Lecture 15: Building a Data Visualization

Let's use the air quality dataset to explore creating more complex plots. Up to this point, we've used base R to make our plots. This is something you have to know how to use! It is arguably not the best way to make plots in R though. We'll introduce a new set of packages here called the `tidyverse`. This loads a set of packages including `ggplot`, `dplyr`, and `magrittr` which gives us a way to map visual features to data feaures a bit easier. The tidyverse is developed from the Grammar of Graphics, which gives us a nice set of verbs to transform and visualize data. It is extremely well documented. Rather than controlling each of the plot parameters individually, the `ggplot` package consists of mapping data features to visual attributes inside an *aesthetic string* and then choose what plot we'd like to produce from a *geometry*. You will notice the syntax looks quite different!

For the example today we'll work with the `airquality` dataset from the `datasets` packages. Also make sure to install the `tidyverse` and load it into your markdown doc.

To begin, we'd like to see how ozone changes over time. Our dataset has records from May to September. Our time information is encoded in a day and month column, and appears to be in a separate columns, so we'll need to make one column that will tell us the index of the date.

```
library(datasets)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
data(airquality)
```

In base R, we make new columns like this, accessing the columns of a data frame with the `$` and using the assignment operator to store values. We also have an example of overwriting

```
airquality[airquality$Month > 6 ,]#subsetting
airquality$Time <- 1:nrow(airquality)#creating a new column
```

`dplyr` gives us a set of standard verbs like `filter` and `mutate` to subset and transform our data. So to do the above example in the tidyverse:

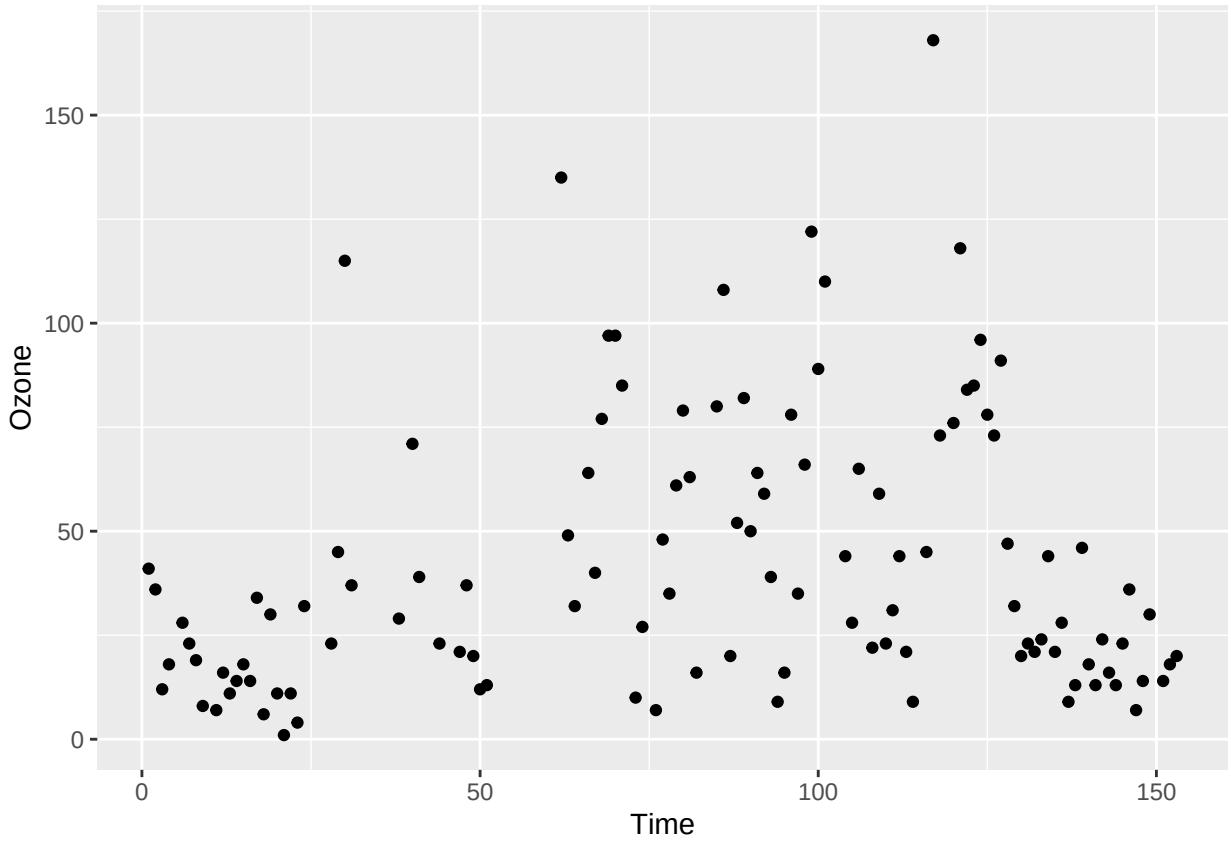
```
airquality %>%
  filter(Month > 6) %>% #subsetting
  mutate(Time = 1:92) -> cleaned_df #creating a new column
```

