Data Visualization

INFO 640 | Pratt Institute | McSweeney

```
#install.packages("ggthemes")
library(ggthemes)
library(tidyverse)
library(dplyr)
library(lubridate)
```

This lab comes in 3 parts: Visualizations about Flowers, Chicken Weights, and Air Quality. You only have to do 2 of the 3. You must do the Flowers part. In Chicken Weights, you will visualize Categorical Variables. In Air Quality, you will visualize Time Series data.

By the end of this lab, you will be familiar with the ggplot library, and some general principles for clear communication through visualizations.

Flowers

We would like to understand how flower petals are related to flower sepals. Petals are the colorful bits, and sepals are the part that hold up the petals. It's often easier to understand a dataset by visualizing it. This is particularly true if there are a lot of datapoints and you are doing an exploratory analysis. Typically in an exploratory analysis, you'd want to see how variables are related to each other, and if there are any trends.

In this tutorial, we will:

- plot sepal length & width against each other
- differentiate by species
- find the "line of best fit" aka the regression line
- format your plot to highlight key aspects

Submit: * Two plots: your final Iris Plot and either the plot of chicken weights or Air quality Lucky for us, R has a dataset already loaded that examines this relationship. It's actually a very famous dataset, so you may have seen it before. In R, it's called the 'iris' dataset. We'd like to know more about what is in the dataset, what is available, and what the variables are. Let's do a little exploration.

```
?iris
```

Load the dataset & inspect it

```
data(iris)

summary(iris)
glimpse(iris)

head(iris)

sum(is.na(iris))
```

Let's get started with an exploratory analysis since we don't really know the story we're going to tell yet. We'll make a scatterplot using the point geomerty.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```

That isn't particularly helpful since so many of the points are on top of each other. We can "jitter" them to move them around ever so slightly. The mathematical representation isn't change, and the point still covers the same place on the cartesian plane, though the center is shifted slightly. If precision is the most essential for your visualization, jittering is not appropriate. But, to visual your data and make generalizations, it is.

Secondly, we will adjust the transparency so we can see through them a little bit

```
ggplot(iris, aes(x= Sepal.Length, y = Sepal.Width)) +
  geom_jitter(alpha = 0.6)
```

This is great, and we've shown 2 variables, but we likely want to be able to encode a 3rd. Let's add in Species with color.

We will also add a title so we know what this plot is about. Always add a title - even for exploratory visualizations, future you will thank you. We do this in the labs() for labels.

```
ggplot(iris, aes(x= Sepal.Length, y = Sepal.Width, color=Species)) +
  geom_jitter() +
  labs(title = "Sepal Length by Sepal Width in Iris Dataset")
```

This is awfully hard to see - we probably need to break these apart. We could do this by filtering the dataframe and creating multiple distinct graphs. There are a few problems with this, the first is scale, the second is formatting, the third is just keeping them together as a unit. A much better solution would be to use small multiples. Small multiples in R is acheived through faceting.

```
ggplot(iris, aes(x= Sepal.Length, y = Sepal.Width)) +
  geom_jitter(alpha = 0.6) +
  facet_grid(.~Species)
```

This is a good start, but a natural next question might be what is the trend here? To visualize this, we will add a 'line of best fit' or a linear model to each plot. The line of best fit uses stat_smooth(). The method is linear, and we don't want to display the standard error (which is the buffer around which the model is 95% accurate), and we'll make the color red.

```
ggplot(iris, aes(x= Sepal.Length, y = Sepal.Width)) +
  geom_jitter(alpha = 0.5) +
  facet_grid(.~Species) +
  stat_smooth(method = "lm", se= FALSE, col = "red")
```

This is beginning to get a bit redundant. ggplot() offers us some more portable ways to manipulate our plots and create similar but slightly different visualizations illustrating different variables.

Every visualization must have data, aesthetics & geometries, but you might want to try different geometries w the same data/aes, or different formats to be sure your visual is communicating what you want it to as clearly as possible.

