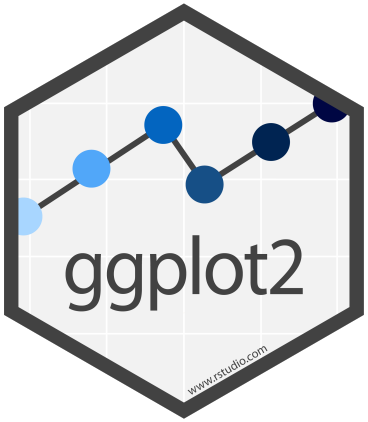
Data Visualization

JSC 370: Data Science II

January 29, 2024

Background



This lecture provides an introduction to `ggplot2`, an R package that provides vastly better graphics options than R's default plots, histograms, etc.

Background

`ggplot2` is part of the Tidyverse. The tidyverse is..."an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures." (https://www.tidyverse.org/)

Core tidyverse includes `dplyr`, which provides the main grammar for data manipulation.

It also includes `stringr` which we will use in a few weeks and `lubridate` that we saw in lab that helps with dates and times.

```
library(tidyverse)
library(nycflights13)
```

```
library(kableExtra)
```

Layers, dplyr and pipes

- We should take a step back and discuss `tidyverse` a bit more.

- `ggplot2` behaves very similarly to `dplyr`. They are part of the tidyverse.

- The first argument in `dplyr` is always a data frame (it can be a data.table).

- Subsequent arguments can also be thought of as layers describe actions to be taken on the data (verbs).

- The output is always a new data frame.

- Layers are connected by a pipe, which until R 4.1.0 was denoted `%>%`.

- The new pipe is `|>`, which works similarly but has some differences likely only noticeable by expert users.

The Pipe %>% and now |>

- The pipe passes the object on its left hand side to the first argument of the function on the right hand side.

- We can kind of think of it as saying 'then'.

Flights data

To illustrate many of today's examples we will look at the `flights` data from the `nycflights13` library. They are all flights that departed from NYC airports (JFK, LGA, EWR) in 2013.

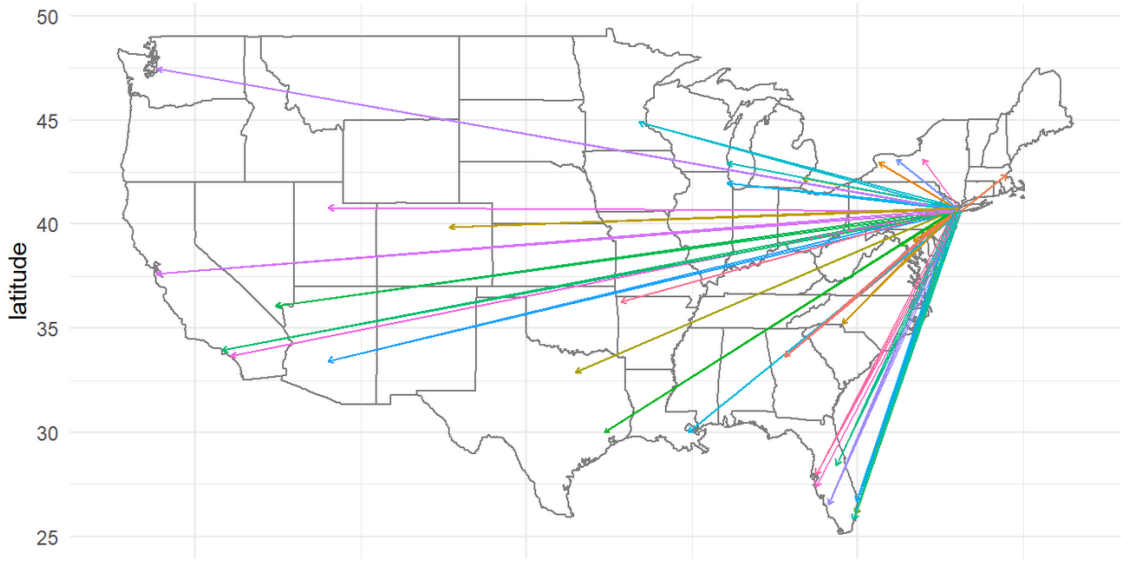
```
names(flights)
```

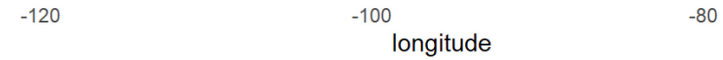
```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

```
dim(flights)
```

```
## [1] 336776      19
```

Flights data





The pipe %>%

- An example using pipes: subset the flights data to LAX, and take the mean arrival delay times by year, month, day.

- We need to start with the data, `filter` to the desired destination (LAX), `group_by` to prepare the groups that we want to summarize over, then `summarize` to take the mean (or whatever function we want) of the variable we are interested in.

```r
nycflights13::flights %>%
  filter(dest == "LAX") %>%
  group_by(year, month, day) %>%
  summarize(
    arr_delay = mean(arr_delay, na.rm = TRUE)
  )
```

The new pipe |>

Let's do the same thing with the new pipe:

```
nycflights13::flights |>
  filter(dest == "LAX") |>
  group_by(year, month, day) |>
  summarize(
    arr_delay = mean(arr_delay, na.rm = TRUE)
  )
```

A few coding style tips

- Variable names (those created by `<-` and those created by `mutate()` and `summarize()`) should use only lowercase letters, numbers, and _

- Use _ to separate words within a name

- `%>%` or `|>` should always have a space before it, and should usually be followed by a new line

- After the first step, each line should be indented by two spaces.

- Note on the previous slides that had long lines, the arguments to a function may not all fit on one line, put each argument on its own line and indent.