You'll also notice we've introduced a wildly different way of writing code, the pipeline `%>%`. The pipeline passes whatever is on the lefthand side to the first argument of whatever function is on the right hand side. This allows us to pass our cleaned data directly into the plot, as in the chunk below. A few things to note here: the `x` and `y` arguments are specified inside the *aesthetic* string `aes()`. Then the *geometry* we've chosen

here is `geom_point()` to create the scatterplot.

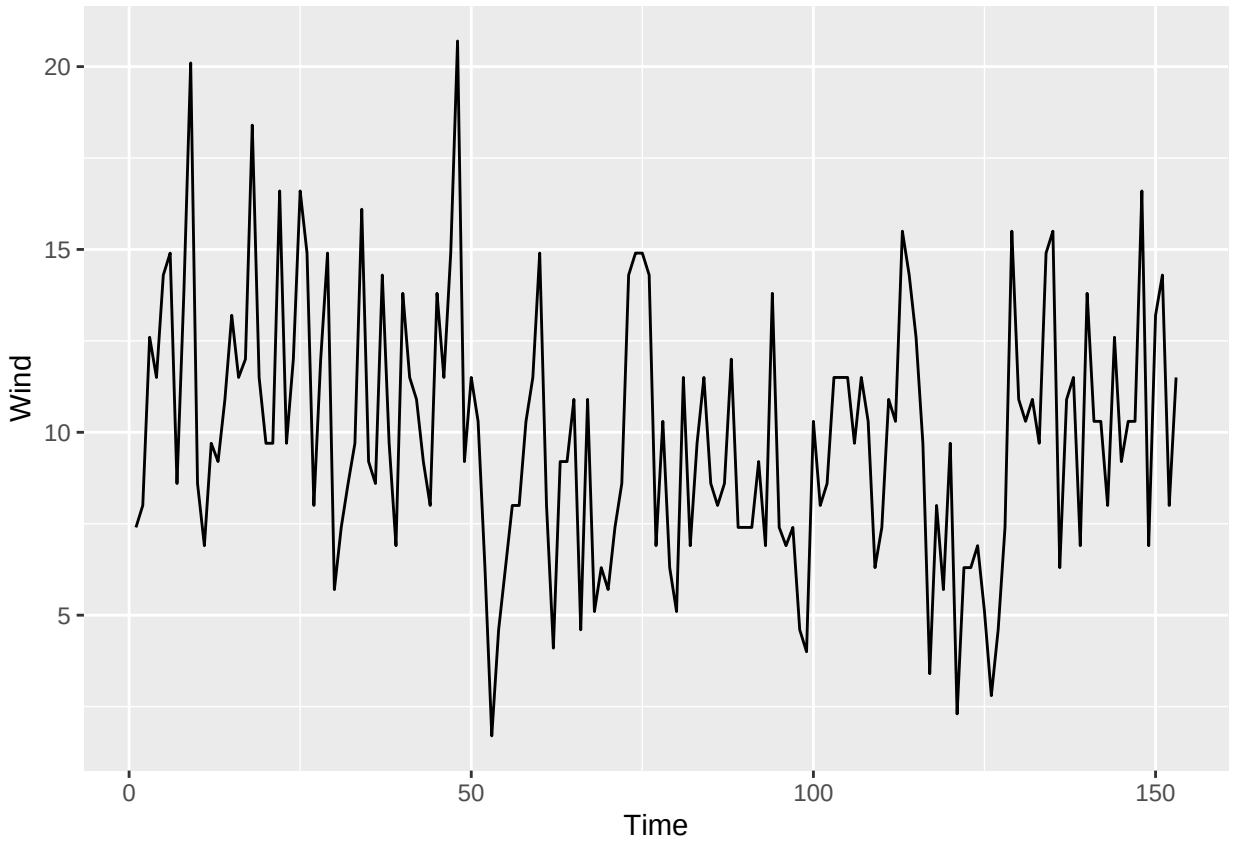
```
airquality %>%
  mutate(Time = 1:153) %>%
  ggplot(aes(x = Time, y = Ozone)) +
