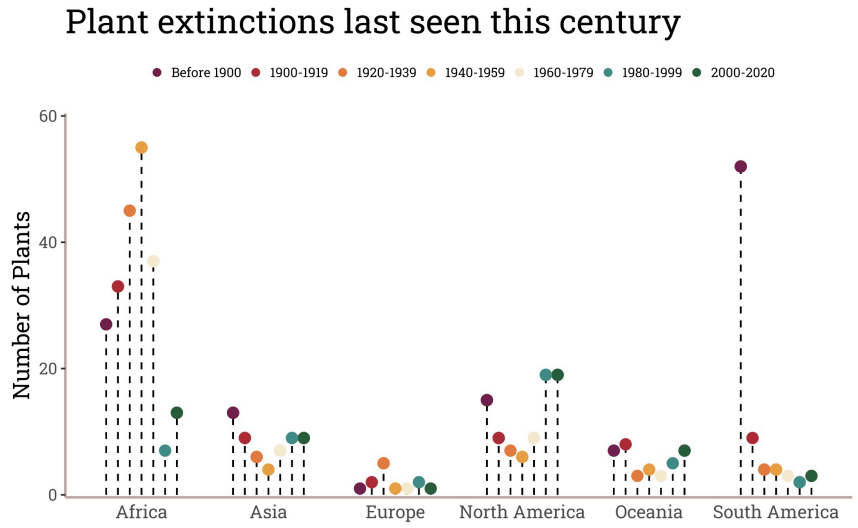
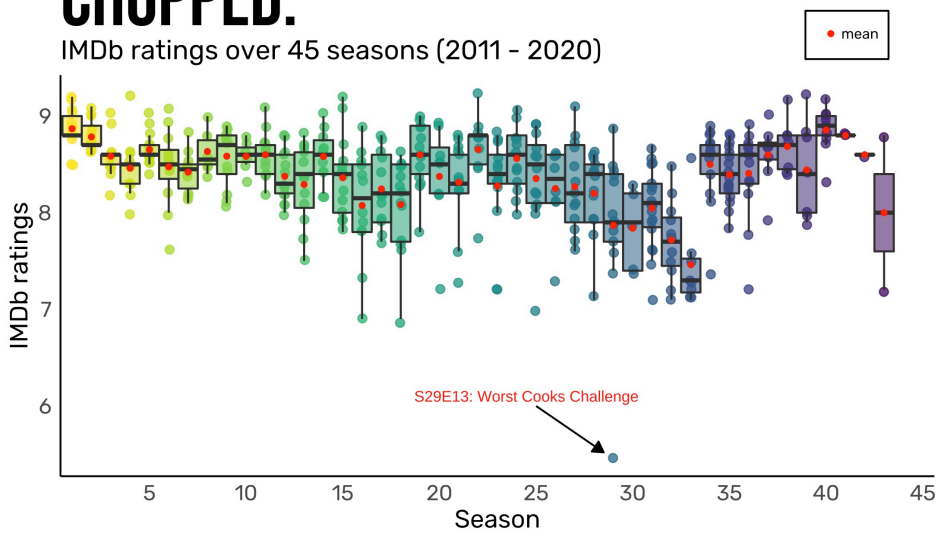



```r
{r}
test %>%
  mutate(DO = ifelse(Defense > Attack,"Defensive", "Offensive")) %>%
  filter(DO == "Defensive") %>%
  ggplot(aes(x = Defense, y = Attack)) +
  geom_point()
```



INTRO TO TIDYVERSE:

Down the %>% pipe!

KELLY MORROW

NACS METHODS SEMINAR

APRIL 2, 2021

GOALS OF THIS TUTORIAL

- Get a basic grasp of how to use tidyverse packages to "wrangle" datasets

- Quickly create beautiful plots from said datasets

- Give some examples to apply to your own data in R

RESOURCES FOR YOU:

- Accompanying R markdown file with all code referenced.

  `# Heavily commented and explanations`

You'll need `Rstudio` to run the `.Rmd` file,
Base R to use the `.R` script version

- Dataset used so all code will run for you.

- This PowerPoint.

  `# Links for programs mentioned are in the presenter notes`

- Some links I've found helpful while learning.

All can be found on: [github page]

MY BACKGROUND WITH R

- Self-taught, first coding language learned starting in 2016.
- Have been using ggplot to visualize data for ~ 4 years.
- Got into using dplyr for data-wrangling in the past year.
- Now create plots for lab publications and am considered the "go to" person for data viz in the lab.

THUS, LEARNING TO USE dplyr & ggplot MAY MAKE YOU THE DATA VIZ SUPERSTAR OF YOUR LAB, PROCEED WITH CAUTION!!

"WHAT ARE YOU EVEN TALKING ABOUT?"

`tidyverse`

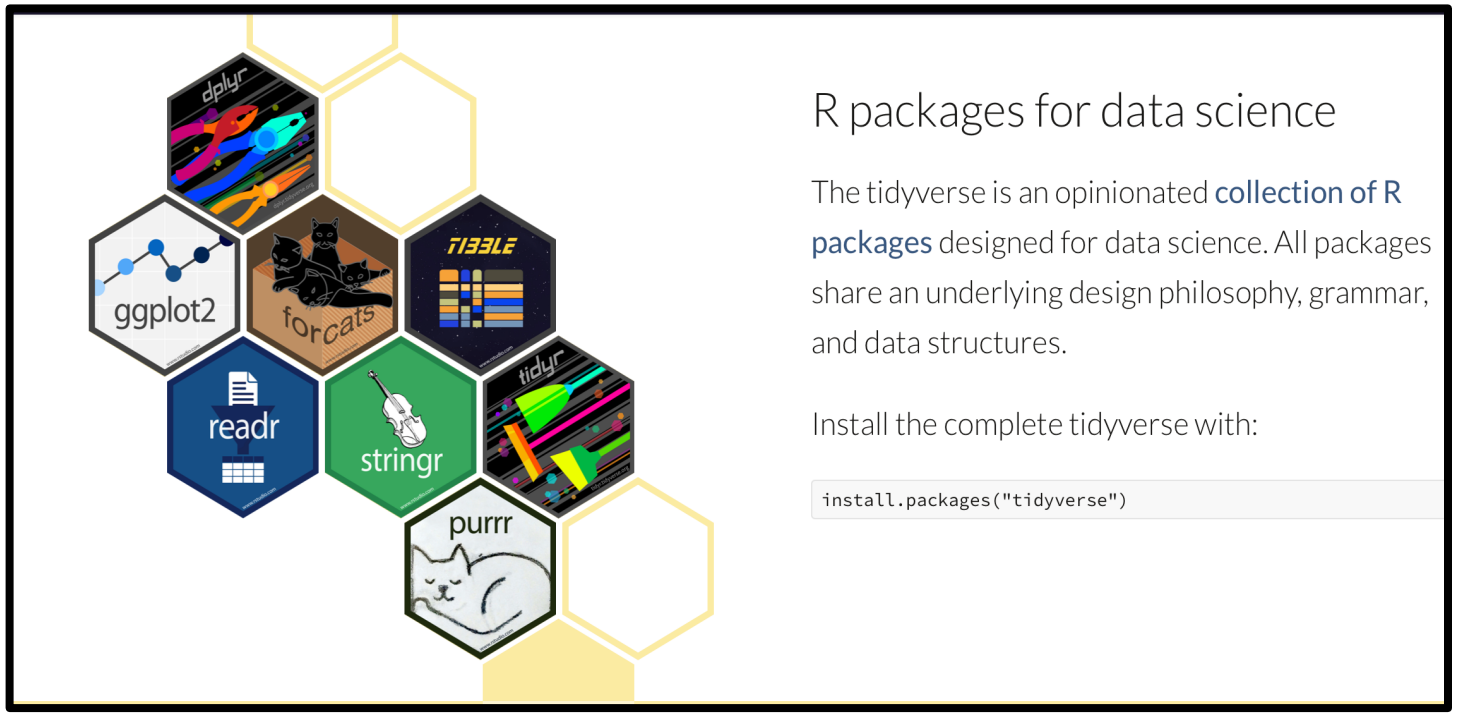
A collection of packages designed to make data science in R easier and prettier.

`dplyr`

Specific package to manipulate and "wrangle" your data for analyses & visualization

`ggplot2`

Specific package to create a multitude of customizable plots of your data



CHECK OUT THE TIDYVERSE WEBSITE: https://www.tidyverse.org

BEFORE WE MOVE TO THE FUN STUFF...

1. SPECIFY PACKAGES

```{r}
# install packages if needed:
install.packages('tidyverse')

# tell R to use tidyverse for this session.
library(tidyverse)
```

REMEMBER THAT `tidyverse` INCLUDES `dplyr` AND `ggplot`, ALONG WITH OTHER PACKAGES. WE COULD SPECIFICALLY INPUT THESE PACAKGES RATHER THAN THE ENTIRE TIDYVERSE BUT I AM LAZY!

2. IMPORT DATA

Today we're importing a .CSV file that lists Pokemon and their corresponding stats

Import a dataset as a dataframe and save as an R variable (df)

`read.delim()` is one of many import functions, we could also use `read.CSV()` or `read.table()`

```{r}
df <- read.delim("Pokemon.csv",
                 sep = ",",                    # tell R cells are separated by commas
                 na.strings = c("", "NA"),     # fill missing values with "NA" character
                 stringsAsFactors = F)         # do not automatically make variables factors


head(df)                                       # look at first 6 rows of the dataframe
```

There are many options you can use to ensure that your dataframe imports properly.

preview your newly imported dataframe

BEFORE WE MOVE TO THE FUN STUFF...

```
head(df)                                    # look at first 6 rows of the dataframe
```

Description: df[,12] [6 × 12]