```
short_flights <- flights |>
  filter(air_time < 60)
```

Flights data in more detail

The `nycflights13` library also provides hourly airport weather data for the three NYC airports (the origin). Let's join the flights data with the `weather` data so we can look at more interesting relationships in our visualizations.

```
names(weather)
```

```
##  [1] "origin"     "year"       "month"      "day"        "hour"
##  [6] "temp"       "dewp"       "humid"      "wind_dir"   "wind_speed"
## [11] "wind_gust"  "precip"     "pressure"   "visib"      "time_hour"
```

```
dim(weather)
```

```
## [1] 26115    15
```

Flights data in more detail

Looks like we can join these datasets on year, month, day, hour and origin (which is the origin airport). We can examine the relationship between flight delays and weather at the origin airports (JFK, LGA, EWR).

A `left_join` will keep all of the observations in x (here, flights) and merge with the observations in y (weather at origin). Because there are many flights originating at the 3 NYC airports on a given year, month, day, hour we want to keep the resolution of the x dataset.

```
flights_weather <-
  left_join(
    flights, weather, by = c("year", "month", "day", "hour", "origin")
    )
```

Flights data in more detail

```
head(flights_weather)
```

```
## # A tibble: 6 × 29
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ℹ 21 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour.x <dttm>, temp <dbl>, dewp <dbl>,
## #   humid <dbl>, wind_dir <dbl>, wind_speed <dbl>, wind_gust <dbl>,
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hour.y <dttm>
```

Daily flights data

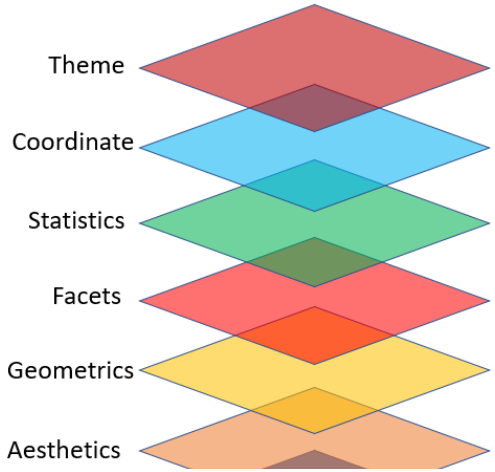
Let's make a more manageable sized dataset and summarize the data by month, day, and origin airport

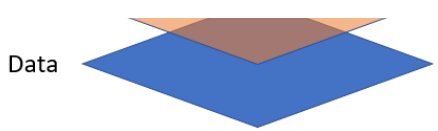
```
flights_weather_day <-
  flights_weather |>
  group_by(year, month, day, origin) |>
  summarize_at(
    vars(dep_delay, arr_delay, temp, dewp, humid, wind_dir,
             wind_speed, wind_gust, precip, pressure, visib), mean, na.rm=TRUE

  )
```

Visualizations

`ggplot2` is designed on the principle of adding layers.





Layers in ggplot2

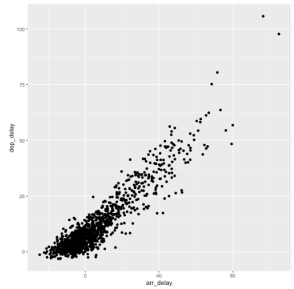
- With `ggplot2` a plot is initiated with the function `ggplot()`
- The first argument of `ggplot()` is the dataset to use in the graph
- We add aesthetics is always paired with `aes()`
- The `aes()` mapping takes the x and y axes of the plot
- Layers are added to `ggplot()` with `+`
- Layers include `geom` functions such as point, lines, etc.

```
ggplot(data = data, mapping = aes(mappings)) +
  geom_function()
```

Basic scatterplot

The first argument of `ggplot()` is the dataset to use, then the aesthetics. With the `+` you add one or more layers.

```
ggplot(data = flights_weather_day, mapping = aes(x = arr_delay, y = dep_delay)) +
  geom_point()
```

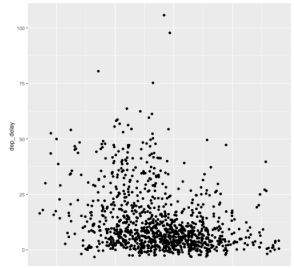


As expected, we see that if a flight has a late departure, it has a late arrival.

Another basic scatterplot

We can drop `data =` and `mapping =`. Now let's see if there is a relationship between departure delays and pressure (low pressure means clouds and precipitation, high pressure means better weather).