  geom_point()
```

```
## Warning: Removed 37 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



We can swap the geometry out for other appropriate options, like `geom_line()`:

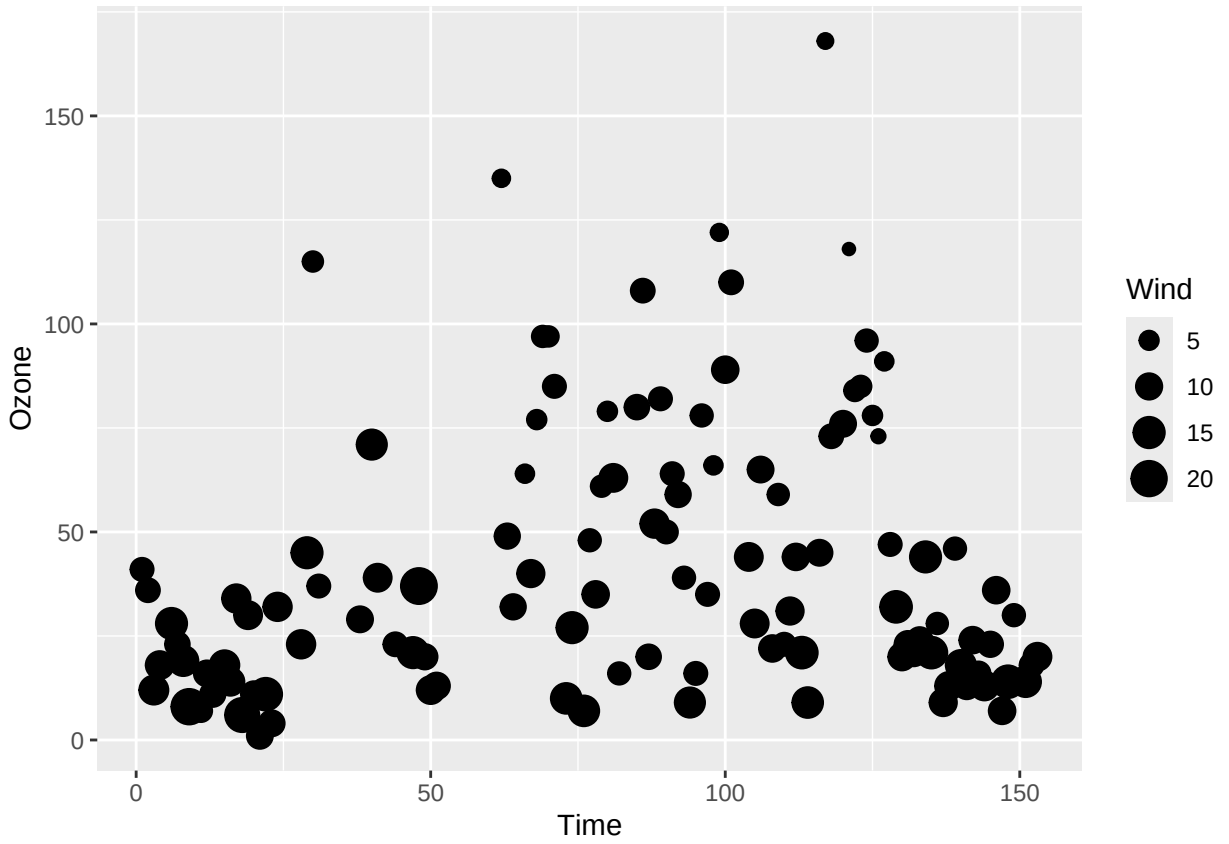
```
airquality %>%
  mutate(Time = 1:153) %>%
  ggplot(aes(x = Time, y = Wind)) +
  geom_line()
```



In the aesthetic string, we put our arguments that **vary according to each row** In the geometry, we put our arguments that should be applied to all rows in the data. To wit, if we'd like to size our points according to the windspeed that day, we add a `size` parameter to our aesthetic string:

```
airquality %>%
  mutate(Time = 1:153) %>%
  ggplot(aes(x = Time, y = Ozone, size = Wind)) +
  geom_point()
```

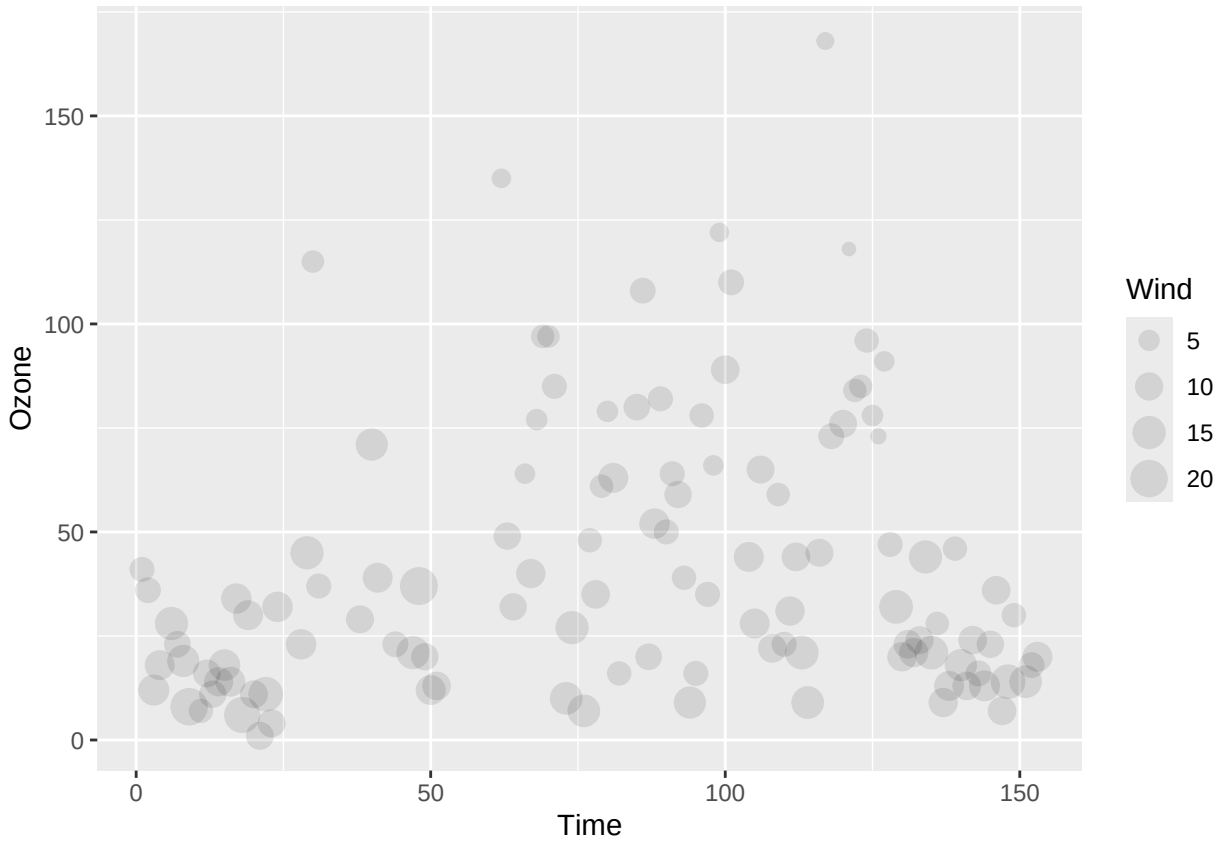
```
## Warning: Removed 37 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



There's some overlap here, so we'd like to change the transparency. This is governed by the `alpha` parameter. We want alpha to be consistent across the board, so we'll add it to the global string.

```
airquality %>%
  mutate(Time = 1:153) %>%
  ggplot(aes(x = Time, y = Ozone, size = Wind)) +
  geom_point(alpha = 0.1) #change *global* aesthetics in the geometry
```