Let's make an object that contains the group of elements that will remain consistent.

```
iris_plot <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color=Species))
```

Now we can try out a few different geometries

```
iris_plot + geom_point()
iris_plot + geom_jitter()
iris_plot + geom_line()
```

So far, the jitter argument seems problematic - right now it's random, we'd like to make it consistent across plots. Save the position argument and pass it to jitter your points

```
posn <- position_jitter(width = 0.1)
iris_plot + geom_point(position =  posn)
```

It would be great if we could find the midpoint of each cluster, essentially an "average" for each metric. Let's do some basic statistics on our plot. We want to find the mid point of Sepal Length & Sepal Width and place a point there so we can visualize the distribution with respect to the mean. First we need to maniputlate the data. We will make a new dataframe with just these points. We will use aggregate(), and specify the columns of the iris dataset that we want to take into consideration. Then find the mean of each of those columns for each species.

```
iris_summary <- aggregate(iris[1:4], list(iris$Species), mean)
names(iris_summary)[1] <- "Species"
iris_summary
```

Then we can plot this aggregated measure using the iris_plot we defined above, and layering a visualization of the iris_summary data.

```
iris_plot + geom_point() +
  geom_point(data = iris_summary,
             shape = 21, size = 5, fill = "#00000080")

iris_plot + geom_point(position =posn, alpha=.6) +
  geom_point(data = iris_summary, shape = 15, size = 5)
```

What if you would rather see these values as a line rather than a point. Use vline and hline as your geometries: (vertical and horizontal, respectively). You can even change the way the line looks with the linetype.

```
iris_plot + geom_point(position =posn, alpha=.6) +
  geom_vline(data = iris_summary, aes(xintercept = Sepal.Length))

#can do this to the y axis, too - creating horizontal lines
iris_plot + geom_point(position =posn, alpha=.6) +
  geom_vline(data = iris_summary, aes(xintercept = Sepal.Length)) +
  geom_hline(data = iris_summary, aes(yintercept = Sepal.Width))

#change the linetypes
iris_plot + geom_point(position =posn, alpha=.6) +
  geom_vline(data = iris_summary,linetype = 2, aes(xintercept = Sepal.Length)) +
  geom_hline(data = iris_summary, linetype = 3, aes(yintercept = Sepal.Width))
```

Up to here, our plots have been strictly exploratory. You can distribute graphs like this, but you'd probably want to alter the formatting to better tell your story.

We'll begin by adding coordinates and specifying the scale. We will add back in the facets and "line of best fit".

```
iris_plot +
  geom_point(position = posn, alpha = 0.5) +
  facet_grid(.~Species) +
  stat_smooth(method = "lm", se= FALSE, col = "red") +
  scale_y_continuous("Sepal Width (cm)",
                     limits = c(1,5),
                     expand = c(0,0)) +
  scale_x_continuous("Sepal length (cm)",
                     limits = c(4,8),
                     breaks = seq(2, 8, 2),
                     expand = c(0,0)) +
  coord_equal()
```

Now we'll modify the labels by adding another phrase to our expression.

```
iris_plot +
  geom_point(position = posn, alpha = 0.5) +
  facet_grid(.~Species) +
  stat_smooth(method = "lm", se= FALSE, col = "red") +
  scale_y_continuous("Sepal Width (cm)",
                     limits = c(1,5),
                     expand = c(0,0)) +
  scale_x_continuous("Sepal length (cm)",
                     limits = c(4,8),
                     breaks = seq(2, 8, 2),
                     expand = c(0,0)) +
  coord_equal() +
  labs(x = "Sepal Length", y = "Sepal Width", col = "Species")
```

R also allows you to specify the "theme" - essentially defining how the grids, backgrounds, and other features of the plot look. Though these features are not the data themselves, controlling them helps readers understand what you are trying to communicate with the data. There is a significant amount of theory on this, of the best way to prepare a plot for maximal communication. For more on this topic, Edward Tufte's work is a good starting place (but by no means the only voice).

```
ggplot(iris, aes(x= Sepal.Length, y = Sepal.Width)) +
  geom_jitter(alpha = 0.5) +
  facet_grid(.~Species) +
  stat_smooth(method = "lm", se= FALSE, col = "red") +
  scale_y_continuous("Sepal Width (cm)",
                     limits = c(1,5),
                     expand = c(0,0)) +
  scale_x_continuous("Sepal length (cm)",
                     limits = c(4,8),
                     breaks = seq(2, 8, 2),
                     expand = c(0,0)) +
  coord_equal() +
  labs(x = "Sepal Length", y = "Sepal Width", col = "Species") +
  theme(panel.background = element_blank(),
        plot.background = element_blank(),
        legend.background = element_blank(),
        legend.key = element_blank(),
        strip.background = element_blank(),
        axis.text = element_text(colour = "black"),
        axis.ticks = element_line(colour = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_line(colour = "black"),
        strip.text = element_blank(),
        panel.spacing = unit(1, "lines")
  )
```

Well that was ridiculous to type and you should probably never do it again (save this lab and use it as a template or use pre-defined themes). But for now, let's save it as a unit that we can call for future plots (you can actually save this to a file and call it later - a technique beyond the scope of this course).