```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point()
```



Adding to a basic scatterplot

- `geom_point()` adds a layer of points to your plot, creating a scatterplot.

- `ggplot2` comes with many geom functions that each add a different type of layer to a plot.

- Each `geom` function in `ggplot2` takes a mapping argument.

- This defines how variables in your dataset are mapped to visual properties.

- The mapping argument is always paired with `aes()`, and the x and y arguments of `aes()` specify which variables to map to the x and y axes.

- One common problem when creating `ggplot2` graphics is to put the `+`

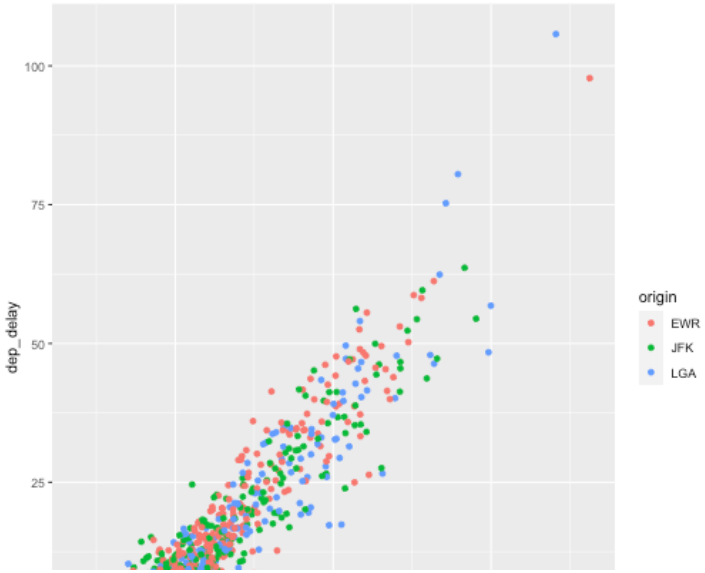
in the wrong place: it has to come at the end of the line, not the start.

Coloring by a variable - using aesthetics

You can convey information about your data by mapping the aesthetics in your plot to the variables in your dataset. For example, you can map the colors of your points to the class variable to reveal groupings. `ggplot` chooses colors, and adds a legend, automatically.

```
ggplot(flights_weather_day, aes(x = arr_delay, y = dep_delay, color = origin)) +
  geom_point()
```

Coloring by a variable - using aesthetics

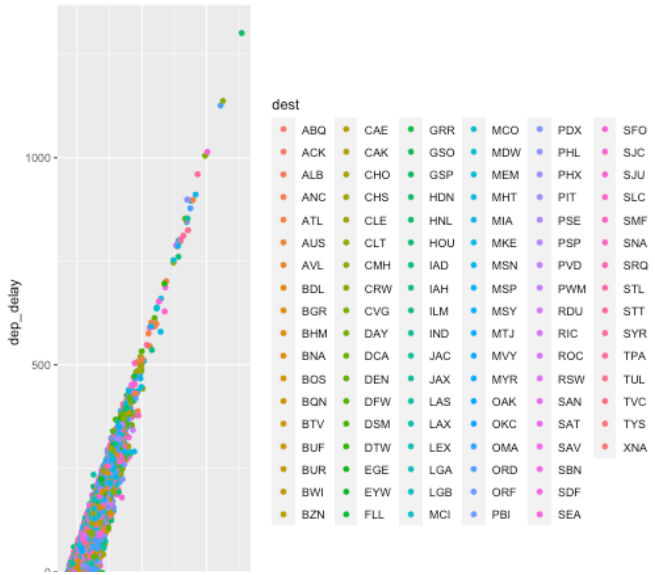


Coloring by a variable - using aesthetics

Note when there are a lot of classes or groups, the coloring is not distinguished well

```
ggplot(flights_weather, aes(x = arr_delay, y = dep_delay, color = dest)) +
  geom_point()
```

Coloring by a variable - using aesthetics

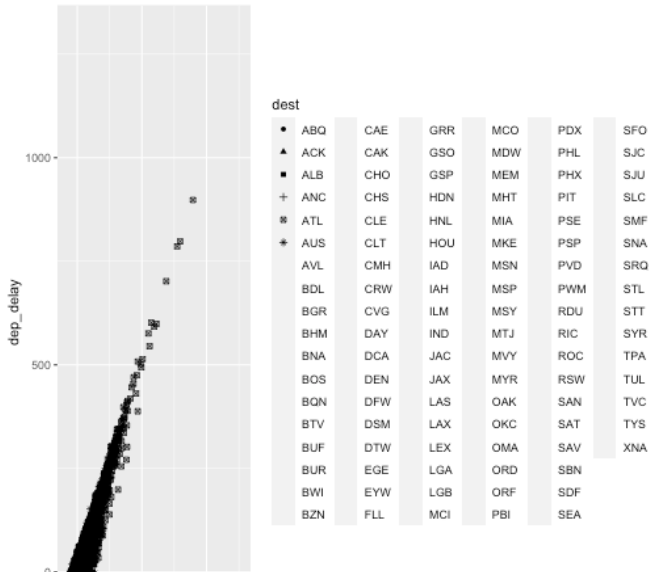


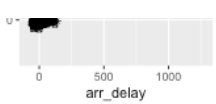
Determining point type using a variable

By default ggplot uses up to 6 shapes. If there are more, some of your data is not plotted!! (At least it warns you.)

