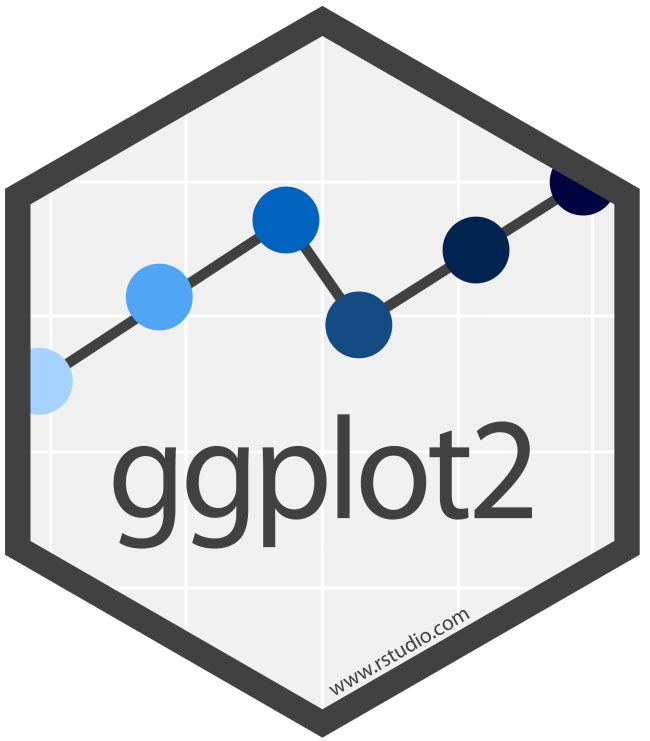
Week 4: Data Visualization

PM 566: Introduction to Health Data Science

Acknowledgment

These slides were originally developed by Meredith Franklin (and Paul Marjoram) and modified by George G. Vega Yon.

Background



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

This section is based on chapter 3 of "R for Data Science"

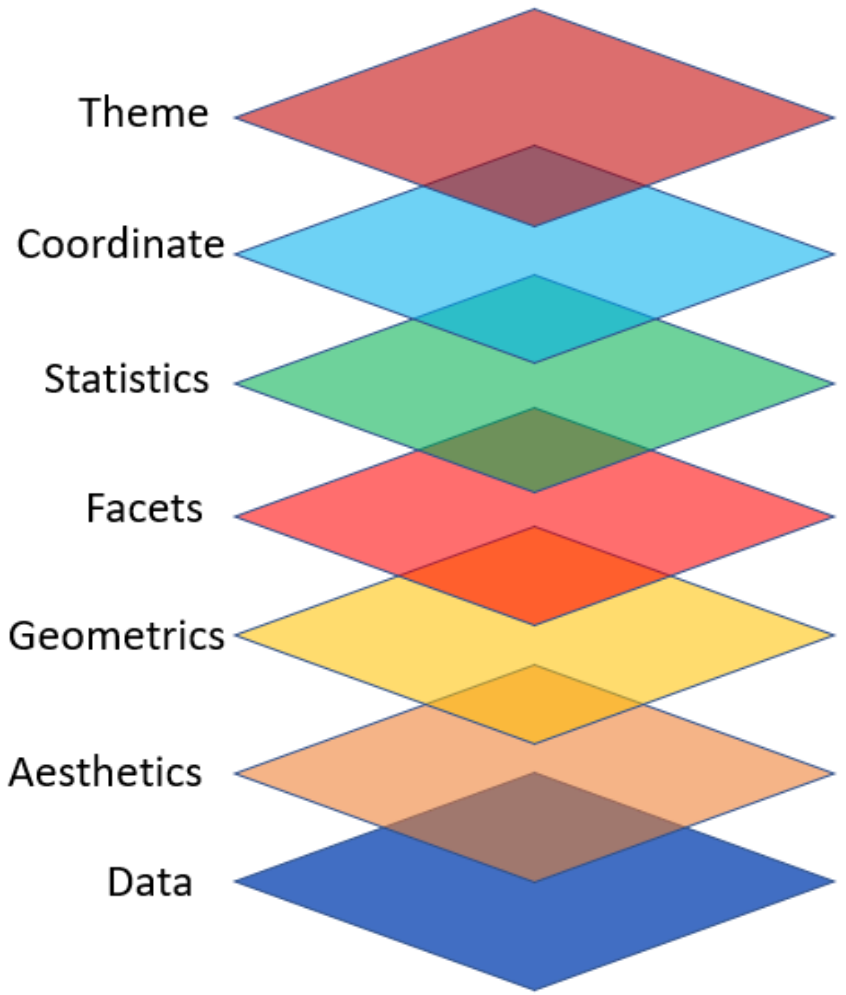
Background

`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/)

```
library(tidyverse)
library(data.table)
```

ggplot2

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



ggplot2

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

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

Data

Continuing with the weather data from last week, let's take the daily averages at each site, keeping some of the variables. Let's also create a new variable for region (east and west), categorize elevation, and create a multi-category variable for visibility for exploratory purposes.

```
# Reading the data, filtering, and replacing bad values with NAs
met <- read.csv('../data/met_all.gz')
met <- met[met$temp > -10, ]
met$elev[met$elev == 9999.0] <- NA

# Creating an aggregated version of the dataset
library(dplyr)
met_avg <- summarize(met,
                     temp =     mean(temp, na.rm = TRUE),
                     rh =       mean(rh, na.rm = TRUE),
                     wind.sp =  mean(wind.sp, na.rm = TRUE),
                     vis.dist = mean(vis.dist, na.rm = TRUE),
                     lat =      mean(lat, na.rm = TRUE),
                     lon =      mean(lon, na.rm = TRUE),
                     elev =     mean(elev, na.rm = TRUE),
                     .by = c(USAFID, day))
```

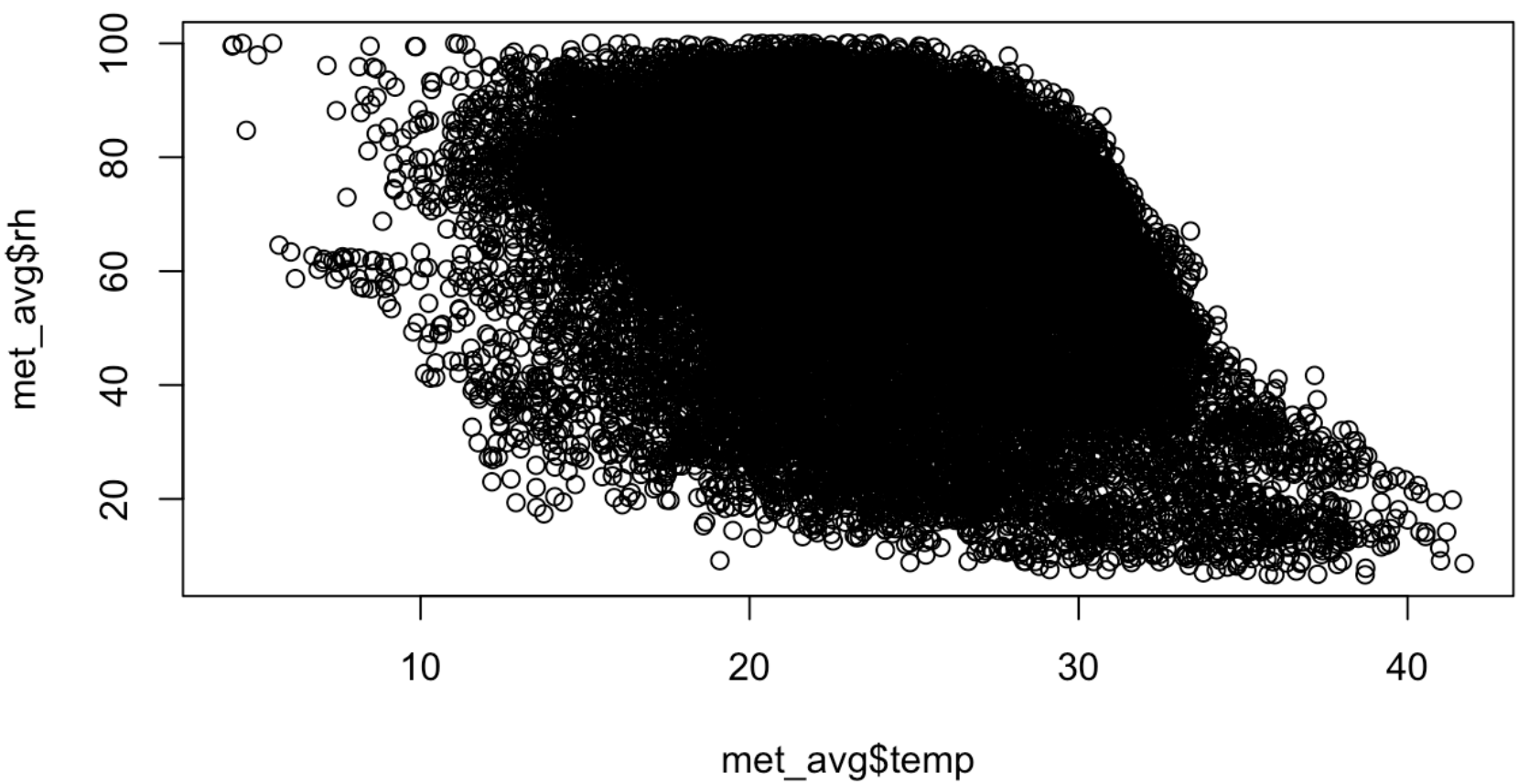
```r
# New variables using ifelse()
met_avg$region <- ifelse(met_avg$lon > -98, "east", "west")
met_avg$elev_cat <- ifelse(met_avg$elev > 252, "high", "low")

# Using the cut() function to create more than 2 categories
met_avg$vis_cat <- cut(met_avg$vis.dist,
                       breaks = c(0, 1000, 6000, 10000, Inf),
                       labels = c("fog", "mist", "haze", "clear"),
                       right = FALSE)
```

The variables we will focus on for this example are temp and rh (temperature in C and relative humidity %)

Basic Scatterplot

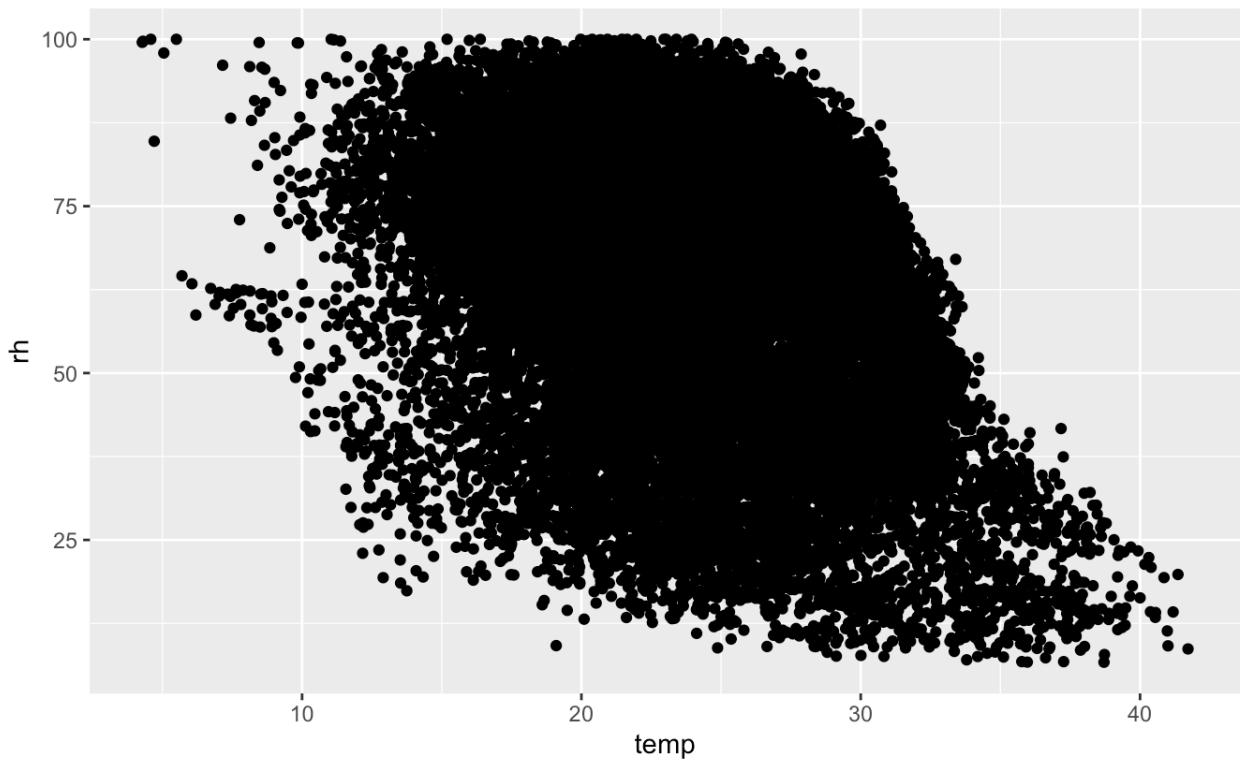
```
1 plot(met_avg$temp, met_avg$rh)
```



Basic Scatterplot

Here's how to create a basic plot in ggplot2

```
1 ggplot(data = met_avg) +
2   geom_point(mapping = aes(x = temp, y = rh))
```



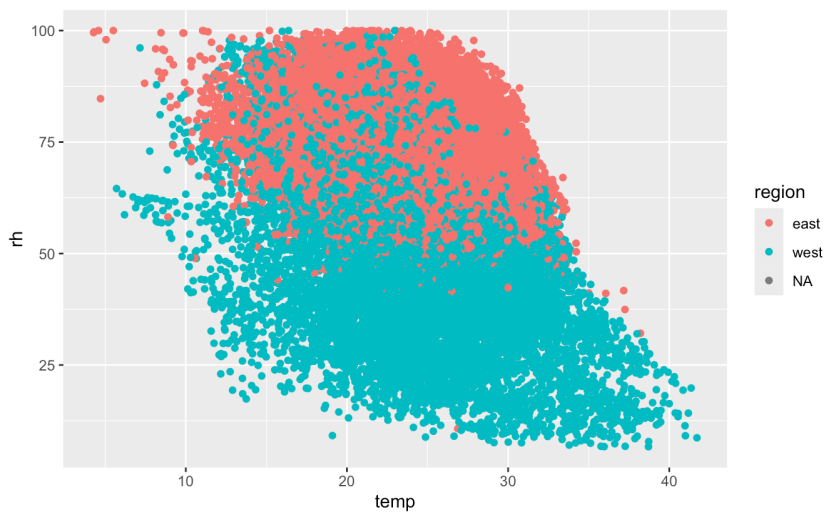
Basic Scatterplot 2

- `geom_point()` adds a layer of points to your plot, to create 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. ggplot2 looks for the mapped variables in the data argument, in this case, met_avg
- 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 map the colors of your points to the class variable to reveal the region of data (west or east). `ggplot2` chooses colors, and adds a legend, automatically.

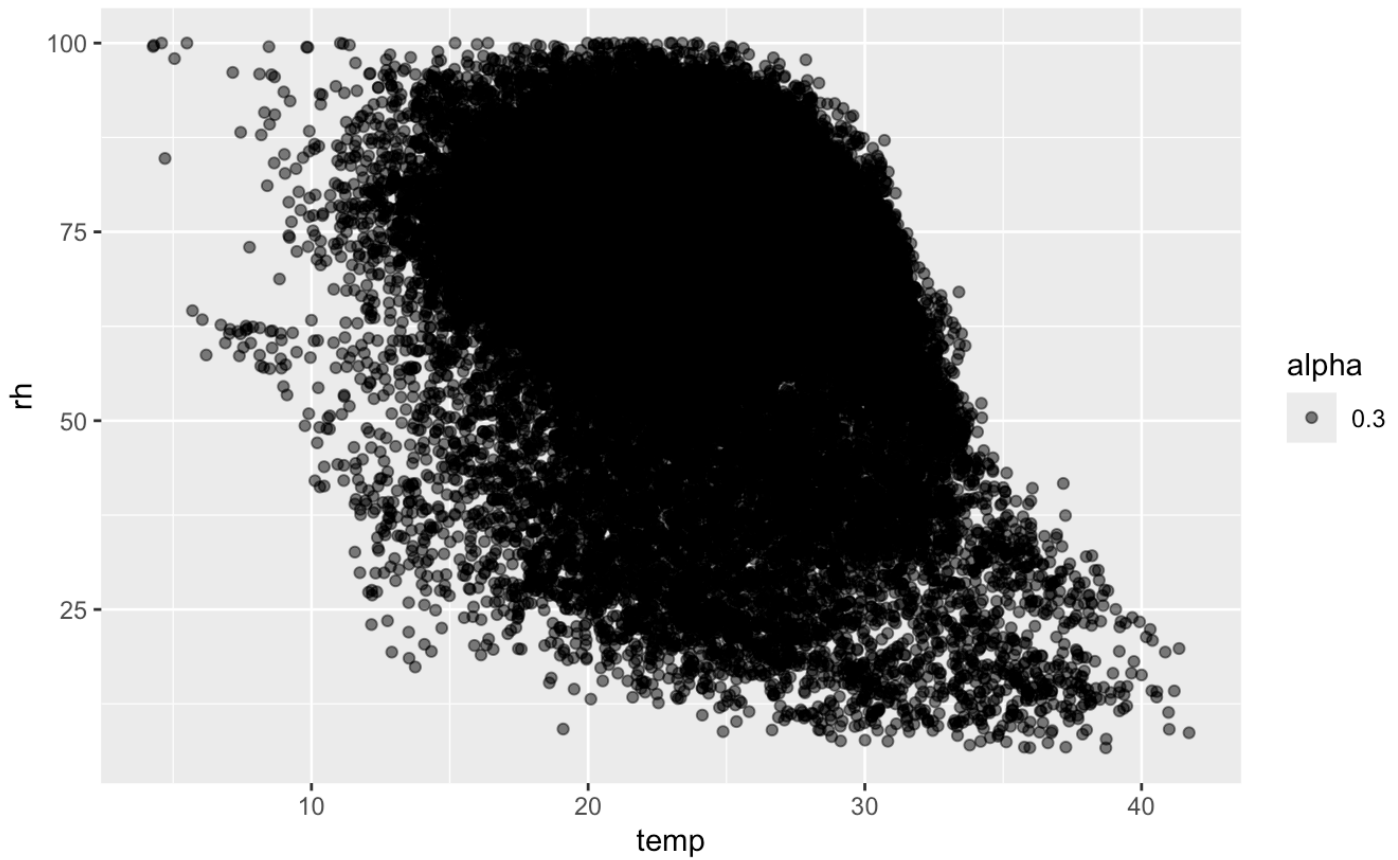
```
ggplot(data = met_avg) +
  geom_point(mapping = aes(x = temp, y = rh, color = region))
```



We see that humidity in the east is generally higher than in the west and that the hottest temperatures are in the west.