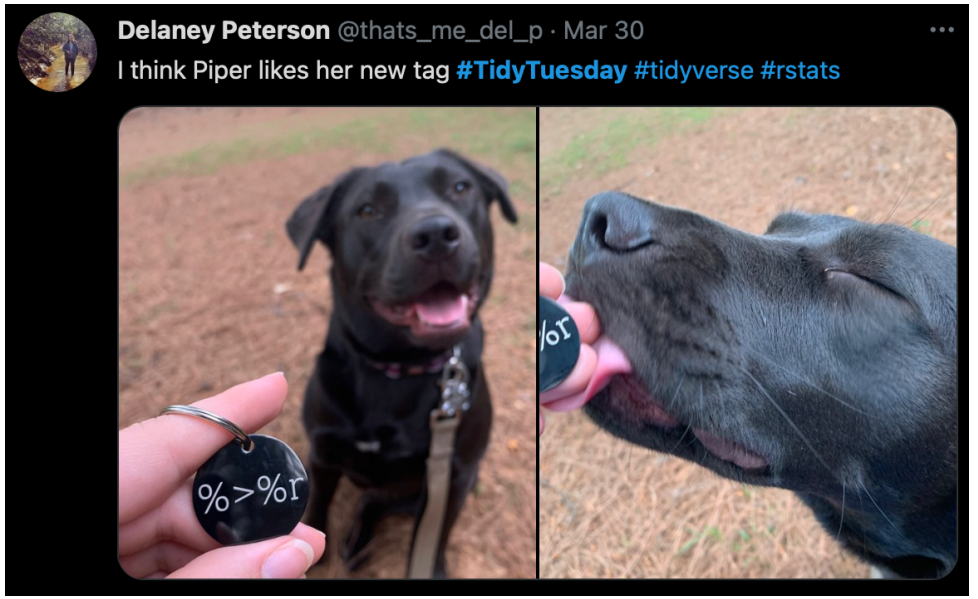
	X. <int>	Name <chr>	Type.1 <chr>	Type.2 <chr>	HP <int>	Attack <int>	Defense <int>	Sp..Atk <int>
1	1	Bulbasaur	Grass	Poison	45	49	49	65
2	2	Ivysaur	Grass	Poison	60	62	63	80
3	3	Venusaur	Grass	Poison	80	82	83	100
4	3	Venusa...	Grass	Poison	80	100	123	122
5	4	Charma...	Fire	*NA*	39	52	43	60
6	5	Charme...	Fire	*NA*	58	64	58	80

6 rows | 1-9 of 12 columns

dplyr: DOWN THE PIPE!

- dplyr uses a variety of verbs and pipes (%>%) to transform the user's data

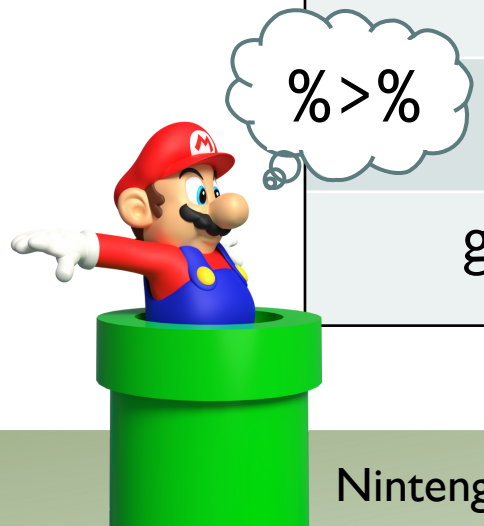
- What is a pipe?
  # %>% which can be thought of as saying "and then"
  # Allows you to string multiple verbs together
  # Quicker than creating "for" or "if/then" loops



dplyr: VERBS

```
dataframe %>% arrange(name)
```

Verb	Description
`select()`	Select specific columns in dataframe
`mutate()`	Create a new column
`summarize()`	Summarize column values
`filter()`	Filter rows of specific values
`arrange()`	Re-order dataframe based on a row
`group_by()`	Allow for group operations



dplyr: GETTING SUMMARY STATISTICS

We want to summarize how Attack stats differ between Pokémon types

STEPS:

1. Select columns wanted to perform actions (optional)
2. Filter rows where Generation value equals 1
3. Group by the Type.1 variable
4. Calculate statistics

```{r}
summary <-  df %>%

            select(-X.) %>%

            filter(Generation == 1) %>%

            group_by(Type.1) %>%

            summarize(n = n(),
              mean.Attack = mean(Attack),
              sd.Attack = sd(Attack))
```

dplyr: GETTING SUMMARY STATISTICS

We want to summarize how Attack stats differ between Pokémon types

```{r}
summary <-  df %>%

            select(-X.) %>%

            filter(Generation == 1) %>%

            group_by(Type.1) %>%

            summarize(n = n(),
              mean.Attack = mean(Attack),
              sd.Attack = sd(Attack))
```

	Type.1	n	mean.Attack	sd.Attack
1	Bug	14	76.42857	45.50691
2	Dragon	3	94.00000	36.05551
3	Electric	9	62.00000	22.27106
4	Fairy	2	57.50000	17.67767
5	Fighting	7	102.85714	18.67644
6	Fire	14	88.64286	26.56373
7	Ghost	4	53.75000	14.36141
8	Grass	13	72.92308	21.14025
9	Ground	8	81.87500	25.20452
10	Ice	2	67.50000	24.74874
11	Normal	24	70.62500	26.57363
12	Poison	14	74.42857	19.21423
13	Psychic	11	79.18182	52.92980
14	Rock	10	87.50000	32.25334
15	Water	31	74.19355	29.44307

dplyr: CREATING NEW VARIABLES

What if we wanted to calculate the total stat points for each Pokémon?

- We can easily create new columns using `mutate()` using this format:

```
mutate(new.variable = function(data)
```

```{r}

df <- df %>%
      group_by(Name) %>%
      mutate(Total = sum(HP, Attack, Defense, Sp..Atk, Sp..Def, Speed))

# side note: if you need to ever implement just one dplyr verb, you can do so without pipes!

mutate(df, Total = sum(HP, Attack, Defense, Sp..Atk, Sp..Def, Speed))
```

dplyr: CREATING NEW VARIABLES

```{r}

df <- df %>%
      group_by(Name) %>%
      mutate(Total = sum(HP, Attack, Defense, Sp..Atk, Sp..Def, Speed))

# side note: if you need to ever implement just one dplyr verb, you can do so without pipes!

mutate(df, Total = sum(HP, Attack, Defense, Sp..Atk, Sp..Def, Speed))
```

A tibble: 800 x 13 | Groups: Name [800]

X. <int>	Name <chr>	Type.1 <chr>	Type.2 <chr>	HP <int>	Attack <int>	Defense <int>	Sp..Atk <int>	Sp..Def <int>	Speed <int>	Generation <int>	Legendary <lgl>	Total <int>
1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	FALSE	318
2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	FALSE	405
3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	FALSE	525
3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	1	FALSE	625
4	Charmander	Fire	*NA*	39	52	43	60	50	65	1	FALSE	309
5	Charmeleon	Fire	*NA*	58	64	58	80	65	80	1	FALSE	405
6	Charizard	Fire	Flying	78	84	78	109	85	100	1	FALSE	534
6	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	FALSE	634

dplyr: CREATING NEW VARIABLES

Perhaps we want to categorized Pokémon based on whether they have greater Defense or Attack stats.

- We also can use conditional statements to create new variables based on existing columns.
  - WITH REAL DATASETS WE WOULD PROBABLY HAVE FINER CLASSIFICATIONS

Because we have a dichotomous new variable (defensive/offensive), we can use `mutate()` and `ifelse()`

```r
{r}
df %>%
      group_by(Name) %>%
      mutate(DO = ifelse(Defense > Attack,"Defensive", "Offensive")) %>%
      filter(DO == "Defensive")
```

A tibble: 298 x 14 | Groups: Name [298]

Attack <int>	Defense <int>	Sp..Atk <int>	Sp..Def <int>	Speed <int>	Genera... <int>	Legend... <lgl>	Total <int>	DO <chr>
62	63	80	80	60	1	FALSE	405	Defensive
82	83	100	100	80	1	FALSE	525	Defensive
100	123	122	120	80	1	FALSE	625	Defensive
48	65	50	64	43	1	FALSE	314	Defensive
63	80	65	80	58	1	FALSE	405	Defensive
83	100	85	105	78	1	FALSE	530	Defensive
103	120	135	115	78	1	FALSE	630	Defensive
30	35	20	20	45	1	FALSE	195	Defensive

Turns out all Pokémon are Defensive by this definition… Oops!

MOVING FROM TRANSFORMATION TO PLOTTING

- We can save different versions of our data frame as we make our transformations, but we can also move straight into plotting

    # this comes in handy if you're simply grouping or performing transformations *just* to plot

```r
{r}
df %>%
  mutate(DO = ifelse(Defense > Attack,"Defensive", "Offensive")) %>%
  filter(DO == "Defensive") %>%
  ggplot(aes(x = Defense, y = Attack)) +
  geom_point()
```

BASICS OF ggplot

ggplot: INTRO

ggplot CREATES PLOTS IN LAYERS

visual items are referred to as **geoms**, or geometric objects

Do we want to use bars, dots, lines?

geoms have aesthetic items that control the appearance of the plot with `aes()`

ggplot: GEOMS

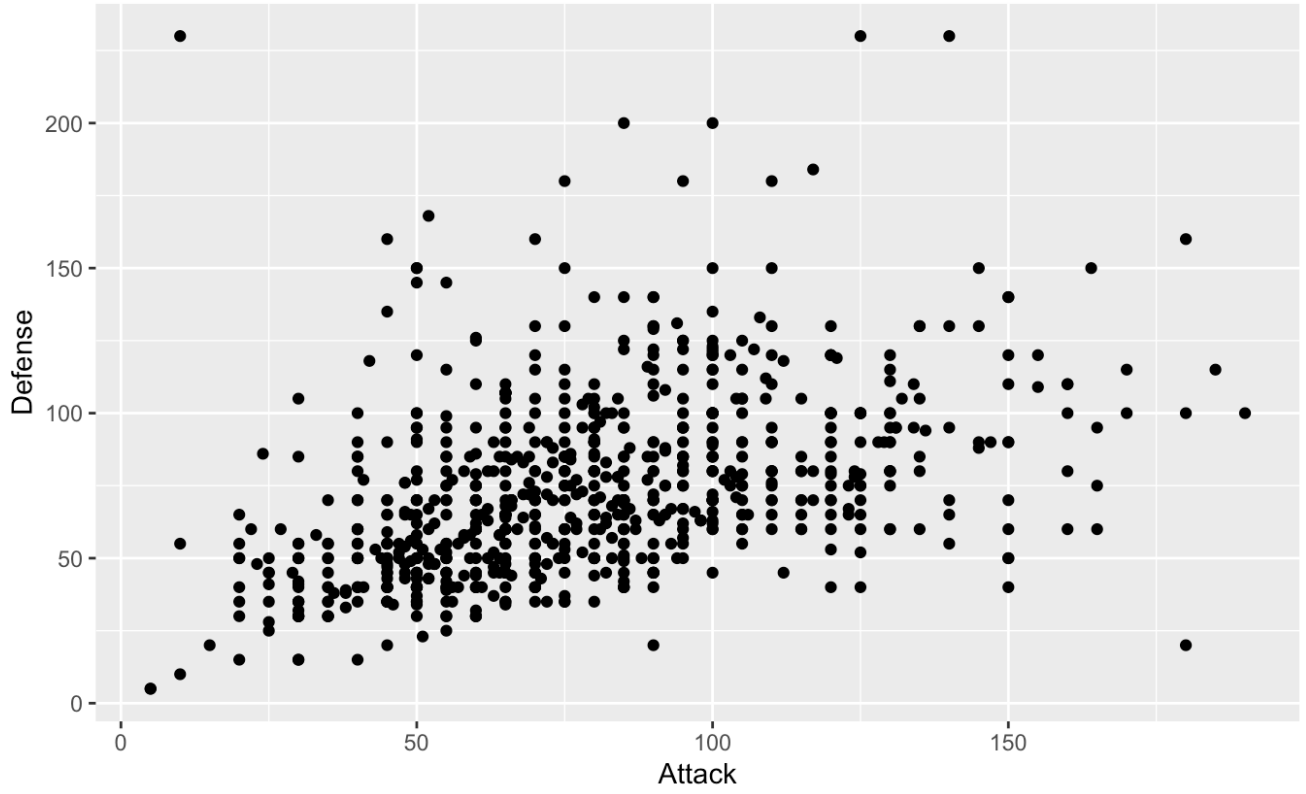
- Geoms come in many shapes
- We don't have to stick to just one!

```
geom_point()
geom_jitter()
geom_line()
geom_boxplot()
geom_violin()
geom_histogram()
geom_density()
geom_text()
geom_errorbar()
```