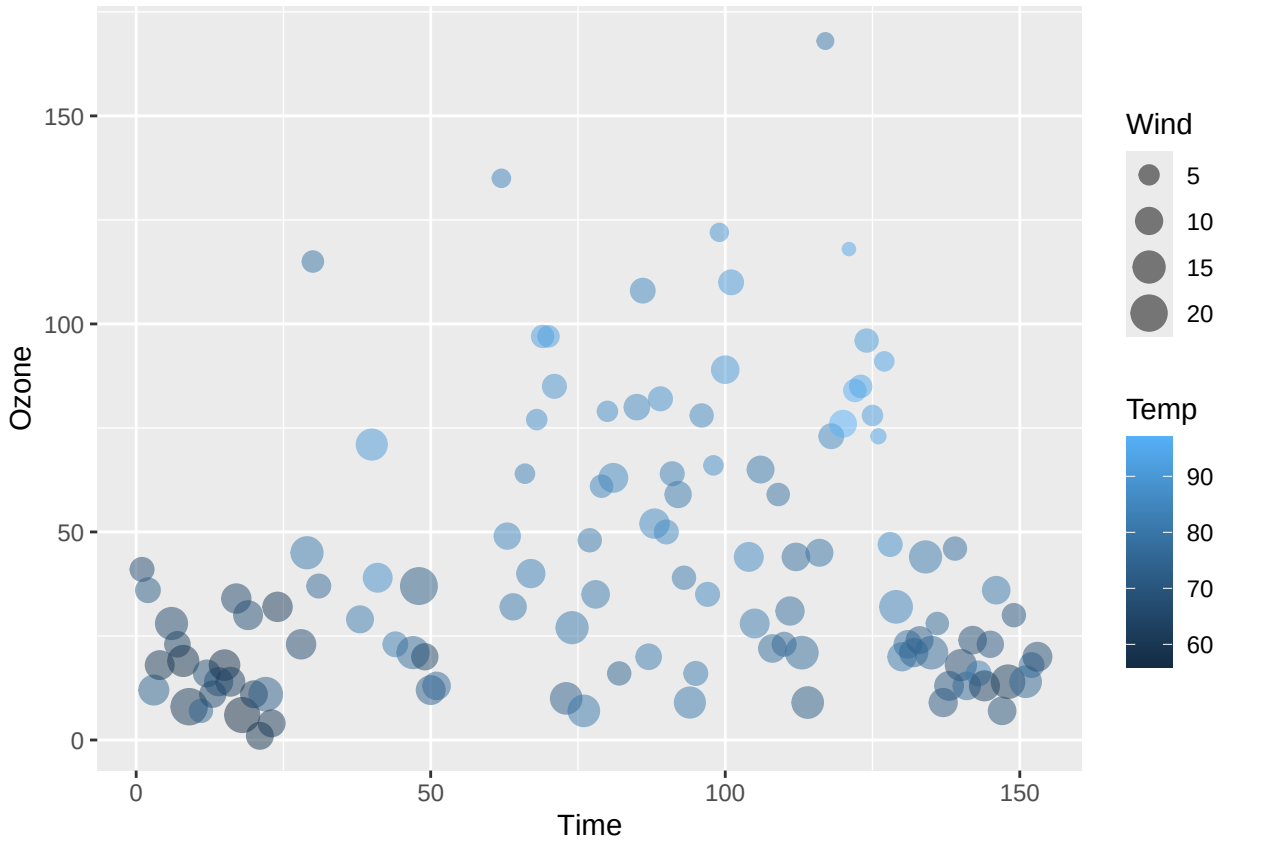
```
## Warning: Removed 37 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



We can also add `color`. Color gets a bit more complicated! When we have points we use `color` as the parameter. For geometries that deal with areas (like in a bar chart) we use the `fill` parameter. The tidyverse supplies a default color palette (a gradient of blues in this example.)