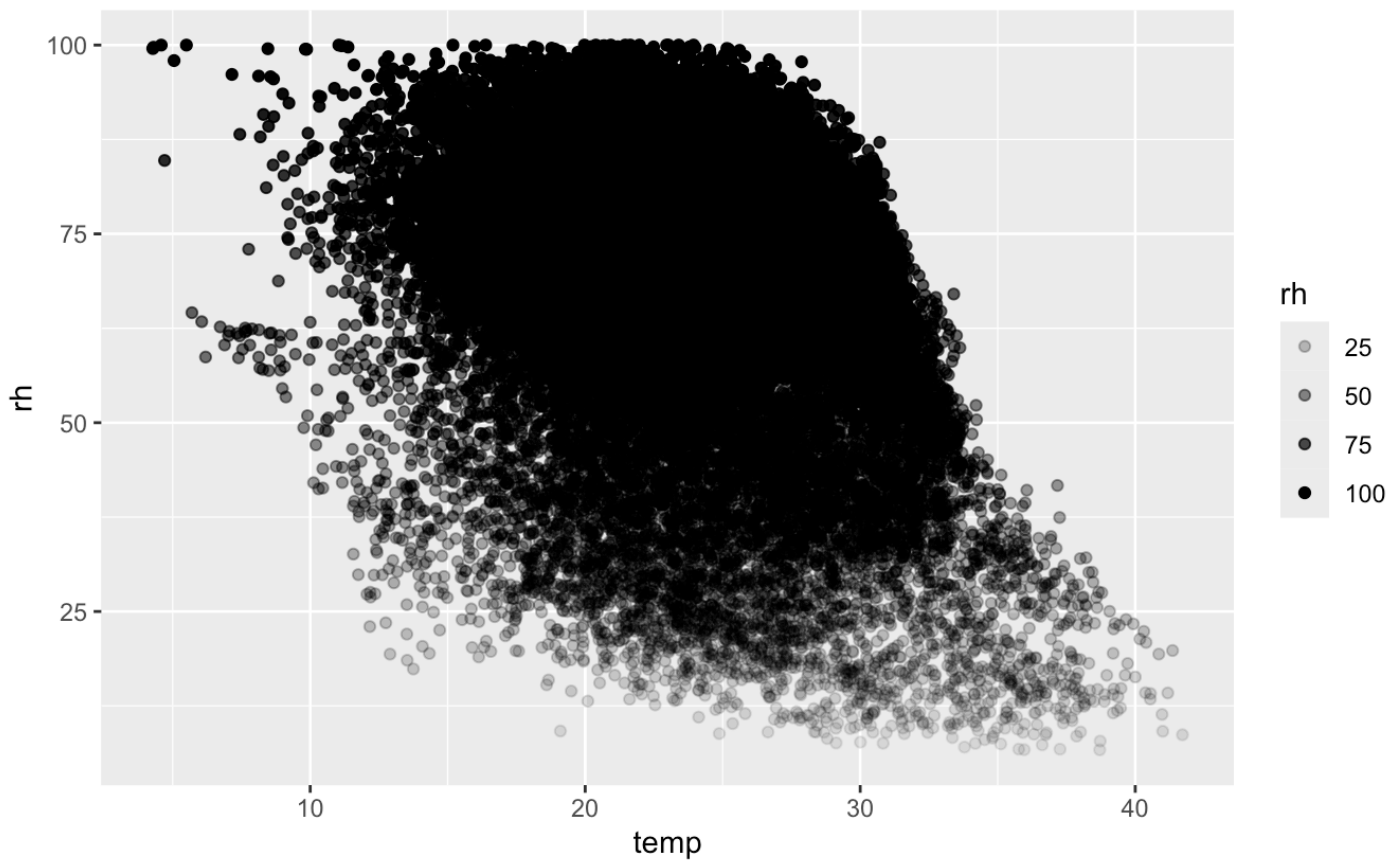
Controlling point transparency using the “alpha” aesthetic

```
1 ggplot(data = met_avg) +
2   geom_point(mapping = aes(x = temp, y = rh, alpha = 0.3))
```



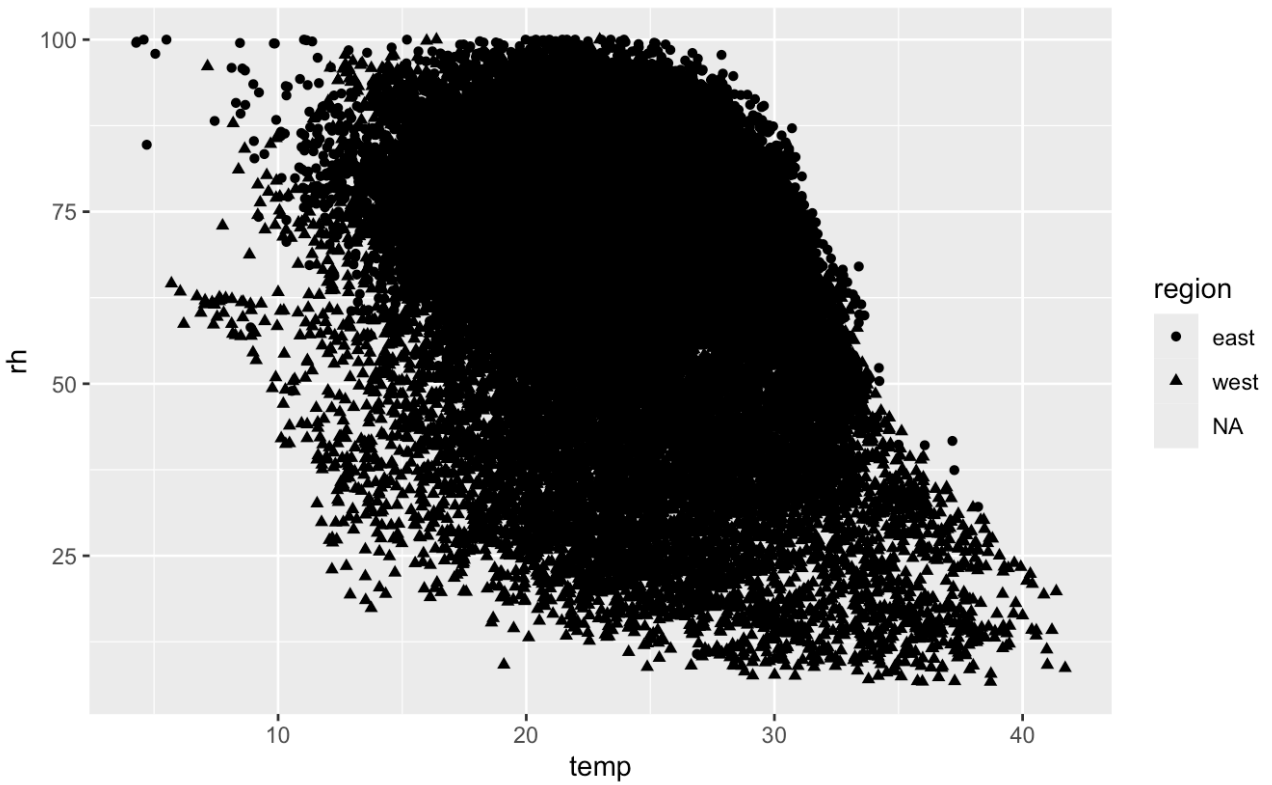
Controlling point transparency using the “alpha” aesthetic

```
1 ggplot(data = met_avg) +
2   geom_point(mapping = aes(x = temp, y = rh, alpha = rh))
```



Controlling point shape:

```
1 ggplot(data = met_avg) +
2   geom_point(mapping = aes(x = temp, y = rh, shape = region))
```

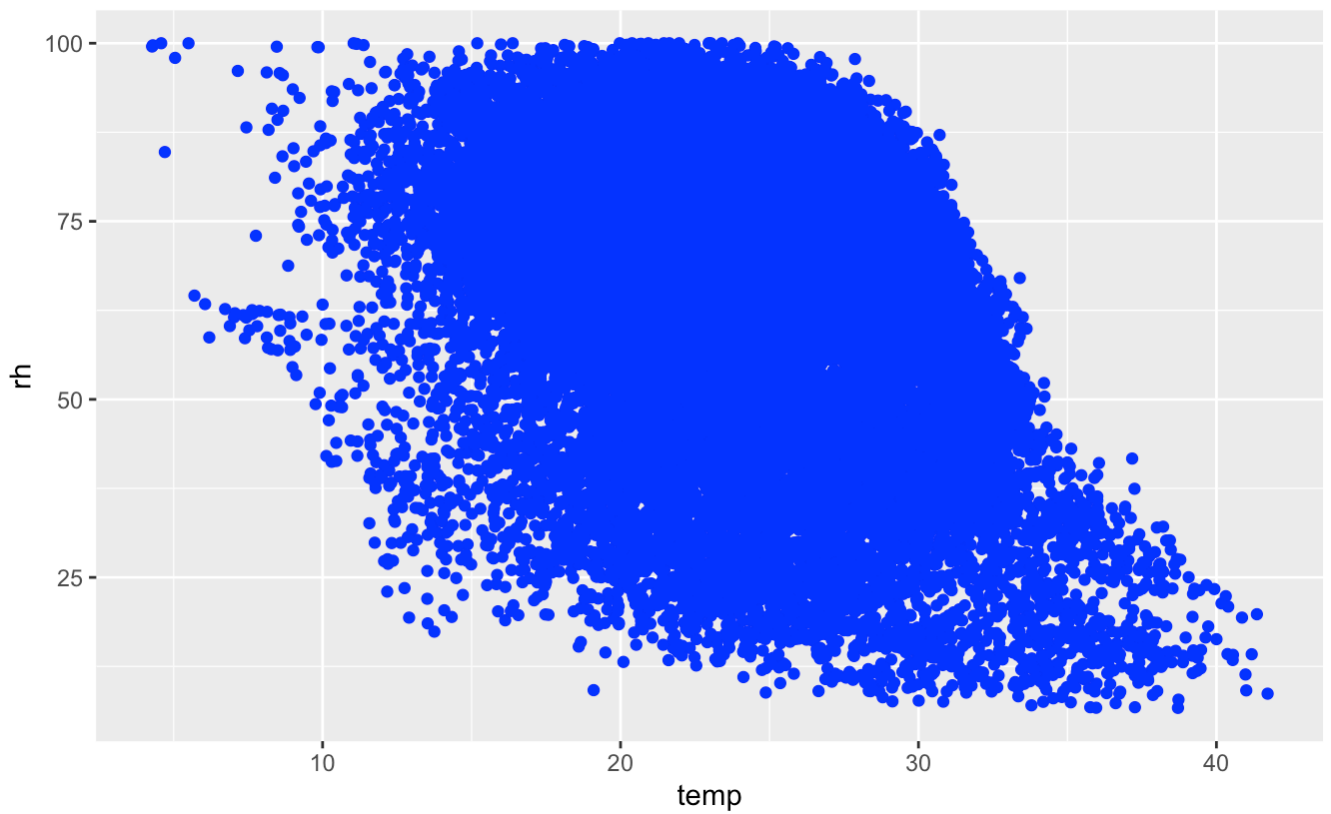


Note that, by default, ggplot uses up to 6 shapes. If there are more, some of your data is not plotted!! (At least it warns you.)

Manual control of aesthetics

To control aesthetics manually, set the aesthetic by name as an argument of your geom function; i.e. it goes outside of aes().

```
1 ggplot(data = met_avg) +
2   geom_point(mapping = aes(x = temp, y = rh), color = "blue")
```



Summary of aesthetics

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

Base plot equivalents

code	description
first arg / x	position on x-axis
second arg / y	position on y-axis
pch	shape
col	color of element borders
fill	color inside of elements
cex	size
scales::alpha	transparency
lty	type of line

Add points to plot

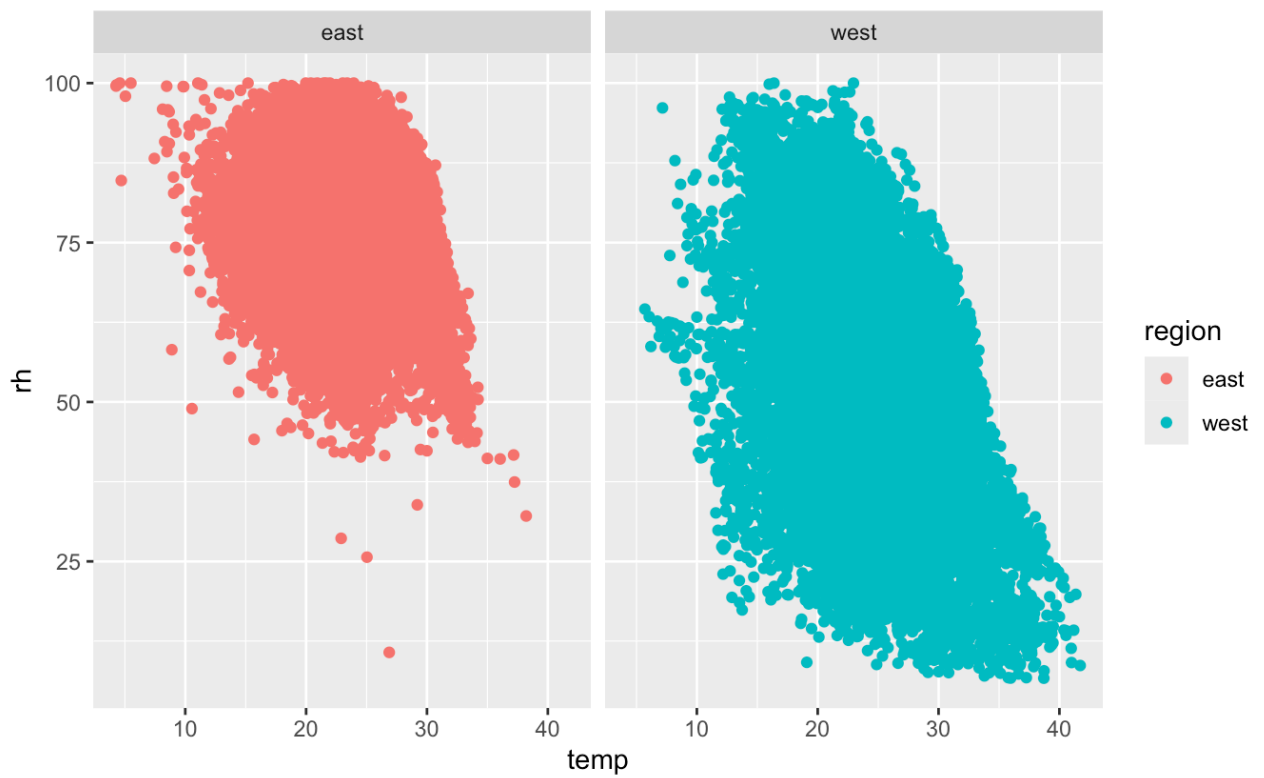
With base plot, you can add points to an existing plot with `points()`, which takes the same arguments as `plot()` for plotting points.

```
1 plot(1:10, pch = 16)
2 points(10:1, pch = 16, col = 2)
```

Facets 1

Facets are particularly useful for categorical variables.

