Let’s do some plotting with ggplot. We begin with histograms.

We first load the data from the Nettle (1999) book about language diversity and create a new object called languages.

languages <- read\_csv("nettle\_1999\_climate.csv")

glimpse(languages)

Rows: 74

Columns: 5

$ Country <chr> "Algeria", "Angola", "Australia", "Bangladesh", "Benin", "Bolivia", "Botswana", "Brazil", …

$ Population <dbl> 4.41, 4.01, 4.24, 5.07, 3.69, 3.88, 3.13, 5.19, 3.97, 3.50, 3.93, 4.09, 3.76, 4.53, 3.37, …

$ Area <dbl> 6.38, 6.10, 6.89, 5.16, 5.05, 6.04, 5.76, 6.93, 5.44, 5.79, 5.26, 5.68, 6.11, 6.06, 5.53, …

$ MGS <dbl> 6.60, 6.22, 6.00, 7.40, 7.14, 6.92, 4.60, 9.71, 5.17, 8.08, 8.44, 9.17, 4.00, 11.37, 9.60,…

$ Langs <dbl> 18, 42, 234, 37, 52, 38, 27, 209, 75, 94, 18, 275, 126, 79, 60, 10, 75, 1, 22, 11, 112, 11…

Histograms are plotted if we use geom\_histogram() in the ggplot() command.

ggplot(languages,

mapping = aes (x = Langs)) + geom\_histogram()

Let's pause for a second to consider themes(). In ggplot, we can make many, many, many, many detailed changes to our graphs, which we cannot cover here. Check out the graph gallery here.

<https://r-graph-gallery.com/ggplot2-package.html>

But we can also make bigger changes easily by adding themes() to our ggplot() commands.

Let's compare the following four types of themes. (There are many other themes.)

ggplot(languages,

mapping = aes (x = Langs)) + geom\_histogram() # No theme

ggplot(languages,

mapping = aes (x = Langs)) + geom\_histogram() + theme\_gray()   
 # Same as before, the default

ggplot(languages,

mapping = aes (x = Langs)) + geom\_histogram() + theme\_minimal() # Theme minimal

ggplot(languages,

mapping = aes (x = Langs)) + geom\_histogram() + theme\_classic() # Theme classic

We can also format the different elements of the histogram. Below, we change the widths of the bins and later add some color, too.

Again, compare what the commands do.

ggplot(languages,

aes(Langs)) + geom\_histogram()

ggplot(languages,

aes(Langs)) + geom\_histogram(binwidth = 50)

ggplot(languages,

aes(Langs)) + geom\_histogram(binwidth = 50,

color = 'turquoise4',

fill = 'paleturquoise')

Sometimes you might prefer using a frequency polygon to display data rather than a histogram. For this, just the geom\_freqpoly() command.

Compare. First, a histogram. (You can see the plot above.)

gplot(languages,

aes(Langs)) + geom\_histogram(binwidth = 50,

color = 'turquoise4',

fill = 'paleturquoise')

Then, a frequency polygon.

ggplot(languages,

aes(Langs)) + geom\_freqpoly(binwidth = 50,

color = 'turquoise4',

fill = 'paleturquoise')

Step 10: *Bar charts with* ggplot

Let’s try out bar charts now. Again, we load data first. This comes from the Andrews (2021) textbook.

titanic <- read\_csv("TitanicSurvival.csv")

glimpse(titanic)

Rows: 1,309

Columns: 5

$ ...1 <chr> "Allen, Miss. Elisabeth Walton", "Allison, Master. Hudson Trevor", "Allison, Miss. Hel…

$ survived <chr> "yes", "yes", "no", "no", "no", "yes", "yes", "no", "yes", "no", "no", "yes", "yes", "…

$ sex <chr> "female", "male", "female", "male", "female", "male", "female", "male", "female", "mal…

$ age <dbl> 29.0000, 0.9167, 2.0000, 30.0000, 25.0000, 48.0000, 63.0000, 39.0000, 53.0000, 71.0000…

$ passengerClass <chr> "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1…

To do bar charts, we use geom\_bar().

