Class 5: Data Viz with ggplot

Andrew Sue (PID: A13006809)

Graphics systems in R

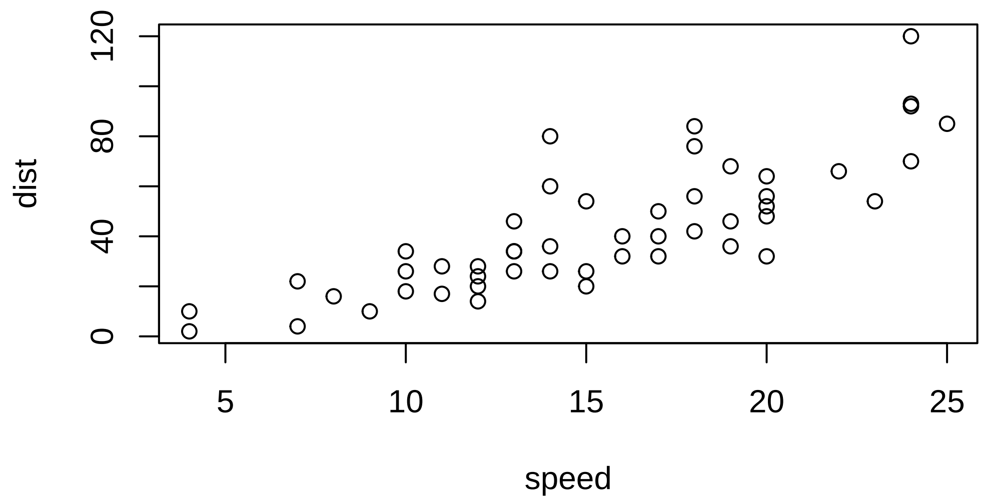
There are many graphic systems in R. This include so-called *"base R"* and those in add-on packages like **`ggplot2`**.

```r
cars
```

```
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
```

```
24      15    20
25      15    26
26      15    54
27      16    32
28      16    40
29      17    32
30      17    40
31      17    50
32      18    42
33      18    56
34      18    76
35      18    84
36      19    36
37      19    46
38      19    68
39      20    32
40      20    48
41      20    52
42      20    56
43      20    64
44      22    66
45      23    54
46      24    70
47      24    92
48      24    93
49      24   120
50      25    85
```

```r
plot(cars)
```



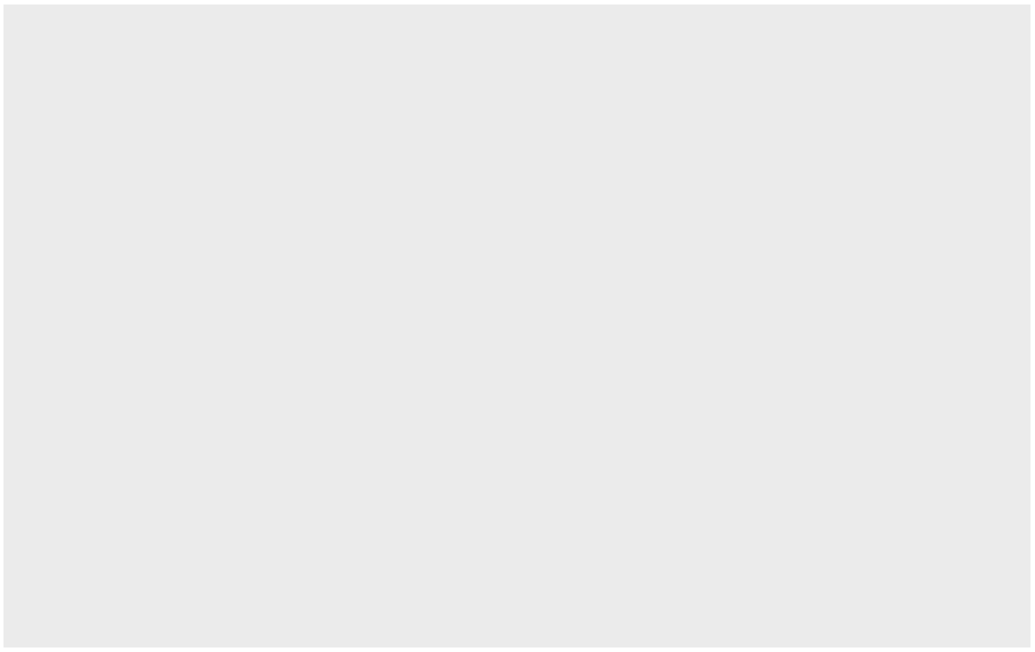
How can we make this same plot in `ggplot2`.

This is an add-on package that must be installed on my computer. This is a one-time install.

To install packages use `install.packages()`. Note: do **NOT** do this in R code chunk. Do it in the terminal.

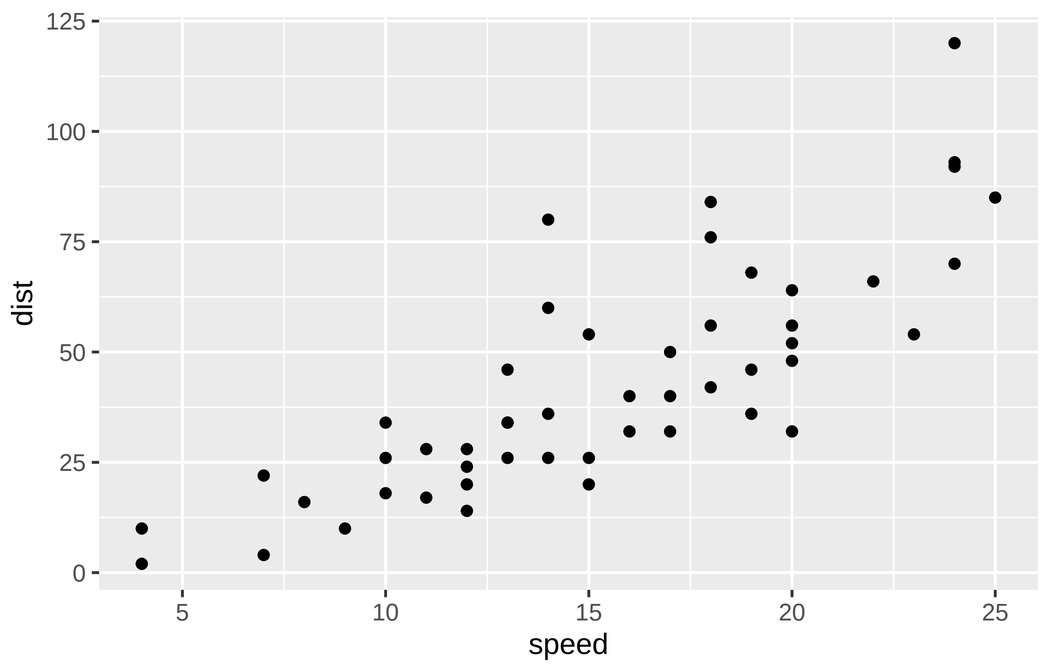
To use packages, you must load it from the library of installed packages. Use `library()`.

```r
library(ggplot2)
ggplot(cars)
```



In order to get a function graph with `ggplot2`, you must specify all 3 ‘layers’ (minimum inputs). These include **`data()`**, **`aes()`**, and **`geom_()`**.

