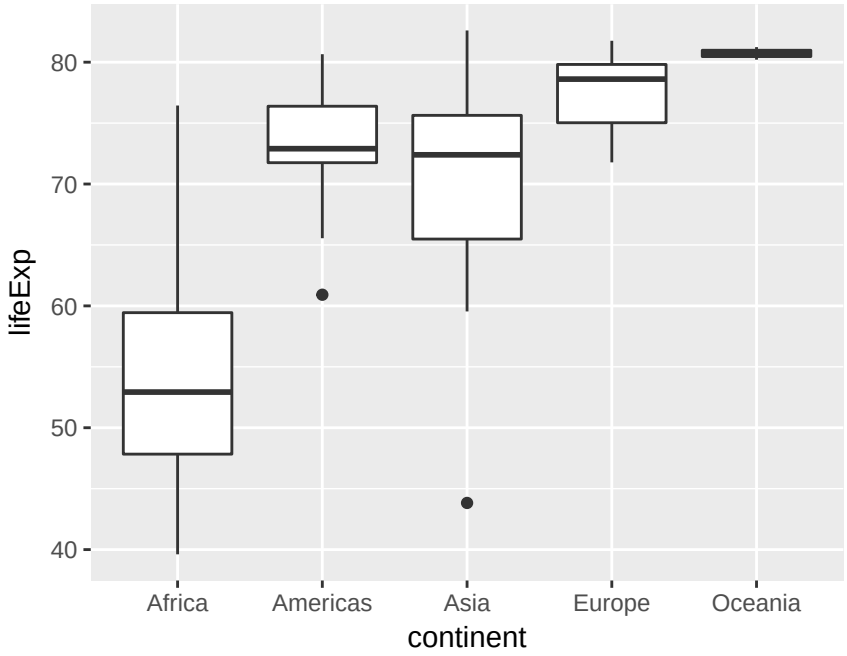


5. No right answer: it's a discussion question! But the idea is to see how taking logs gets rid of the huge positive skewness in GDP per capita.

Boxplots

The final kind of `ggplot` we'll learn how to produce is a boxplot. Recall from Econ 103 that a boxplot is a visualization of the *five-number summary* of a variable: minimum, 25th percentile, median, 75th percentile, and maximum. To make a boxplot in `ggplot` we use the function `geom_boxplot()`, for example:

```
ggplot(gapminder_2007) +
  geom_boxplot(aes(x = continent, y = lifeExp))
```



Compared to histograms, boxplots provide less detail but allow us to easily compare across groups.

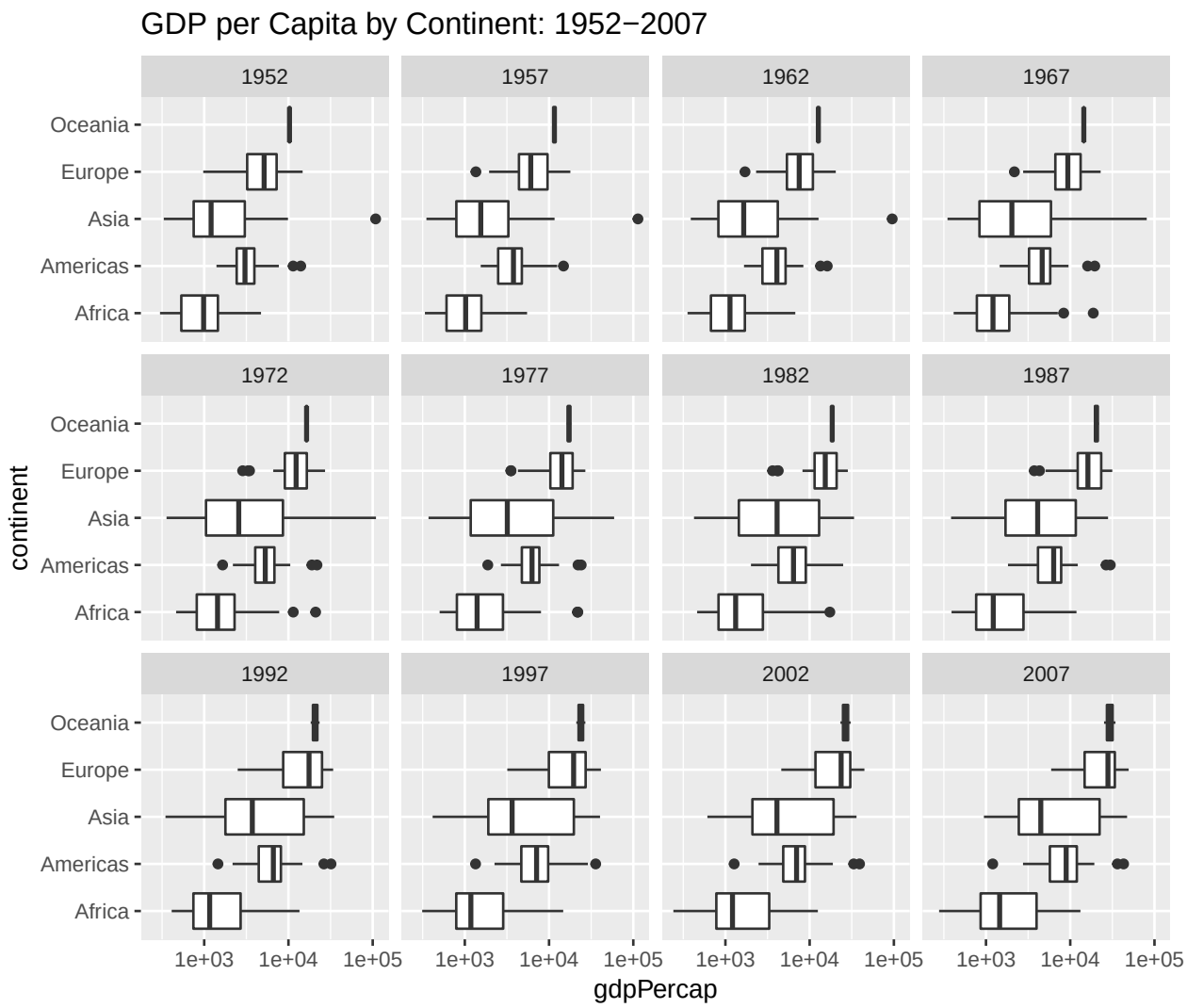
Exercise # 9

1. What is the meaning of the little "dots" that appear in the boxplot above? Use a Google search to find out what they are and how they are computed.
2. Use faceting to construct a collection of boxplots, each of which compares log GDP per capita across continents in a given year.
3. Use a Google search to find out how to add a title to a `ggplot`. Use it to add a title to the plot you created in 2.

Solution to Exercise #9

1. They are outliers: `ggplot` considers any observation that is more than 1.5 times the interquartile range away from the "box" to be an outlier, and adds a point to indicate it. Turn your boxplots sideways to make it easier to read the continent labels.
2. Use the following code:

```
ggplot(gapminder) +
  geom_boxplot(aes(x = continent, y = gdpPercap)) +
  facet_wrap(~ year) +
  scale_y_log10() +
  coord_flip() +
  ggtitle('GDP per Capita by Continent: 1952-2007')
```



3. Use `ggtitle('YOUR TITLE HERE')` as I did in my solution to 2. above.