```
ggplot(flights_weather, aes(x = arr_delay, y = dep_delay, shape = dest)) +
  geom_point()
```

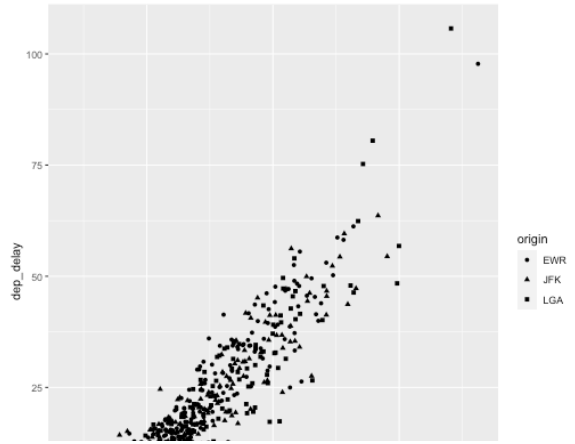
Determining point type using a variable

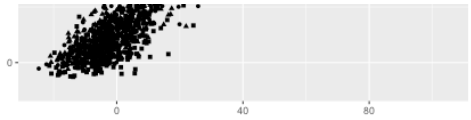




Determining point type using a variable

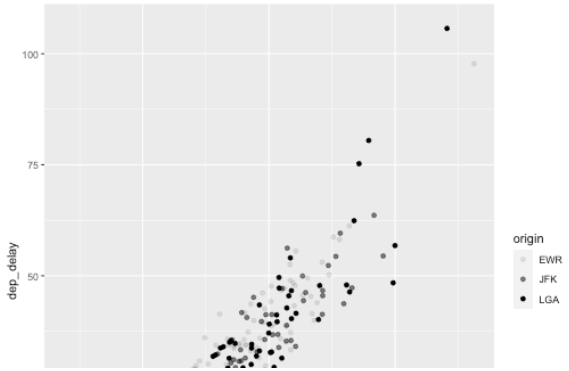
```
ggplot(flights_weather_day, aes(x = arr_delay, y = dep_delay, shape = origin)) +
  geom_point()
```

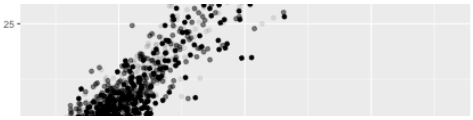




Controlling point transparency using the "alpha" aesthetic

```
ggplot(flights_weather_day, aes(x = arr_delay, y = dep_delay, alpha = origin)) +
  geom_point()
```

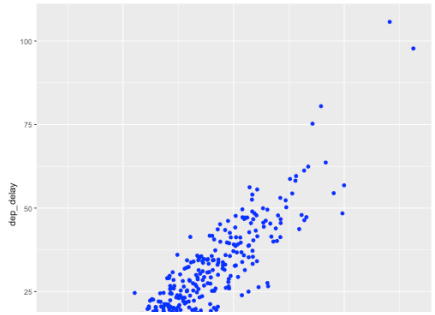


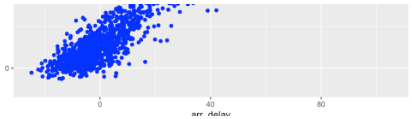


Manual control of aesthetics

To control aesthetics manually, we do it in `geom_point`, so set the aesthetic by name outside `aes()`

```
ggplot(flights_weather_day, aes(x = arr_delay, y = dep_delay)) +
  geom_point(color = "blue")
```





Summary of aesthetics

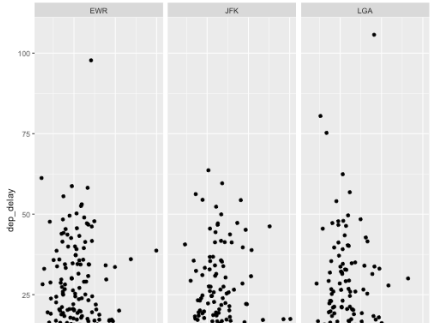
The various aesthetics...

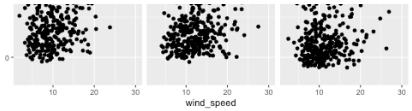
Code	Description
x	position on x-axis
x	position on y-axis
shape	shape
color	color of element borders
fill	color inside elements
size	size
alpha	transparency
linetype	type of line

Facets 1

Facets are particularly useful for categorical variables...

```
ggplot(flights_weather_day, aes(x = wind_speed, y = dep_delay)) +
  geom_point() +
  facet_wrap(~origin, nrow = 1)
```

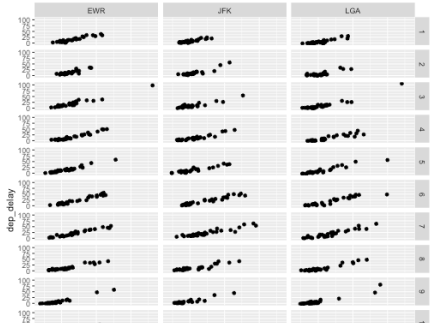


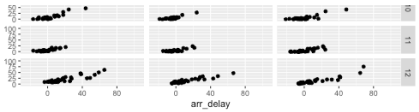


Facets 2

Or you can facet on two variables...