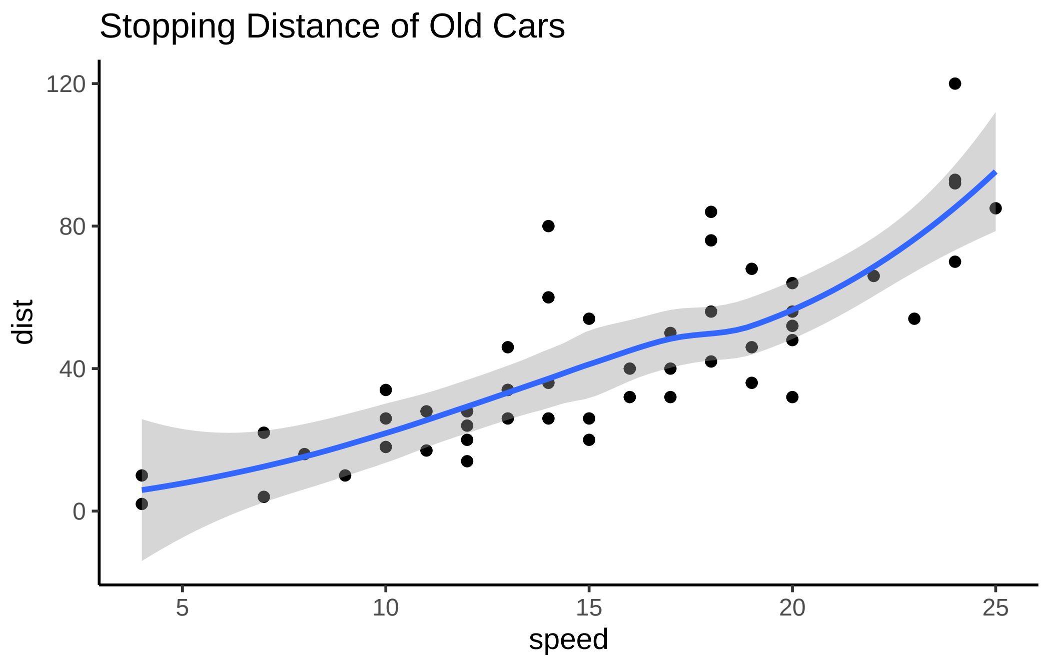
```
library(ggplot2)
ggplot(cars) +
  aes(x=speed,y=dist) +
  geom_point()
```



To make more complicated graphs with `ggplot2`, it is as simple as adding more layers to your code. This is what makes `ggplot2` powerful.

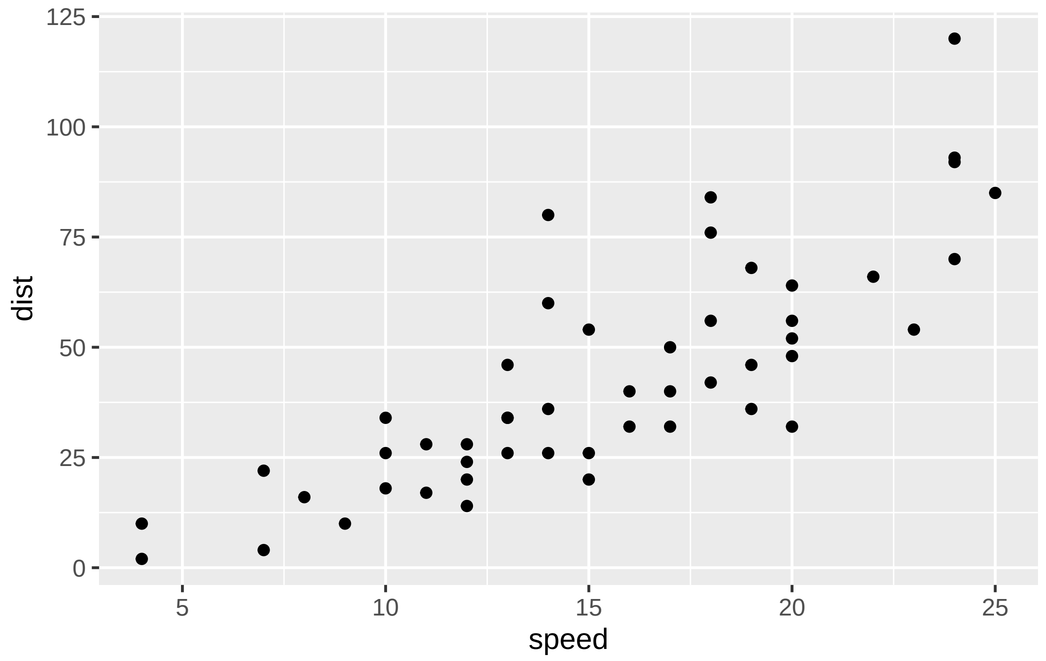
```
ggplot(cars) +
  aes(speed,dist) +
  geom_point() +
  geom_smooth() +
  labs(title= "Stopping Distance of Old Cars") +
  theme_classic()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Code can get long when writing a custom ggplot graph. It is useful to save codes as variable to clean up your space.

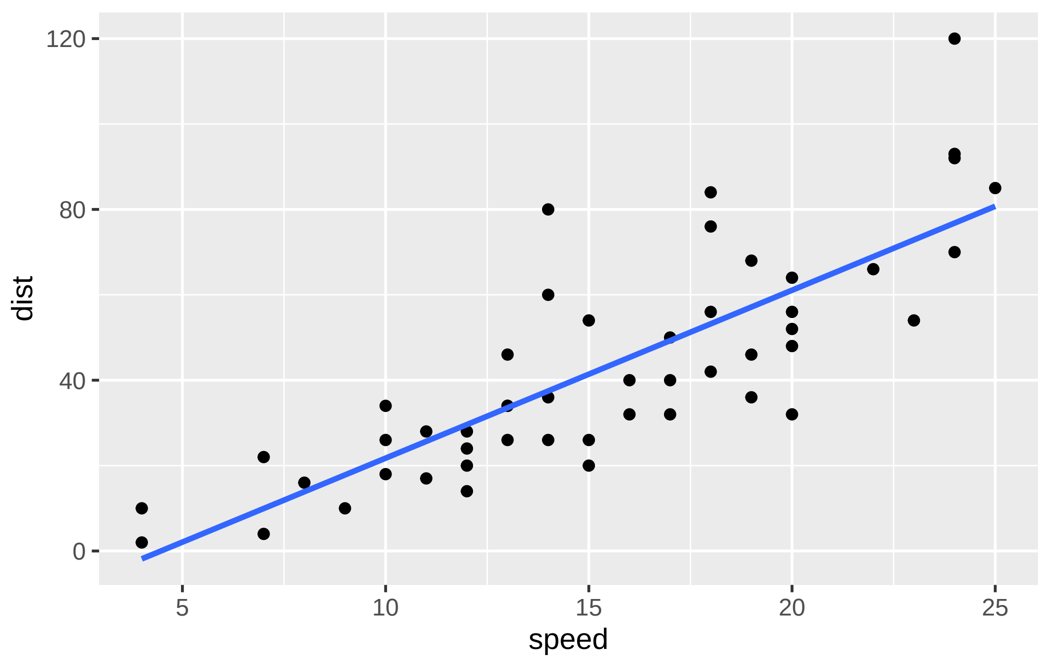
```
base_code<-ggplot(cars) +
  aes(x=speed,y=dist) +
  geom_point()
base_code
```



Customize the graph now to add a trendline and get rid of the standard error (se).