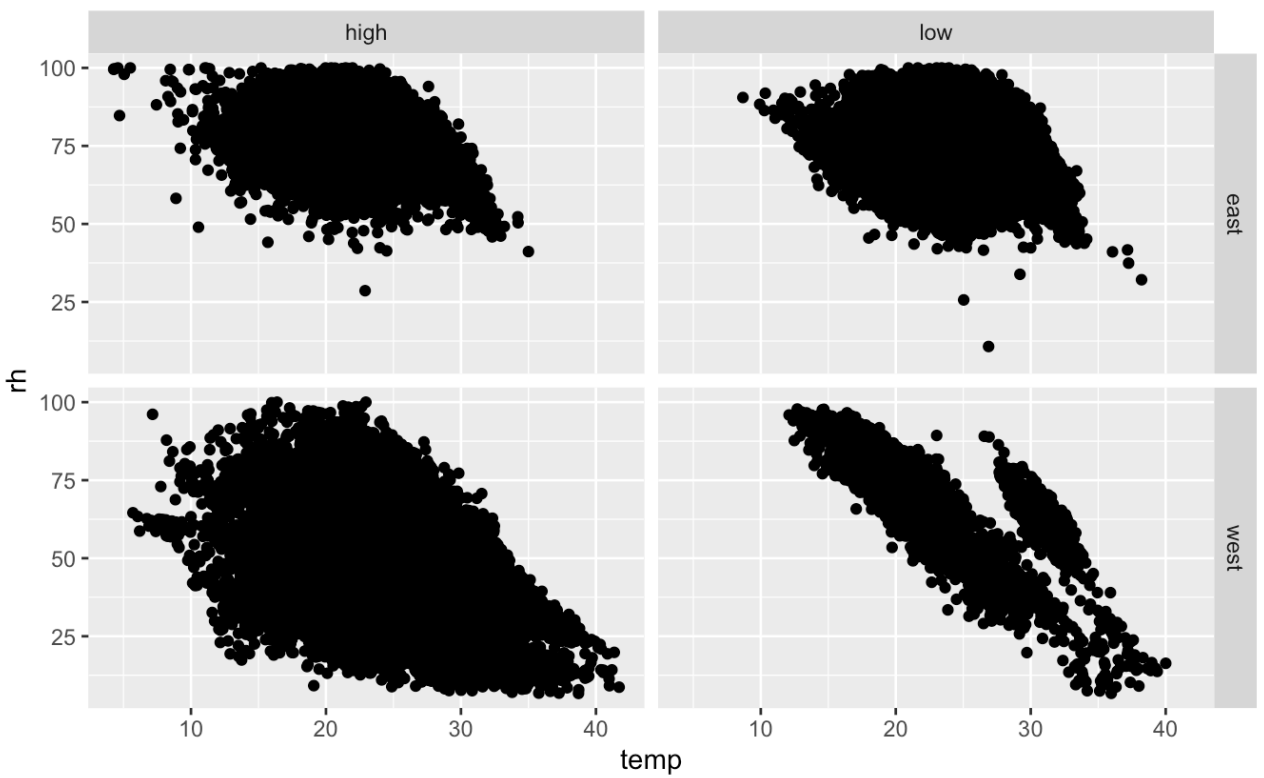
```
met_avg[!is.na(met_avg$region), ] %>%
  ggplot() +
  geom_point(mapping = aes(x = temp, y = rh, color=region)) +
  facet_wrap(~ region, nrow = 1)
```



Facets 2

Or you can facet on two variables…

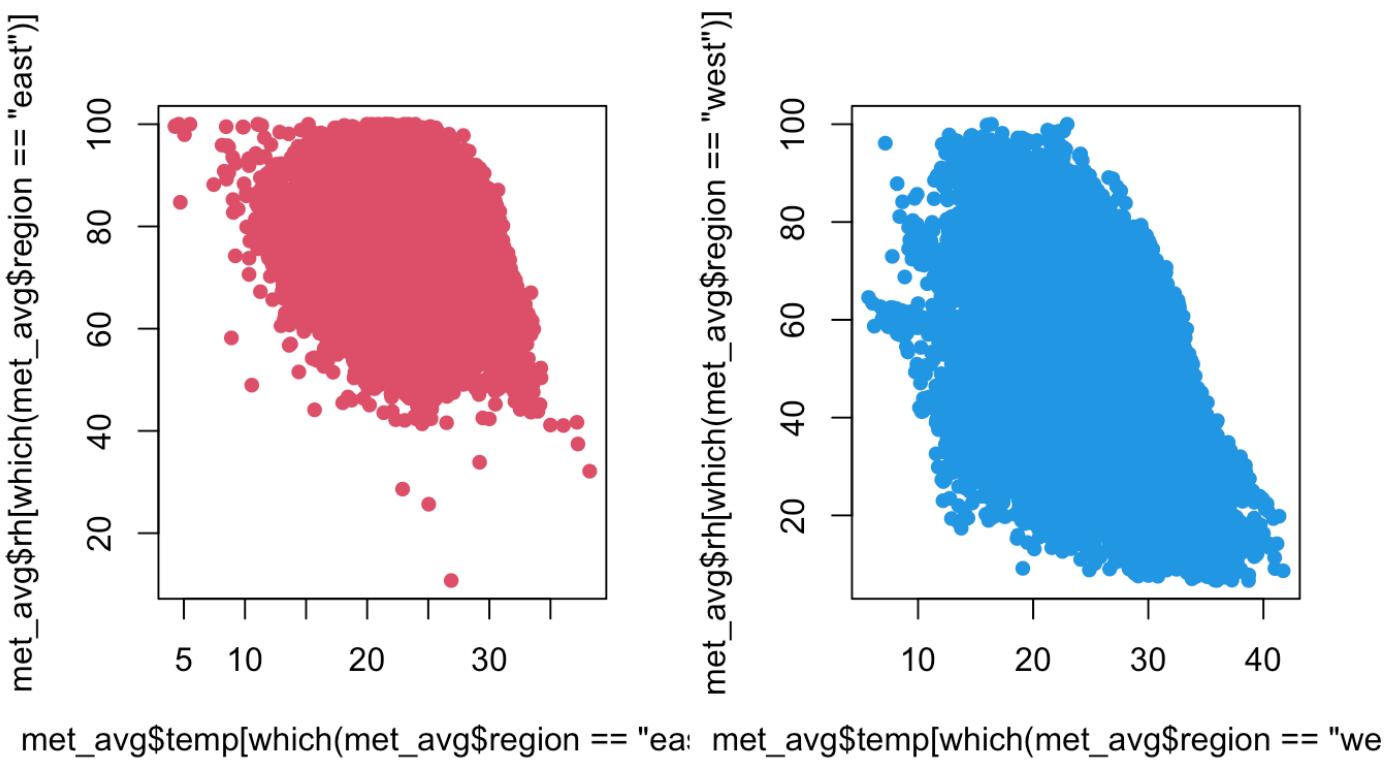
```
met_avg[!is.na(met_avg$region) & !is.na(met_avg$elev_cat), ] %>%
  ggplot() +
  geom_point(mapping = aes(x = temp, y = rh)) +
  facet_grid(region ~ elev_cat)
```



Facets 3

Base plot is not good at this! You can make multiple plots within a single plotting window by utilizing the `layout()` function, but you will still have to make each plot manually.

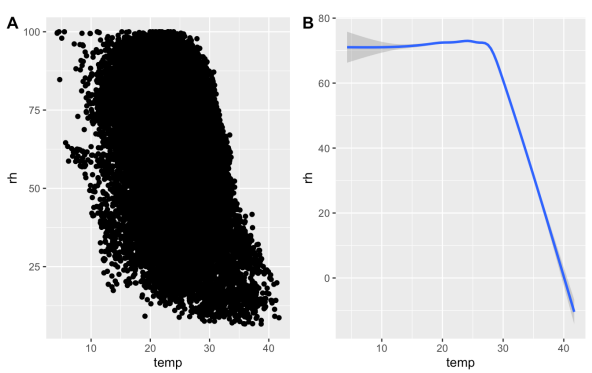
```
1 layout(matrix(1:2, nrow=1))
2 plot(met_avg$temp[which(met_avg$region == 'east')], met_avg$rh[which(met_avg$region == 'east')], pch = 16, c
3 plot(met_avg$temp[which(met_avg$region == 'west')], met_avg$rh[which(met_avg$region == 'west')], pch = 16, c
```



Geometric objects 1

Geometric objects are used to control the type of plot you draw. So far we have used scatterplots (via `geom_point`). But now let's try plotting a smoothed line fitted to the data (and note how we do side-by-side plots)

```
1  library(cowplot)
2
3  scatterplot <- ggplot(data = met_avg) + geom_point(mapping = aes(x = temp,
4  lineplot    <- ggplot(data = met_avg) + geom_smooth(mapping = aes(x = temp,
5
6  plot_grid(scatterplot, lineplot, labels = "AUTO")
```



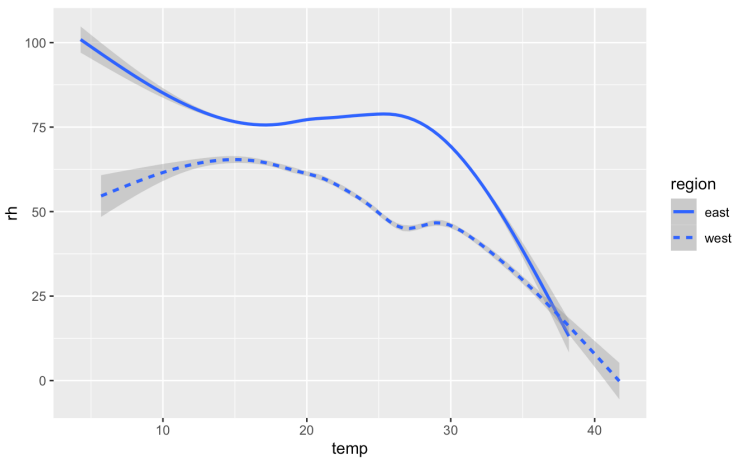
Geometric objects 1

`cowplot` is a package due to Claus Wilke, it "… is a simple add-on to `ggplot`. It provides various features that help with creating publication-quality figures, such as a set of themes, functions to align plots and arrange them into complex compound figures, and functions that make it easy to annotate plots and or mix plots with images."

Geometric objects 2

Note that not every aesthetic works with every geom function. But now there are some new ones we can use.

```
ggplot(data = met_avg) +
  geom_smooth(mapping = aes(x = temp, y = rh, linetype = region))
```

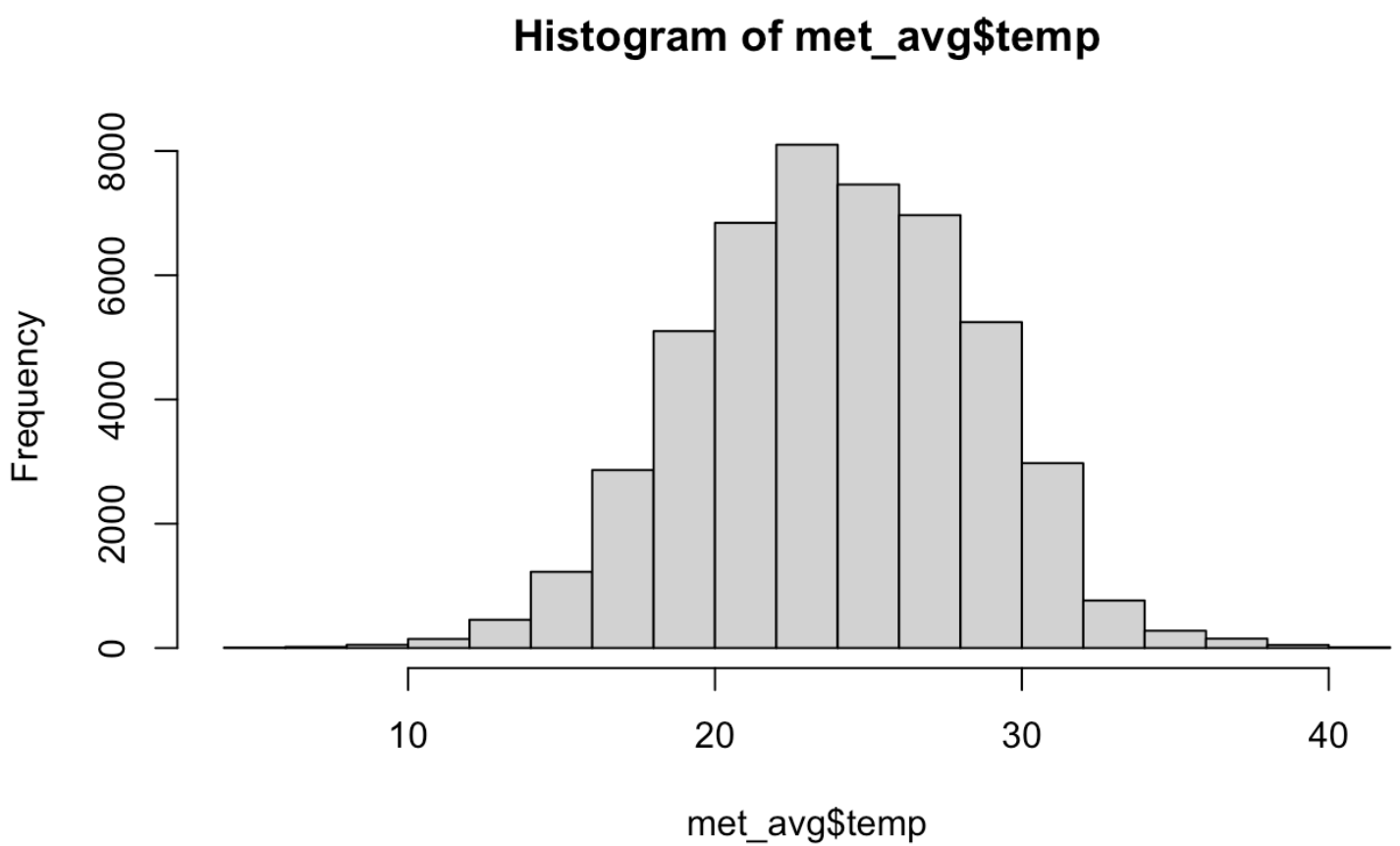


Here we make the line type depend on the region and we clearly see east has higher rh than west, but generally as temperatures increase rh decreases in both regions.