Also, you can find more about themes on the ggplot documentation

```
iris_theme <-   theme(panel.background = element_blank(),
                      plot.background = element_blank(),
                      legend.background = element_blank(),
                      legend.key = element_blank(),
                      strip.background = element_blank(),
                      axis.text = element_text(colour = "black"),
                      axis.ticks = element_line(colour = "black"),
                      panel.grid.major = element_blank(),
                      panel.grid.minor = element_blank(),
                      axis.line = element_line(colour = "black"),
                      strip.text = element_blank(),
                      panel.spacing = unit(1, "lines")
)

iris_plot + geom_point(posn) + labs(x = "Sepal Length", y = "Sepal Width", col = "Species")
```

There are some built in themes! Check out the docs.

```
??ggthemes
```

Let's see the Tufte theme

```
iris_plot + geom_jitter() + theme_tufte()
```

Chickens

We'd like to understand how the amount that a chicken is fed affects its weight. In this visualization, we will work with categorical variables. We already loaded the packages, but in case you're starting fresh, you can reload them.

```
library(ggthemes)
library(tidyverse)
library(dplyr)
library(lubridate)
```

R has a dataset about Chicken Weights that we will utilize here.

```
??ChickWeight
data("ChickWeight")
```

Let's first establish our base plot specifying that we will use the ChickWeight dataset and the x axis will contain the weight.

```
chicks <- ggplot(ChickWeight, aes(x=ChickWeight$weight))
```

First we'd like to see the distribution of chicken weights, so we will make a histogram. A histogram is different from a bar chart because in a histogram, you can specify the number of bins you want your data broken down into whereas with a bar chart, those bins are pre-defined. Chicken Weight is a QUANTITATIVE variable, which is why a histogram (rather than a bar chart) is appropriate.

```
chicks + geom_histogram()
```

That caused a warning. R is often very helpful, so let's take its advice here and calculate the best binwidth. 30 is a good default number of bins, but maybe we'd like a finer distinction. Let's look at 40.

```
diff(range(ChickWeight$weight)) / 40
```

Looks like the bin width for 40 bins is is 8.45pt

```
chicks + geom_histogram(binwidth = 8.45)
```

Notice how the bins are a bit thinner.

That gave us a good view into the distribution, but now let's investigate weight as a function of diet type. We will again make a histogram, but this time fill each bin with the color of the Diet. Diet is our categorical variable.

```
ggplot(ChickWeight, aes(x=ChickWeight$weight, fill = factor(ChickWeight$Diet))) +
  geom_histogram(binwidth = 8.45)
```

If instead you want to only show the breakdown of the fill - irrespective of how many chicks are in each category (similar to 100% bar chart), you'd take the count for each item and divide it by the sum. The .. on eiter side of each metric tells R to do this on the fly for each unique item in the variable, in this case, 'Diet'.

```
ggplot(ChickWeight, aes(x=ChickWeight$weight, fill = factor(ChickWeight$Diet))) +
  geom_histogram(aes(y = ..count../sum(..count..)),
                 binwidth = 8.45, position = "fill")
```

Notice that in the first plot, we see that there is probably only one chick that has a near zero weight. This could be an error, or it could be that the chick never grew. This datum presents more questions than answers. It's probably an anomoly. Unlike the datum at about Chick weight 245. Even though there is also only 1 chick in that category, it's really just a statistical chance. Even without looking at the distribution, we can infer that since there are reasonably small counts on either side of it. Finally, the few outliers are probably true outliers, but definitely interesting.

Notably, looking at the second plot, it seems that Diet type 1 causes low weight, but really, it's just one datum that represents 100%. Vsualization choices affect the interpretation.