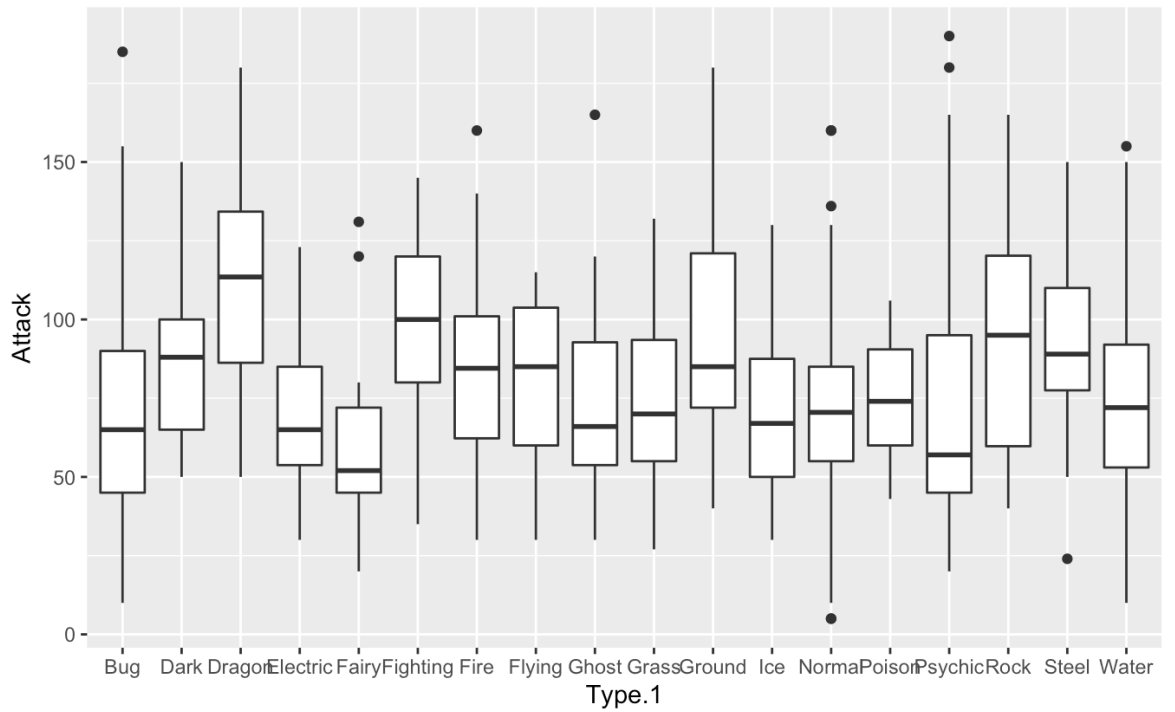
ggplot: BASIC STRUCTURE

Already not to shabby...

```{r}
ggplot(df, aes(x = Attack, y = Defense)) +
  geom_point()
```




```{r}
ggplot(df, aes(x = Type.1, y = Attack)) +
  geom_boxplot()
```



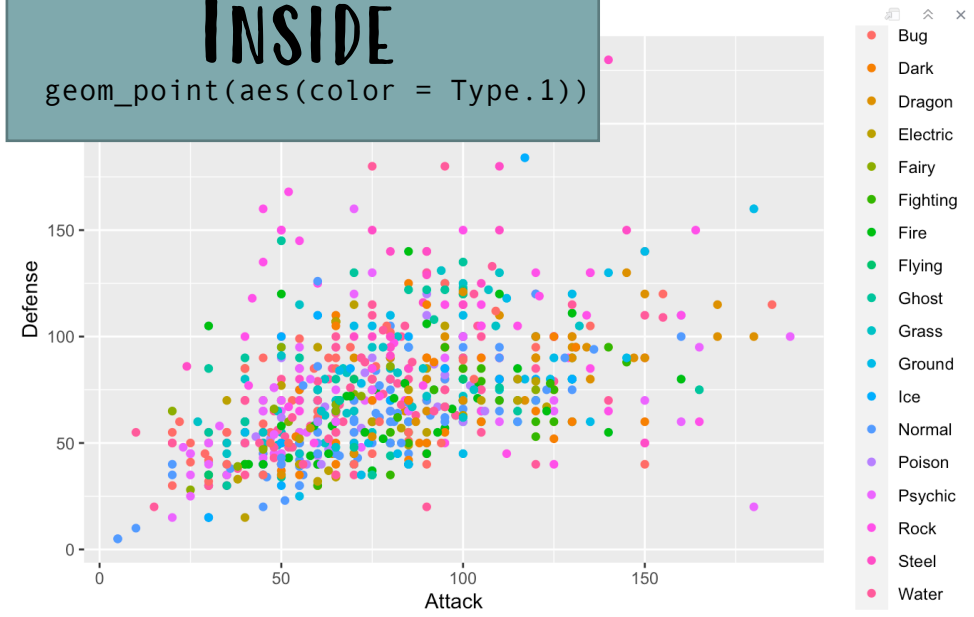
ggplot: AESTHETICS

- We can make basic plots, but it's obvious they could be improved upon
- Aesthetics can be used to vary positions, colors, shapes, sizes, transparencies

- **Here's the tricky part**: some aesthetics go inside parenthesis, some do not!
  - What determines this?