Geometric objects 3

Histograms

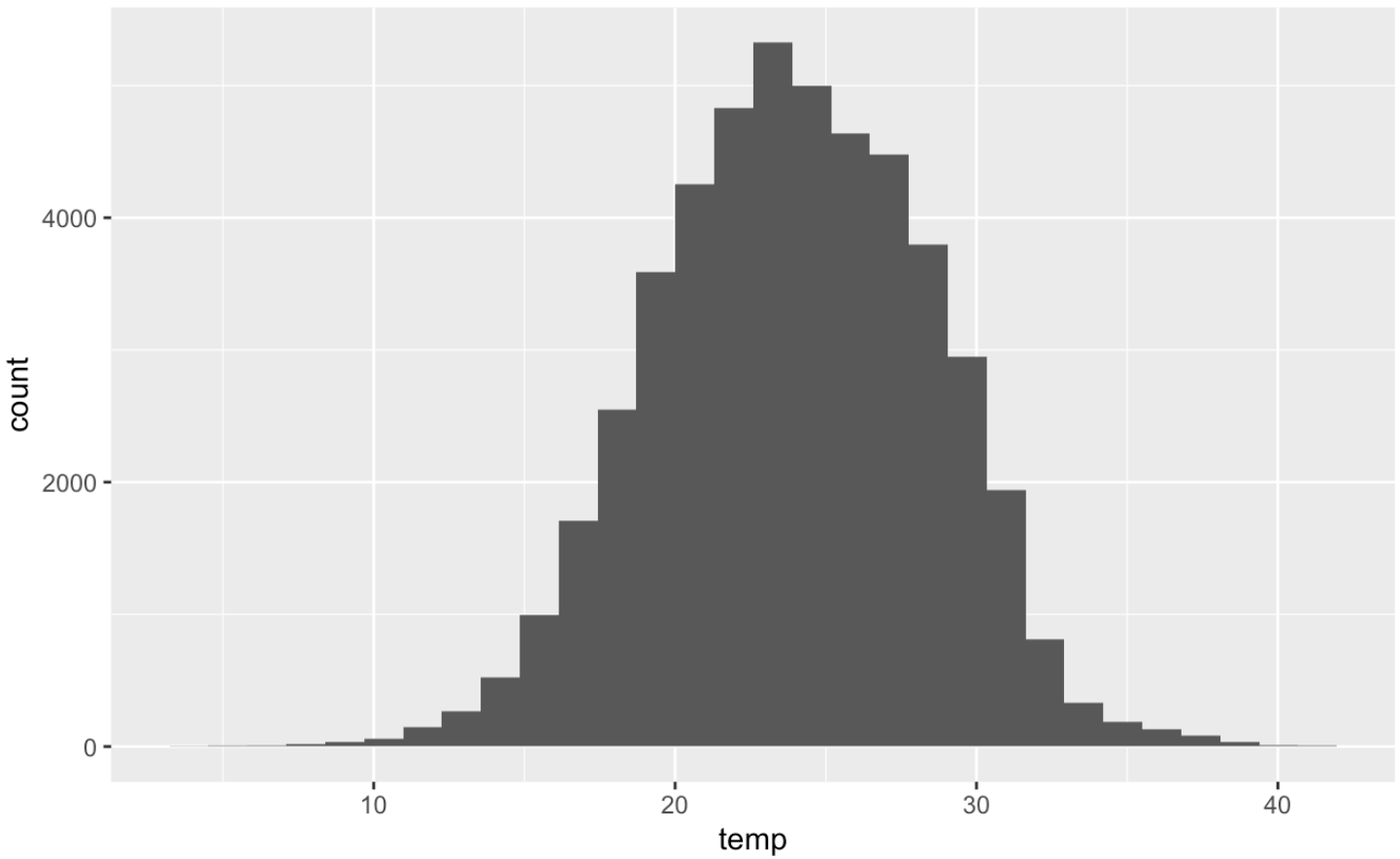
```
1 hist(met_avg$temp)
```



Geometric objects 3

Histograms

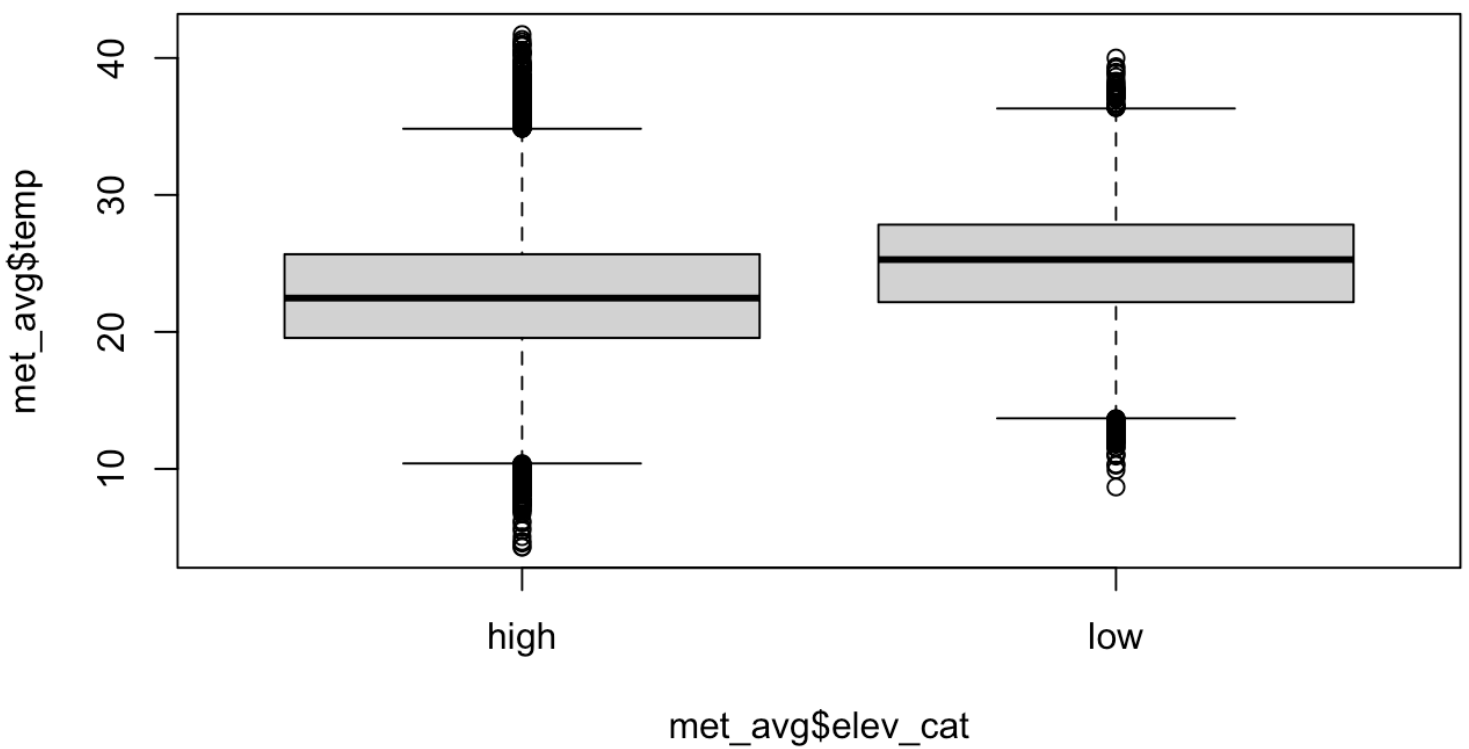
```
1 ggplot(met_avg) +
2   geom_histogram(mapping = aes(x = temp))
```



Geometric objects 4

Boxplots

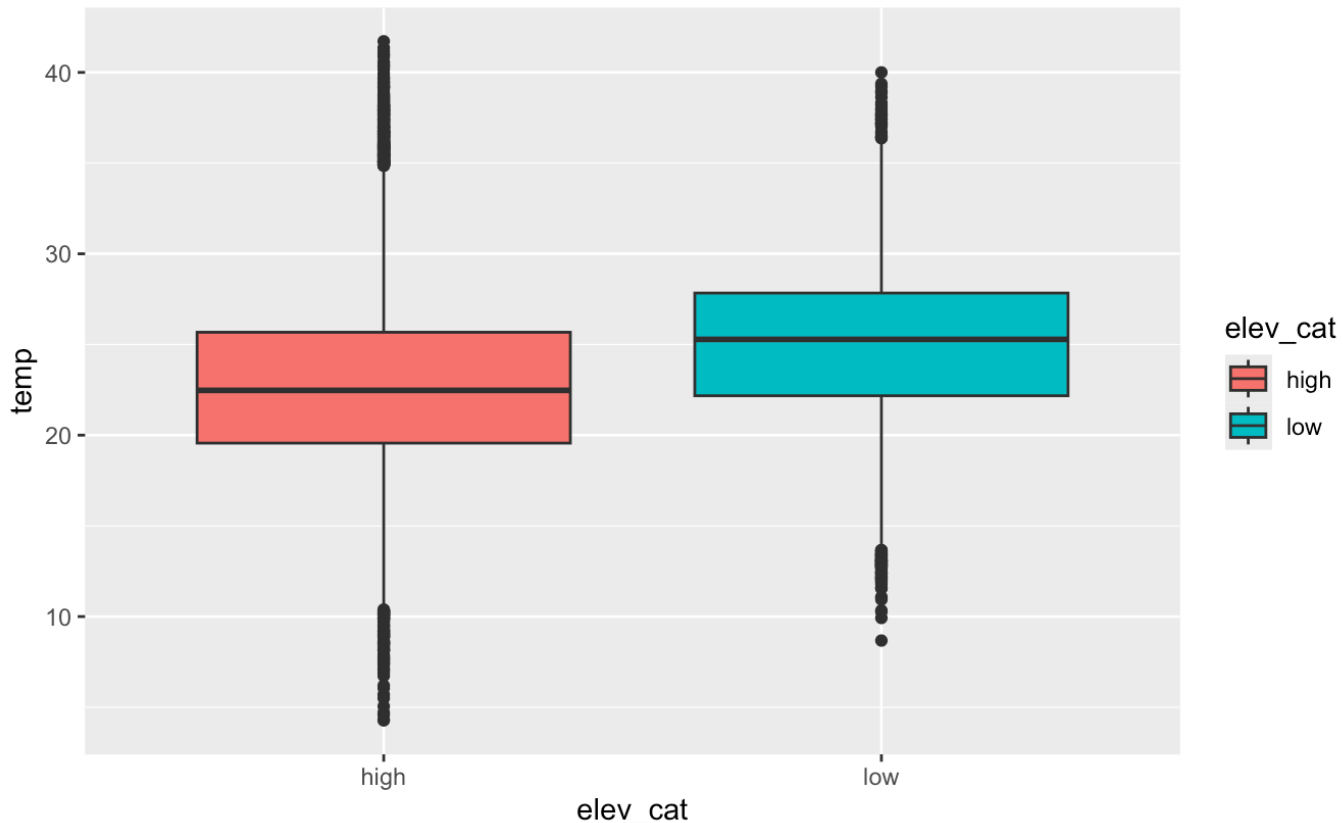
```
1 boxplot(met_avg$temp ~ met_avg$elev_cat)
```



Geometric objects 4

Boxplots

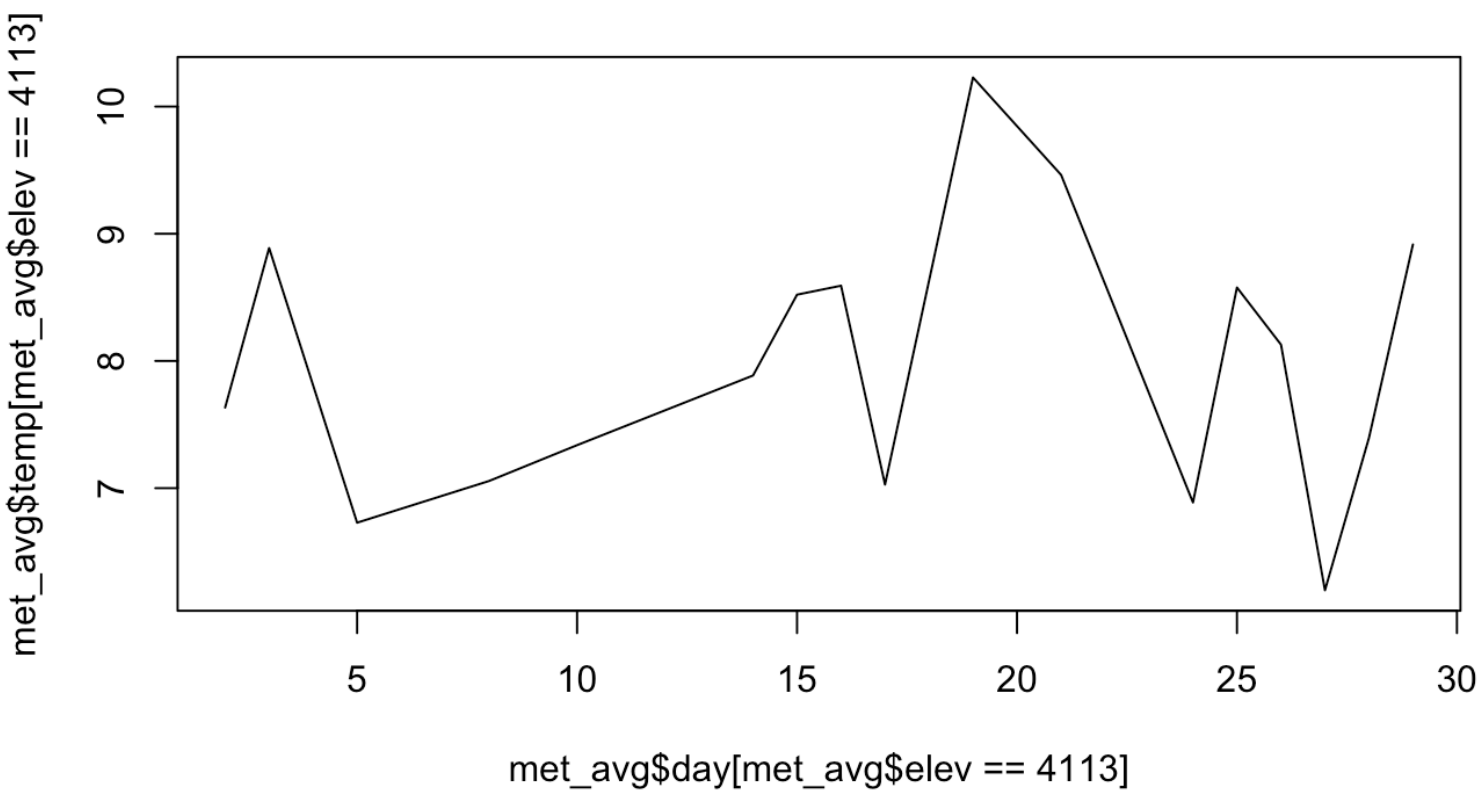
```
met_avg[!is.na(met_avg$elev_cat), ] %>%
  ggplot()+
  geom_boxplot(mapping=aes(x=elev_cat, y=temp, fill=elev_cat))
```



Geometric objects 5

Lineplots

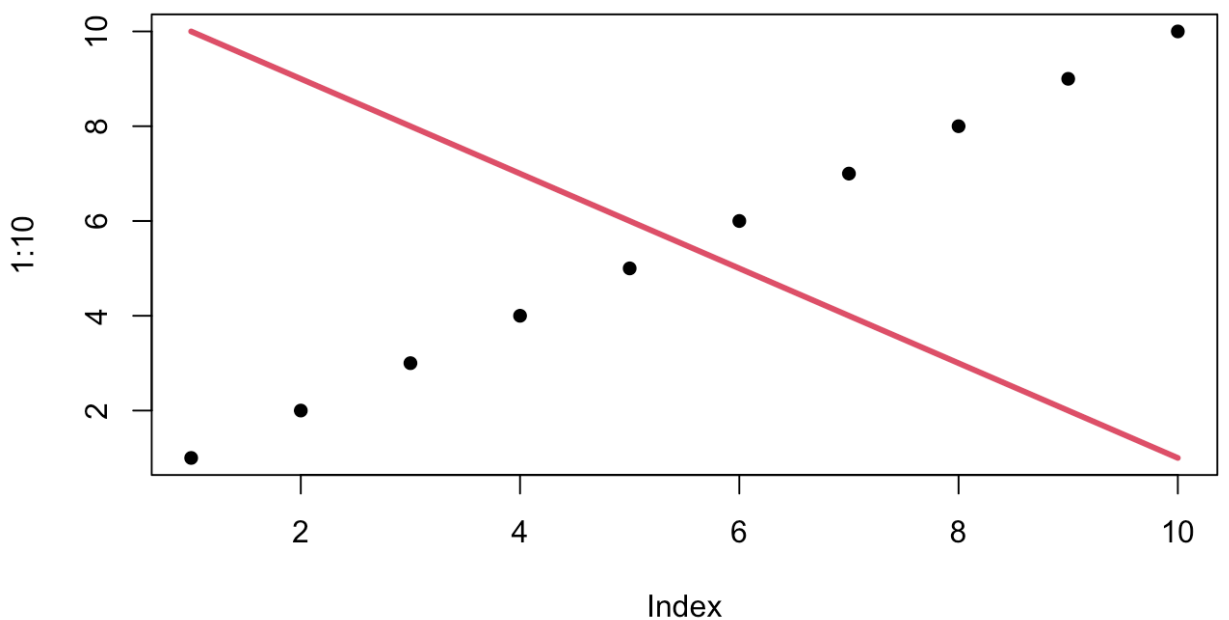
```
1 plot(met_avg$day[met_avg$elev==4113], met_avg$temp[met_avg$elev==4113], typ
```



Geometric objects 5

Just as you can add points to an existing plot, you can also add `lines()`

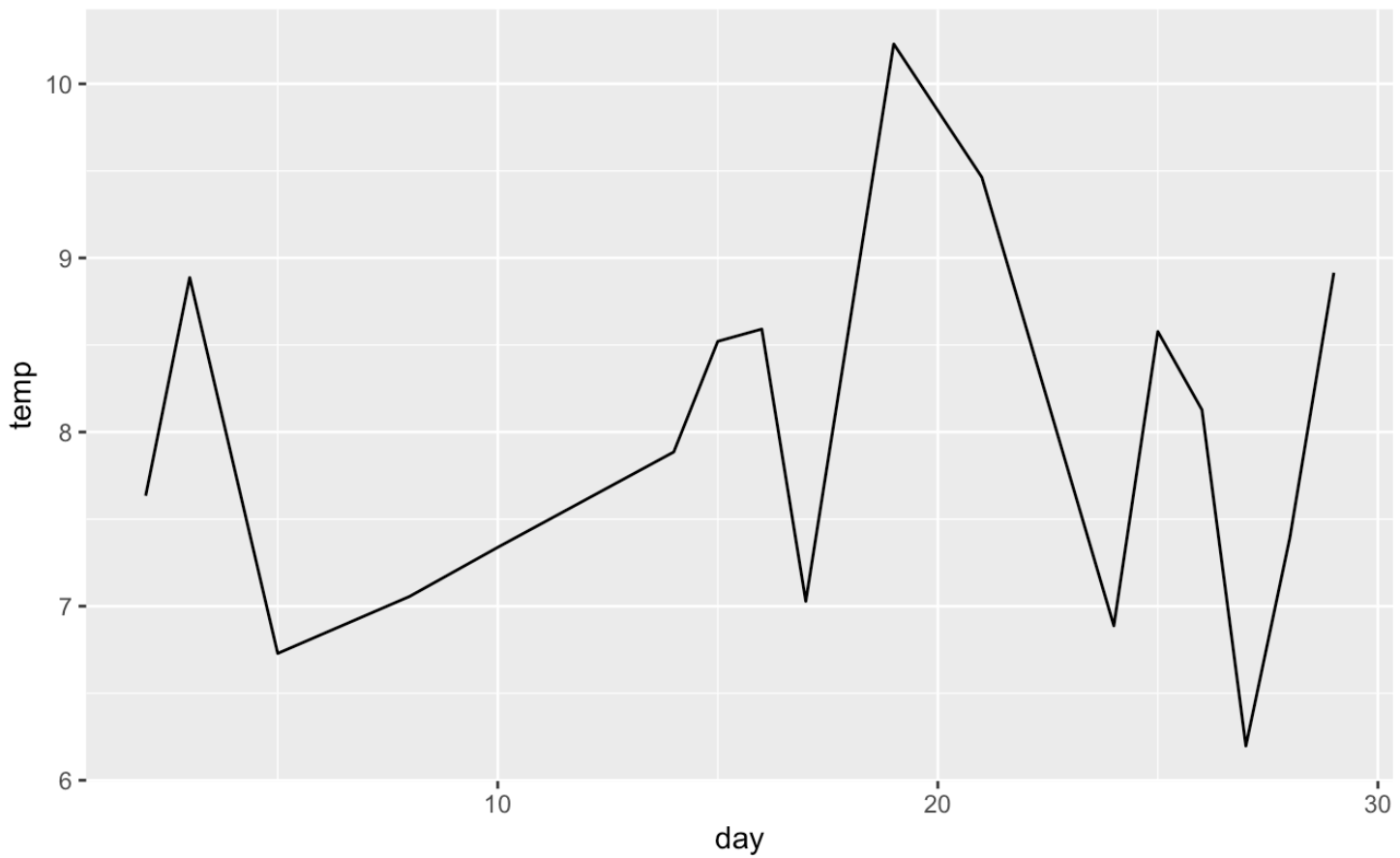
```
1 plot(1:10, pch = 16)
2 lines(10:1, col = 2, lwd = 3) # add a thick red line
```