```
ggplot(flights_weather_day, aes(x = arr_delay, y = dep_delay)) +
  geom_point() +
  facet_grid(month ~ origin)
```



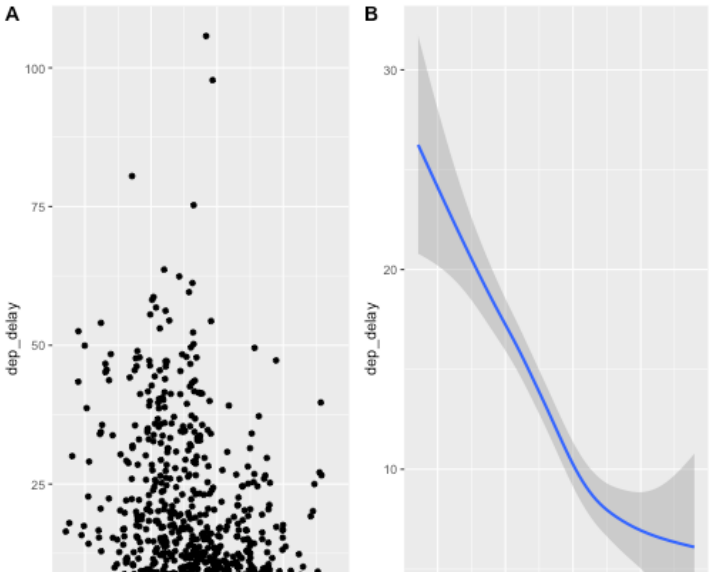


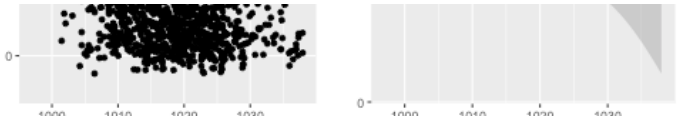
Geometric Objects 1

Geometric objects are used to control the type of plot you draw. Let's try plotting a smoothed line fitted to the data (and note how we do side-by-side plots).

```
library(cowplot)
scatterplot <- ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point()
lineplot <- ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_smooth()
plot_grid(scatterplot, lineplot, labels = "AUTO")
```

Geometric Objects 1

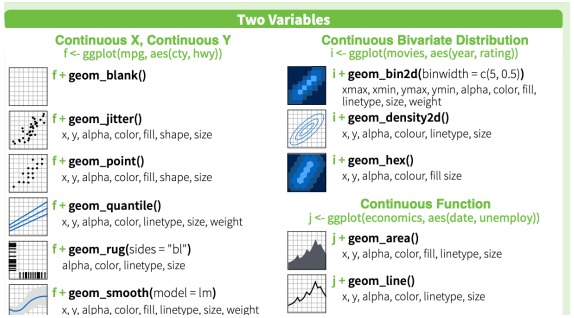




Geoms - Reference

`ggplot2` provides over 40 geoms, and extension packages provide even more (see https://ggplot2.tidyverse.org/reference/ for a sampling).

The best way to get a comprehensive overview is the `ggplot2 cheatsheet`, which you can find at https://posit.co/resources/cheatsheets/

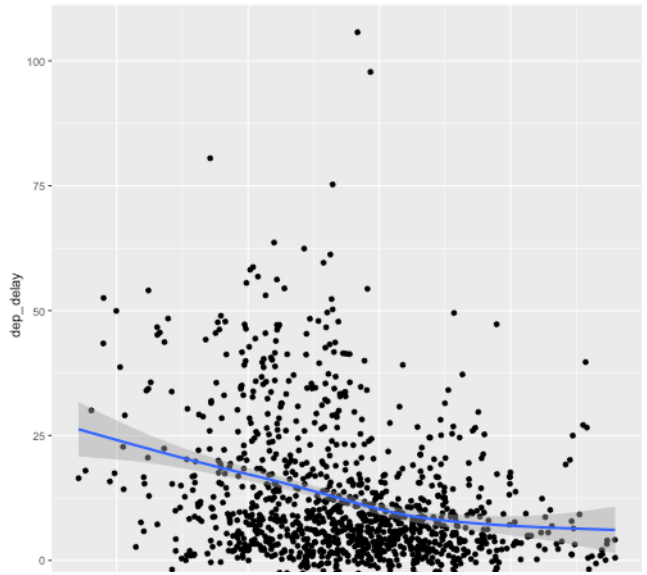


Multiple Geoms 1

To display multiple geoms in the same plot, add multiple `geom` functions to `ggplot()`:

```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point() +
  geom_smooth()
```

Multiple Geoms 1

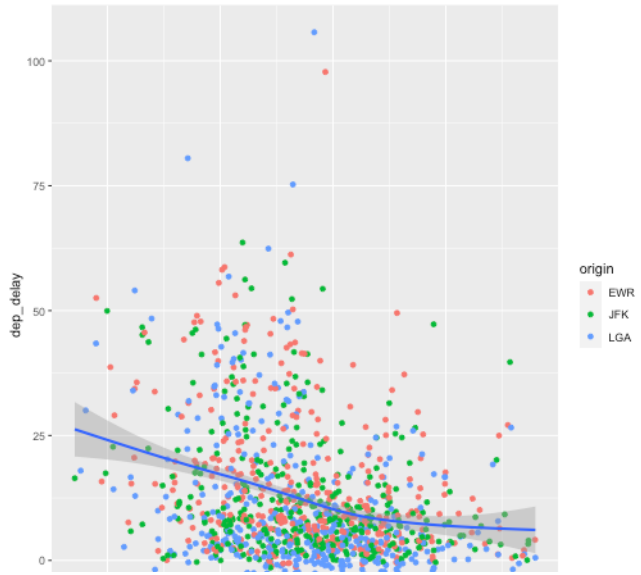


Multiple Geoms 2

If you place mappings in a `geom` function, `ggplot2` will use these mappings to extend or overwrite the global mappings for that layer only. This makes it possible to display different aesthetics in different layers.