INSIDE

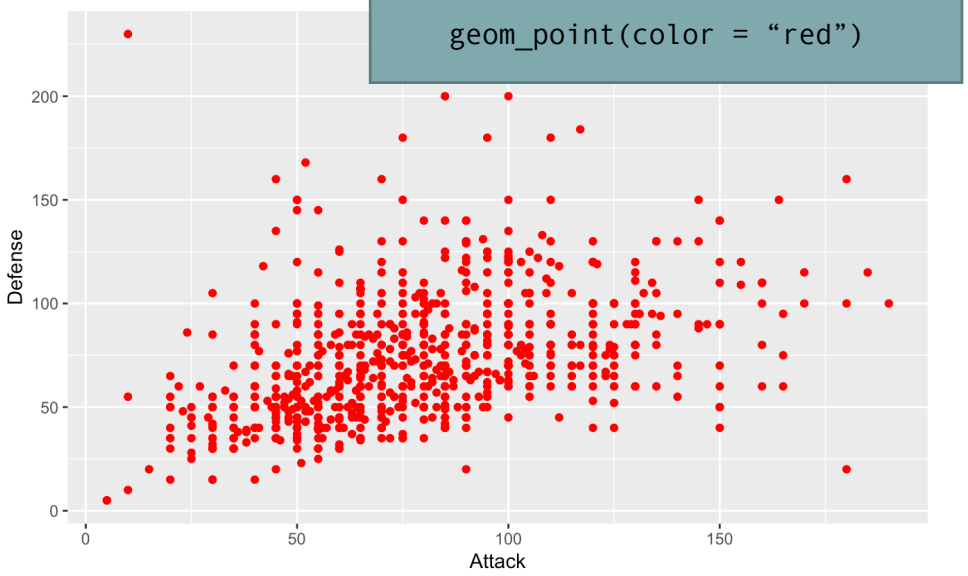
`geom_point(aes(color = Type.1))`



You want the aesthetic set based on a variable

OUTSIDE

`geom_point(color = "red")`



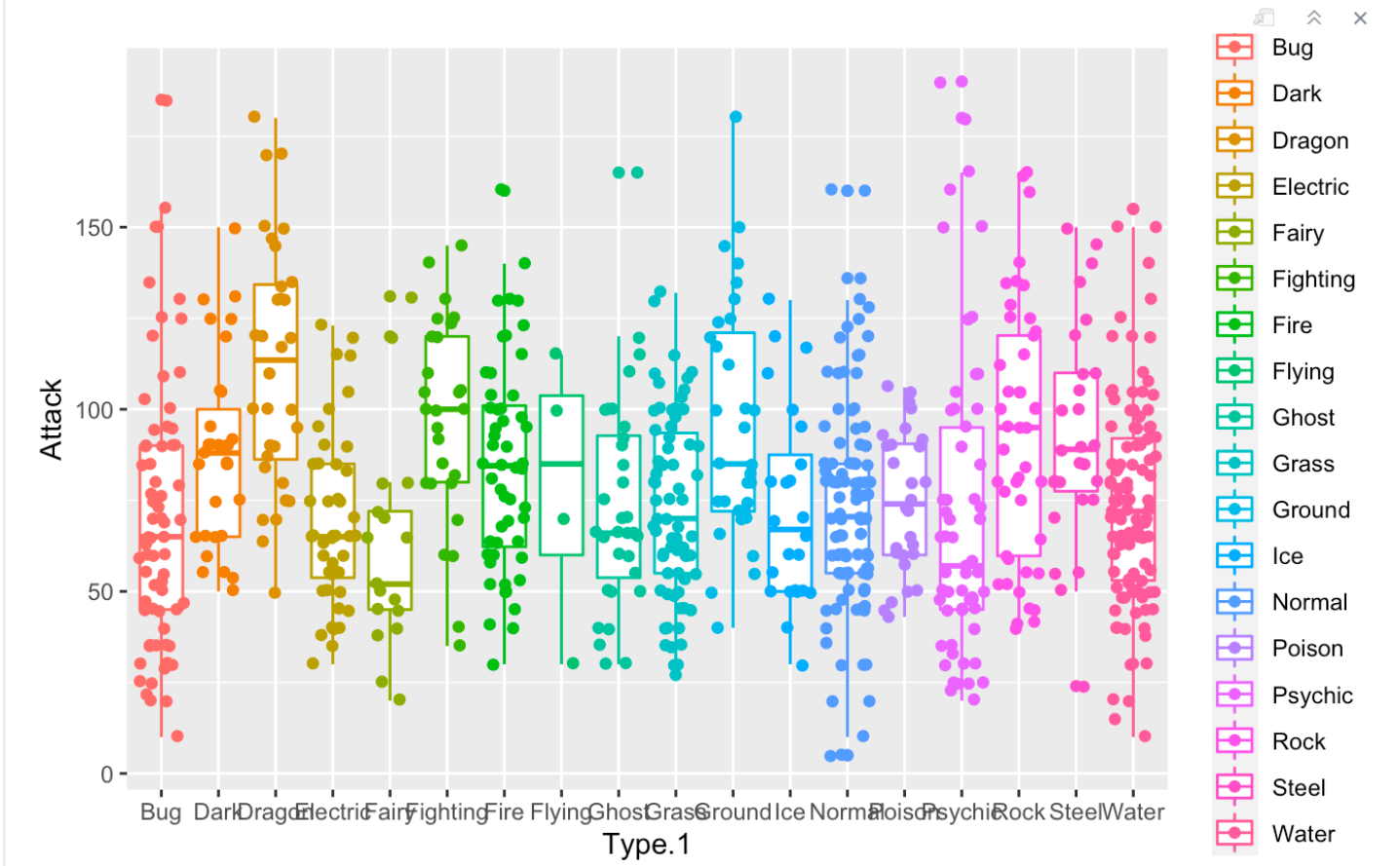
You specifically want to set the aesthetic

ggplot: AESTHETICS

- We can set global aesthetics in our set-up as well
  - All subsequent geoms will follow this aesthetic

Wow, okay that's a lot... but you get the point

```r
{r}
ggplot(df, aes(x = Type.1, y = Attack, color = Type.1)) +
  geom_boxplot() +
  geom_jitter()
```

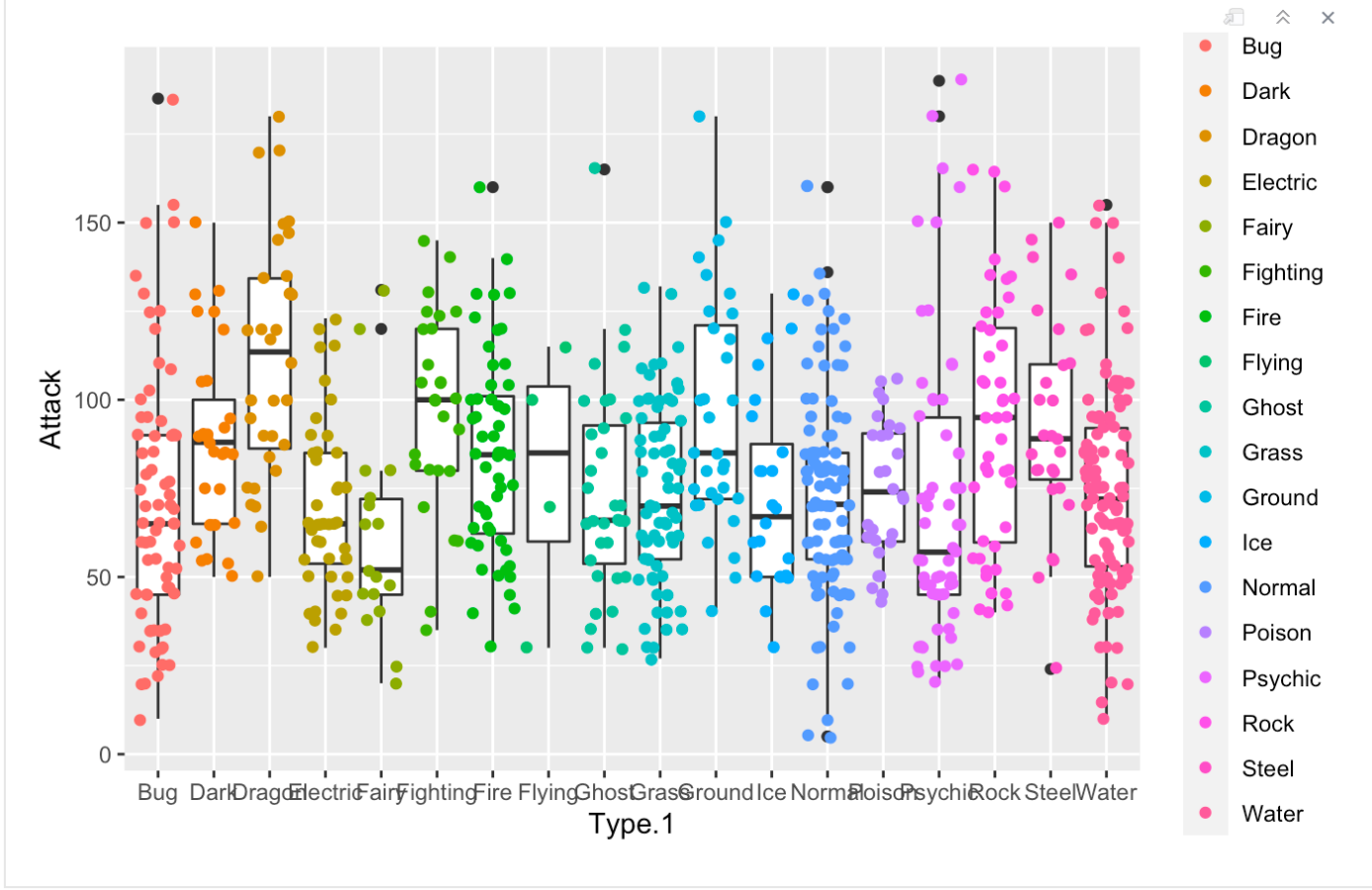


ggplot: AESTHETICS

- We can set global aesthetics in our set-up as well
  - All subsequent geoms will follow this aesthetic

- Sometimes it is obviously better to specify aesthetics within a geom

```r
{r}
ggplot(df, aes(x = Type.1, y = Attack)) +
  geom_boxplot() +
  geom_jitter(aes( color = Type.1))
```



ggplot: AESTHETICS

- We can make this plot better by modifying with *other* aesthetics outside of parentheses

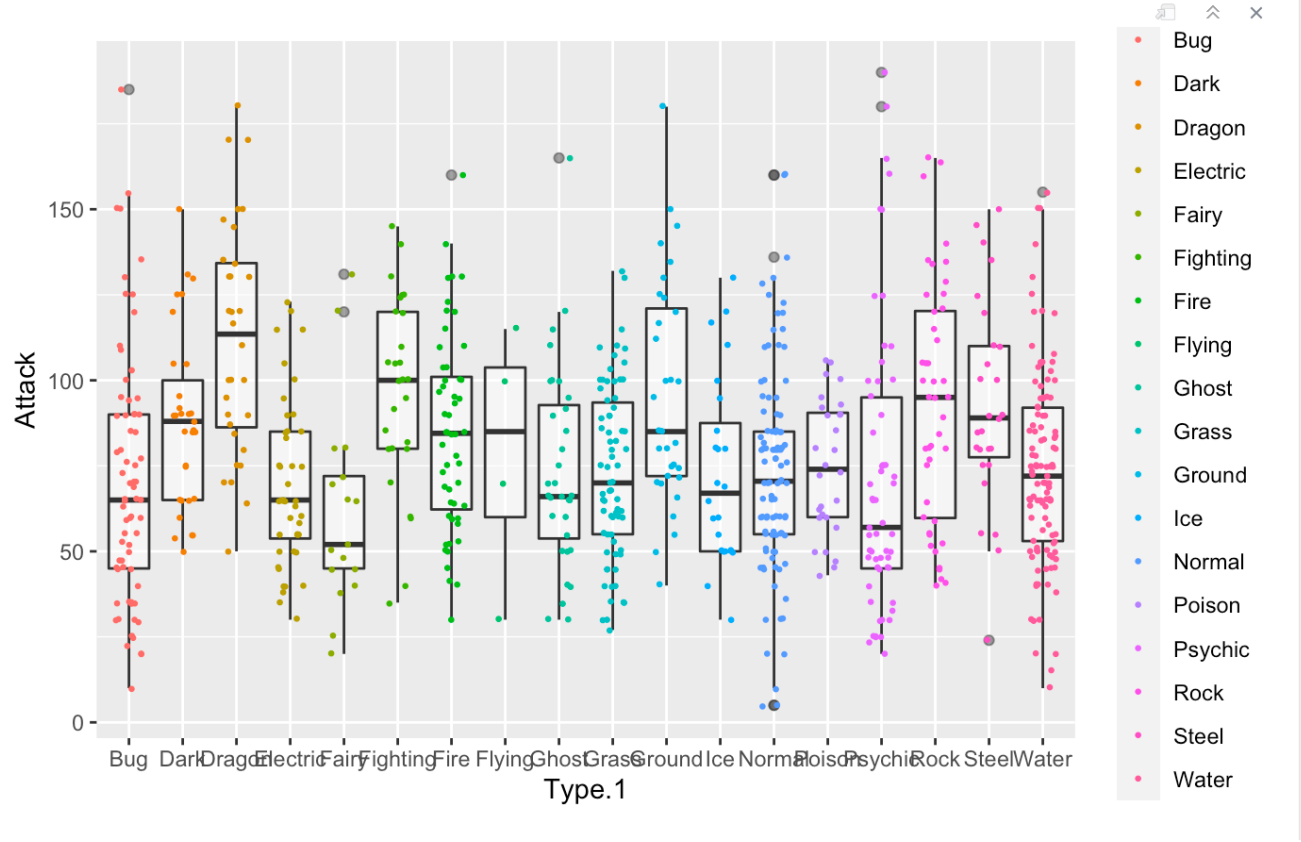
Things we can manipulate:

Transparency	`alpha`
Size	`size`
Line	`linetype`
Shape	`shape`

There are other aesthetics that are specific to a geom

- For example, we can set the width of a jitter for `geom_jitter()`, but you'll get an error if you try this with `geom_point()`

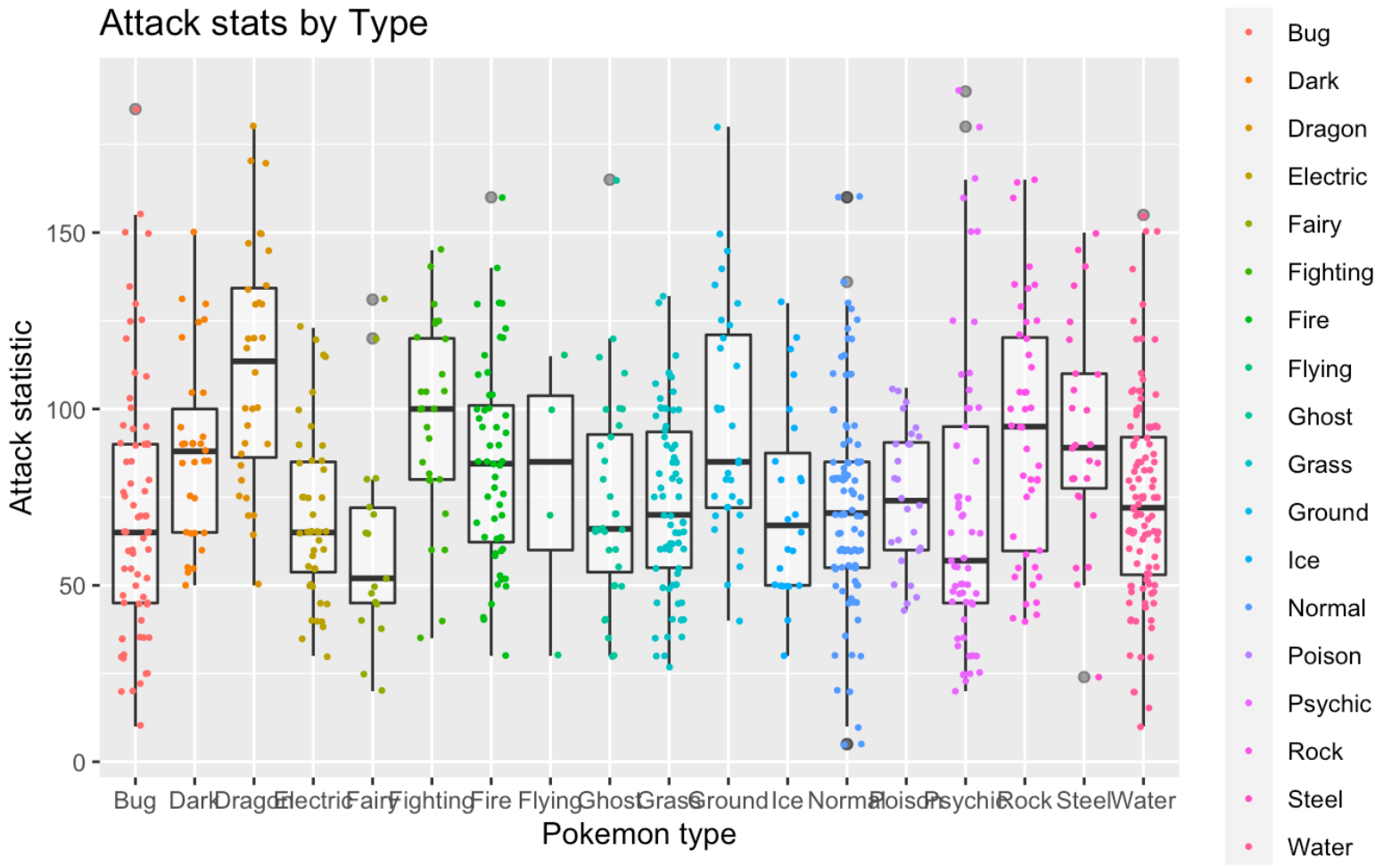
```r
{r}
ggplot(df, aes(x = Type.1, y = Attack)) +
  geom_boxplot(alpha = 0.5) +
  geom_jitter(aes( color = Type.1),
              width = 0.25,
              size = 0.5)
```



ggplot: LABELS

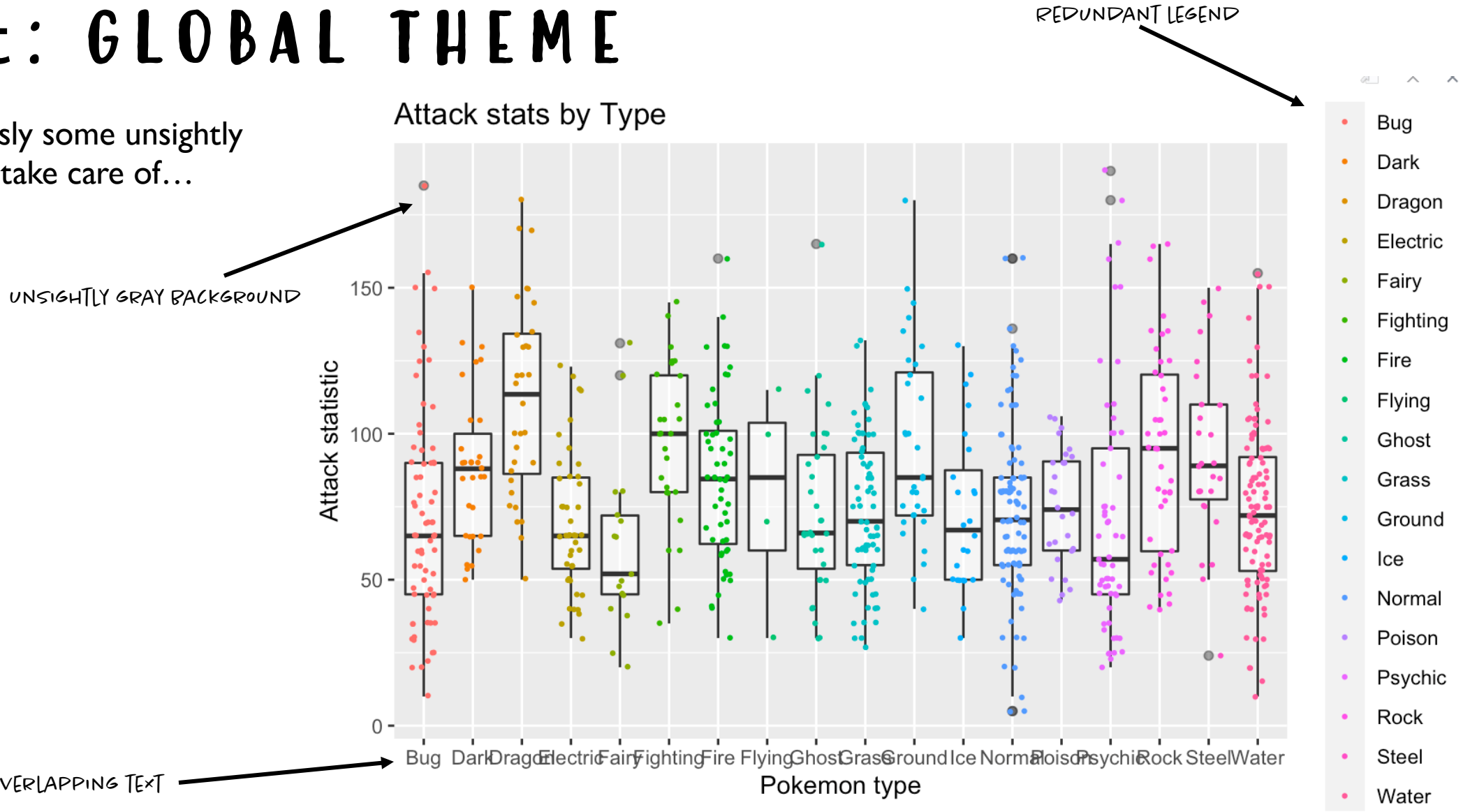
- We can also specify our labels quite simply, despite the name of the variable in the dataframe

```{r}
ggplot(df, aes(x = Type.1, y = Attack)) +
  geom_boxplot(alpha = 0.5) +
  geom_jitter(aes( color = Type.1),
              width = 0.25,
              size = 0.5) +
  labs(x = "Pokemon type",
       y = "Attack statistic",
       title = "Attack stats by Type")
```



ggplot: GLOBAL THEME

- There are obviously some unsightly things we should take care of…



ggplot: GLOBAL THEME

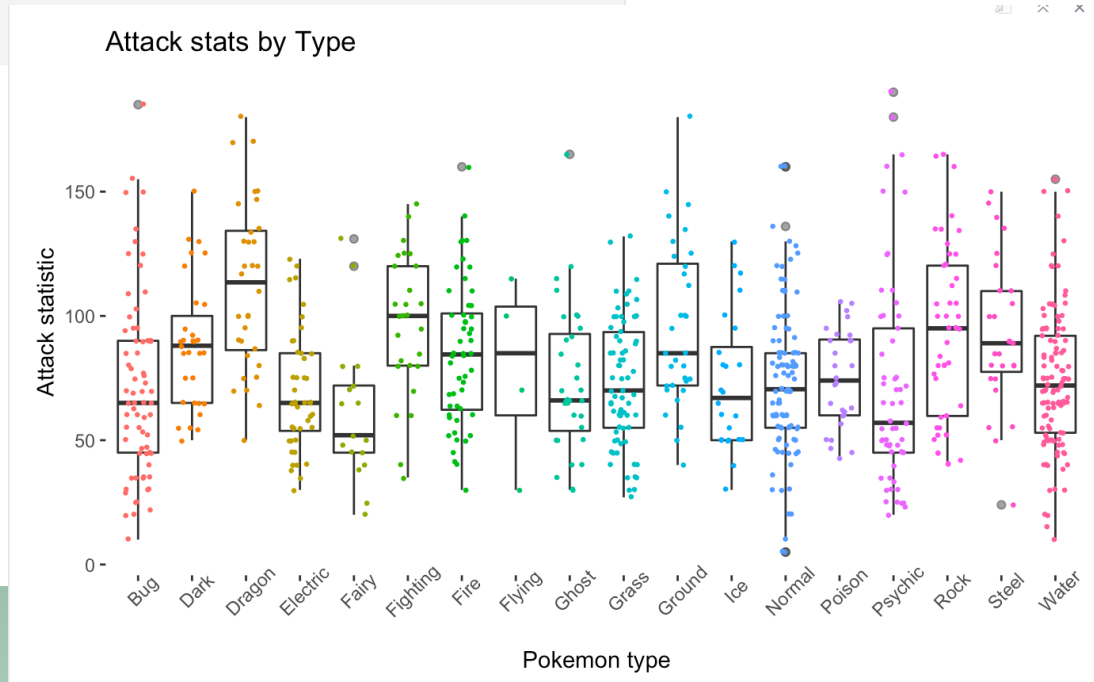
As the name implies, global theme options will change more general things about the plot