Geometric objects 5

Lineplots

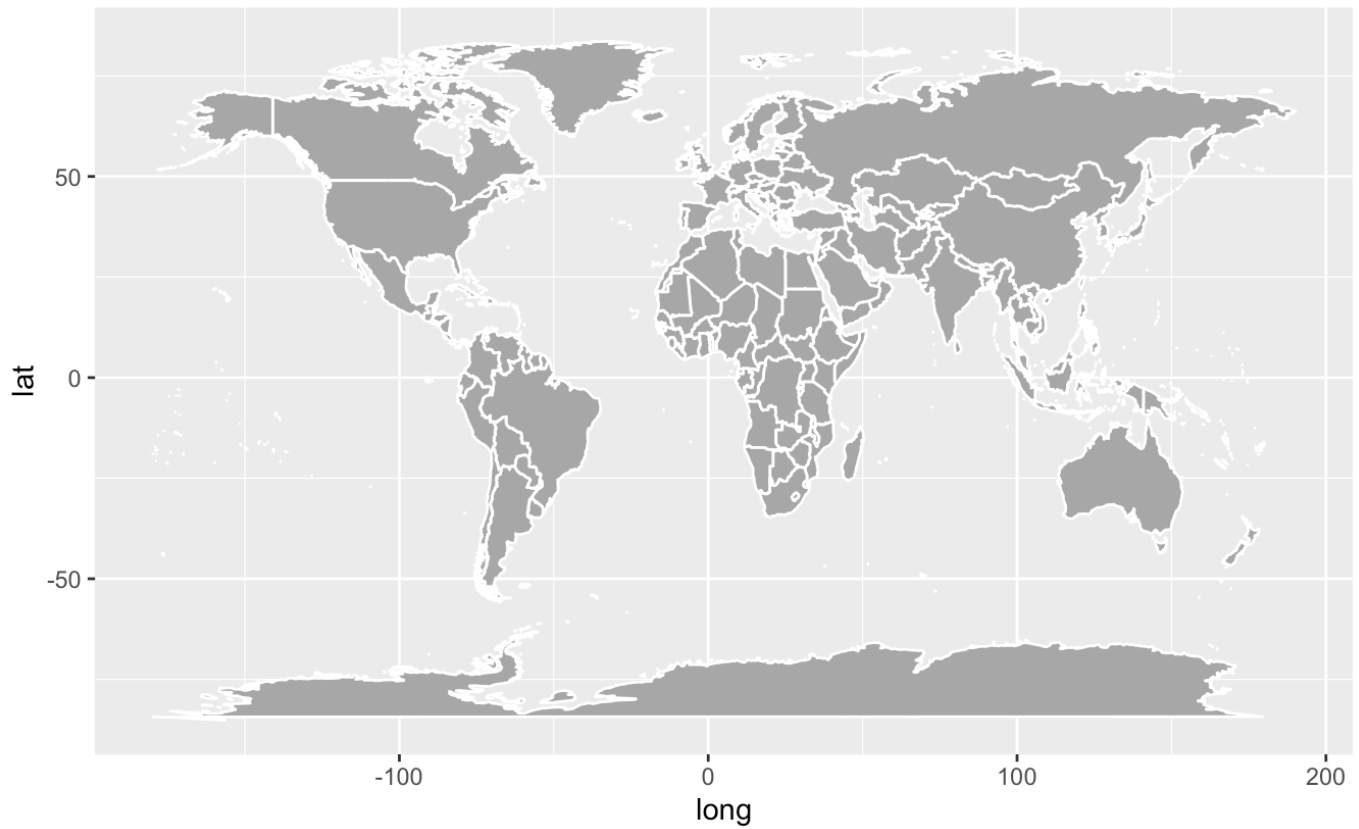
```
ggplot(data = met_avg[met_avg$elev==4113, ])+
 geom_line(mapping=aes(x=day, y=temp))
```



Geometric objects 5

Polygons

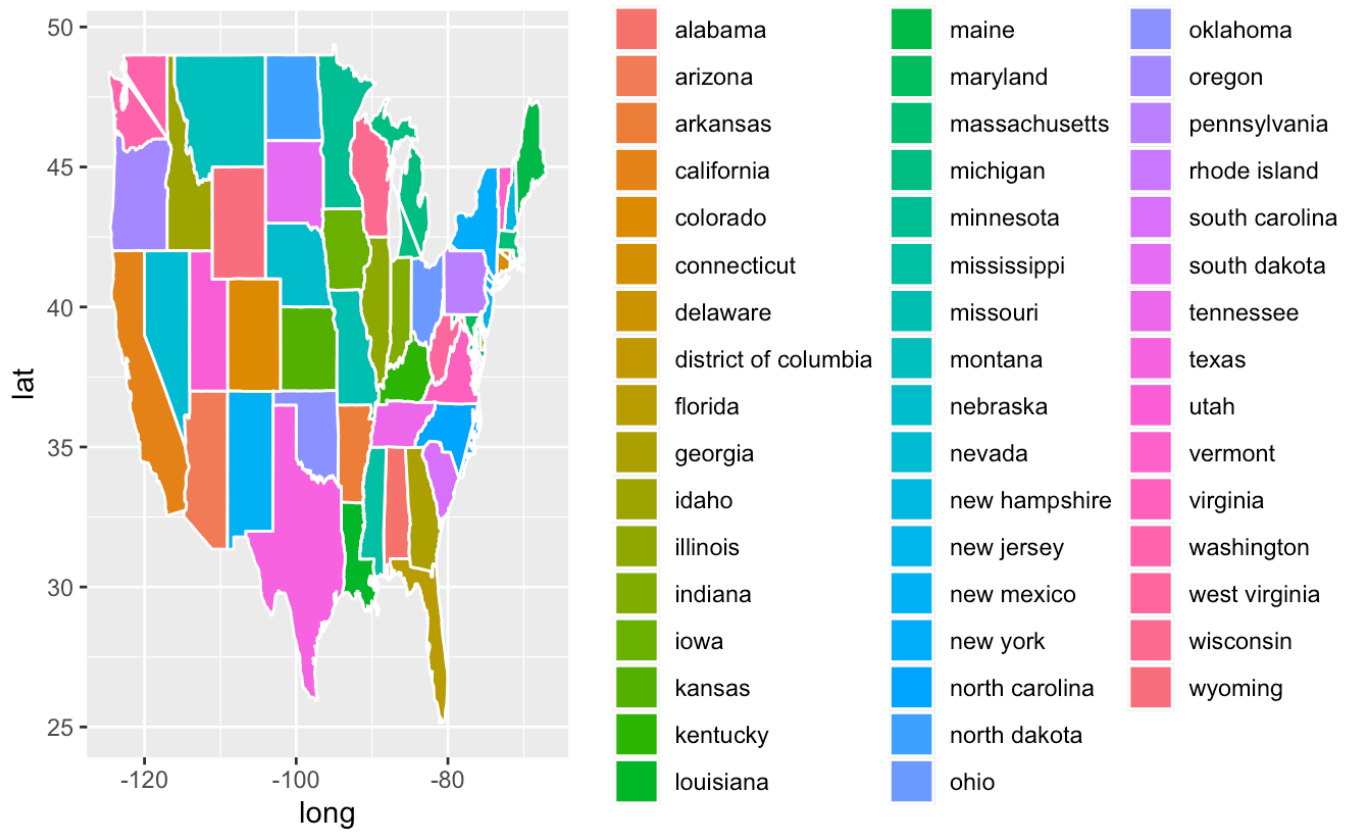
```
world_map <- map_data("world")
ggplot(data = world_map, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "darkgray", color = "white")
```



Geometric objects 5

Polygons

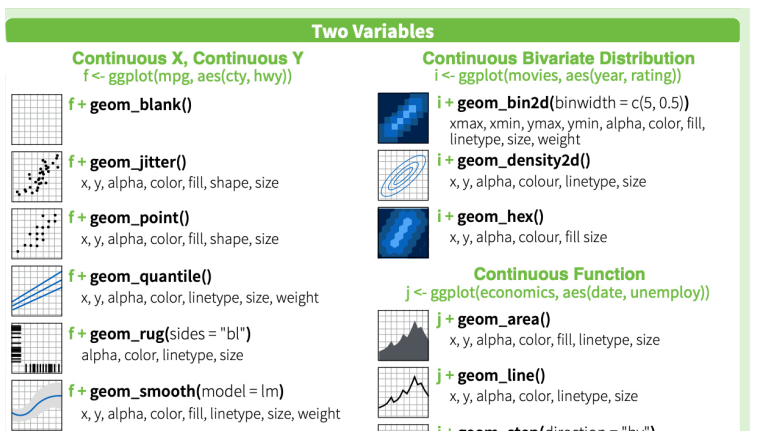
```
us_map <- map_data("state")
ggplot(data = us_map, aes(x = long, y = lat, fill = region)) +
  geom_polygon(color = "white")
```



Geoms - reference

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

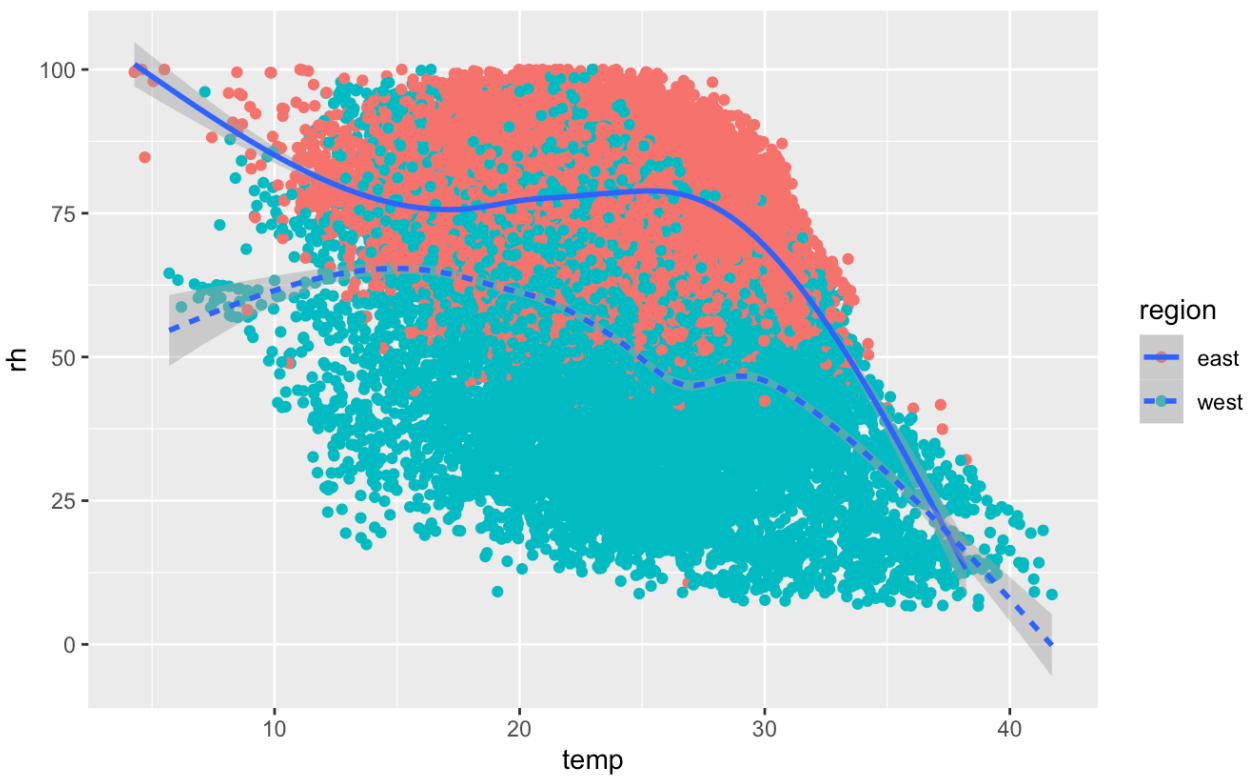
The best way to get a comprehensive overview is the ggplot2 cheatsheet, which you can find at https://github.com/rstudio/cheatsheets/blob/main/data-visualization-2.1.pdf



Multiple geoms 1

Let’s layer geoms

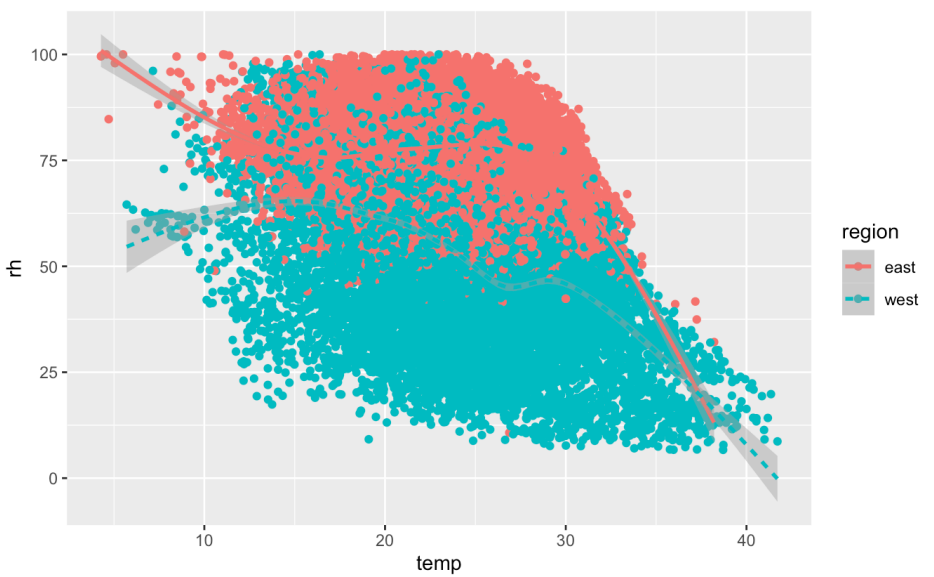
```
met_avg[!is.na(met_avg$region), ] %>%
  ggplot() +
  geom_point(mapping = aes(x = temp, y = rh, color = region))+
  geom_smooth(mapping = aes(x = temp, y = rh, linetype = region))
```



Multiple geoms 2

We can avoid repetition of aesthetics by passing a set of mappings to ggplot(). ggplot2 will treat these mappings as global mappings that apply to each geom in the graph.