Compare what the different commands do.

ggplot(titanic,

aes(passengerClass)) + geom\_bar()

ggplot(titanic,

aes(passengerClass, fill = survived)) + geom\_bar()

ggplot(titanic,

aes(passengerClass, fill = survived)) + geom\_bar(position = "dodge")

As you can see, we can use nominal categories as fill in aesthetic mapping (fill = survived), and we can manipulate the position of the bars (position = “dodge”).

Let’s more data before we illustrate further. This dataset is also from Andrew (2021), it displays information about cars (lots of them).

car\_prices <- read\_csv("carprice.csv")

glimpse(car\_prices)

The following command creates a bar chart visualizing the prices (y-axis) according to type of car (x-axis).

ggplot(car\_prices,

aes(Type, Price)) + geom\_bar(stat = "identity")

We can also group the data to further explore. The command below creates a new object called cars\_prices\_grouped, loads the car price data, groups it by car type, and calculates the mean prices for each car type. The %>% operator allows you to connect multiple commands that you would normally have to run separately. It is part of the magritte package, which we already installed at the beginning of the session.

cars\_prices\_grouped <- read\_csv("carprice.csv") %>%

group\_by(Type) %>%

summarize(Price = mean(Price))

If you now type the name of the new object cars\_prices\_grouped, you will get a tibble with the average price.

cars\_prices\_grouped # Compare to car\_prices above

We can now do another bar chart with the grouped data.

ggplot(cars\_prices\_grouped,

aes(Type, Price)) + geom\_bar(stat = "identity")

Step 11: *Tukey boxplots with* ggplot

For the boxplots, we will use the built-in dataset ToothGrowth.

First, let’s convert the variable dose from a numeric to a factor variable. These are categorical variables that can be either numeric or string variables and that can be used more easily in different types of graphics.

ToothGrowth

ToothGrowth$dose <- as.factor(ToothGrowth$dose)

head(ToothGrowth)

len supp dose

1 4.2 VC 0.5

2 11.5 VC 0.5

3 7.3 VC 0.5

4 5.8 VC 0.5

5 6.4 VC 0.5

6 10.0 VC 0.5

The following command will generate a boxplot.

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot()

To make our life a bit easier, let's assign the command above (which produces a plot) to a new object called our\_boxplot.

This command creates an object called our\_boxplot, which consists of … a boxplot.

our\_boxplot <- ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot()

If you now run this command…

our\_boxplot

… you create the boxplot.

To rotate a boxplot, just coord\_flip()to your boxplot, as in the example below.

our\_boxplot + coord\_flip()

Notched box plots are also useful. The notch refers to the narrowing around the median. You can create a notched box plots as follows.

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(notch = TRUE)

Did you notice that there is an outlier? (See above the top whisker.) By default, outliers are in the color of the box. But we can change color, shape and size of the outlier.

Let's try out different shapes. See how the different values for the outlier.shape argument affect the plot. What happens when you run the following commands?

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 0,

# square

outlier.size = 4

)

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 1,

# circle

outlier.size = 4

)

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 2,

# triangle point up

outlier.size = 4

)

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 5,

# diamond

outlier.size = 4

)

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 8,

# star

outlier.size = 4

)

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 17,

# filled triangle point-up

outlier.size = 4

)

ggplot(ToothGrowth, aes(x = dose, y = len)) +

geom\_boxplot(

outlier.colour = "darkblue",

outlier.shape = 25,

# filled triangle point down blue

outlier.size = 4

)

The function stat\_summary() can be used to add mean points to a box plot, as in the following command.

our\_boxplot + stat\_summary(

fun.y = mean,

geom = "point",

shape = 23,

size = 4

)

We can points to a box plot by using the functions geom\_dotplot() or geom\_jitter().