- Text sizes
- Text rotation
- How far font is from axis
- Background color
- Title height
- Where the legend goes
- Orientation of the legend
- Graph lines
- Size of legend
- Tick marks

ALMOST ANYTHING YOU CAN IMAGINE

You can explore options by typing ?theme()

```{r}
ggplot(df, aes(x = Type.1, y = Attack)) +
  geom_boxplot(alpha = 0.5) +
  geom_jitter(aes( color = Type.1),
              width = 0.25,
              size = 0.5) +
  labs(x = "Pokemon type",
       y = "Attack statistic",
       title = "Attack stats by Type") +
  theme(panel.background = element_rect(fill = "white"),
        legend.position = 'none',
        axis.text.x = element_text(angle = 45))
```



DPLYR & GGPLOT TOGETHER

Let's say we want to look at how stats differ between Pokémon types

- We want to create a stacked bar plot, which requires a bit of manipulation
- Stacked bar plots require what needs to be "stacked" in a single column, their labels in another

Steps

1. Create a dataframe that includes averages of each stat for each pokemon type
2. Transpose data from wide to long format
3. Plot!

STEP 1: CREATE AVERAGE DATAFRAME

```r
{r}
df %>%
  group_by(Type.1) %>%
  summarize(Avg.Total = mean(Total)
            Avg.HP = mean(HP),
            Avg.Attack = mean(Attack),
            Avg.Defense = mean(Defense),
            Avg.SpAtk = mean(Sp..Atk),
            Avg.SpDef = mean(Sp..Def)) %>%
  gather(key = "Stat", value = "Average", Avg.HP:Avg.SpDef)
```

A tibble: 90 x 4

Type.1 <chr>	Avg.Total <dbl>	Stat <chr>	Average <dbl>
Bug	378.9275	Avg.HP	56.88406
Dark	445.7419	Avg.HP	66.80645
Dragon	550.5312	Avg.HP	83.31250
Electric	443.4091	Avg.HP	59.79545
Fairy	413.1765	Avg.HP	74.11765
Fighting	416.4444	Avg.HP	69.85185
Fire	458.0769	Avg.HP	69.90385
Flying	485.0000	Avg.HP	70.75000
Ghost	439.5625	Avg.HP	64.43750
Grass	421.1429	Avg.HP	67.27143

1–10 of 90 rows Previous 1 2 3 4 5 6 ... 9 Next

We're using the `gather()` function which takes three arguments:

1. Key: name of new column storing old column names

2. **Value:** name of new column that will store the old column values

3. Which columns you would like to collapse

THE OPPOSITE OF `gather()` IS `spread()` WHICH TAKES YOUR DATAFRAME FROM LONG TO WIDE FORM

STEP 1: CREATE AVERAGE DATAFRAME

```r
df %>%
  group_by(Type.1) %>%
  summarize(Avg.Total = mean(Total),
            Avg.HP = mean(HP),
            Avg.Attack = mean(Attack),
            Avg.Defense = mean(Defense),
            Avg.SpAtk = mean(Sp..Atk),
            Avg.SpDef = mean(Sp..Def))
```

A tibble: 18 x 7

	Type.1 <chr>	Avg.Total <dbl>	Avg.HP <dbl>	Avg.Attack <dbl>	Avg.Defe... <dbl>	Avg.SpAtk <dbl>	Avg.SpDef <dbl>
1	Bug	378.9275	56.88406	70.97101	70.72464	53.86957	64.79710
2	Dark	445.7419	66.80645	88.38710	70.22581	74.64516	69.51613
3	Dragon	550.5312	83.31250	112.12500	86.37500	96.84375	88.84375
4	Electric	443.4091	59.79545	69.09091	66.29545	90.02273	73.70455
5	Fairy	413.1765	74.11765	61.52941	65.70588	78.52941	84.70588
6	Fighting	416.4444	69.85185	96.77778	65.92593	53.11111	64.70370
7	Fire	458.0769	69.90385	84.76923	67.76923	88.98077	72.21154
8	Flying	485.0000	70.75000	78.75000	66.25000	94.25000	72.50000
9	Ghost	439.5625	64.43750	73.78125	81.18750	79.34375	76.46875
10	Grass	421.1429	67.27143	73.21429	70.80000	77.50000	70.42857

1–10 of 18 rows

Previous 1 2 Next

Using `summarize()`

STEP 2: SWITCH TO LONG FORM

```{r}
df %>%
  group_by(Type.1) %>%
  summarize(Avg.Total = mean(Total)
            Avg.HP = mean(HP),
            Avg.Attack = mean(Attack),
            Avg.Defense = mean(Defense),
            Avg.SpAtk = mean(Sp..Atk),
            Avg.SpDef = mean(Sp..Def)) %>%
  gather(key = "Stat", value = "Average", Avg.HP:Avg.SpDef)
```

A tibble: 90 x 4

Type.1 <chr>	Avg.Total <dbl>	Stat <chr>	Average <dbl>
Bug	378.9275	Avg.HP	56.88406
Dark	445.7419	Avg.HP	66.80645
Dragon	550.5312	Avg.HP	83.31250
Electric	443.4091	Avg.HP	59.79545
Fairy	413.1765	Avg.HP	74.11765
Fighting	416.4444	Avg.HP	69.85185
Fire	458.0769	Avg.HP	69.90385
Flying	485.0000	Avg.HP	70.75000
Ghost	439.5625	Avg.HP	64.43750
Grass	421.1429	Avg.HP	67.27143

1–10 of 90 rows Previous 1 2 3 4 5 6 ... 9 Next

We're using the `gather()` function which takes three arguments:

1. Key: name of new column storing old column names

2. **Value:** name of new column that will store the old column values

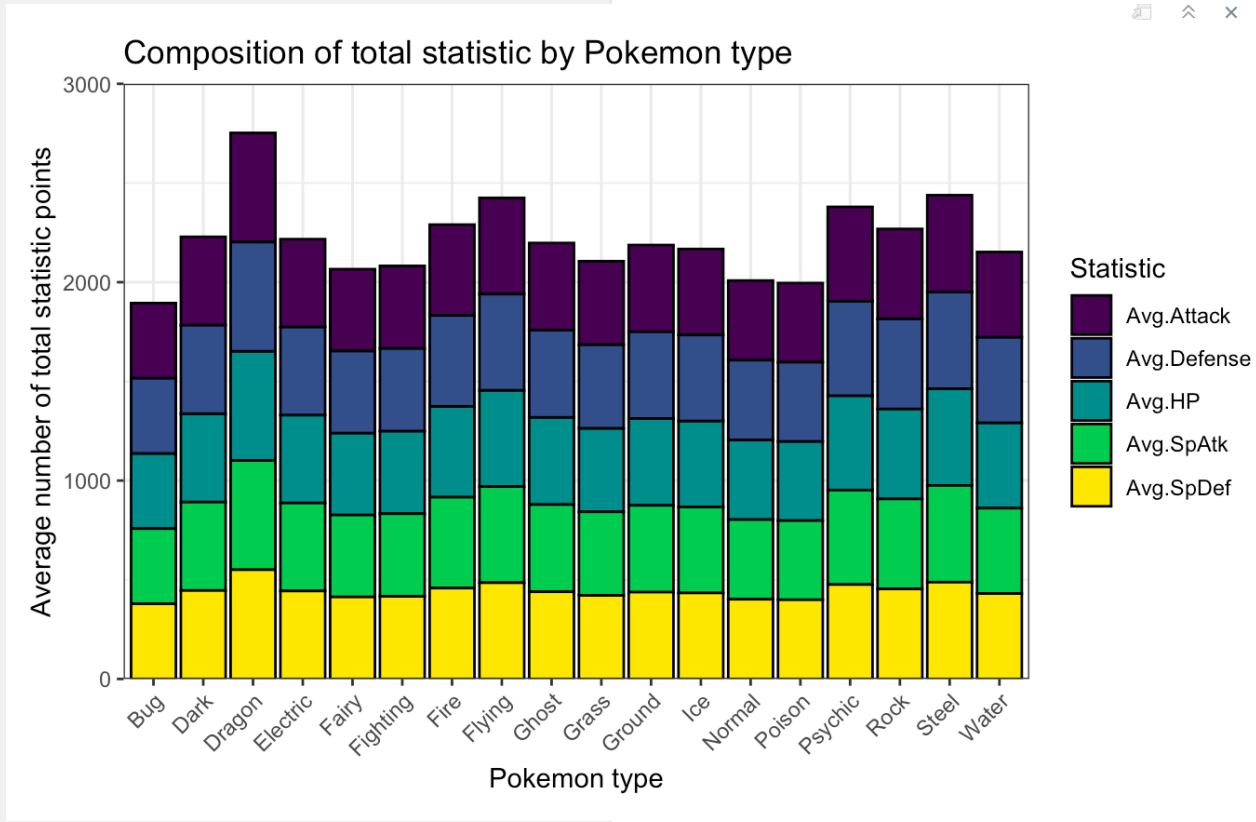
3. Which columns you would like to collapse

THE OPPOSITE OF `gather()` IS `spread()` WHICH TAKES YOUR DATAFRAME FROM LONG TO WIDE FORM

STEP 3: PLOT! MAKE IT FANCY!