```
p<- base_code + geom_smooth(method = "lm", se = FALSE)
p
```

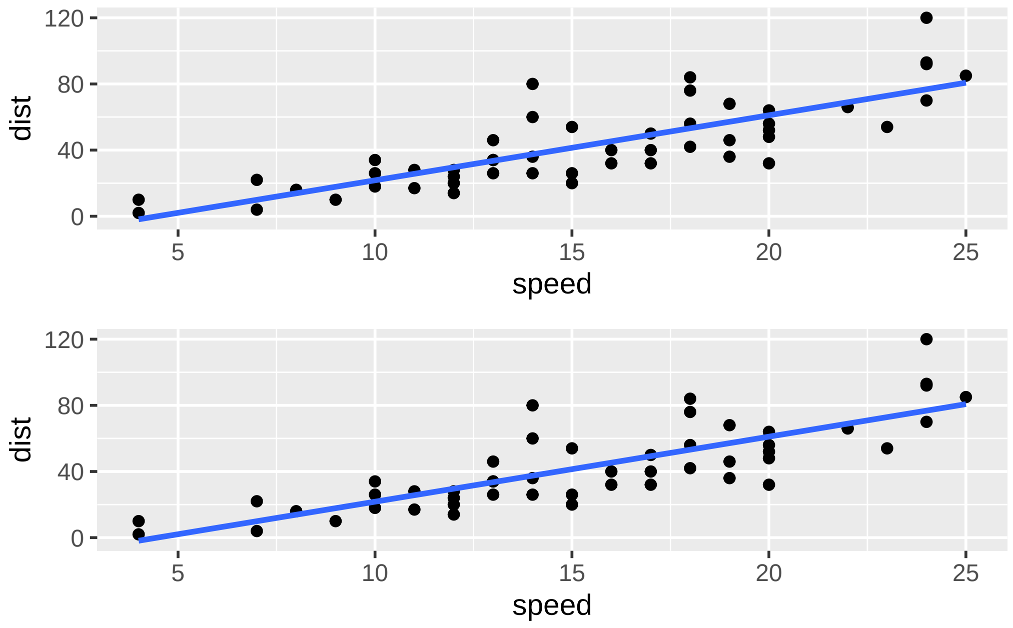
```
`geom_smooth()` using formula = 'y ~ x'
```



To make panel figures, use **`patchwork`** library to stitch graphs together. Note how you can change they way they are patched using slashes.

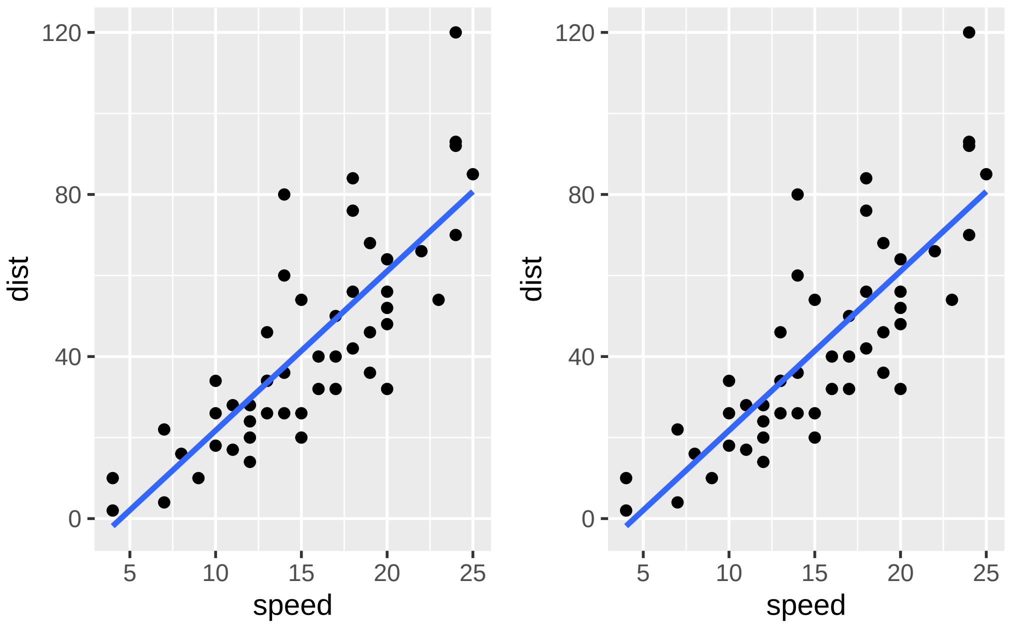
```r
library(patchwork)
p/p
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```



