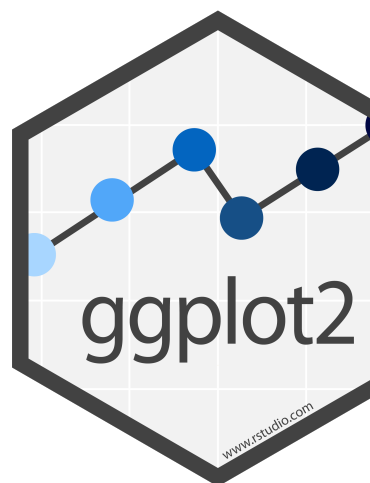


# Data Visuali

JSC 370: Data Sc

January 29, 20

## Background



This lecture provides an introduction to `ggplot2`, which provides vastly better graphics options than R's default

## Background

`ggplot2` is part of the Tidyverse. The tidyverse is a collection of R packages designed for data science. All packages follow the same design philosophy, grammar, and data structure. Visit [www.tidyverse.org/](http://www.tidyverse.org/)

Core tidyverse includes `dplyr`, which provides data manipulation.

It also includes `stringr` which we will use in a moment. 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
- `ggplot2` behaves very similarly to `dplyr`. The
- The first argument in `dplyr` is always a data frame (or a `data.table`).
- Subsequent arguments can also be thought of as actions to be taken on the data (verbs).
- The output is always a new data frame.
- Layers are connected by a pipe, which un

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

## The Pipe `%>%` and now `|>`

- The pipe passes the object on its left hand 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 use the `nycflights13` library. They are all flights that departed from New York City (JFK, LGA, EWR) in 2013.

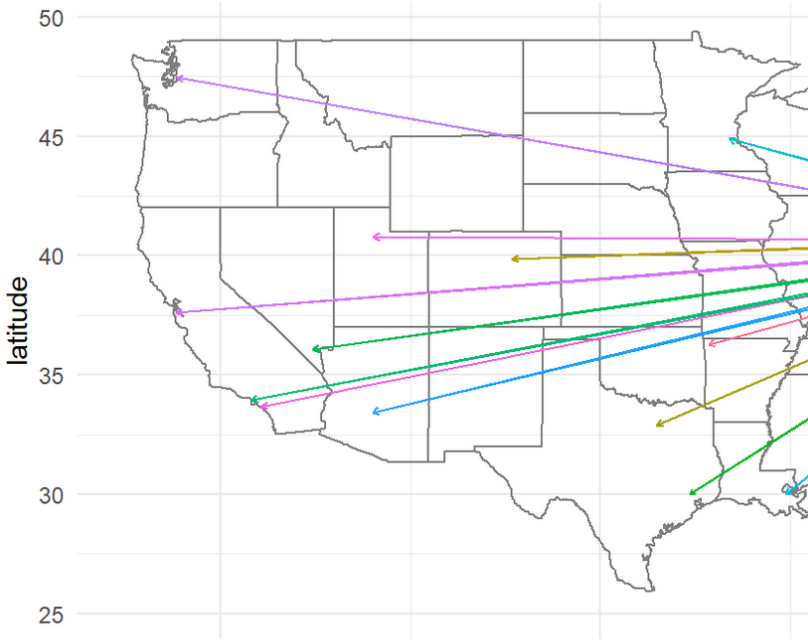
```
names(flights)
```

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

```
dim(flights)
```

## [1] 336776 19

# Flights data



-120

-100

longitude

## The pipe %>%

- An example using pipes: subset the flight data, calculate the mean arrival delay times by year, month, and carrier.
- We need to start with the data, filter to get the data we want, group\_by to prepare the groups that we want to summarize to take the mean (or whatever function we are interested in).



```
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 `summarize()`) should use only lowercase
- Use `_` to separate words within a name
- `%>%` or `|>` should always have a space before followed by a new line
- After the first step, each line should be indented
- Note on the previous slides that had long function may not all fit on one line, put each line on a new line and indent.

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

## Flights data in more detail

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