In the following example, our box plot also has a dot plot.

our\_boxplot + geom\_dotplot(binaxis = 'y',

stackdir = 'center',

dotsize = 0.5)

And in this example, it has jittered points with 0.2 degree of jitter in x direction. The jitter geom adds a bit of random variation to the location of each point. This is useful when too many points are overlapping.

our\_boxplot + geom\_jitter(shape = 16, position = position\_jitter(0.2))

Step 12: *Scatterplots with* ggplot

Let’s now turn to scatterplots. Again, let’s some data, this time the Nettle (1999) data about language diversity, used in Winter (2019).

languages <- read\_csv("nettle\_1999\_climate.csv")

glimpse(languages)

The geom\_point() command creates scatterplots. This first one below uses points.

ggplot(languages, aes(MGS, Langs)) + geom\_point()

But this one uses the text, drawn from the Country variable in our dataset languages.

ggplot(languages, aes(MGS, Langs, label = Country)) + geom\_text()

Let’s load some additional data.

language\_exams <- read\_csv("language\_exams\_shorter.csv")

glimpse(language\_exams)

In the following scatterplots, we will use the variable age cohort for color.

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort)) + geom\_point()

There are many parameters you can add, delete or edit in your plots. The cheat sheet is very helpful in that regard.

By default, the size of your points are in size 2. You can see this by comparing the scatterplot above, which doesn’t have a size specification, to the one created by the following command. (The plot looks the same, so we won’t plot this here.)

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort))   
 + geom\_point(size = 2)

Let’s play around with size and shapes in scatterplots. See what happens when you run these commands.

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort))   
 + geom\_point(size = 3)

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort))   
 + geom\_point(size = 4)

And now let’s try out different shapes. How do the commands change your plots?

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort))   
 + geom\_point(size = 3, shape = 'triangle') # Same as writing 2

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort))   
 + geom\_point(size = 4, shape = 'diamond') # Same as writing 5

ggplot(language\_exams,

aes(x = exam\_1, y = exam\_2, color = age\_cohort))   
 + geom\_point(size = 3, shape = 'square') # Same as writing 0

We can also use labels for data points.

ggplot(language\_exams,

aes(

x = exam\_1,

y = exam\_2,

label = student,

color = age\_cohort

)) + geom\_point(size = 3, shape = 'triangle') +

geom\_text(size = 4)

The geom\_text\_repel() function is useful if we intend to use labels.

library(ggrepel)

install.packages("ggrepel")

ggplot(our\_data,

aes(

x = exam\_1,

y = exam\_2,

label = student,

color = age\_cohort

)) + geom\_point(size = 3) +

geom\_text\_repel(size = 4, segment.alpha = 2)

Step 13: *Saving our graphs with* ggplot

This is the ggsave() command. Let's try saving our plots in a few different file formats (png, pdf, jpeg). Have a look at the working directory to see if you can find your three new files.

ggsave('our\_plot.png', width = 8, height = 6)

ggsave('our\_plot.pdf', width = 8, height = 6)

ggsave('our\_plot.jpeg', width = 8, height = 6)

You can also save plots with specific resolutions. They default is dpi = 300.

Alternatively, you can also write string input: “retina” (dpi 320), “print” (dpi 300), or “screen” (dpi 72). Have a go at the following.

ggsave(

'our\_plot\_300.jpeg',

width = 8,

height = 6,

dpi = 300

)

ggsave(

'our\_plot\_screen.jpeg',

width = 8,

height = 6,

dpi = 'screen'

)

ggsave(

'our\_plot\_retina.jpeg',

width = 8,

height = 6,

dpi = 'retina'

)

As mentioned, we have only touched very lightly on ggplot. I recommend you check out online materials and start exploring! ☺

Step 14: *Appendix on colors*

In R, you can either specify colors by writing their names (e.g., “mistyrose”) or you can write the hexadecimal code (#ffe4e1).

You can try out compare the following.

The first uses the color names. The second uses the same colors but refers to them hexadecimal code.

barplot(c(2,5), col=c("paleturquoise", "mistyrose "))

barplot(c(2,5), col=c("#30D5C8 ", "#FFE4E1"))

The following website provides color names and a hex code finder:

<https://r-graph-gallery.com/ggplot2-color.html>