I got a little fancy and added in some more plot details

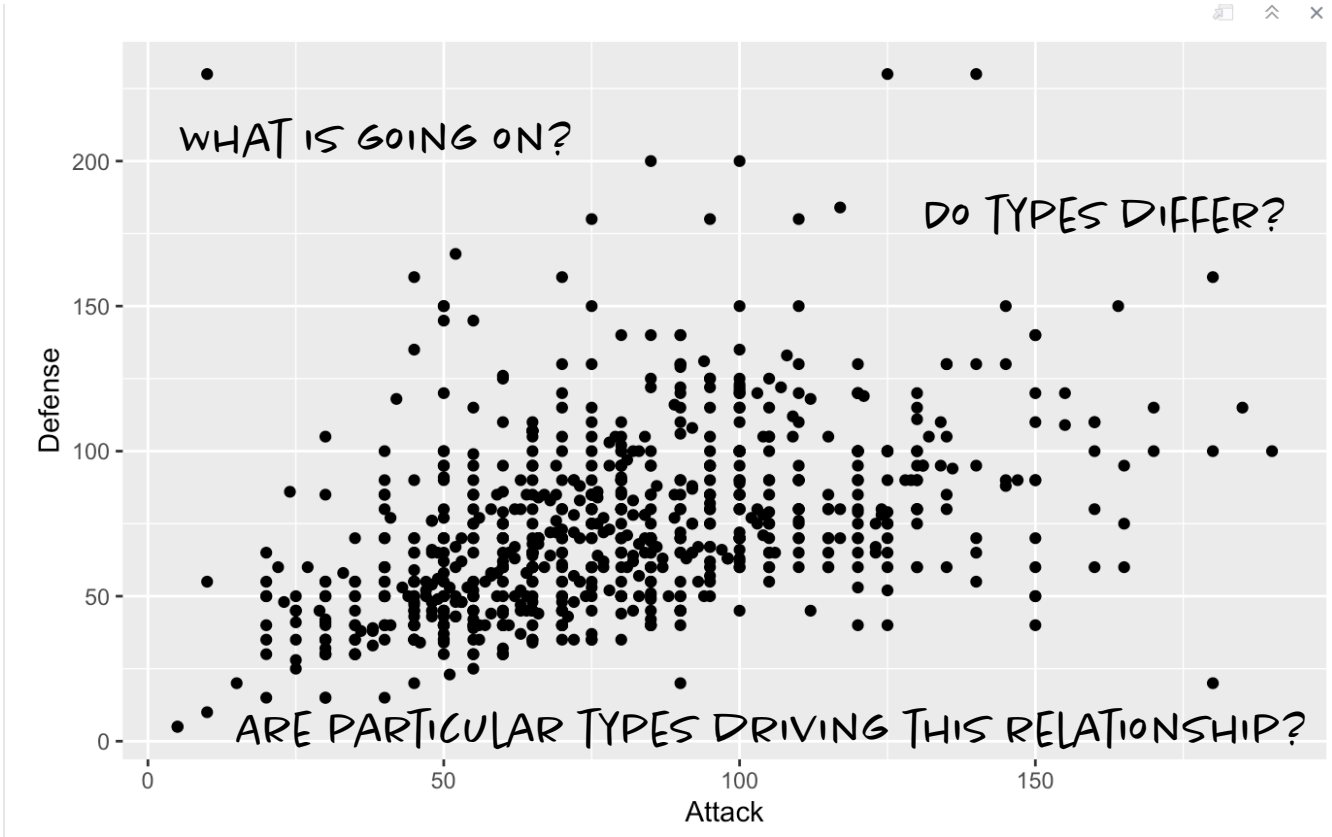
```r
{r}
library(viridis)

df %>%
  group_by(Type.1) %>%
  summarize(Avg.Total = mean(Total),
            Avg.HP = mean(HP),
            Avg.Attack = mean(Attack),
            Avg.Defense = mean(Defense),
            Avg.SpAtk = mean(Sp..Atk),
            Avg.SpDef = mean(Sp..Def)) %>%
  gather(key = "Stat", value = "Average", Avg.HP:Avg.SpDef) %>%

  ggplot(aes(x = Type.1, y = Avg.Total, fill = Stat)) +
  geom_bar(stat="identity", color = "black") +
  labs(x = "Pokemon type",
       y = "Average number of total statistic points",
       title = "Composition of total statistic by Pokemon type",
       fill = "Statistic") +
  coord_cartesian(ylim = c(0, 3000)) +
  scale_fill_viridis(discrete = TRUE) +
  scale_y_continuous(expand = c(0,0)) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



ggplot: FACET-NATING

Sometimes its better to display data in subplots to more clearly display results

For instance, there is a clear relationship between Defense and Attack stats, but can we get a clearer picture?

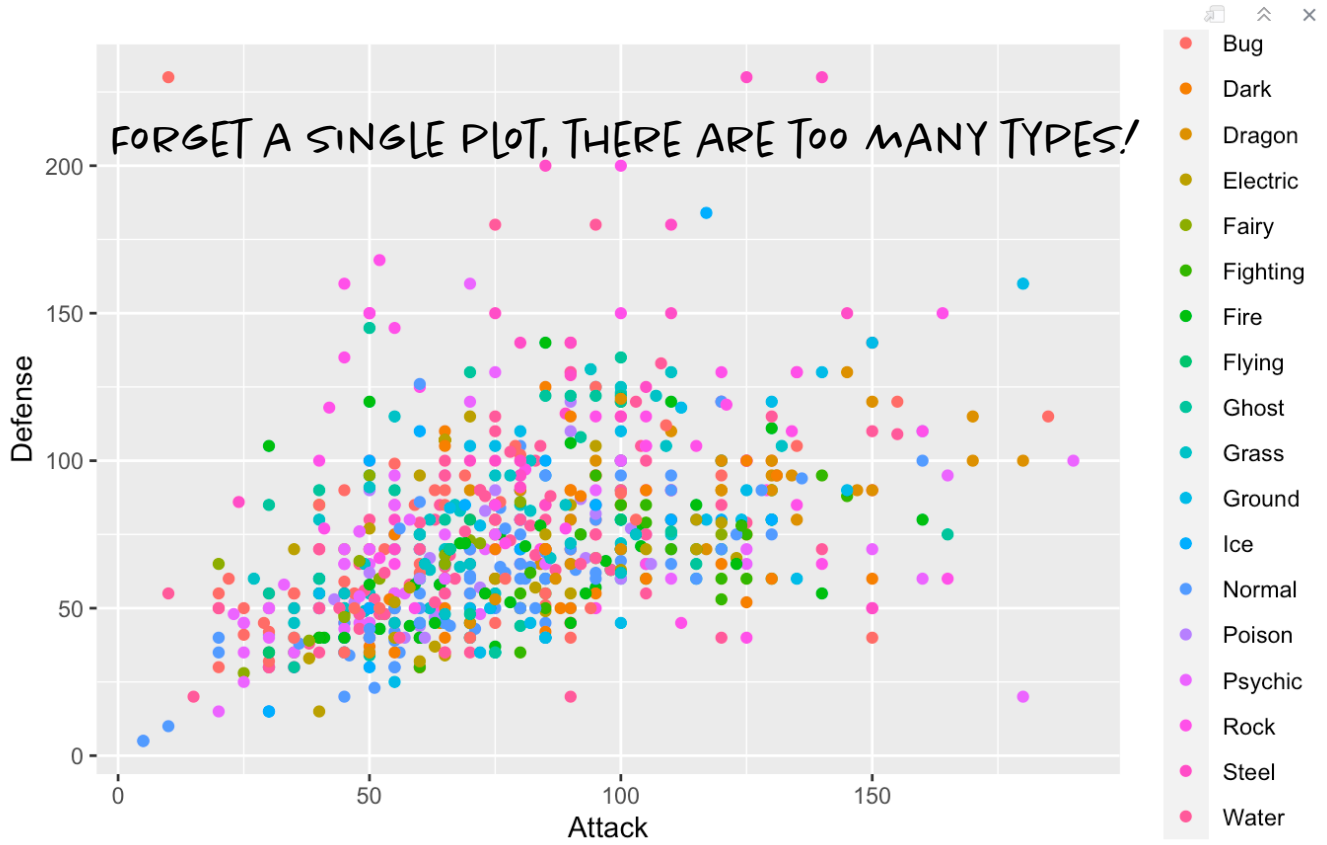


ggplot: FACET-NATING

Sometimes its better to display data in subplots to more clearly display results

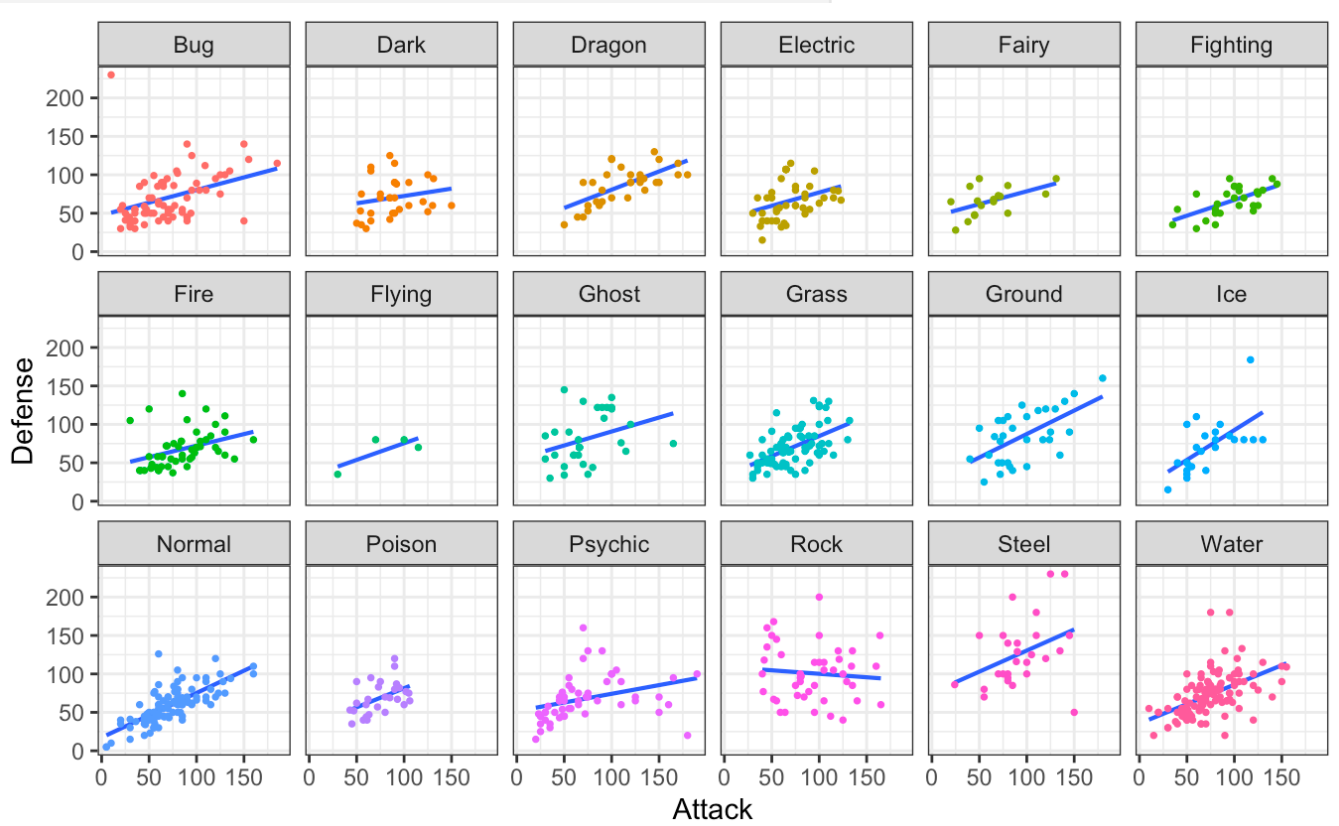
For instance, there is a clear relationship between Defense and Attack stats, but can we get a clearer picture?

ENTER `facet_wrap()`



ggplot: FACET-NATING

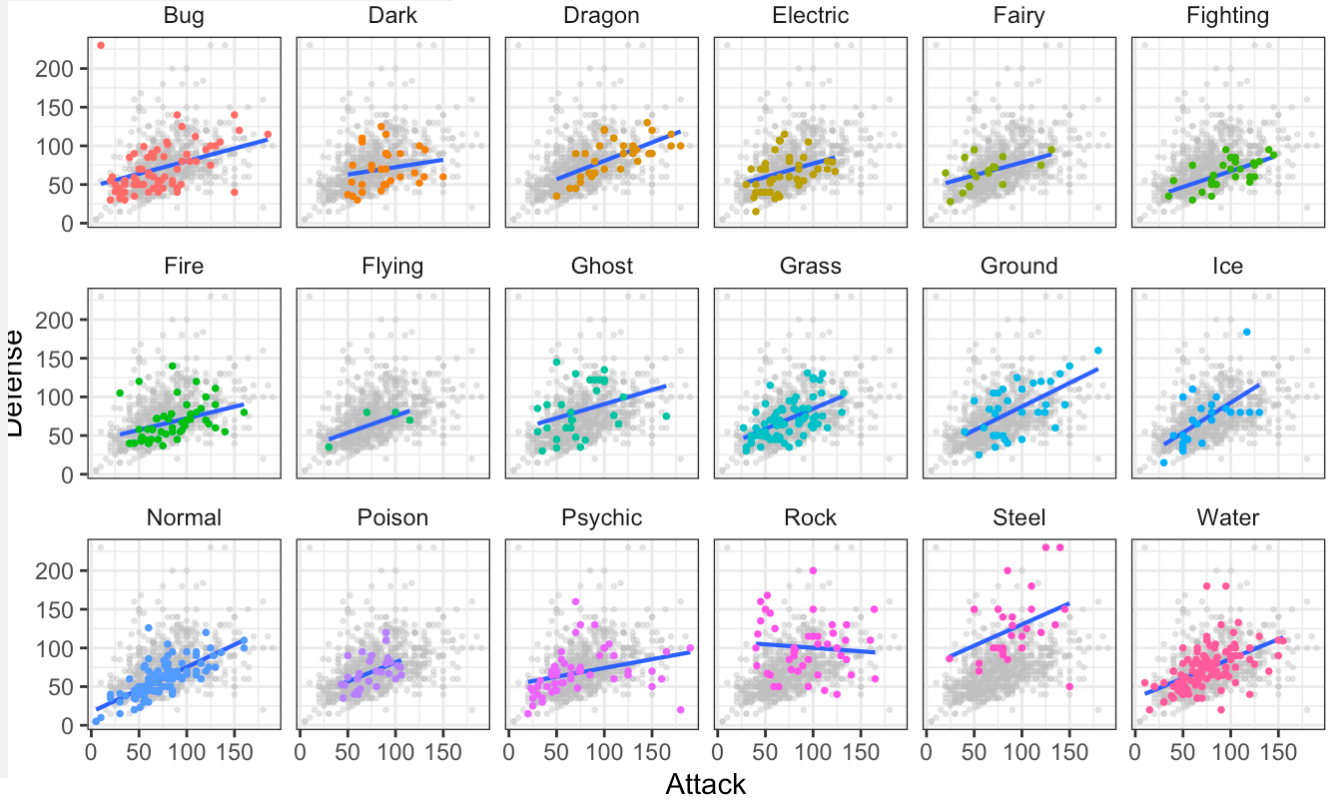
```{r}
ggplot(df, aes(x = Attack, y = Defense)) +
  facet_wrap(~Type.1, nrow = 3) +
  geom_smooth(method = 'lm',
              se = FALSE,
              size = 0.75) +
  geom_point(aes(color = Type.1),
             size = 0.65,) +
  theme_bw() +
  theme(legend.position = 'none')
```



ggplot: FACET-NATING FANCIFIED

Display all data but highlight each type!

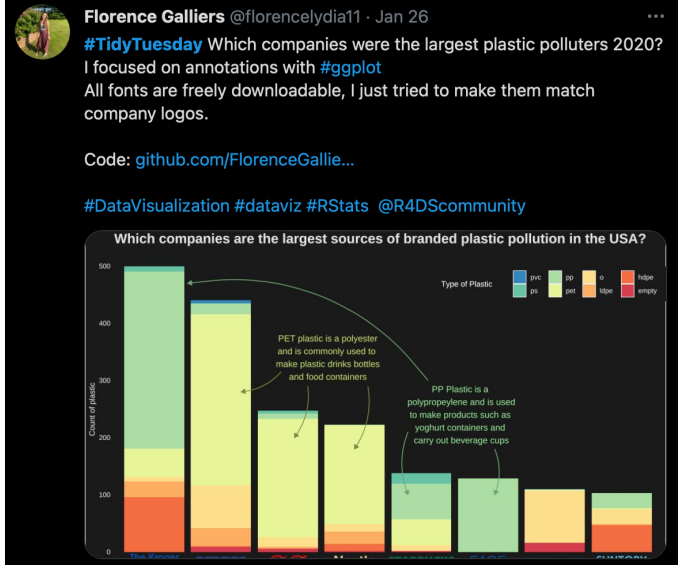
```{r}
df2 <- select(df, -Type.1)              # grab all data besides our normal grouping variable