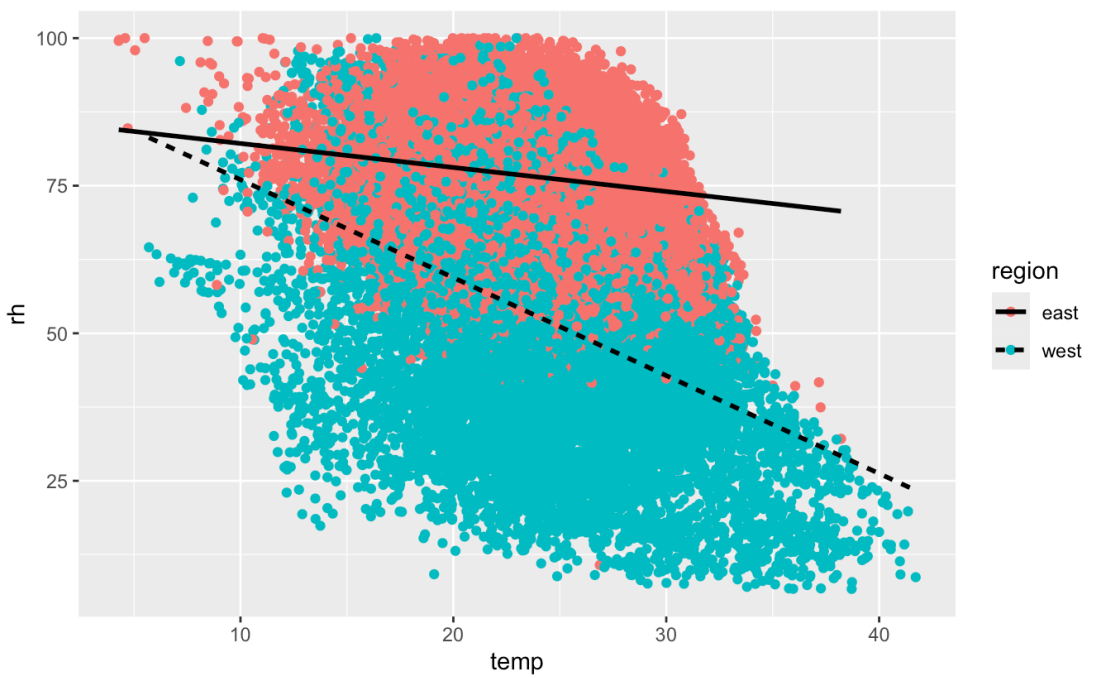
```
1 met_avg[!is.na(met_avg$region), ] %>%
2   ggplot(mapping = aes(x = temp, y = rh, color=region, linetype=region)) +
3   geom_point() +
4   geom_smooth()
```



Multiple geoms 2

`geom_smooth()` has options. For example if we want a linear regression line we add `method=lm`

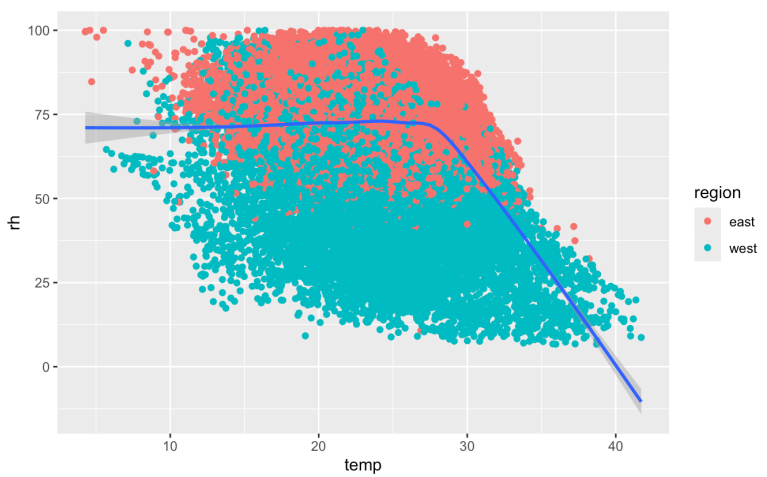
```
1 met_avg[!is.na(met_avg$region), ] %>%
2   ggplot(mapping = aes(x = temp, y = rh, color = region, linetype = region)
3   geom_point() +
4   geom_smooth(method = lm, se = FALSE, col = "black")
```



Multiple geoms 3

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.

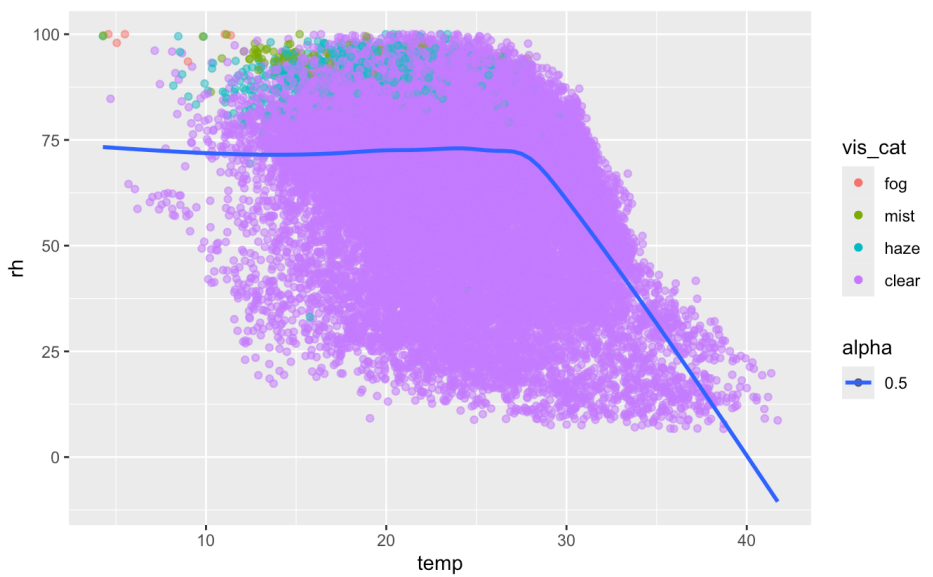
```
met_avg[!is.na(met_avg$region), ] %>%
  ggplot(mapping = aes(x = temp, y = rh)) +
  geom_point(mapping = aes(color = region)) +
  geom_smooth()
```



Multiple geoms 4

You can use the same idea to specify different data for each layer. Here, our smooth line displays the full met dataset but the points are colored by visibilty.

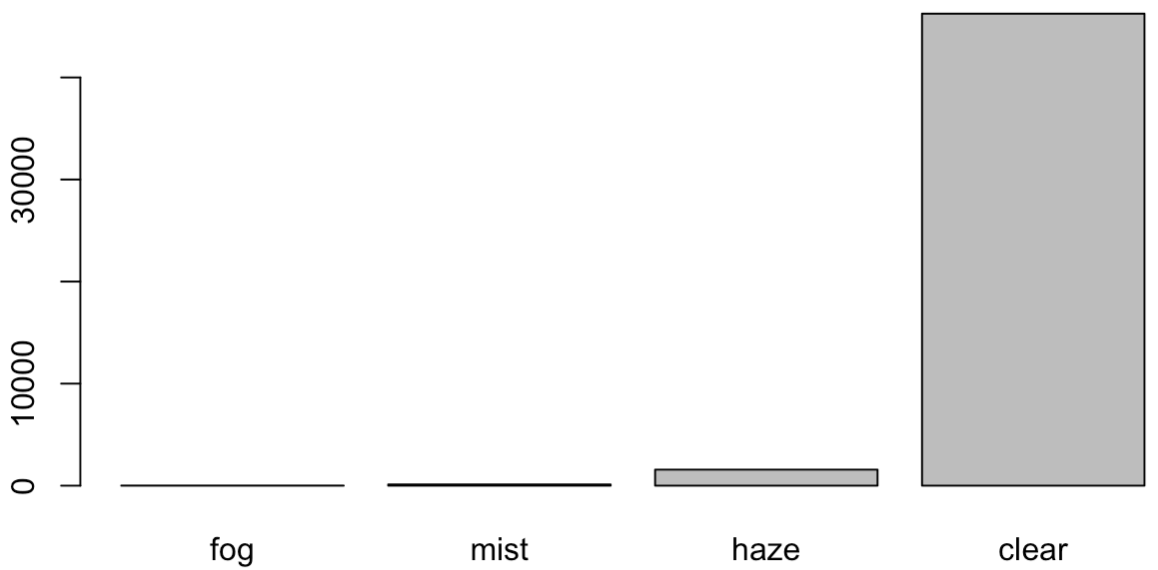
```
met_avg[!is.na(met_avg$vis_cat), ] %>%
  ggplot(mapping = aes(x = temp, y = rh, alpha = 0.5)) +
  geom_point(mapping = aes(color = vis_cat)) +
  geom_smooth(se = FALSE)
```



Statistical transformationas - e.g. Bar charts

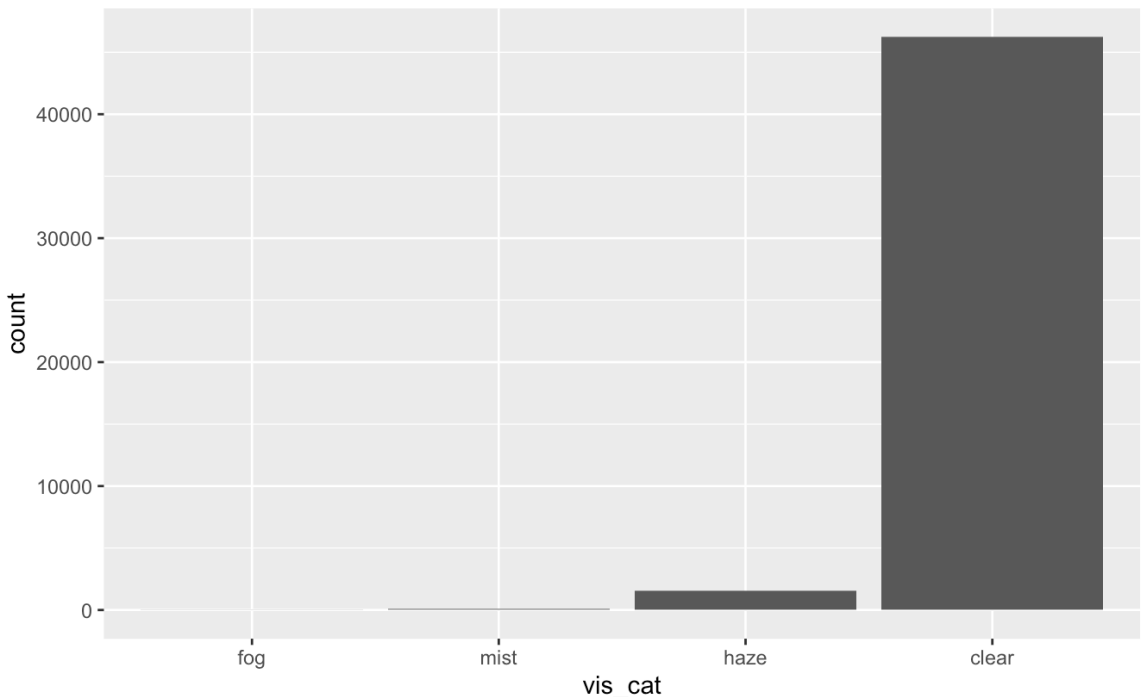
Let’s say we want to know the frequencies of the different visibility categories.

```
1 tab <- table(met_avg$vis_cat)
2 barplot(tab)
```



Statistical transformationas - e.g. Bar charts

```
met_avg %>%
filter(!is.na(met_avg$vis_cat)) %>%
  ggplot() +
  geom_bar(mapping = aes(x = vis_cat))
```

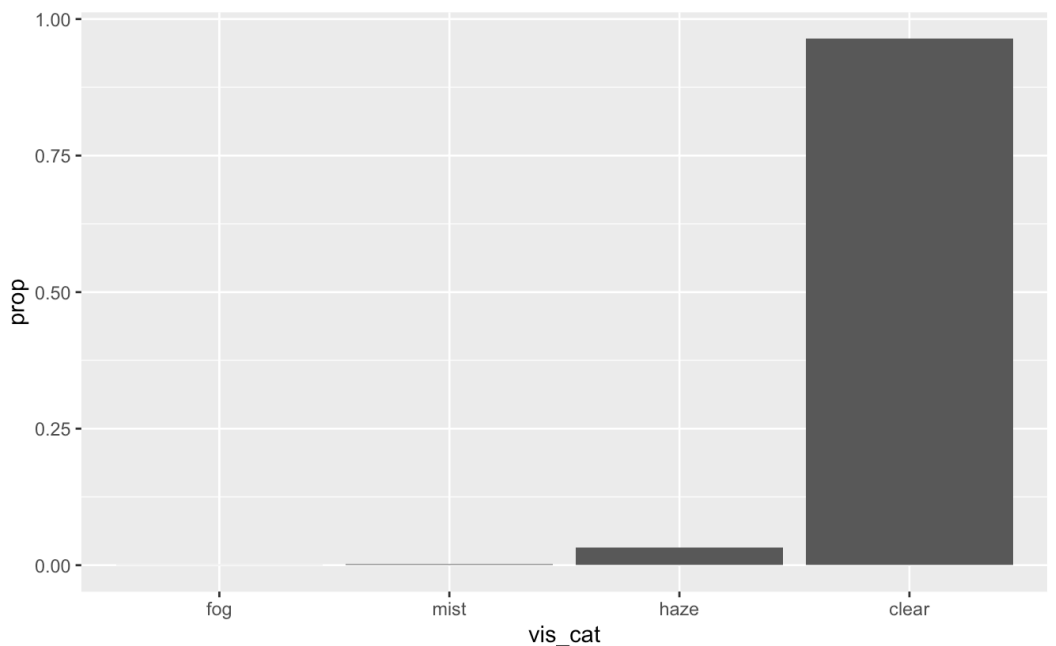


The algorithm uses a built-in statistical transformation, called a “stat”, to calculate the counts.

Bar charts 2

You can over-ride the stat a geom uses to construct its plot.
e.g., if we want to plpot proportions, rather than counts:

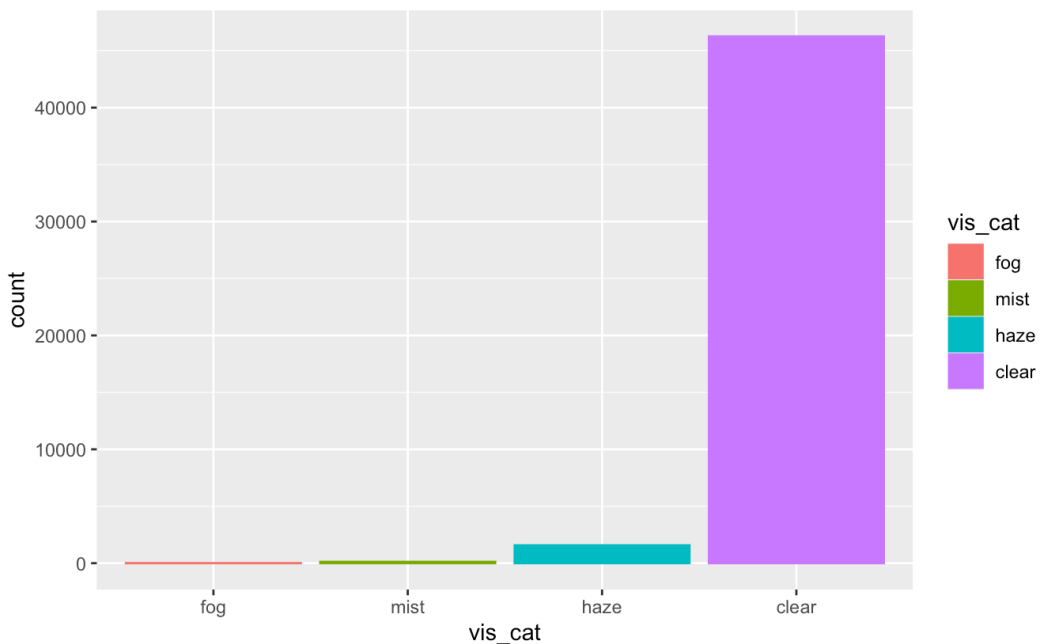
```
1 met_avg[!is.na(met_avg$vis_cat), ] %>%
2   ggplot() +
3   geom_bar(mapping = aes(x = vis_cat, y = stat(prop), group = 1))
```



Coloring barcharts

You can color a bar chart using either the color aesthetic, or, more usefully, fill:

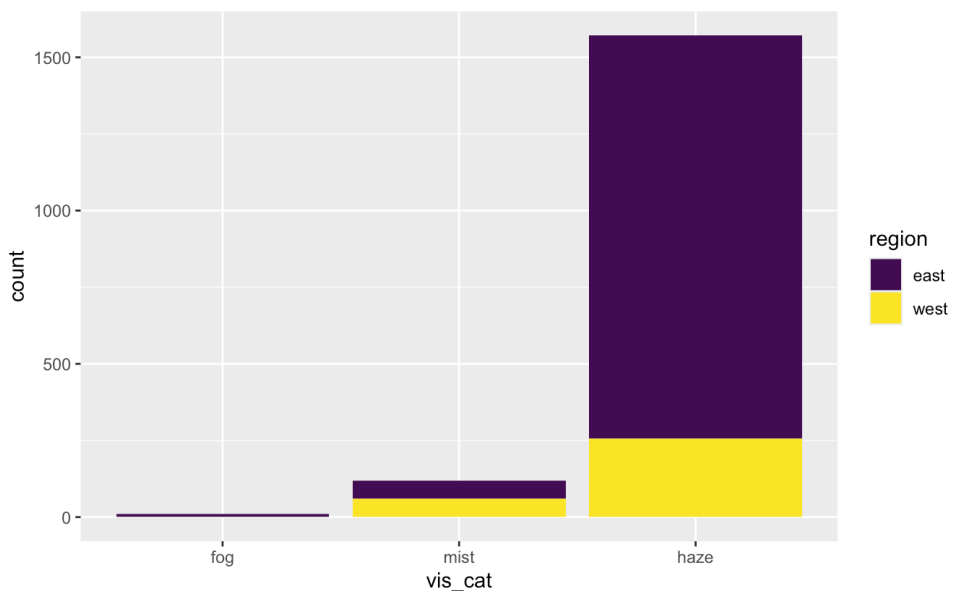
```
met_avg[!is.na(met_avg$vis_cat), ] %>%
  ggplot() +
  geom_bar(mapping = aes(x = vis_cat, color = vis_cat, fill=vis_cat))
```