```
names(weather)
```

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

```
dim(weather)
```

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

## Flights data in more detail

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

A `left_join` will keep all of the observations in `x` (flights) with the observations in `y` (weather at origin). We can examine flight delays originating at the 3 NYC airports on a given day. We want to keep the resolution of the `x` dataset.

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

## Flights data in more detail

```
head(flights_weather)
```

```
## # A tibble: 6 × 29
##   year month   day dep_time sched_dep_time dep_delay a
##   <int> <int> <int>   <int>         <int>      <dbl>
## 1  2013     1     1     517           515         2
## 2  2013     1     1     533           529         4
## 3  2013     1     1     542           540         2
## 4  2013     1     1     544           545        -1
## 5  2013     1     1     554           600        -6
## 6  2013     1     1     554           558        -4
## # i 21 more variables: arr_delay <dbl>, carrier <chr>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <
## #   hour <dbl>, minute <dbl>, time_hour.x <dtm>, temp
## #   humid <dbl>, wind_dir <dbl>, wind_speed <dbl>, wind
## #   precip <dbl>, pressure <dbl>, visib <dbl>, time_hou
```

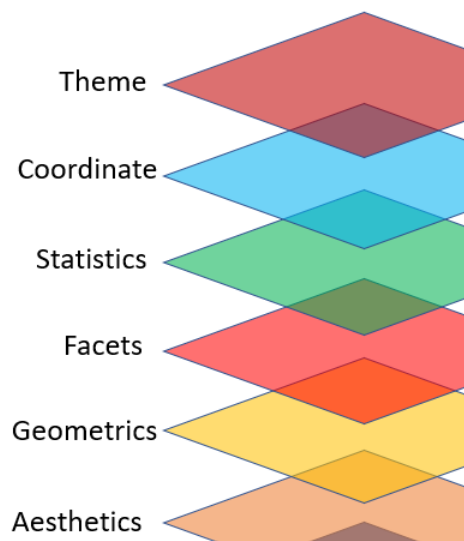
## Daily flights data

Let's make a more manageable sized dataset  
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, vis)  
  )
```

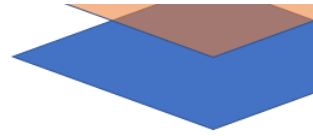
# Visualizations

`ggplot2` is designed on the principle of adding





Data



## Layers in ggplot2

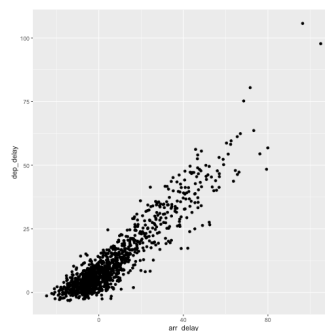
- With `ggplot2` a plot is initiated with the function `ggplot()`
- The first argument of `ggplot()` is the data
- We add aesthetics is always paired with `aes()`
- The `aes()` mapping takes the x and y axes
- Layers are added to `ggplot()` with `+`
- Layers include `geom` functions such as `point`

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

## Basic scatterplot

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

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

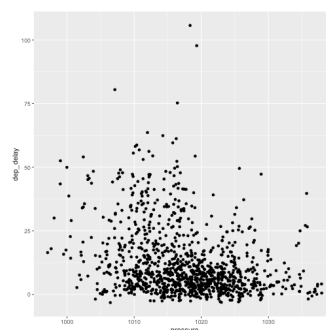


As expected, we see that if a flight has a late arrival, it also tends to have a late departure.

