Lab 4: Visualization

Mark Huber

Summary

In this lab you will learn how to create many of the common visualizations using the `ggplot2` package.

Setup

First let's make sure that the `ggplot2` package has been installed. Try

```
library(ggplot2)
```

If you receive an error when you try this command, you might have to install the package using

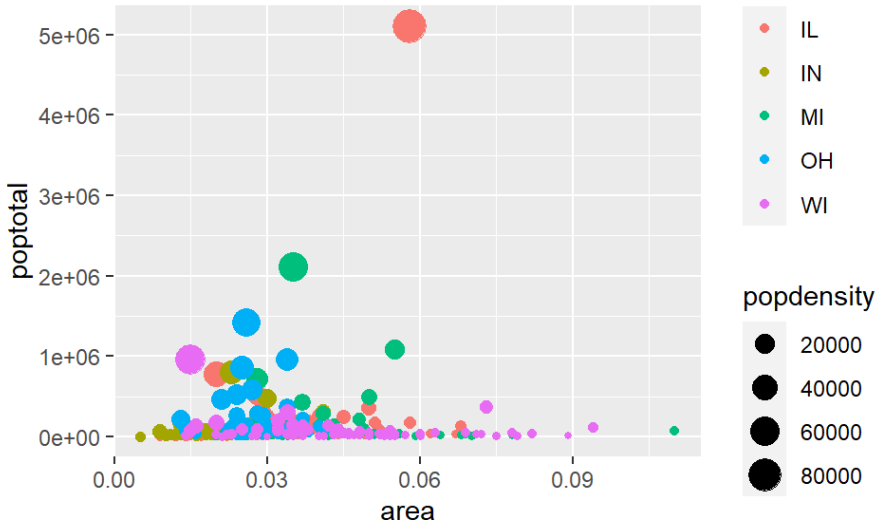
```
install.packages("ggplot2")
```

Once you have completed the installation, try the `library(ggplot2)` command again.

Scatterplots

Our code for graphs begins with a call to the `ggplot` function. This sets up the canvas upon which we shall create our graphs. Next we use various `geom_` functions to say how the data is to be presented. The plot created will be assigned to the variable `g1`, so that it can be expanded upon later. The `midwest` dataset is part of the ggplot2 package.

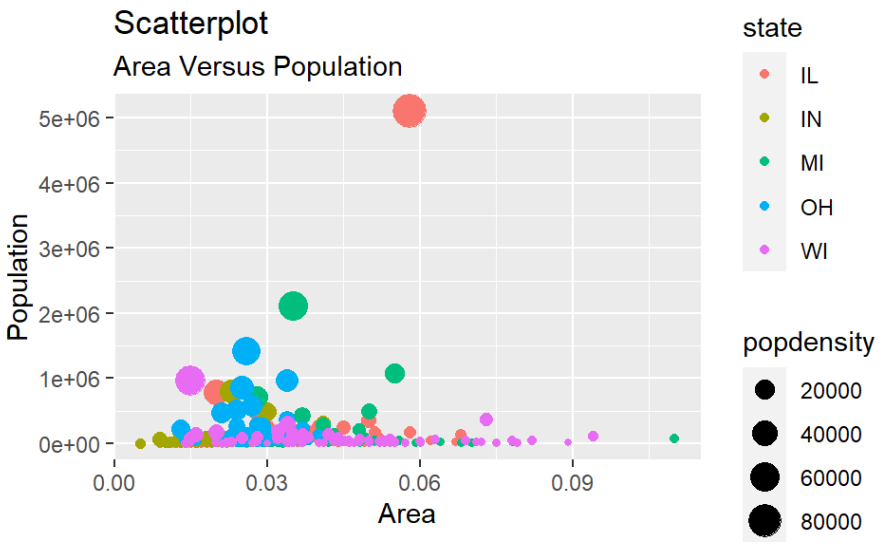
```
g1 <-
  midwest |>
  ggplot(aes(x = area, y = poptotal)) +
  geom_point(aes(col = state, size = popdensity))
g1
```



Changing labels

There are many ways to change the x and y labels on the plot. One can use the `labs` function, which transforms the plot and allows for changing everything from the axis labels to the title.

```
g2 <- g1 +
  labs(subtitle = "Area Versus Population",
       y = "Population",
       x = "Area",
       title = "Scatterplot"
  )
g2
```



Problem 1

Create a plot that has the same data and geom as `g1`, but which has the title `"County Populations"`. Assign this graphic to variable `answer01`.