Coloring barcharts

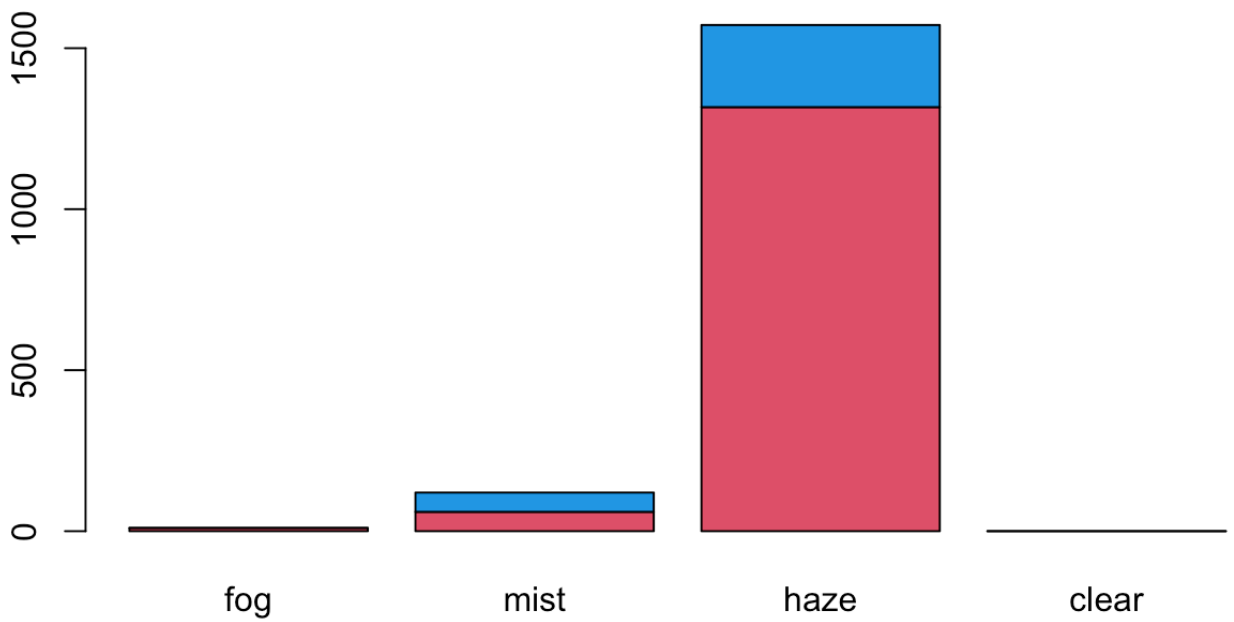
More interestingly, you can fill by another variable (here, ‘region’). We also show that we can change the color scale.

```
met_avg[!is.na(met_avg$vis_cat) & met_avg$vis_cat != "clear", ] %>%
  ggplot() +
  geom_bar(mapping = aes(x = vis_cat, fill = region))+
  scale_fill_viridis_d()
```



Coloring barcharts

```
1 tab <- table(met_avg$region[!is.na(met_avg$vis_cat) & met_avg$vis_cat != "c
2              met_avg$vis_cat[!is.na(met_avg$vis_cat) & met_avg$vis_cat != "
3 barplot(tab, col = c(2,4))
```



Coloring barcharts

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

```
met_avg[!is.na(met_avg$vis_cat) & met_avg$vis_cat != "clear", ] %>%
  ggplot() +
  geom_bar(mapping = aes(x = elev_cat, fill = vis_cat), position = "dodge")
```

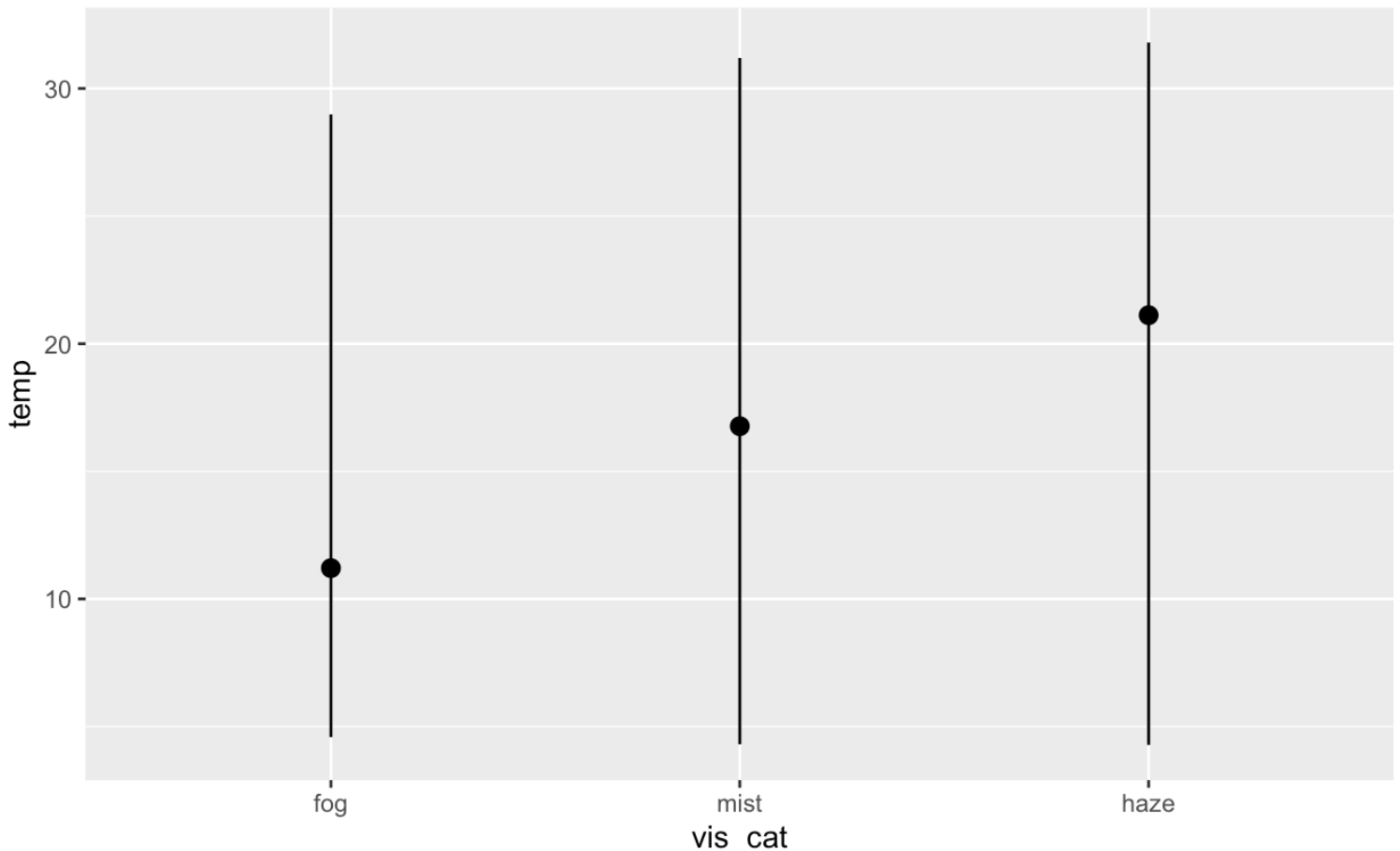
Statistical transformations - another example

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

```
l <- met_avg[!is.na(met_avg$vis_cat) & met_avg$vis_cat != "clear", ] %>%
  ggplot() +
    stat_summary(mapping = aes(x = vis_cat, y = temp),
    fun.min = min,
    fun.max = max,
    fun = median)
```

Statistical transformations - another example

```
1 l
```



Position adjustments

An option that can be very useful is `position = "jitter"`. This adds a small amount of random noise to each point. This spreads out points that might otherwise be overlapping.

```
nojitter <- ggplot(data = met_avg[1:1000,]) +
  geom_point(mapping = aes(x = vis_cat, y = temp))

jitter <- ggplot(data = met_avg[1:1000,]) +
  geom_point(mapping = aes(x = vis_cat, y = temp), position = "jitter")
```

Position adjustments

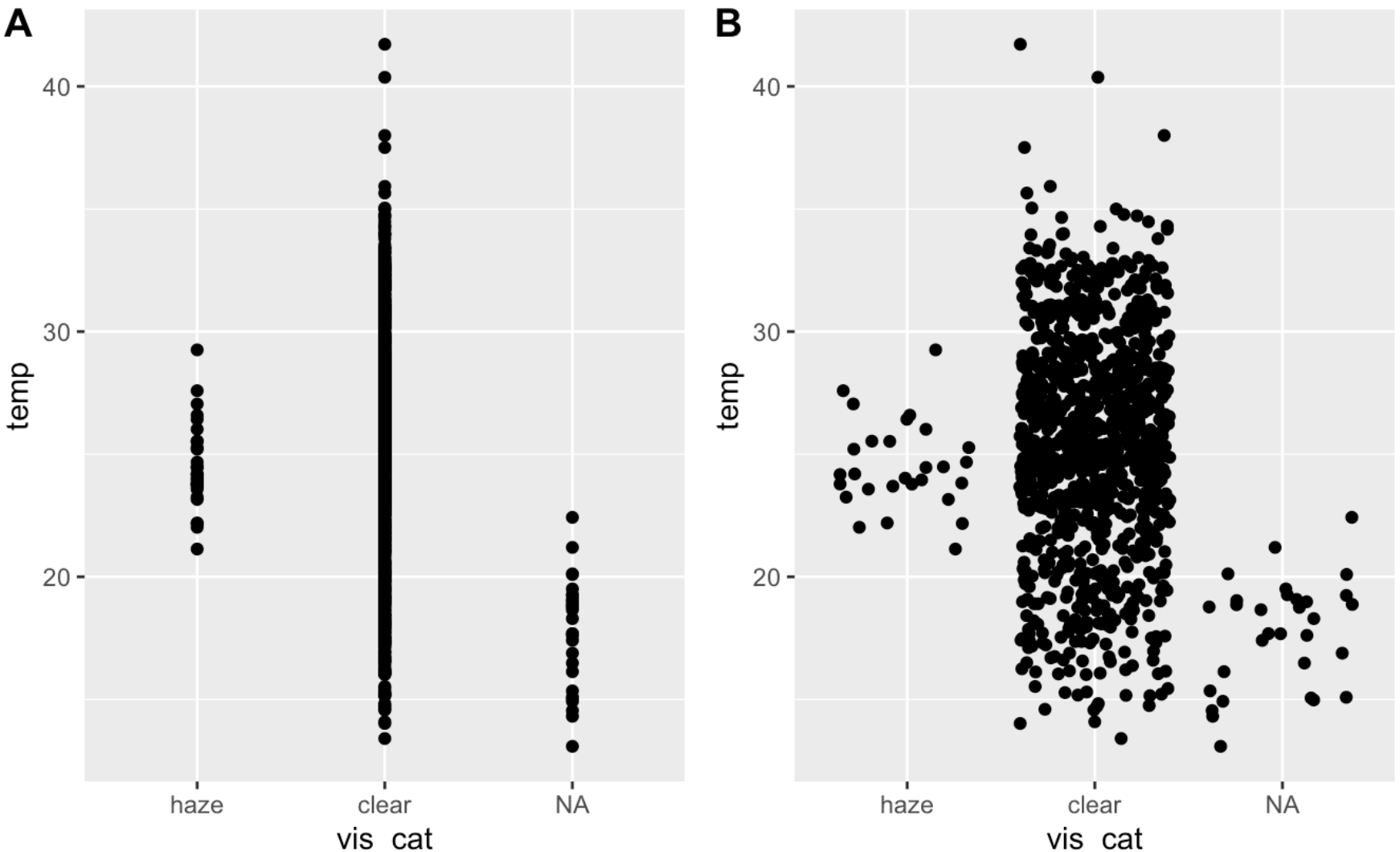
An option that can be very useful is `position = "jitter"`. This adds a small amount of random noise to each point. This spreads out points that might otherwise be overlapping.

```
nojitter <- ggplot(data = met_avg[1:1000,]) +
  geom_point(mapping = aes(x = vis_cat, y = temp))

jitter <- ggplot(data = met_avg[1:1000,]) +
  geom_point(mapping = aes(x = vis_cat, y = temp), position = "jitter")
```

Position adjustments

```
1 plot_grid(nojitter, jitter, labels = "AUTO")
```



Coordinate systems

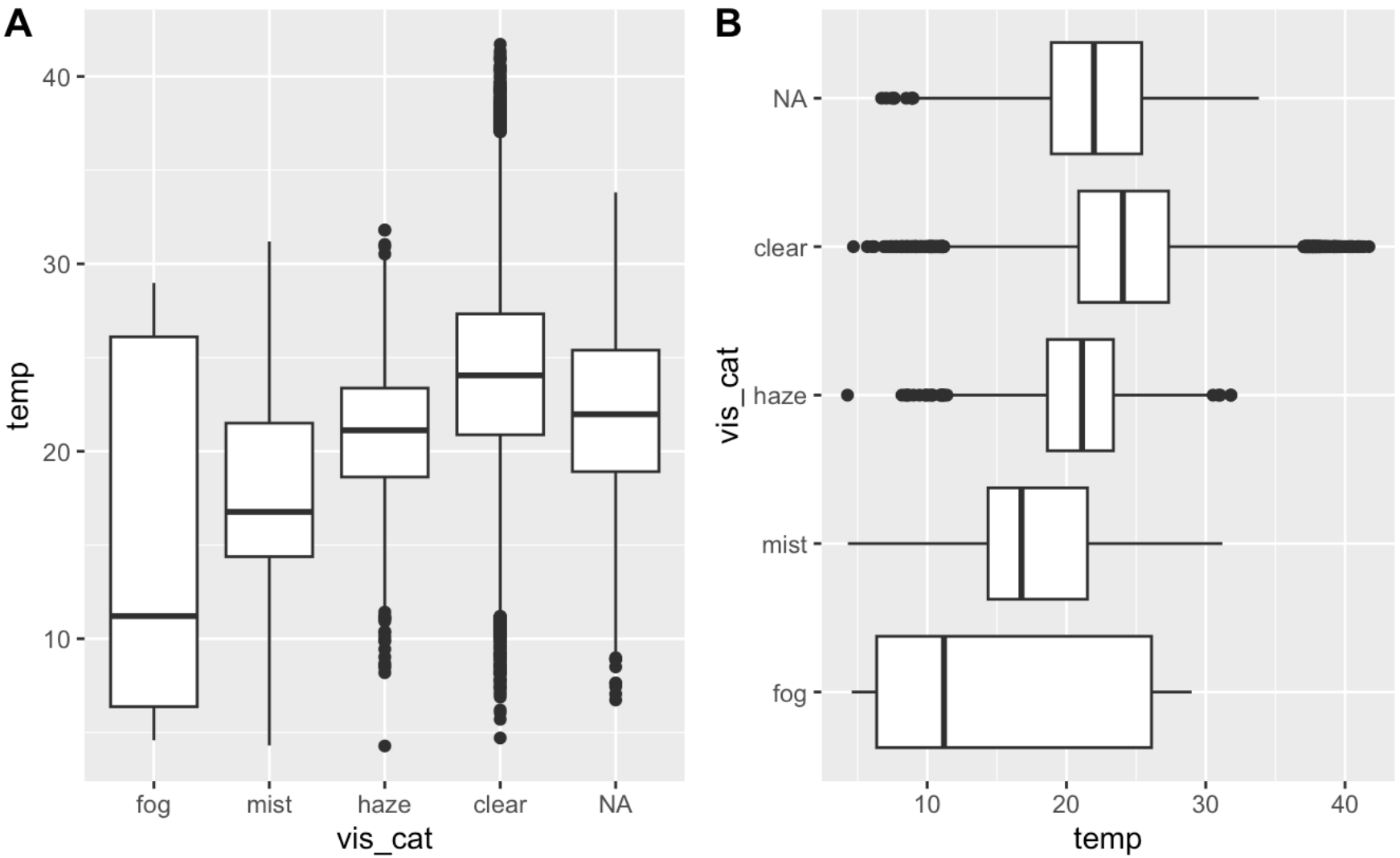
Coordinate systems are one of the more complicated corners of ggplot. To start with something simple, here's how to flip axes:

```
unflipped <- ggplot(data = met_avg) +
  geom_boxplot(mapping = aes(x = vis_cat, y = temp))

flipped <- ggplot(data = met_avg) +
  geom_boxplot(mapping = aes(x = vis_cat, y = temp)) +
  coord_flip()
```

Coordinate systems

```
1 plot_grid(unflipped, flipped, labels = "AUTO")
```



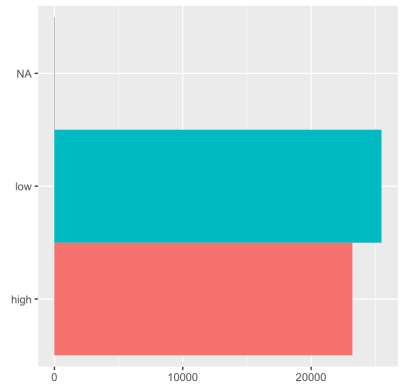
Coordinate systems

There is also the ability to control the aspect ratio using `coord_quickmap()` and to use polar coordinates with `coord_polar()`.