```
airquality %>%
  mutate(Time = 1:153) %>%
  ggplot(aes(x = Time, y = Ozone, size = Wind, color = Temp)) +
  geom_point(alpha = 0.5)
```

```
## Warning: Removed 37 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

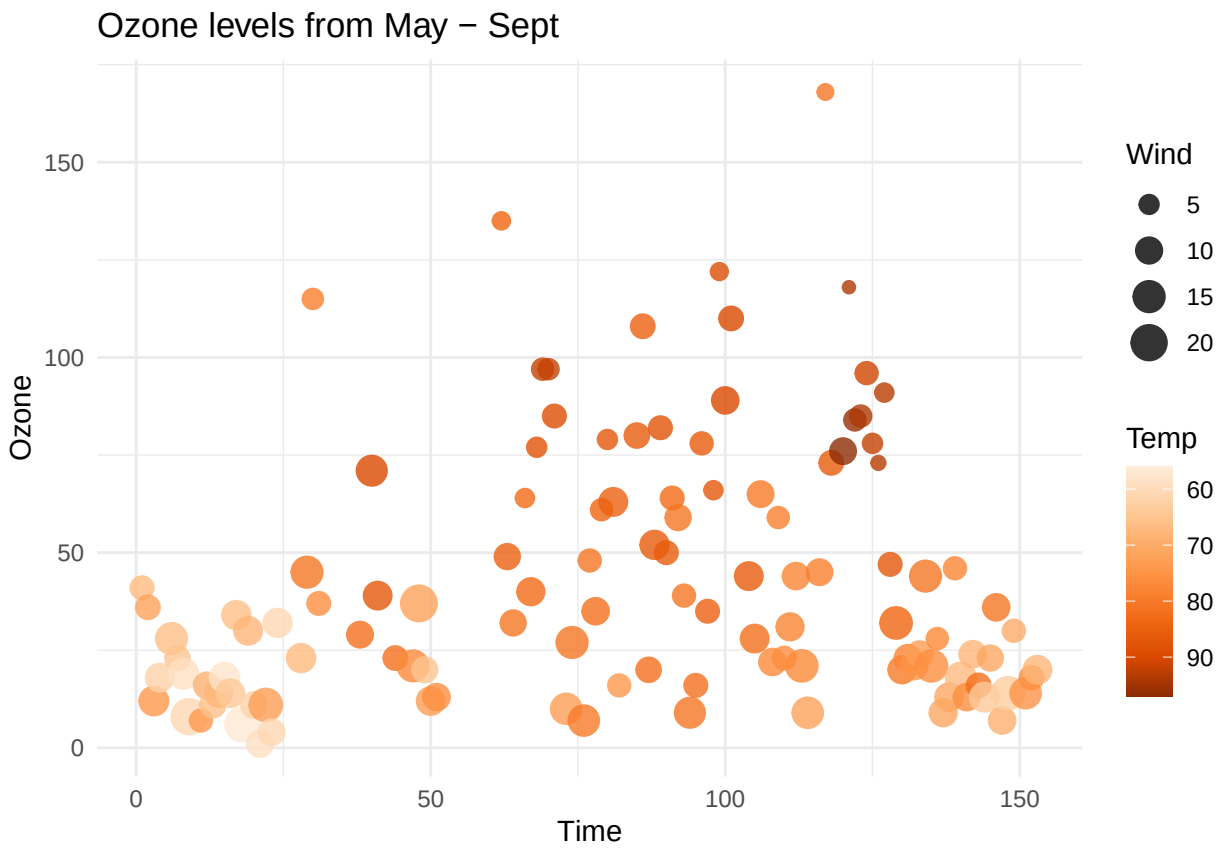


Suppose we'd like to change the color palette: we can do so by using the suite of functions from `RColorBrewer`. If I want to color my points orange instead, I can supply a new argument for `palette`. Refer to the documentation to see available colors. You'll want to be sure to match *continuous data* to *continuous scales* and likewise, *discrete data* to *discrete scales*. (I've also flipped the color axis here with the `trans = 'reverse'` argument.)

A few other features to note from ggplot. We can change the theme to adjust how the background of the chart fills. Try changing the theme in this example to `theme_bw()` or `theme_classic()`. I've also added a title with the `ggtitle()` function.

```
airquality %>%
  mutate(Time = 1:153) %>%
  ggplot(aes(x = Time, y = Ozone, size = Wind, color = Temp)) +
  geom_point(alpha = 0.8) +
  scale_color_distiller(palette = "Oranges", trans = "reverse") +
  theme_minimal() +
  ggtitle("Ozone levels from May - Sept")
```

```
## Warning: Removed 37 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



To sum up: `ggplot2` allows us to map data features to visual features. This is key to developing compelling visualizations!