Problem 2

Create a plot that has the same data and geom as `g1`, but the x-axis is labeled “Area of County” and the y-axis is labeled “Population count”. Assign this graphic to variable `answer02`.

Problem 3

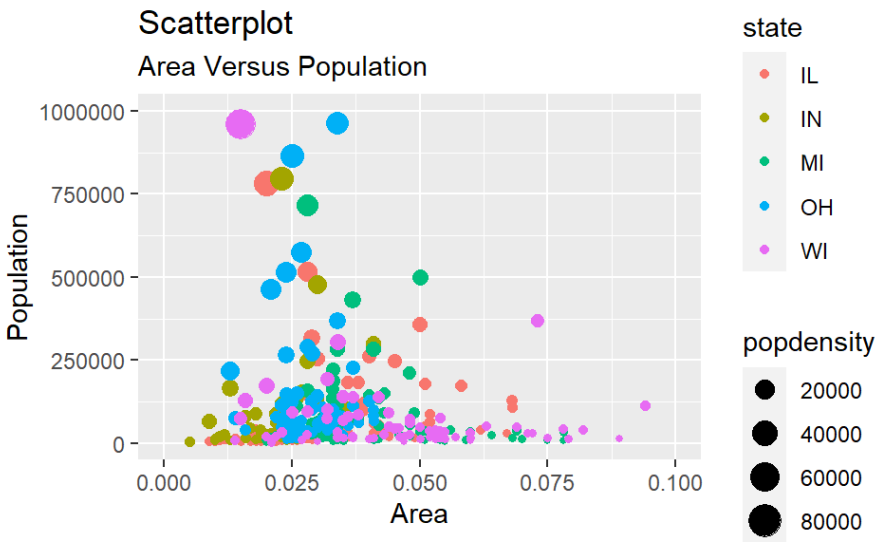
Looking at the parameters of labs using `?labs`, you will find that there are several more things that you can change. Create a plot that has the same data and geom as `g1`, but with the caption “Source: midwest dataset in R”. Assign this graphic to variable `answer03`.

Changing the limits of the plot

Now let's change the y-axis so that it only covers values up to 1,000,000, and the x-axis up to 0.1.

```
g3 <- g2 +
  xlim(c(0, 0.1)) +
  ylim(c(0, 10^6))
g3
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```



Problem 4

How many data points were removed from the plot by the restrictions on x and y? Assign this integer to `answer04`.

Highlighting selected values

Suppose that we wanted to circle some of the values in this plot that have high populations, say greater than 800,000. The `filter` function from the `dplyr` package can be used to find these points, and then the geom_encircle function from the ggalt package can be used to to draw the circle on the graph.

First let's load in our libraries (remember that if they are not already installed on your system, you will need to use the `install.packages` command to do that first.)

```
library(dplyr)
library(ggalt)
```

Okay, now filter out those points with a high population.

```
midwest_filter <-
  midwest |>
  filter(poptotal > 8*10^5)
```

Problem 5

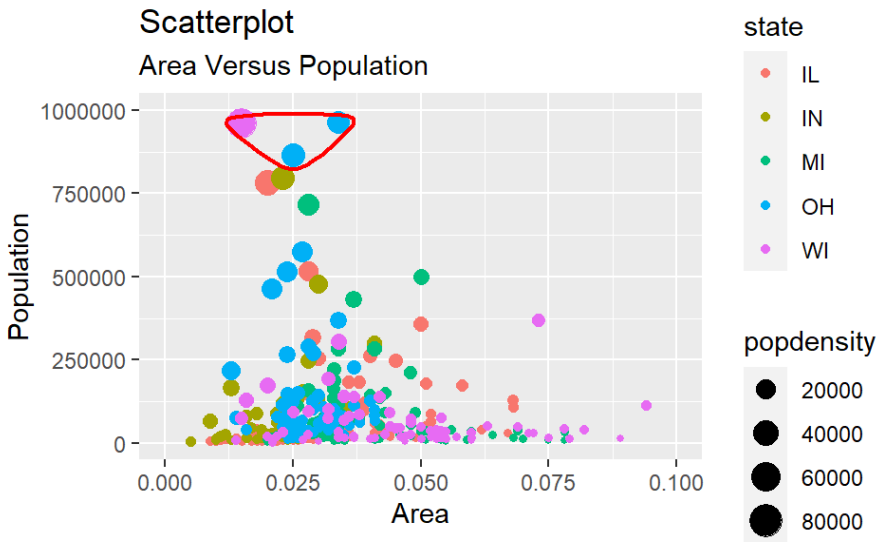
Looking at the tibble `midwest_filter`, how many data points have population greater than 800,000? Assign this integer value to `answer05`.

Now encircle our points. Because the new data set for the encircle is `midwest_filter`, this will be directly passed to the [geom_encircle]{classKeywordTok} function.