## Another basic scatterplot

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

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



## Adding to a basic scatterplot

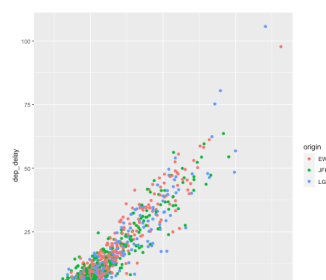
- `geom_point()` adds a layer of points to your plot.
- `ggplot2` comes with many geom functions to add different type of layer to a plot.
- Each geom function in `ggplot2` takes a mapping argument.
- This defines how variables in your dataset are mapped to the plot's properties.
- The mapping argument is always paired with the `aes()` arguments of `aes()` specify which variables are mapped to the plot's properties.
- One common problem when creating ggplots is that the mapping argument is not paired with the `aes()` arguments.

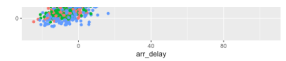
in the wrong place: it has to come at the end

## Coloring by a variable - using aesthetic

You can convey information about your data in your plot to the variables in your dataset. For example, you can map the colors of your points to the class variable to represent different categories. This choice of colors, and adds a legend, automatically.

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

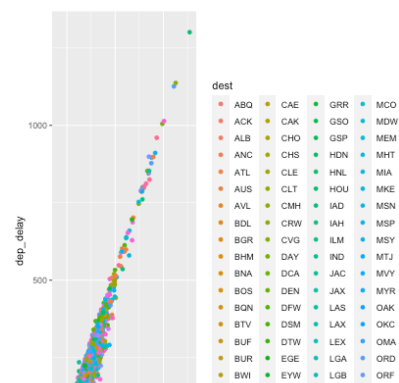


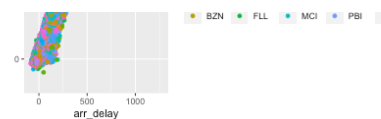


## Coloring by a variable - using aesthetic

Note when there are a lot of classes or groups distinguished well

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





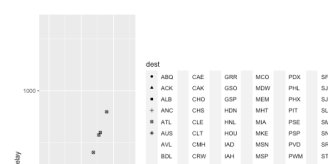
## Determining point type using a variable

By default ggplot uses up to 6 shapes. If there are more than 6 destinations, some points will not be plotted!! (At least it warns you.)

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

```
## Warning: The shape palette can deal with a maximum of 6
## more than 6 becomes difficult to discriminate; you have
## specifying shapes manually if you must have them.
```

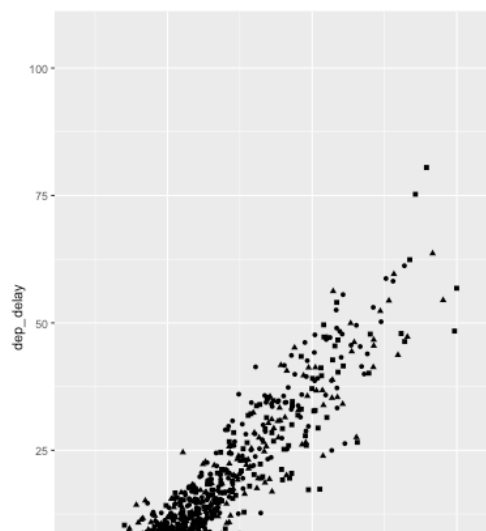
```
## Warning: Removed 316584 rows containing missing values
```