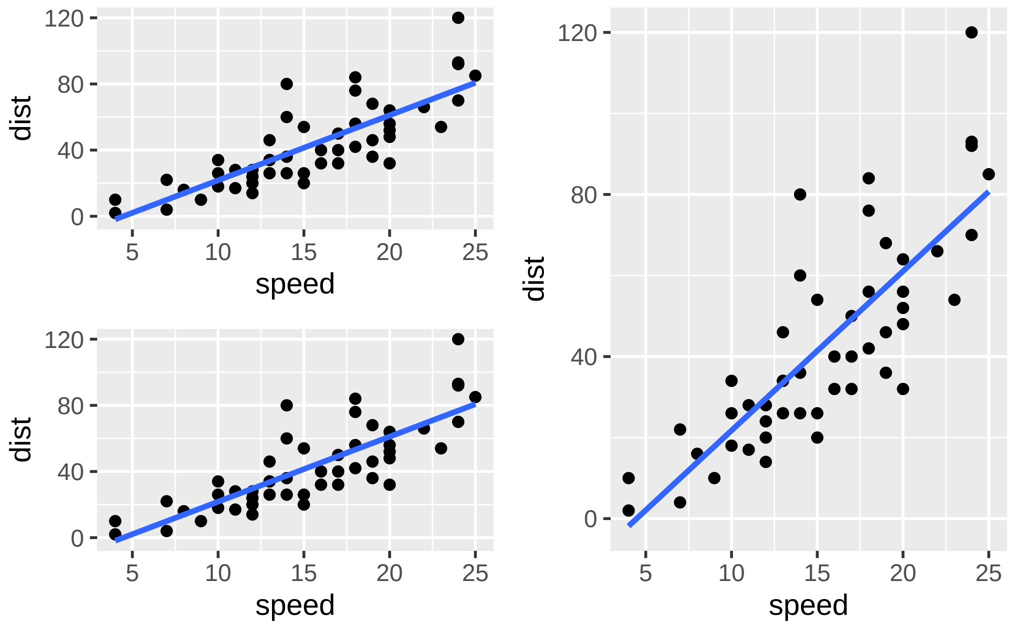
```
p|p
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```



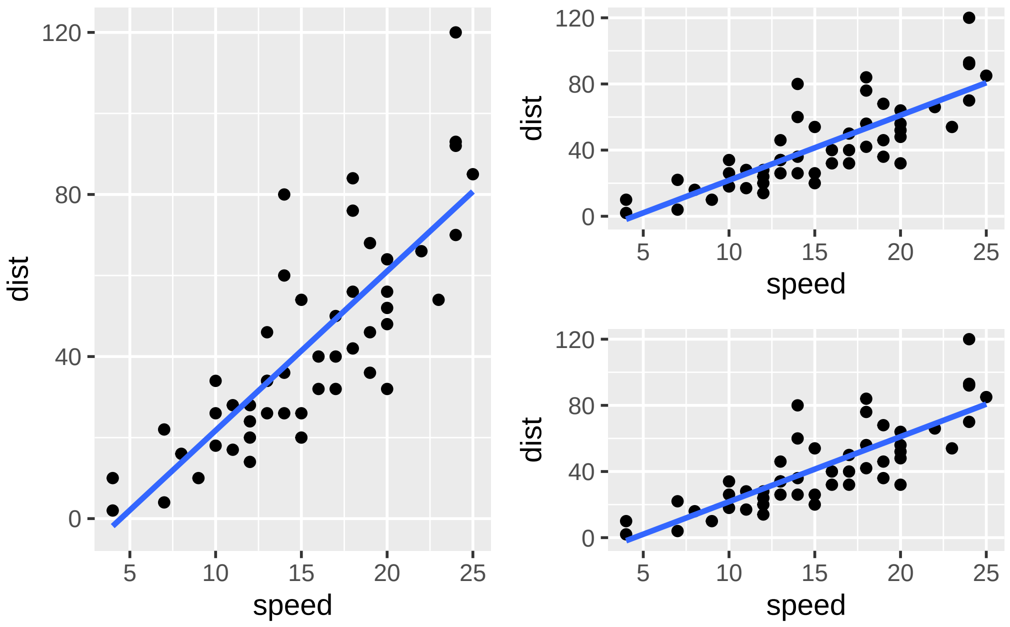
```
p/p|p
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```



```
p|p/p
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```



Adding more plots through `aes()`

Lets move forward with a more complicated dataset.

```r
url <- "https://bioboot.github.io/bimm143_S20/class-material/up_down_expression.txt"
genes <- read.delim(url)
head(genes)
```

```
        Gene Condition1 Condition2      State
1      A4GNT -3.6808610 -3.4401355 unchanging
2       AAAS  4.5479580  4.3864126 unchanging
3      AASDH  3.7190695  3.4787276 unchanging
4       AATF  5.0784720  5.0151916 unchanging
5       AATK  0.4711421  0.5598642 unchanging
6 AB015752.4 -3.6808610 -3.5921390 unchanging
```

Q. Use the nrow() function to find out how many genes are in this dataset. What is your answer?

```r
nrow(genes)
```

```
[1] 5196
```

Q. Use the colnames() function and the ncol() function on the genes data frame to find out what the column names are (we will need these later) and how many columns there are. How many columns did you find?

```
ncol(genes)
```

```
[1] 4
```

```
colnames(genes)
```

```
[1] "Gene"       "Condition1" "Condition2" "State"
```

Q. Use the table() function on the State column of this data.frame to find out how many 'up' regulated genes there are. What is your answer?

```
table(genes$State)
```

```

      down unchanging         up 
        72       4997        127 
```

Q. Using your values above and 2 significant figures. What fraction of total genes is up-regulated in this dataset?