```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point(aes(color = origin)) +
  geom_smooth()
```

Multiple Geoms 2

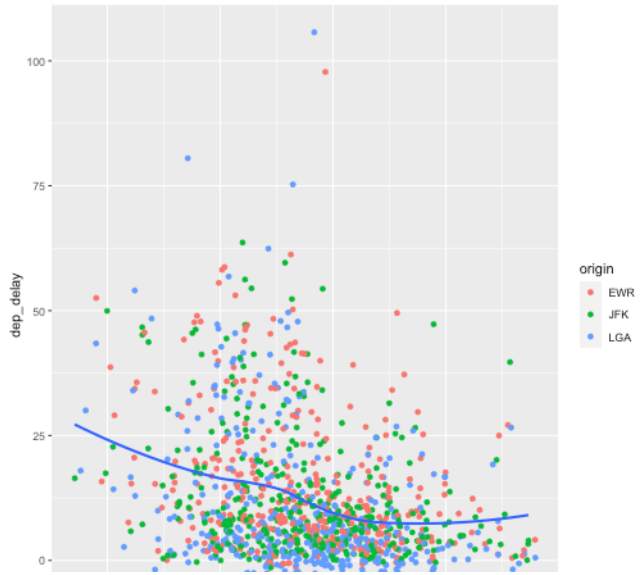


Multiple Geoms 3

You can use the same idea to specify different data for each layer. Here, our smooth line displays just a subset of the dataset, the flights departing from JFK. The local data argument in `geom_smooth()` overrides the global data argument in `ggplot()` for that layer only.

```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point(mapping = aes(color = origin)) +
  geom_smooth(data = filter(flights_weather_day, origin == "JFK"), se = FALSE)
```

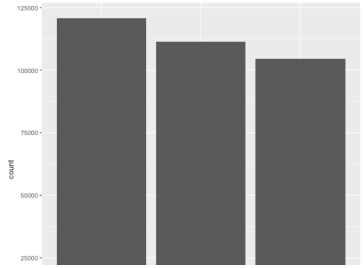
Multiple Geoms 3

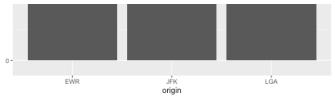


Statistical Transformations - e.g. Bar charts

Let's make a bar chart of the number of flights at each origin airport in 2013. The algorithm uses a built-in statistical transformation, called a "stat", to calculate the counts.

```
ggplot(flights_weather, aes(x = origin)) +
  geom_bar()
```

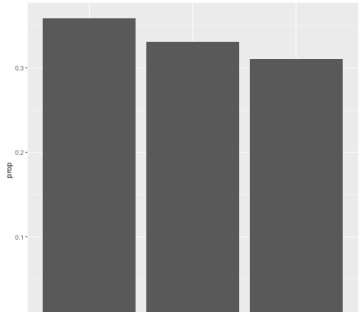




Bar charts 2

You can override the statistic a `geom` uses to construct its plot. e.g., if we want to plot proportions, rather than counts:

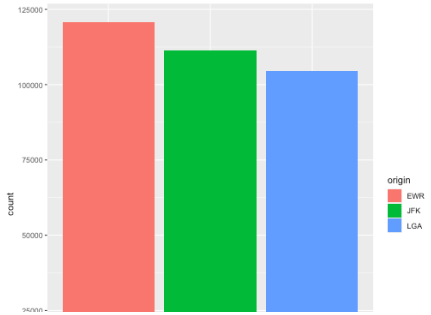
```
ggplot(flights_weather, aes(x = origin, y = stat(prop), group = 1)) +
  geom_bar()
```



Coloring barcharts

You can color a bar chart using either the `color` aesthetic (only does the outline), or, more usefully, `fill`:

```
ggplot(flights_weather, aes(x = origin, fill= origin)) +
  geom_bar()
```



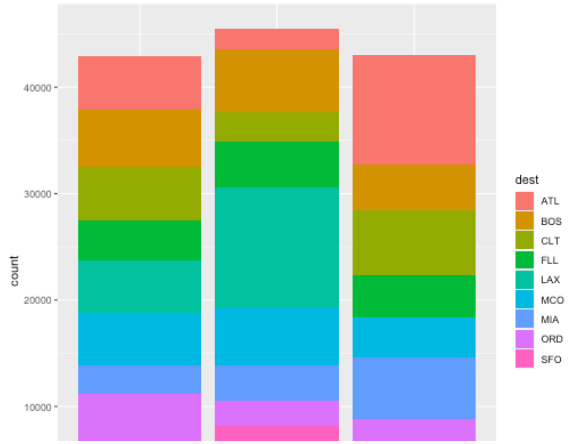
Coloring barcharts

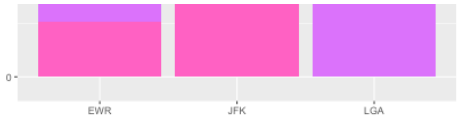
More interestingly, you can fill by another variable. Let's first subset our data to look at the destination airports with a lot of flights (more than 10,000 flights)

```
flights_weather_ss <- flights_weather |>
  group_by(dest) |>
  filter(n() > 10000)
```

Coloring barcharts

```
ggplot(flights_weather_ss, aes(x = origin, fill= dest)) +
  geom_bar()
```

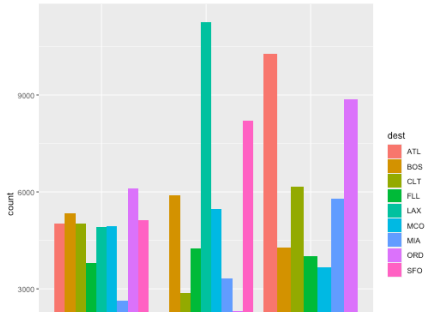


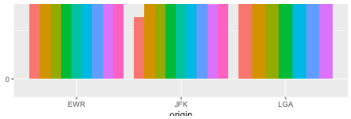


Coloring barcharts

The `position = "dodge"` places overlapping objects directly beside one another. This makes it easier to compare individual values.