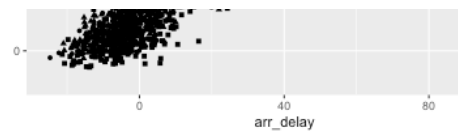


## Determining point type using a variable

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

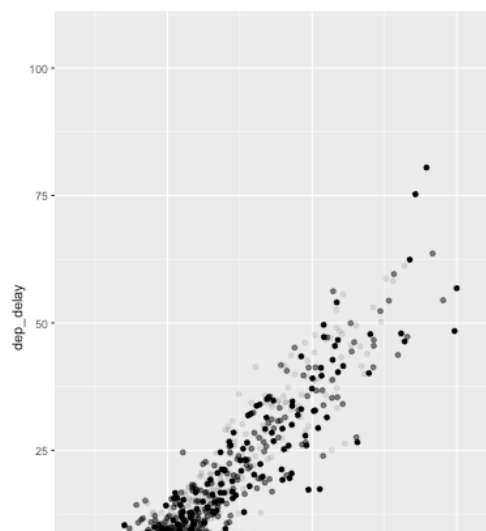


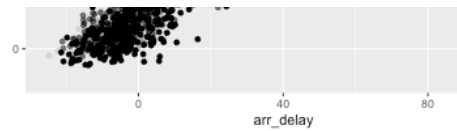




## Controlling point transparency using

```
ggplot(flights_weather_day, aes(x = arr_delay, y = dep_delay))  
  geom_point(alpha = 0.5)
```

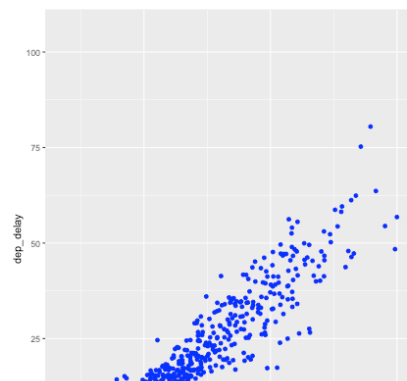


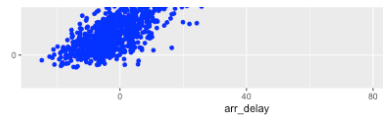


## Manual control of aesthetics

To control aesthetics manually, we do it in `geom_point()` name outside `aes()`

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





## Summary of aesthetics

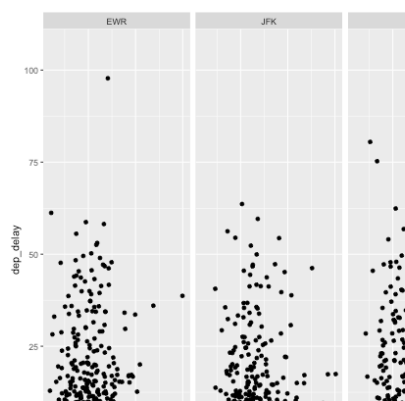
The various aesthetics...

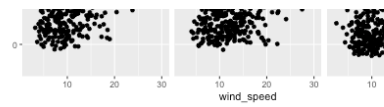
Code	Description
x	position on x-axis
y	position on y-axis
shape	shape
color	color of elements
fill	color inside elements
size	size
alpha	transparency
linetype	type of line

## Facets 1

Facets are particularly useful for categorical v

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

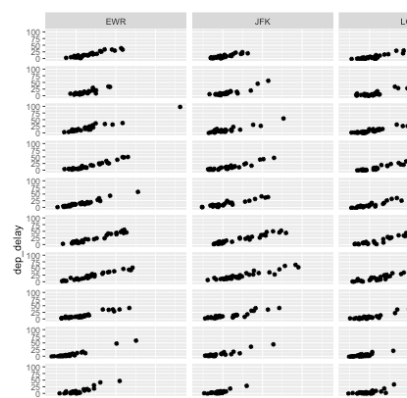


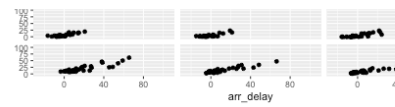


## Facets 2

Or you can facet on two variables...