```
round(table(genes$State)/nrow(genes) *100,2)
```

```

      down unchanging         up 
      1.39      96.17       2.44 
```

We can make a first basic scatter plot of this dataset by following the same recipe we have already seen, namely:

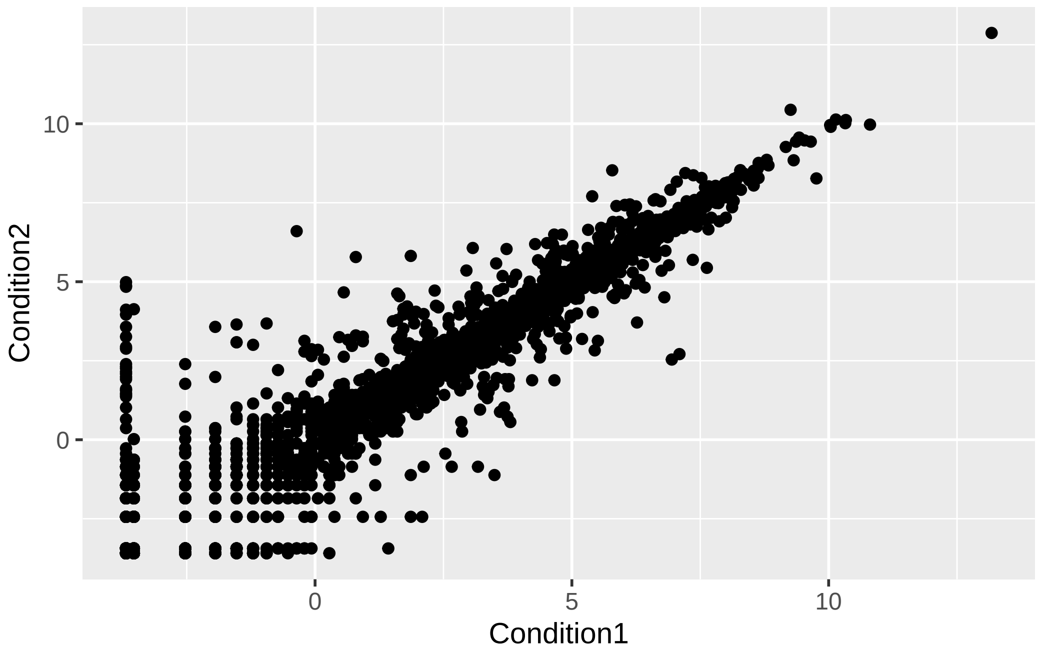
Pass the genes data.frame as input to the ggplot() function.

Then use the aes() function to set the x and y aesthetic mappings to the Condition1 and Condition2 columns.

Finally add a geom_point() layer to add points to the plot.

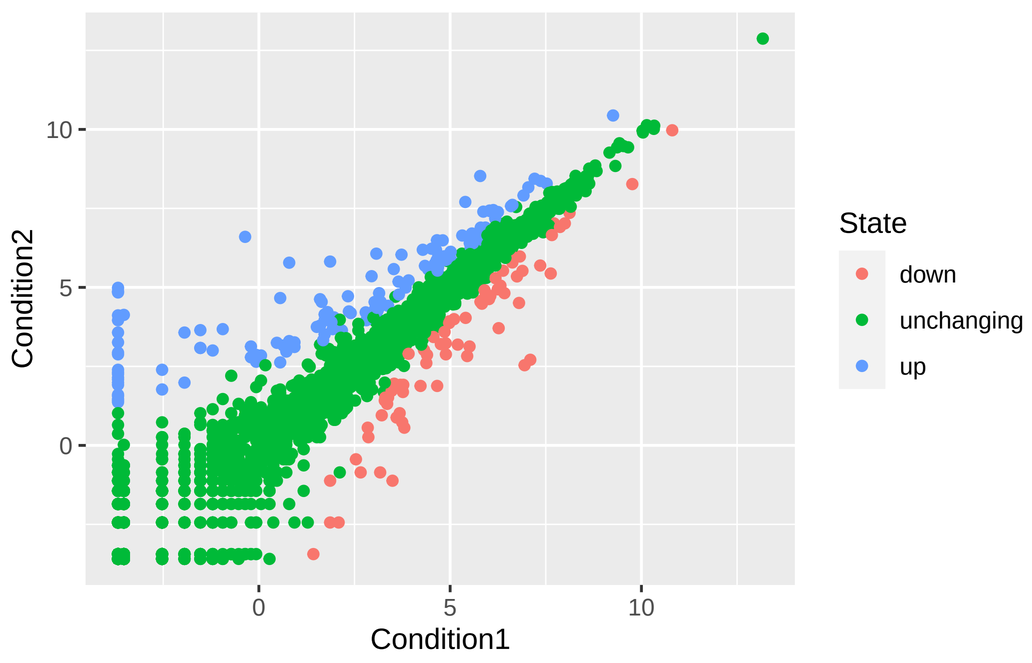
Don't forget to add layers step-wise with the + operator at the end of each line.

```
ggplot(genes) +
  aes(Condition1,Condition2) +
  geom_point()
```



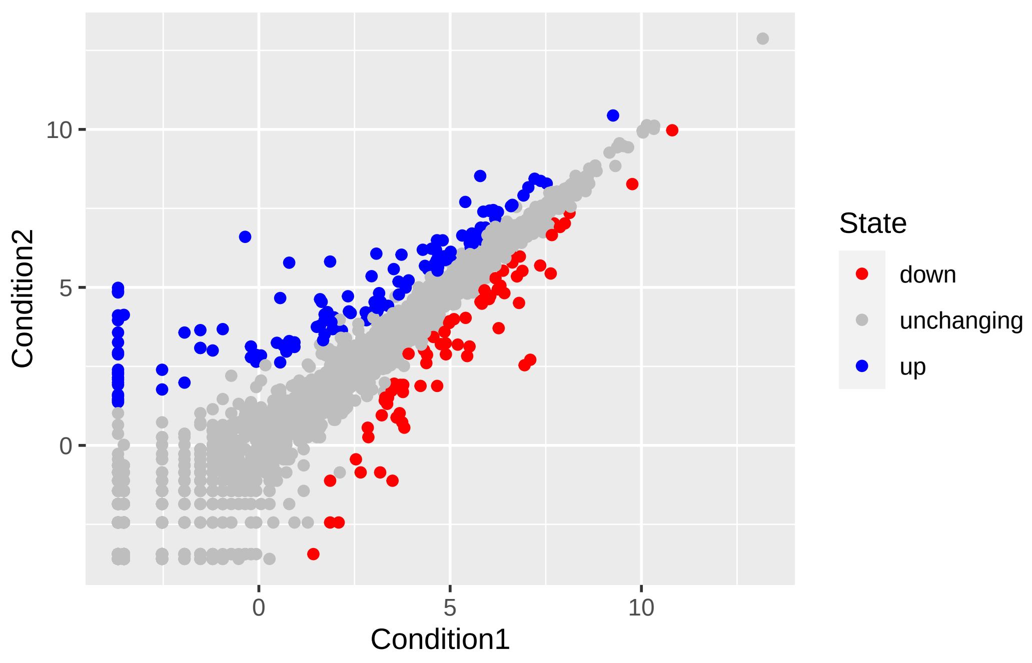
There is extra information in this dataset, namely the State column, which tells us whether the difference in expression values between conditions is statistically significant. Let's map this column to point color:

```
p<- ggplot(genes) +
  aes(Condition1,Condition2, col= State, name = Gene) +
  geom_point()
p
```



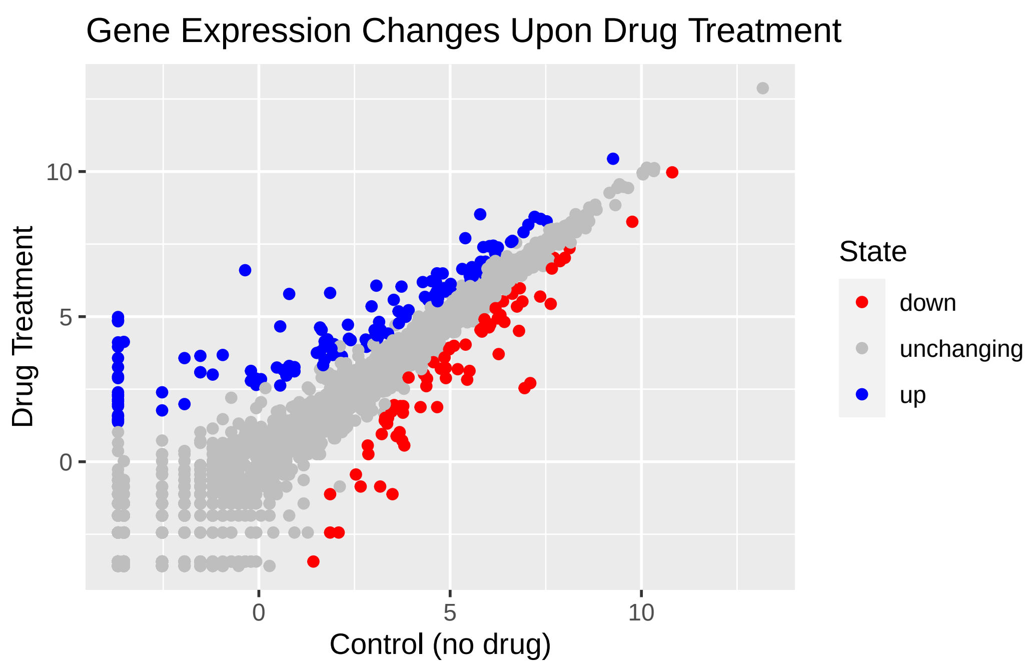
I am not a big fan of these default colors so let's change them up by adding another layer to explicitly specifcy our color scale. Note how we saved our previous plot as the object p and can use it now to add more layers:

```
p + scale_colour_manual(values=c("red","grey","blue"))
```



Q. Nice, now add some plot annotations to the p object with the labs() function so your plot looks like the following:

```
p2 <- p + scale_colour_manual(values=c("red","grey","blue")) +
  labs(x = "Control (no drug)", y = "Drug Treatment", title = "Gene Expression Changes Upo
p2
```



Lets make the graphs more fun and interactive (**very useful!**) Use the package `plotly`. We also are able to hide the code from being printed in our report by using `#/ message: false`. Old school way is using `echo= false` within the {r} bracket.

```
library(plotly)
```

```
#ggplotly(p2)
```

Going Further

The gapminder dataset contains economic and demographic data about various countries since 1952. This dataset features in your DataCamp course for this week and you can find out more about the portion we are using here.

The data itself is available as either a tab-delimited file online, or via the gapmider package. You can use whichever method you are more comfortable with to obtain the dataset. I show both below:

install.packages("gapminder") library(gapminder) Or read the TSV file from online:

```
# File location online
url <- "https://raw.githubusercontent.com/jennybc/gapminder/master/inst/extdata/gapminder.

gapminder <- read.delim(url)
```

This dataset covers many years and many countries. Before we make some plots we will use some dplyr code to focus in on a single year. You can install the dplyr package with the command install.packages("dplyr"). We will learn more about dplyr in next weeks class. For now, feel free to just copy and paste the following line that takes gapminder data frame and filters to contain only the rows with a year value of 2007.

```
library(dplyr)
```

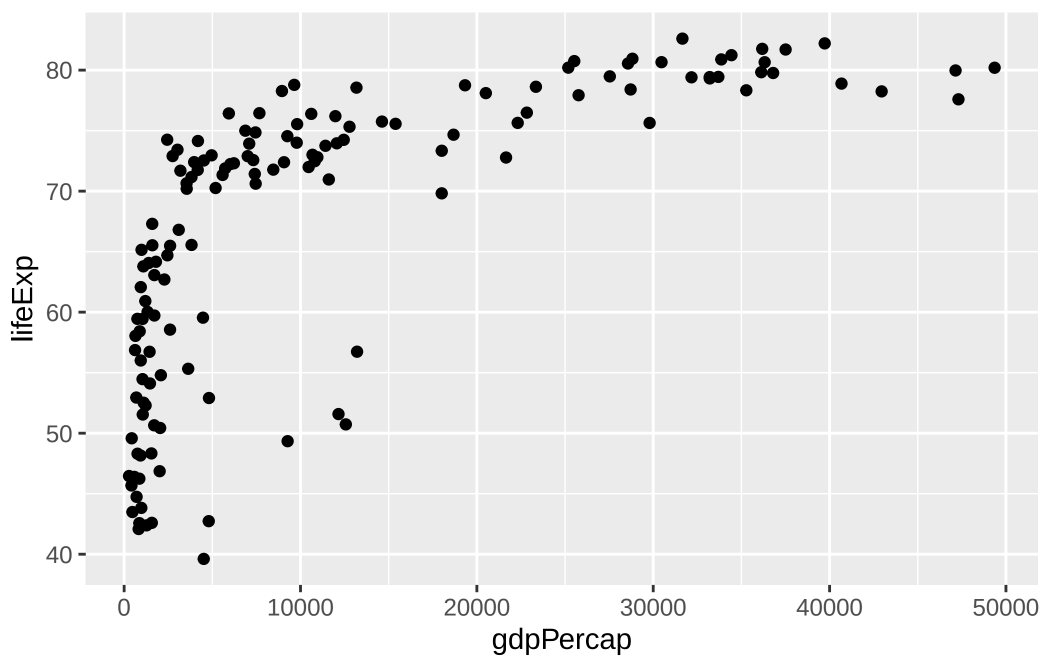
```


Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

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

Let's consider the gapminder_2007 dataset which contains the variables GDP per capita gdpPercap and life expectancy lifeExp for 142 countries in the year 2007

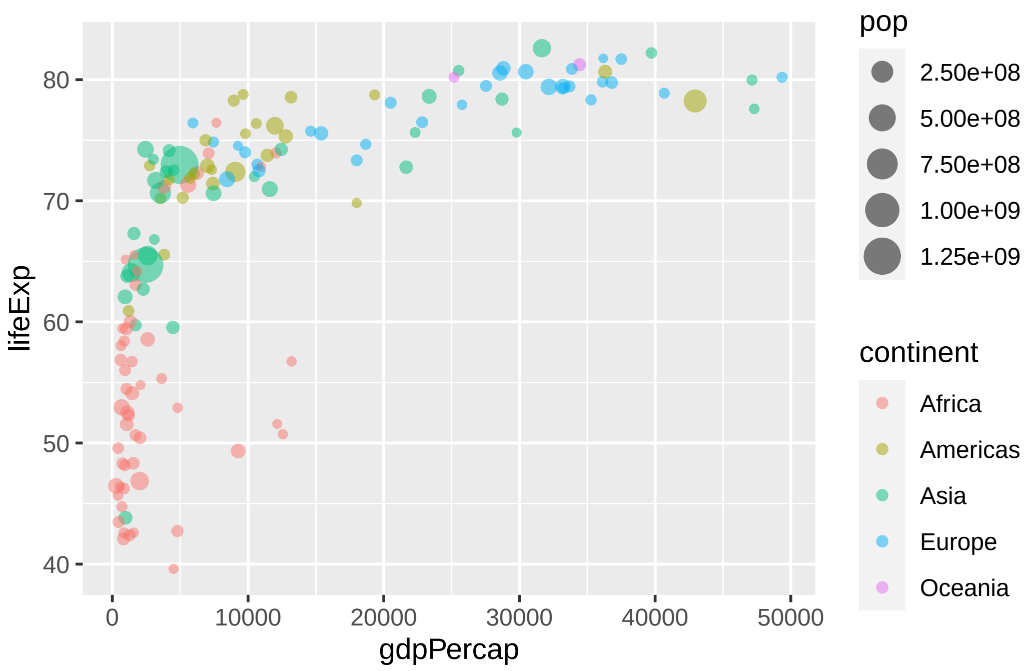
Q. Complete the code below to produce a first basic scater plot of this gapminder_2007 dataset:

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



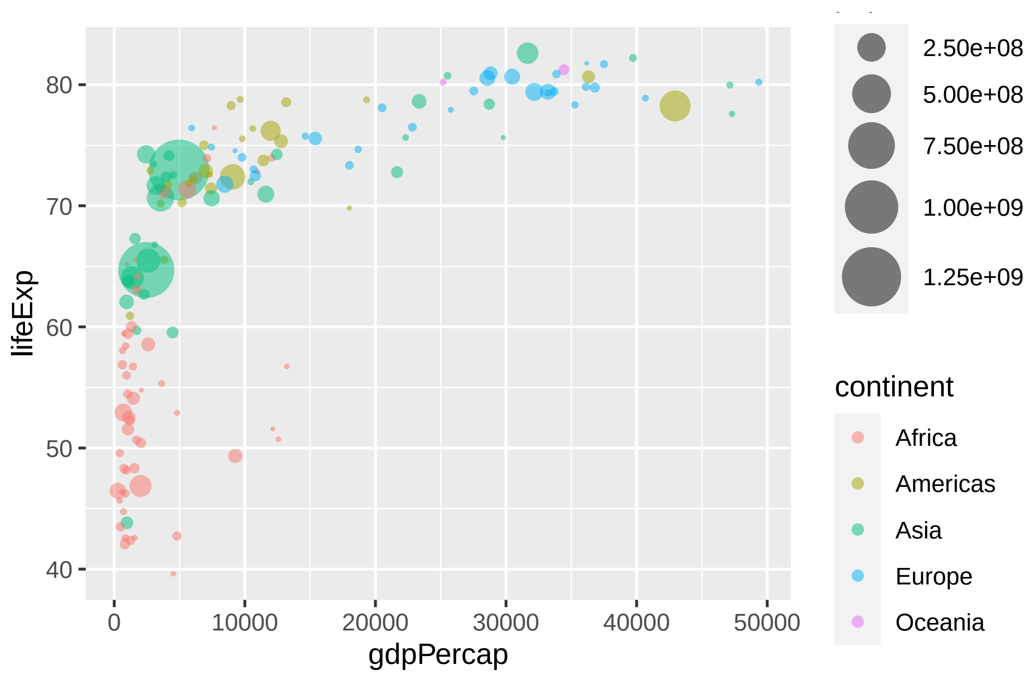
Now add layers and make it cool.

```
ggplot(gapminder_2007) +
  aes(x=gdpPercap, y=lifeExp, color= continent, size=pop) +
  geom_point(alpha=0.5)
```



However, if you look closely we see that the point sizes in the plot above do not clearly reflect the population differences in each country. If we compare the point size representing a population of 250 million people with the one displaying 750 million, we can see, that their sizes are not proportional. Instead, the point sizes are binned by default. To reflect the actual population differences by the point size we can use the scale_size_area() function instead. The scaling information can be added like any other ggplot object with the + operator:

```
ggplot(gapminder_2007) +
  aes(x=gdpPercap, y=lifeExp, color= continent, size=pop) +
  geom_point(alpha=0.5) +
  scale_size_area(max_size = 10)
```

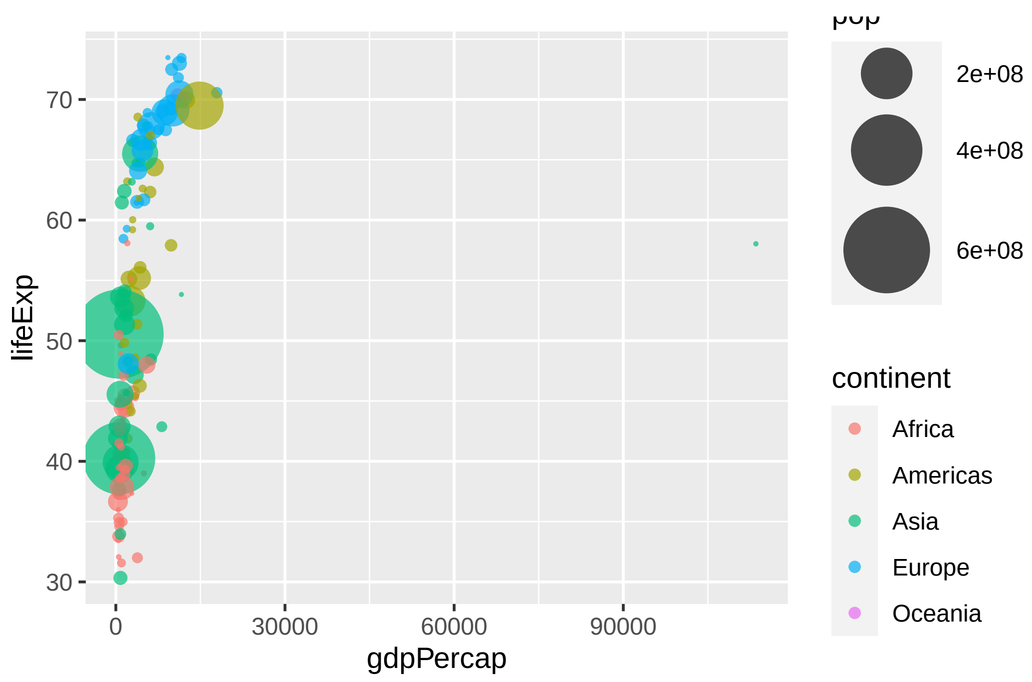


Q. Can you adapt the code you have learned thus far to reproduce our gapminder scatter plot for the year 1957? What do you notice about this plot is it easy to compare with the one for 2007?