```
ggplot(flights_weather_ss, aes(x = origin, fill= dest)) +
  geom_bar(position="dodge")
```



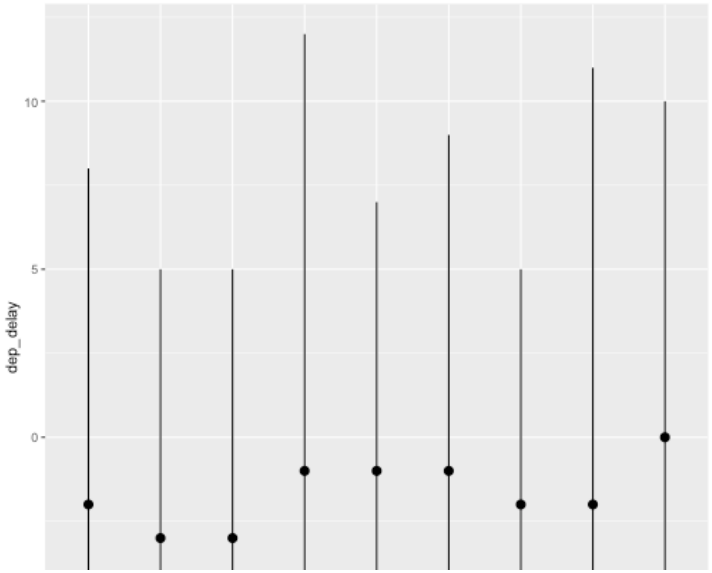


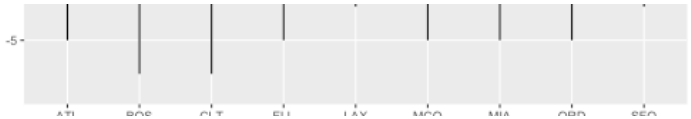
Statistical transformations - another example

You might want to draw greater attention to the statistical transformation in your code. For example, you might use `stat_summary()`, which summarizes the y values for each unique x value, to draw attention to the summary that you're computing:

```
ggplot(flights_weather_ss, aes(x = dest, y = dep_delay)) +
  stat_summary(fun = median,
               fun.min = function(z) { quantile(z,0.25) },
               fun.max = function(z) { quantile(z,0.75) })
```

Statistical transformations - another example



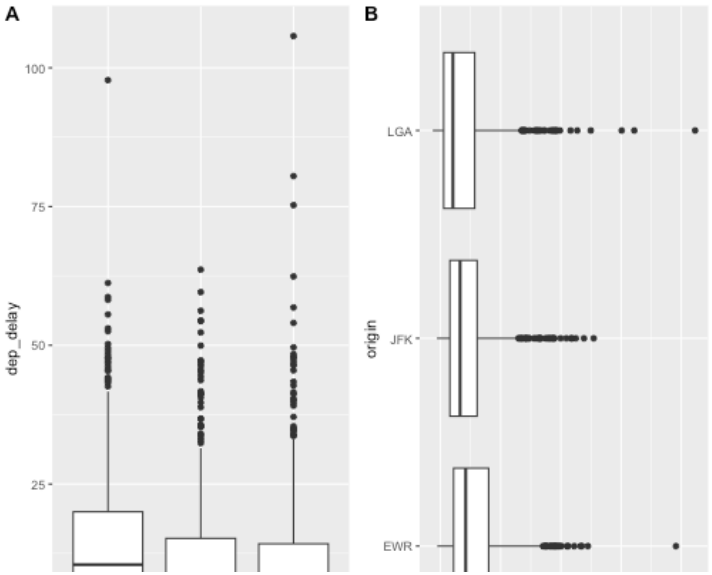


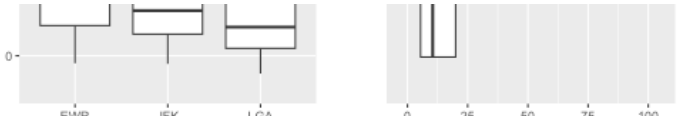
Coordinate systems

Coordinate systems are one of the more complicated corners of ggplot. To start with something simple, here's how to flip axes:

```
unflipped <- ggplot(flights_weather_day, aes(x = origin, y = dep_delay)) +
  geom_boxplot()
flipped <- ggplot(flights_weather_day, aes(x = origin, y = dep_delay)) +
  geom_boxplot() +
  coord_flip()
plot_grid(unflipped, flipped, labels = "AUTO")
```

Coordinate systems





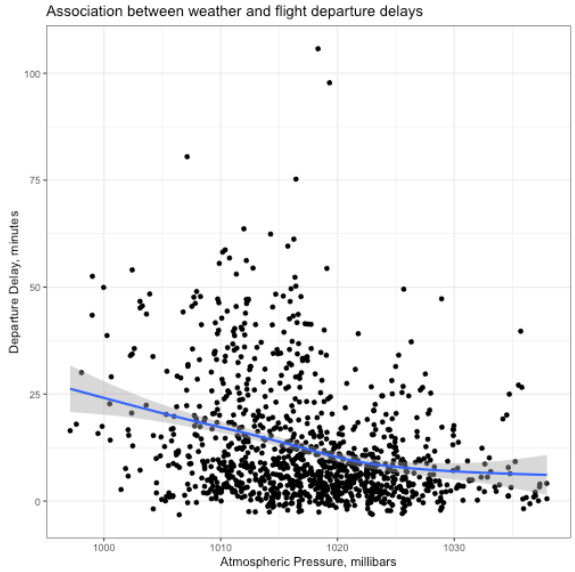
Adding labels

You can make nicer axes and add titles with the `labs` layer

Also showing a minimal theme that removes the background grey `theme_bw()`