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

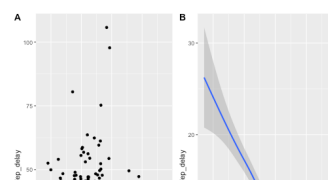


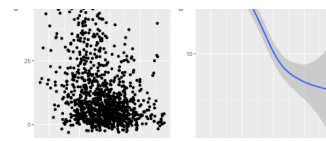


## Geometric Objects 1

Geometric objects are used to control the type of plot (e.g., scatterplot vs. lineplot) and the type of geometric object used for plotting a smoothed line fitted to the data (e.g., points vs. lines in 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")
```

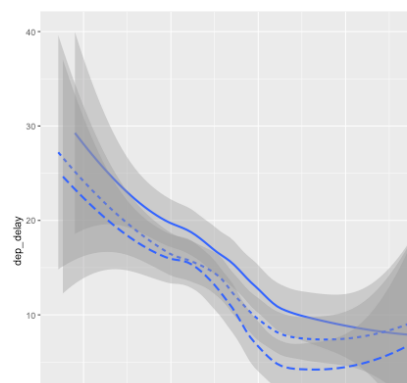


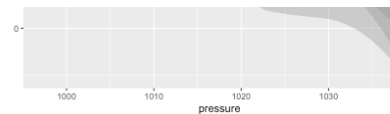


## Geometric Objects 2

Note that not every aesthetic works with every geom, but there are some new ones we can use: `linetype`

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
















# Geoms - Reference

ggplot2 provides over 40 geoms, and extensions add more (see <https://ggplot2.tidyverse.org/reference>)

The best way to get a comprehensive overview of all geoms is the [ggplot2 geom reference](https://posit.co/resources/ggplot2-geom-reference/), which you can find at <https://posit.co/resources/ggplot2-geom-reference/>

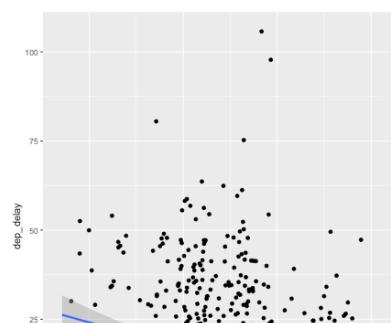
Two Variables	
Continuous X, Continuous Y f <- ggplot(mpg, aes(cty, hwy))	Continuous Bivariate i <- ggplot(movies, aes(rank, gross))
 f+ geom_blank() x, y, alpha, color, fill, shape, size	 i+ geom_bin2d() xmax, xmin, ymax, ymin, linetype, size, weight
 f+ geom_jitter() x, y, alpha, color, fill, shape, size	 i+ geom_density() x, y, alpha, colour, fill, linetype, size, weight
 f+ geom_point() x, y, alpha, color, fill, shape, size	 i+ geom_hex() x, y, alpha, colour, fill, linetype, size, weight
 f+ geom_quantile() x, y, alpha, color, linetype, size, weight	 j+ geom_area() x, y, alpha, color, fill, linetype, size, weight
 f+ geom_rug(sides = "bl") alpha, color, linetype, size	 j+ geom_line() x, y, alpha, color, fill, linetype, size, weight
 f+ geom_smooth(model = lm) x, y, alpha, color, fill, linetype, size, weight	

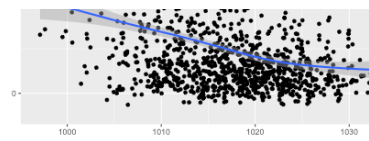


# Multiple Geoms 1

To display multiple geoms in the same plot, a  
`ggplot()`:

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

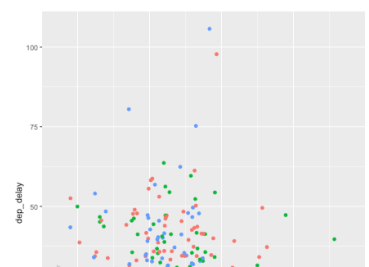




## Multiple Geoms 2

If you place mappings in a `geom` function, `ggplot` will extend or overwrite the global mappings for that layer. It is 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()
```