Steps to produce your 1957 plot should include:

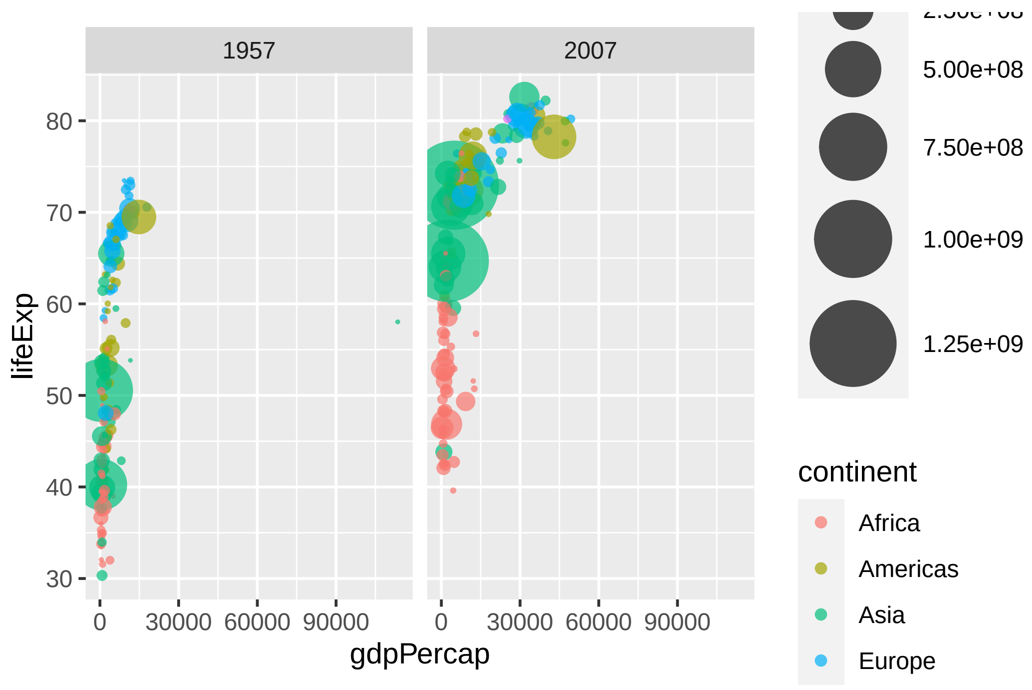
Use dplyr to filter the gapmider dataset to include only the year 1957 (check above for how we did this for 2007). Save your result as gapminder_1957. Use the ggplot() function and specify the gapminder_1957 dataset as input Add a geom_point() layer to the plot and create a scatter plot showing the GDP per capita gdpPercap on the x-axis and the life expectancy lifeExp on the y-axis Use the color aesthetic to indicate each continent by a different color Use the size aesthetic to adjust the point size by the population pop Use scale_size_area() so that the point sizes reflect the actual population differences and set the max_size of each point to 15 -Set the opacity/transparency of each point to 70% using the alpha=0.7 parameter

```
gapminder_1957<- gapminder %>% filter(year==1957)
ggplot(gapminder_1957) +
  aes(gdpPercap,lifeExp, color = continent, size = pop) +
  geom_point(alpha= 0.7) +
  scale_size_area(max_size = 15)
```



Q. Do the same steps above but include 1957 and 2007 in your input dataset for ggplot(). You should now include the layer `facet_wrap(~year)` to produce the following plot:

```
gapminder_1957v2007<- gapminder %>% filter(year == 1957 | year ==2007)
ggplot(gapminder_1957v2007) +
  aes(gdpPercap,lifeExp, color = continent, size = pop) +
  geom_point(alpha= 0.7) +
  scale_size_area(max_size = 15) +
  facet_wrap(~year)
```



Lets get into animations and extensions

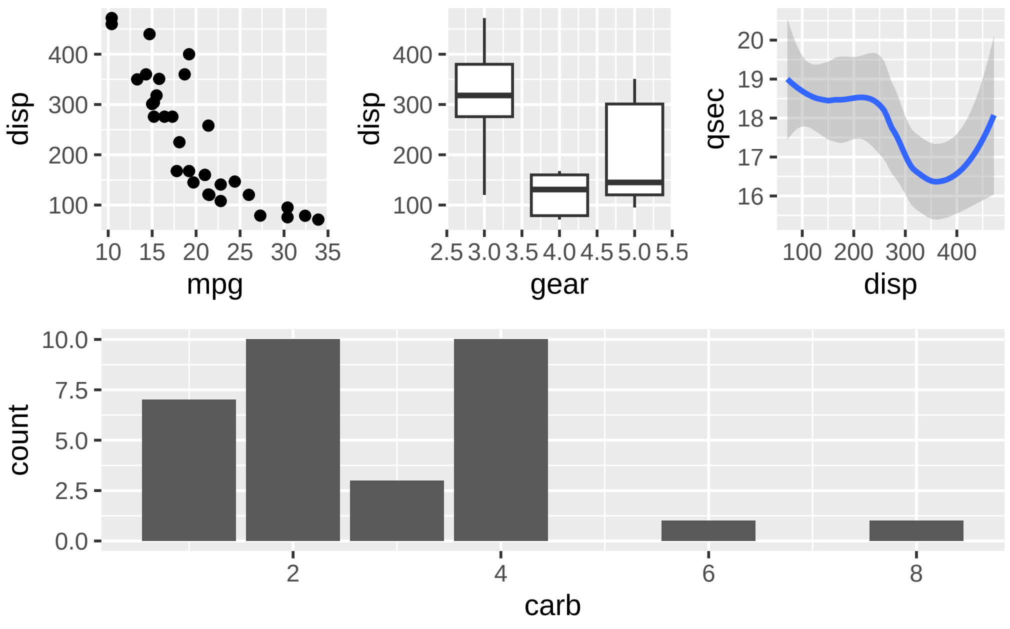
```
library(gapminder)
library(gganimate)
```

```
# Setup nice regular ggplot of the gapminder data
# ggplot(gapminder, aes(gdpPercap, lifeExp, size = pop, colour = country)) +
#   geom_point(alpha = 0.7, show.legend = FALSE) +
#   scale_colour_manual(values = country_colors) +
#   scale_size(range = c(2, 12)) +
#   scale_x_log10() +
#   # Facet by continent
#   facet_wrap(~continent) +
#   # Here comes the gganimate specific bits
#   labs(title = 'Year: {frame_time}', x = 'GDP per capita', y = 'life expectancy') +
#   transition_time(year) +
#   shadow_wake(wake_length = 0.1, alpha = FALSE)
```

Combining plots

```
# Setup some example plots
p1 <- ggplot(mtcars) + geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) + geom_smooth(aes(disp, qsec))
p4 <- ggplot(mtcars) + geom_bar(aes(carb))

# Use patchwork to combine them here:
(p1 | p2 | p3) /
      p4
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```


disp
mpg
disp
gear
qsec
disp
count
carb