```
g4 <- g3 +
  geom_encircle(aes(x=area, y=poptotal),
    data = midwest_filter,
    color = "red",
    size = 2,
    expand = 0.04)
g4
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```

```
## Warning: Removed 4 rows containing missing values (geom_encircle).
```



Problem 6

Looking at `g4`, how many data points were encircled? Assign this integer to `answer06`.

Kernel Density plots

Given data from a distribution, it is helpful to have a way of estimating the density of the distribution. To see how this can be accomplished with `ggplot2`, first let us create some random data. This data will be placed into a container known as a *tibble*. To do this, we will need the `tibble` package.

```
library(tibble)
```

Next we generate our data. Note that we use `set.seed` so that the random numbers generated are the same every time you run the code.

```
set.seed(1234)
df <- tibble(
  gender = factor(rep(c("F", "M"), each = 200)),
  weight = round(c(rnorm(200, mean=55, sd=5),
                   rnorm(200, mean=65, sd=5)))
  )
df
```

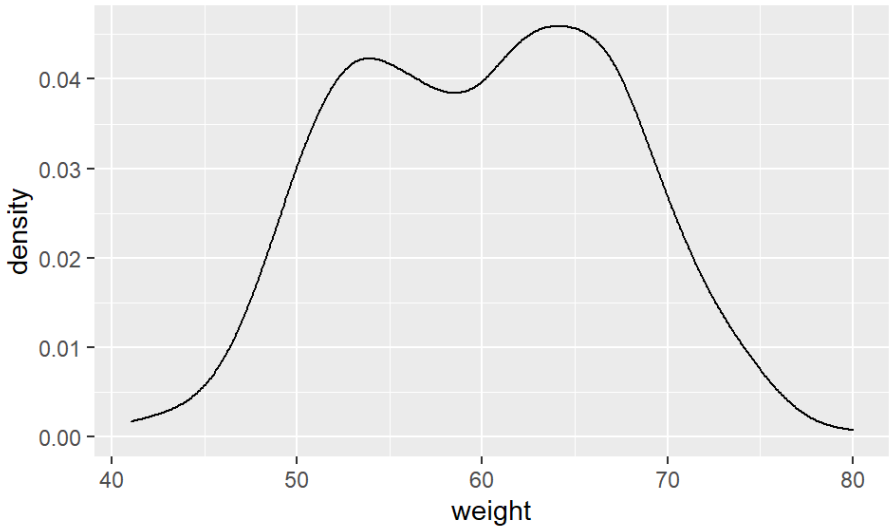
gender <fct>	weight <dbl>
F	49
F	56
F	60
F	43
F	57
F	58
F	52
F	52
F	52
F	51

1-10 of 400 rows Previous **1** 2 3 4 5 6 … 40 Next

This simulates 200 draws to serve as the weights of the male subjects, and 200 draws for the weights of the female subjects.

The basic density plot in `ggplot2` is called `geom_density()`. Try

```
p <- ggplot(df, aes(x = weight)) +
  geom_density()
p
```



We can also add in a vertical line indicating the mean.

```
p + geom_vline(aes(xintercept = mean(weight)),
  color = "blue", linetype = "dashed", size  =1)
```

As with most `geom` functions, the `color` parameter changes the color of the line, while `fill` changes the color of the area under the line.

```
ggplot(df, aes(x = weight))+
  geom_density(color = "darkblue", fill = "lightblue")
```