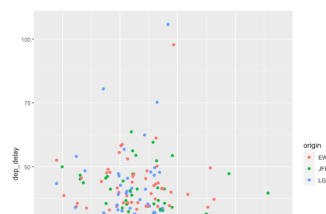


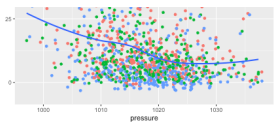


## Multiple Geoms 3

You can use the same idea to specify different smooth line displays just a subset of the data. JFK. The local data argument in `geom_smooth()` 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"))
```

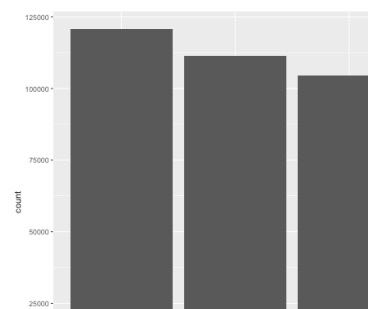


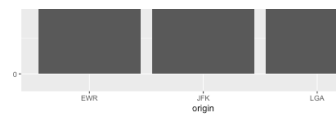


## Statistical Transformations –

Let's make a bar chart of the number of flights  
The algorithm uses a built-in statistical transform  
calculate the counts.

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

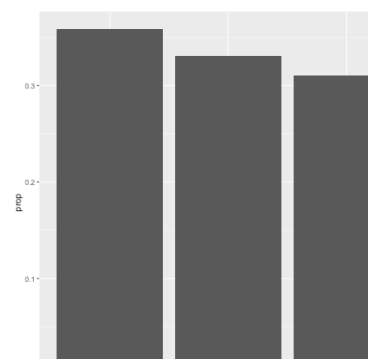




## Bar charts 2

You can override the statistic a geom uses to compute the y values. If you want to plot proportions, rather than counts:

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

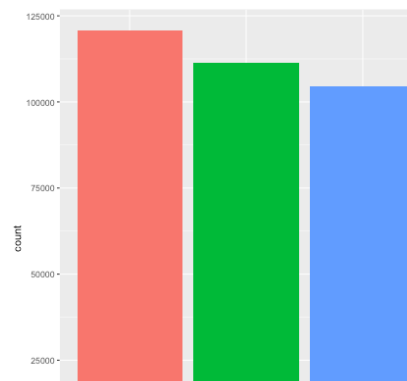


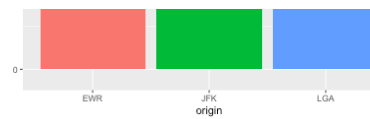


## Coloring barcharts

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

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





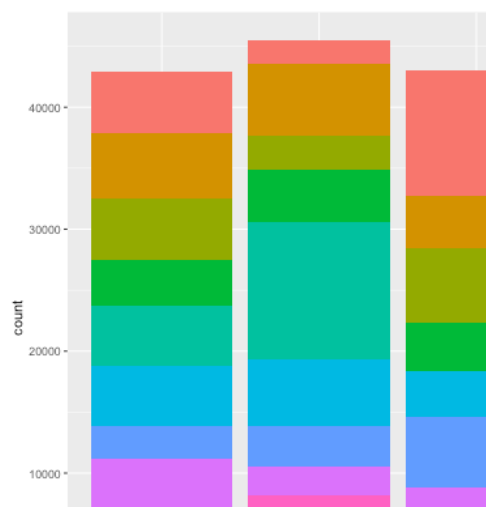
## Coloring barcharts

More interestingly, you can fill by another variable to look at the destination airports with a lot of flights)

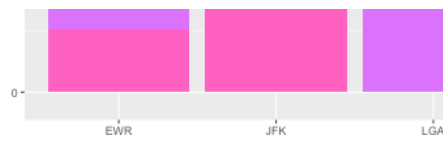
```
flights_weather_ss <- flights_weather |>  
  group_by(dest) %>%  