```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point() +
  geom_smooth() +
  labs(
    x = "Atmospheric Pressure, millibars",
    y = "Departure Delay, minutes",
    title = "Association between weather and flight departure delays"
  )+
  theme_bw()
```

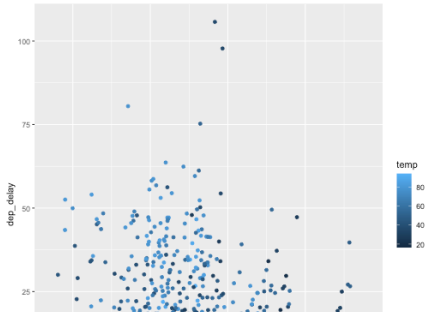
Adding labels

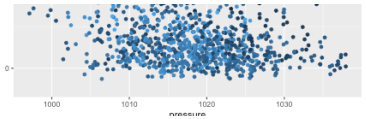


Color ramps

If you add a continuous variable in your color aesthetic, it will create a color ramp

```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point(aes(colour = temp))
```

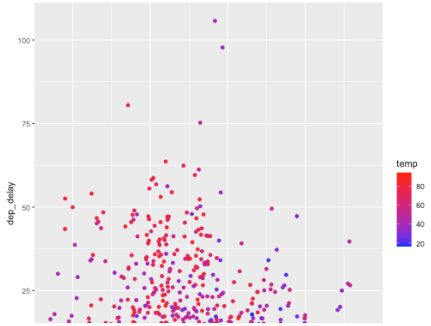


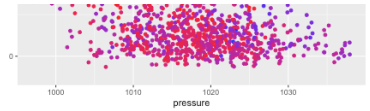


Color palettes

You can define your own color ramp or use one of the palettes

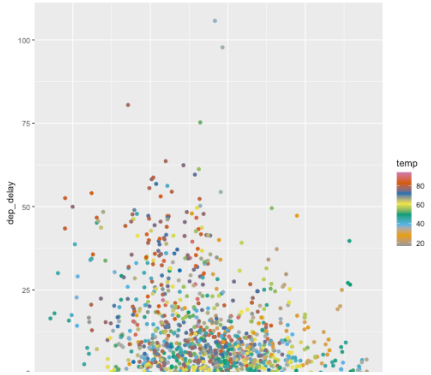
```
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point(aes(colour = temp)) +
  scale_colour_gradient(low = "blue", high = "red")
```

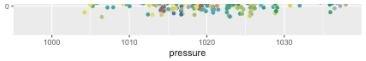




Color palettes

```
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC7
ggplot(flights_weather_day, aes(x = pressure, y = dep_delay)) +
  geom_point(aes(colour = temp)) +
  scale_colour_gradientn(colours = cbPalette)
```





A Great reference

A great (comprehensive) reference for everything you can do with `ggplot2` is the R Graphics Cookbook:

https://r-graphics.org/

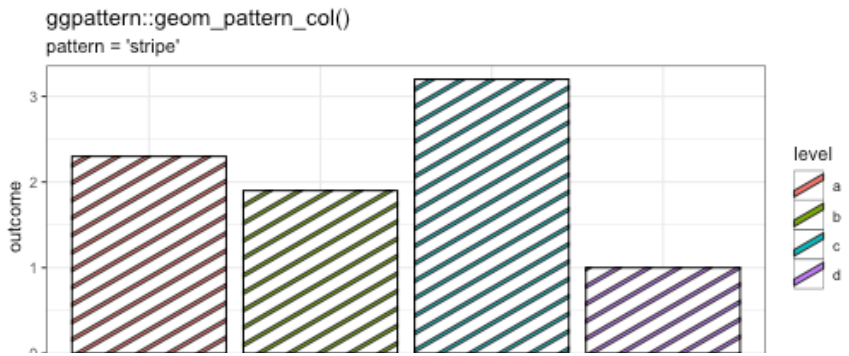
Finally, file under "useless but cool"

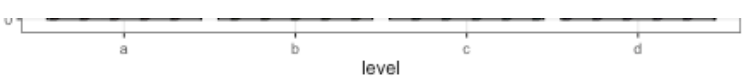
ggpattern - is a library for adding pattern fills to histograms. e.g.,

```
  library(ggpattern)
df <- data.frame(level = c("a", "b", "c", "d"), outcome = c(2.3, 1.9, 3.2, 1))

ggplot(df, aes(level, outcome, pattern_fill = level)) +
  geom_col_pattern(pattern = "stripe", fill = "white", colour = "black") +
  theme(legend.position = "none") +
  labs(title = "ggpattern::geom_pattern_col()", subtitle = "pattern = 'stripe'") +
  coord_fixed(ratio = 1 / 2)+
  theme_bw() +
```

Finally, file under "useless but cool"




0
a
b
c
d
level