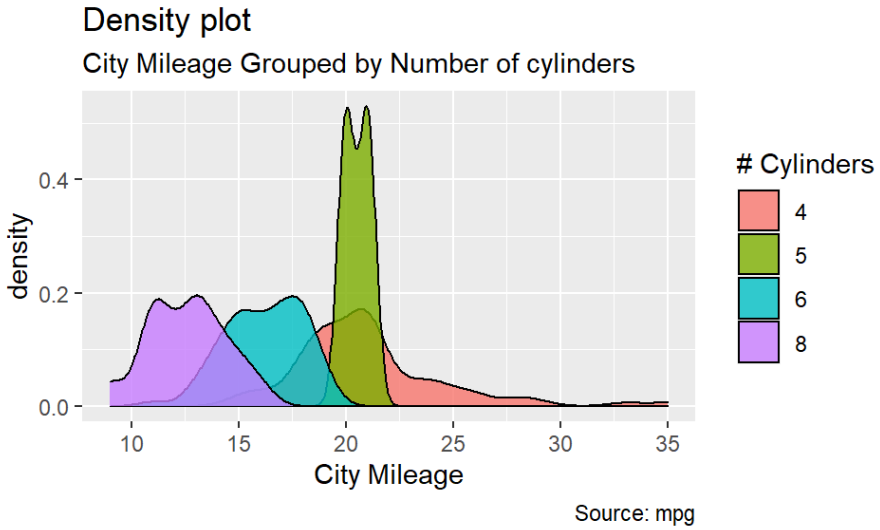
This mix of normals is hiding the difference in average weight between men and women. To break out the data, we need only declare that the two groups should be treated separately in the plot:

```
ggplot(df, aes(x = weight, color = gender)) +
  geom_density()
```

Problem 7

Consider the following plot.

```
# Plot
g <- ggplot(mpg, aes(cty))
g + geom_density(aes(fill = factor(cyl)), alpha=0.8) +
    labs(title="Density plot",
         subtitle = "City Mileage Grouped by Number of cylinders",
         caption = "Source: mpg",
         x = "City Mileage",
         fill = "# Cylinders")
```



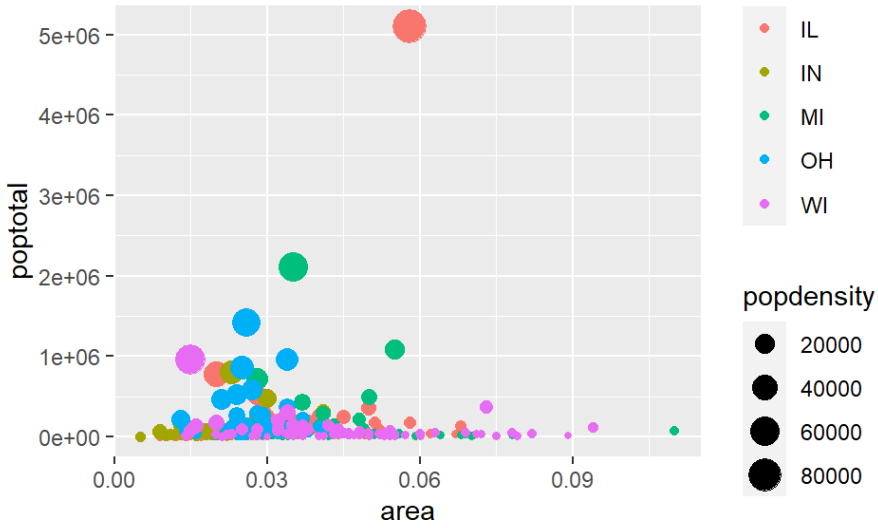
Looking at this plot, which number of cylinders has greater variation, 5 or 6? Assign this number to `answer07`.

Using theme

The `theme` set of functions can alter the entire aspect of a plot.

Recall our `g1` plot.

```
g1
```



Now try it with a classic background.

```
g1 + theme_classic()
```

Now a minimalist background.

```
g1 + theme_minimal()
```

The `theme` function can change all aspects of the plot. For instance, the `aspect.ratio` is the ratio between the vertical and horizontal size of the plot. (Note that this is the multiplicative inverse of how it is usually used in the graphics industry.)

You can get a widescreen plot suitable for 4K TV's with:

```
g1 + theme(aspect.ratio = 9 / 16)
```

Problem 8

The theme_linedraw function is a theme that places a grid of lines onto your graphic. Use this function to add a theme to the `g1` variable as created above. Assign the resulting graphical object to `answer08`.

Problem 9

Using the `theme` function, change the aspect ratio of the plot in `g1` to 4:3. Assign the resulting graphical object to `answer09`.

Moving the legend around

By default the legend is on the right hand side, but can be moved or eliminated with the theme function.

```
g1 + theme(legend.position="top")
g1 + theme(legend.position="bottom")
g1 + theme(legend.position="none") # Remove legend
```

Problem 10

Use the theme to put the legend in `g1` on the left side of the plot. Assign the result to `answer10`.