  filter(n() > 10000)
```

## Coloring barcharts

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



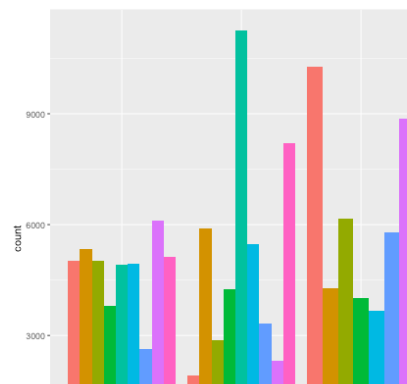


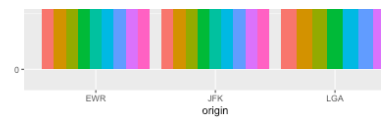


## Coloring barcharts

The `position = "dodge"` places overlapping objects next to each other. This makes it easier to compare individual

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



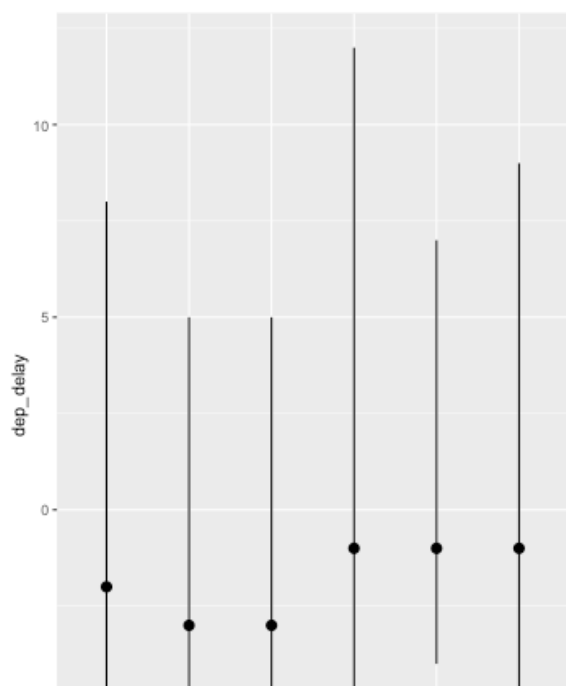


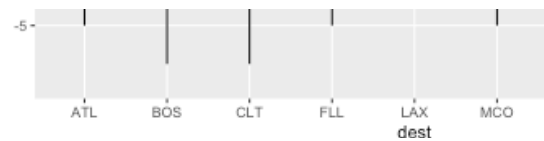
## Statistical transformations - another

You might want to draw greater attention to the distribution of the y values in your code. For example, you might use `stat_summary()` to draw the y values for each unique x value, to draw a boxplot or other summary 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

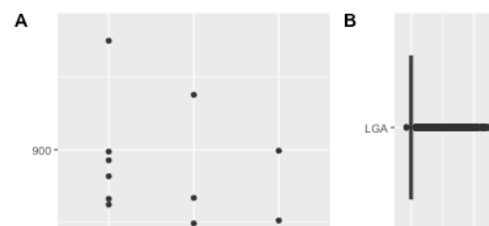


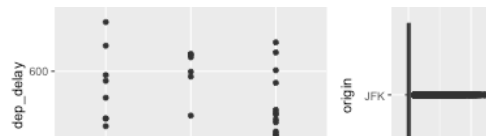


## Coordinate systems

Coordinate systems are one of the more common things you can do in ggplot2. To start with something simple, here's how to flip

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





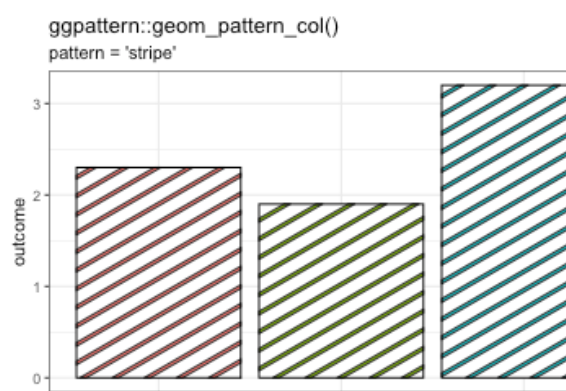
## A Great reference

A great (comprehensive) reference for everything  
the R Graphics Cookbook:

<https://r-graphics.org/>

Finally, file under "useless but cool"

ggpattern - is a library for adding pattern fills



a

b

level

c