ggplot(df, aes(x = Attack, y = Defense)) +
  facet_wrap(~Type.1,
             nrow = 3) +

  geom_point(data = df2,                # plot our new df as a big
             color = "gray",
             alpha = 0.35,
             size = 0.50) +

  geom_smooth(method = 'lm',
              se = FALSE,
              size = 0.75) +

  geom_point(aes(color = Type.1),
             size = 0.65,) +

  theme_bw() +

  theme(legend.position = 'none',
        strip.background = element_blank()) # get rid of facet title
```

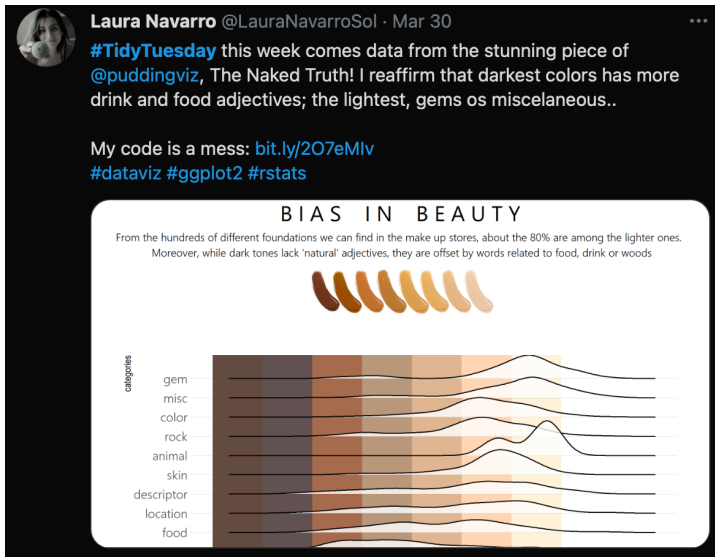


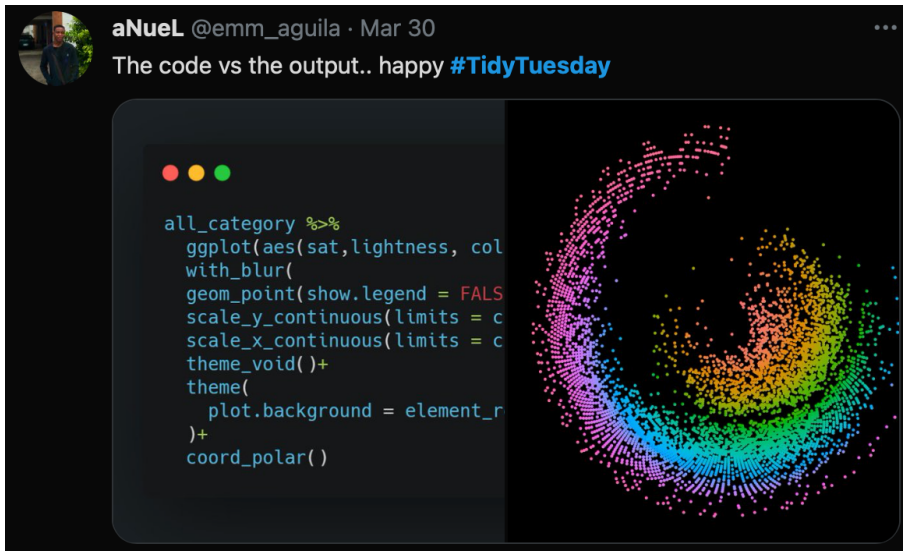
ggsave: SAVING YOUR MASTERPIECE

- You don't want to make beautiful things and leave them in R to rot!
- Adding ggsave() to the bottom of your plot will save as the specified file type, size, *and* DPI!

```r
{r}
df2 <- dplyr::select(df, -Type.1)

ggplot(df, aes(Attack, Defense)) +
  geom_point(data = df2, colour = "grey70", alpha = .5, size = .5) +
  geom_point(aes(colour = Type.1), size = .75) +
  facet_wrap(~Type.1, ncol = 4) +
  scale_color_viridis(discrete = TRUE) +
  theme_bw() +
  theme(legend.position = 'none',
        strip.background = element_blank())

ggsave("cool_plot.pdf", dpi = 600)
```







THAT'S IT! GET OUT THERE AND MAKE SOME BEAUTIFUL PLOTS!