Now let's turn our attention to weight vs age. We might make some predictions about this relationship. For example, we may predict that older chicks weigh more. Let's see if our prediction is correct. We will move Time (age) to the x axis since any kind of time metric is an independent variable.

```
ggplot(ChickWeight, aes(x=ChickWeight$Time, y=ChickWeight$weight)) +
  geom_jitter(alpha=.6)
```

It looks like our prediction was correct: older chicks weigh more. But it may not be that clear cut. Notice how the older the, chicks are the greater the range of weights. This is also not surprising, but we may ask if there's any trend in diet versus weight at those ages. Let's check on age versus weight by diet.

We'll also add a trend line to give our eyes a point of reference - even though, as we will see, the trend line doesn't fit the data.

```
ggplot(ChickWeight, aes(x=ChickWeight$Time, y = ChickWeight$weigh, color = ChickWeight$Diet
  geom_jitter(alpha = 0.6) +
  stat_smooth(method = "lm", se=FALSE,col = "grey")
```

Even though it looks like the Diet #1 chicks are smaller than the others, it's really hard to see. In situations like this, were you are trying to distinguish between categorical variables, encoding the information in BOTH color and facet is often helpful. This serves dua purposes: it makes it easier to read for ALL sighted people. It also makes it accessible for individuals with colorblindness. Faceting is a win for universal design. You may be thinking "well, why do we need color?" Answer is that we actually don't but it does make it visually easier to understand for individuals without colorblindess, so there's no good reason to remove it.

```
ggplot(ChickWeight, aes(x=ChickWeight$weight, y = ChickWeight$Time, color = ChickWeight$Die
  geom_jitter(alpha = 0.6) +
  facet_grid(.~ChickWeight$Diet) +
  stat_smooth(method = "lm", se=FALSE, col = "grey")
```

Notice that even with faceting, it's clear that a linear model doesn't fit this data. This is something we will address in a future lab.

TimeSeries Visualizations in ggplot

Our last visualization technique will be a time series. We'll use a very simple line chart for both of these datasets.

There are 2 datasets in this section: First, we're interested in internet usage. Specifically, we want to know minute-by-minute how many server calls were made.

Second, we're interested in dataquality in NYC.

Internet Usage

Time series data is often presented in a TimeSeries format. Call str() to see what format it is in.

```
data("WWWusage")
??WWWusage
str(WWWusage)
```

Note that this data was collected in 1998. The internet was a different place and a different technology than it is today. What do we expect this to mean for our analysis?

Note secondly that it is not a dataframe. We will have to transform it if we want to use ggplot() (and we do…). First make our dataset a matrix, specifying the column names. Then make it a dataframe.

```
www <- data.frame(usage = as.matrix(WWWusage), use_time=time(WWWusage))
```

Now we can visualize this with ggplot the same way we did the flower data.

```
ggplot(www, aes(x=use_time, y = usage)) +
  geom_line()
```

What might have happened at the 55 minutes time and at the 95 minutes time?

Air quality

The Air Quality in NYC is monitored and recorded everyday. Today, you can get air quality alerts on your phone. Let's look at a historic record and see if there were any trends in the airquality in NYC for 6 months in 1973 (seemingly random, I know, but the principles apply).

```
data("airquality")
??airquality
str(airquality)
```

Looks like this data is already in a dataframe, and has quite a few variables that we are not going to work with, though we could.

Let's start with a simple visualization of a line chart with time on the x axis and air quality on the y. Note: Run this, but it'll look crazy.

```
ggplot(airquality, aes(x=Day, y=Temp)) +
  geom_line()
```

Why didn't that work?!? It's not a time series!! We're just looking at the day of each month - not the day of the year.

We need to manipulate the data to have an actual month/day/year combination. First we will paste the Month and Day columns with 1973, separating each variable by a hyphen. This will give us something that looks like a date.

Then, transform the date into a time stamp using mdy() (since our data is in month-day-year format). Helpful Hint: For more date/time manipulations, check out the lubridate package.

```
airquality$new_date <- paste(airquality$Month, airquality$Day, "1973", sep="-")
airquality$new_date <- mdy(airquality$new_date)
glimpse(airquality)
```

That looks much better! Our new_date column has the datatype . Now we can go ahead and use this dataframe in ggplot with our well defined time variable.

```
ggplot(airquality, aes(x=new_date, y=Temp)) + geom_line()
```

Congratulations! You now have the tools necessary to make visualizations in R!