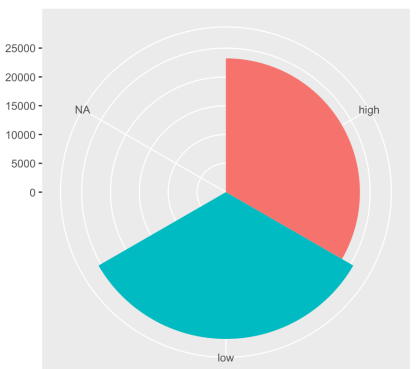
```
bar <- ggplot(data = met_avg) +
  geom_bar(mapping = aes(x = elev_cat, fill = elev_cat), show.legend = FALS
  theme(aspect.ratio = 1) +
  labs(x = NULL, y = NULL)
```

Coordinate systems

```
1 bar + coord_flip()
```

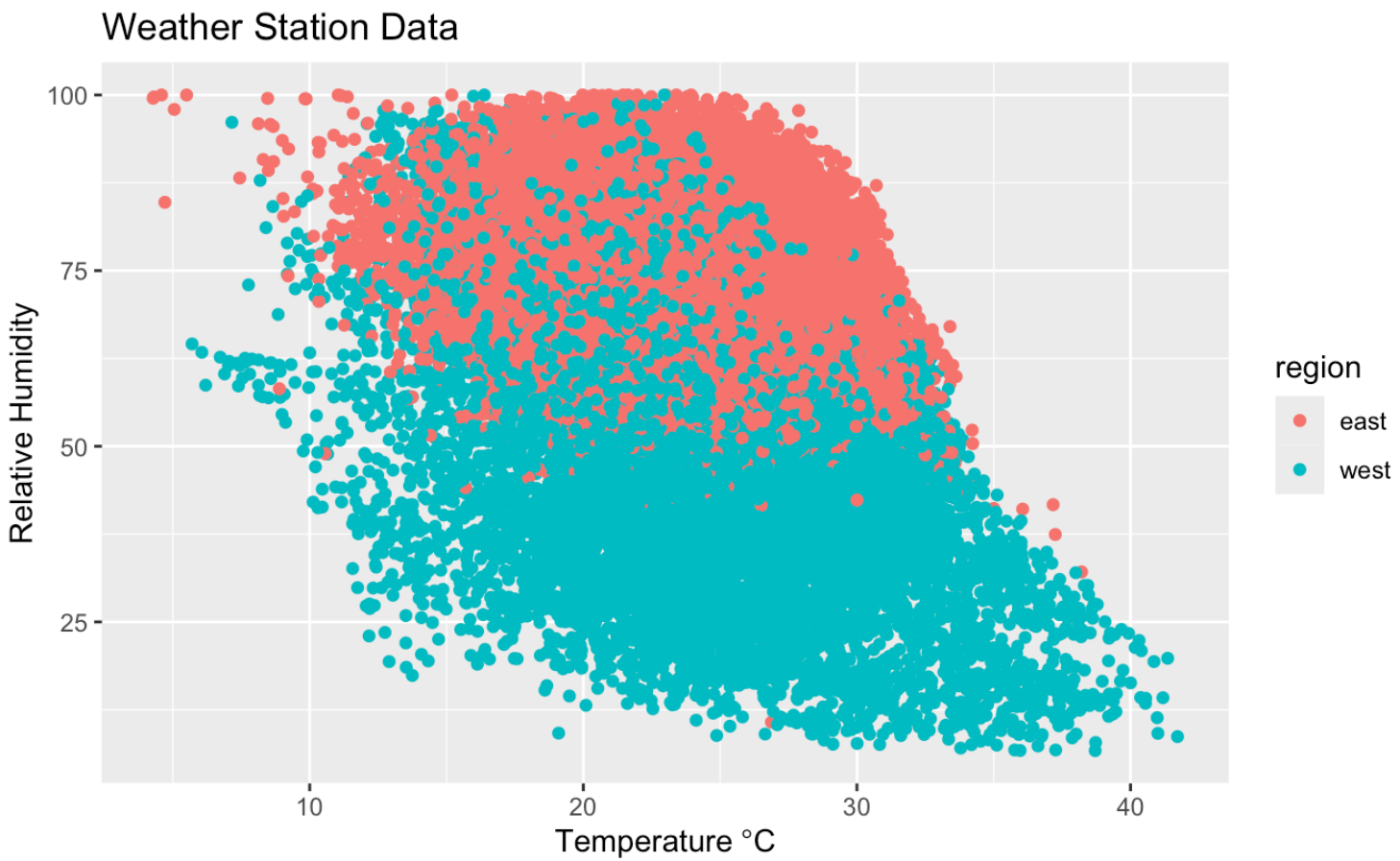



```
1 bar + coord_polar()
```



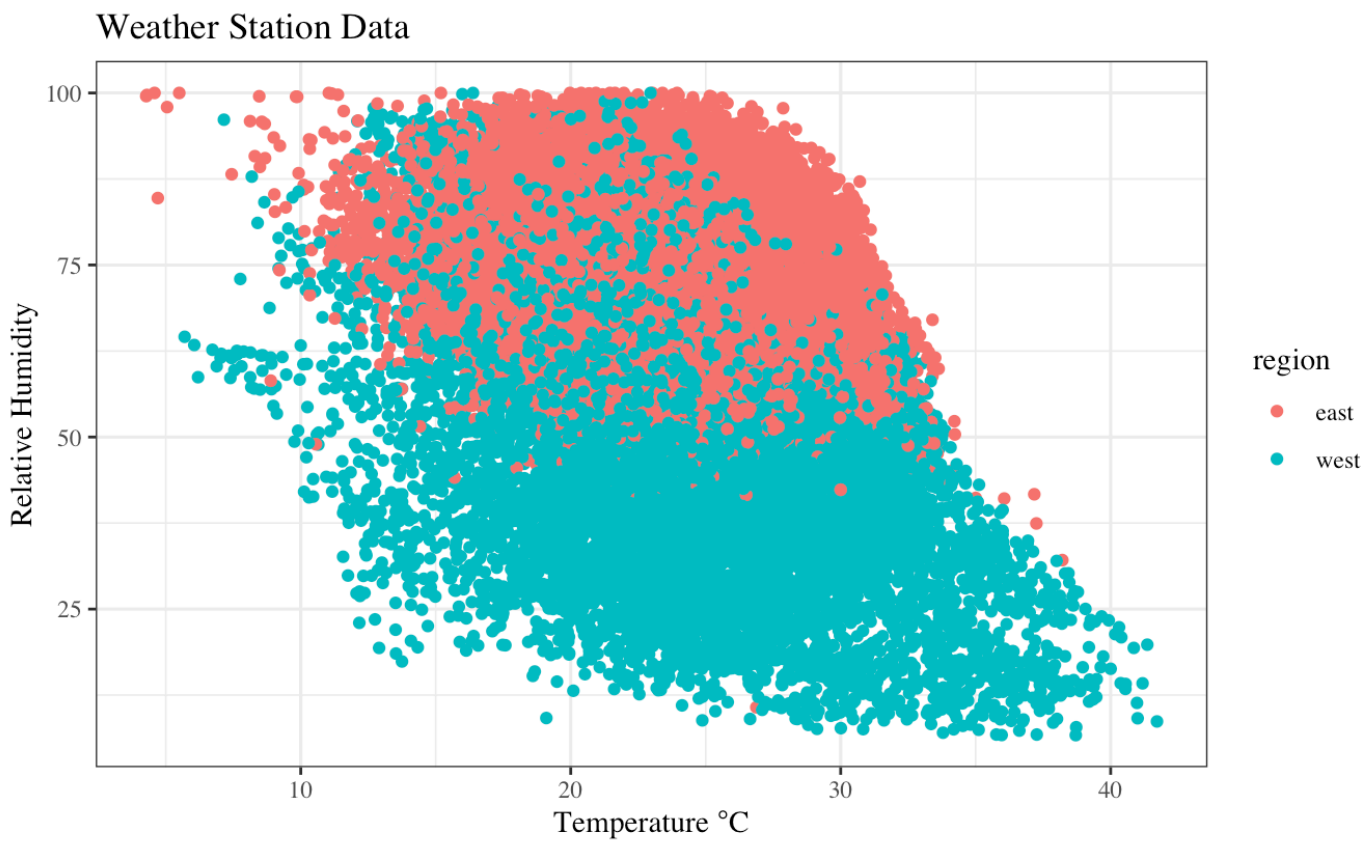
Modifying labels

```
1 ggplot(met_avg[!is.na(met_avg$region), ]) +
2   geom_point(aes(temp, rh, color = region)) +
3   labs(title = "Weather Station Data") +
4   labs(x = expression("Temperature" *~ degree * C), y = "Relative Humidity"
```



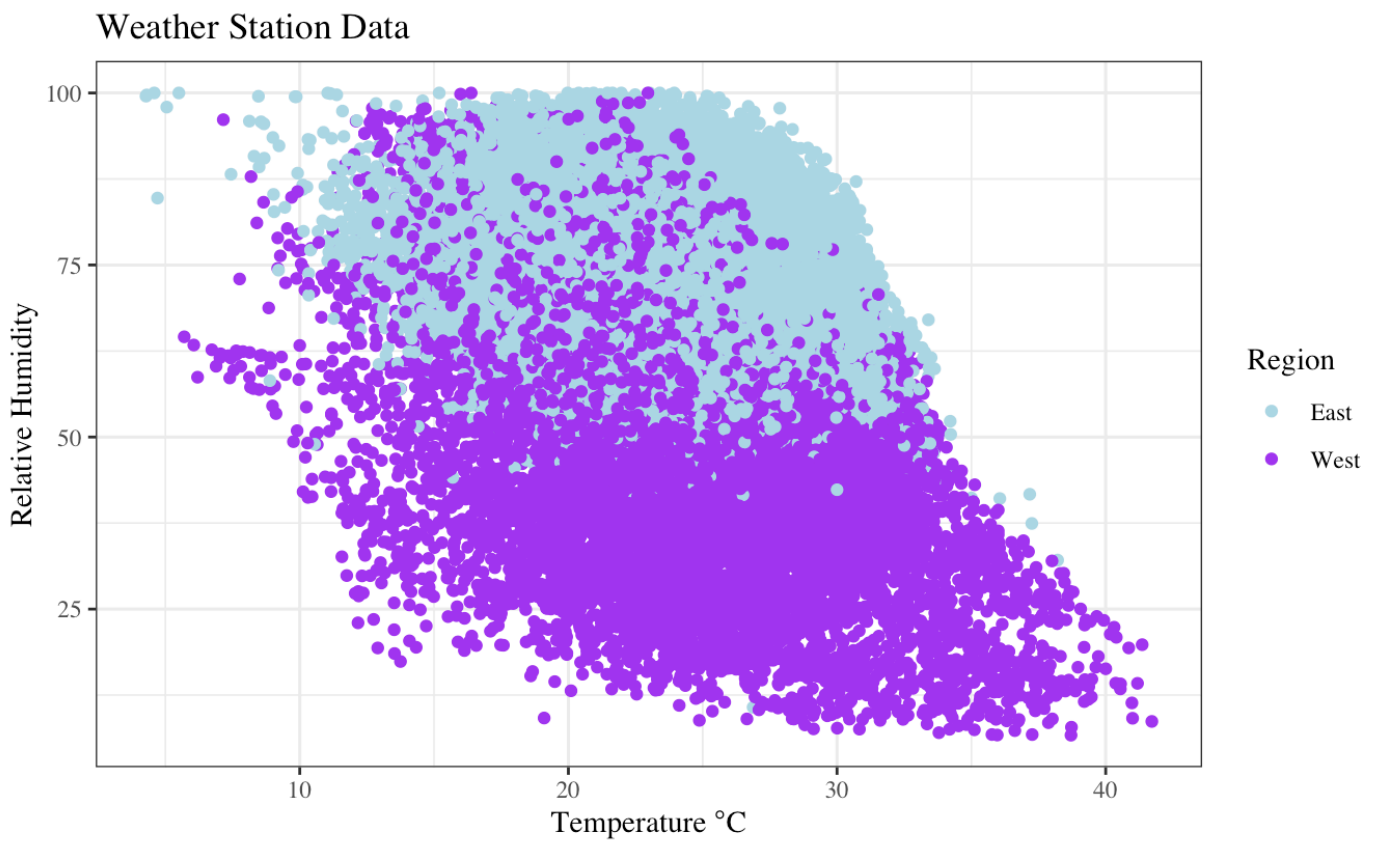
Changing the Theme

```
1 ggplot(met_avg[!is.na(met_avg$region), ]) +
2   geom_point(aes(temp, rh, color = region)) +
3   labs(title = "Weather Station Data") +
4   labs(x = expression("Temperature"*~degree*C), y = "Relative Humidity")+
5   theme_bw(base_family = "Times")
```



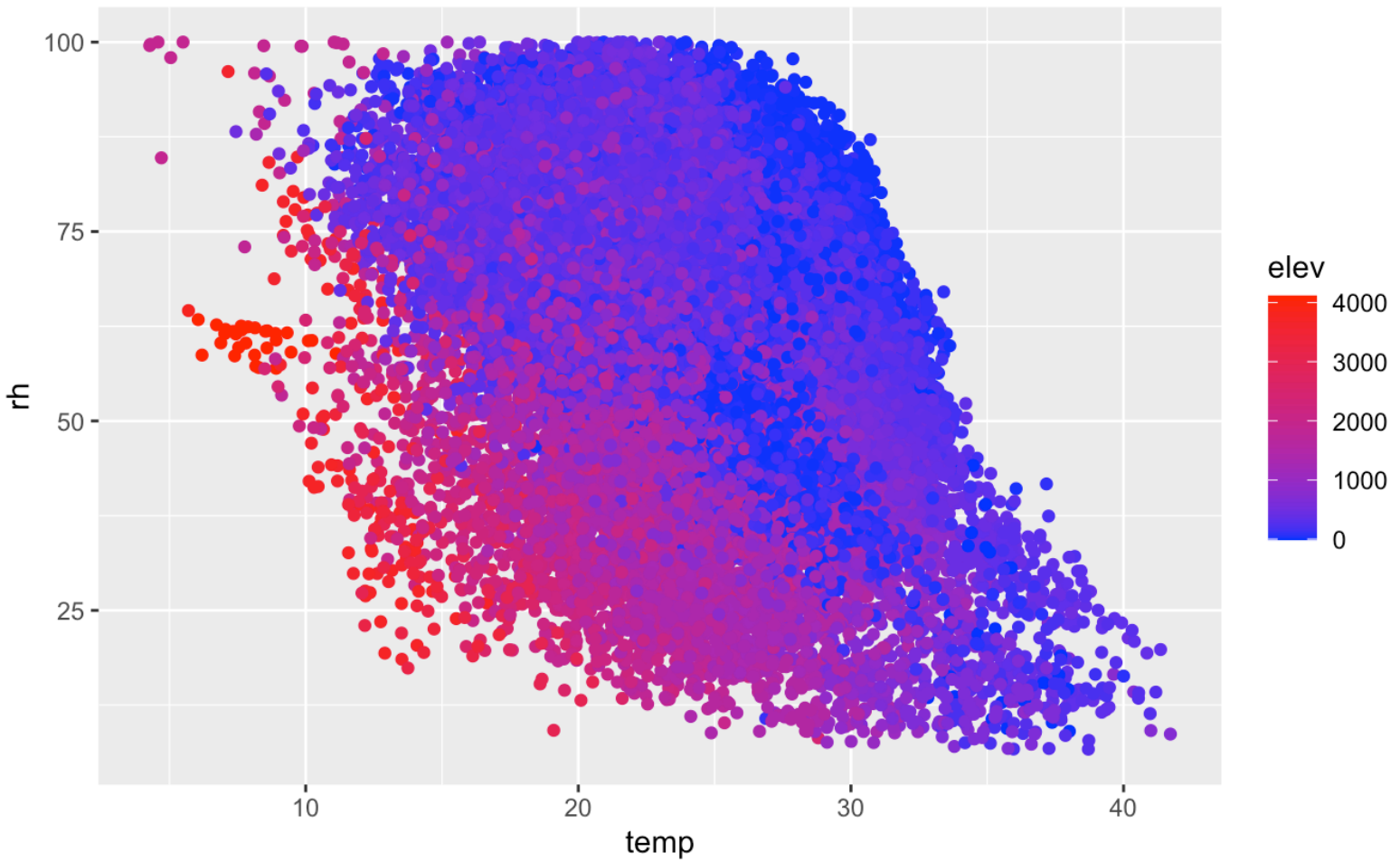
Changing the Legend

```
1 ggplot(met_avg[!is.na(met_avg$region), ]) +
2   geom_point(aes(temp, rh, color = region)) +
3   labs(title = "Weather Station Data",x = expression("Temperature"*~degree*
4   scale_color_manual(name="Region", labels=c("East", "West"), values=c("eas
5   theme_bw(base_family = "Times")
```



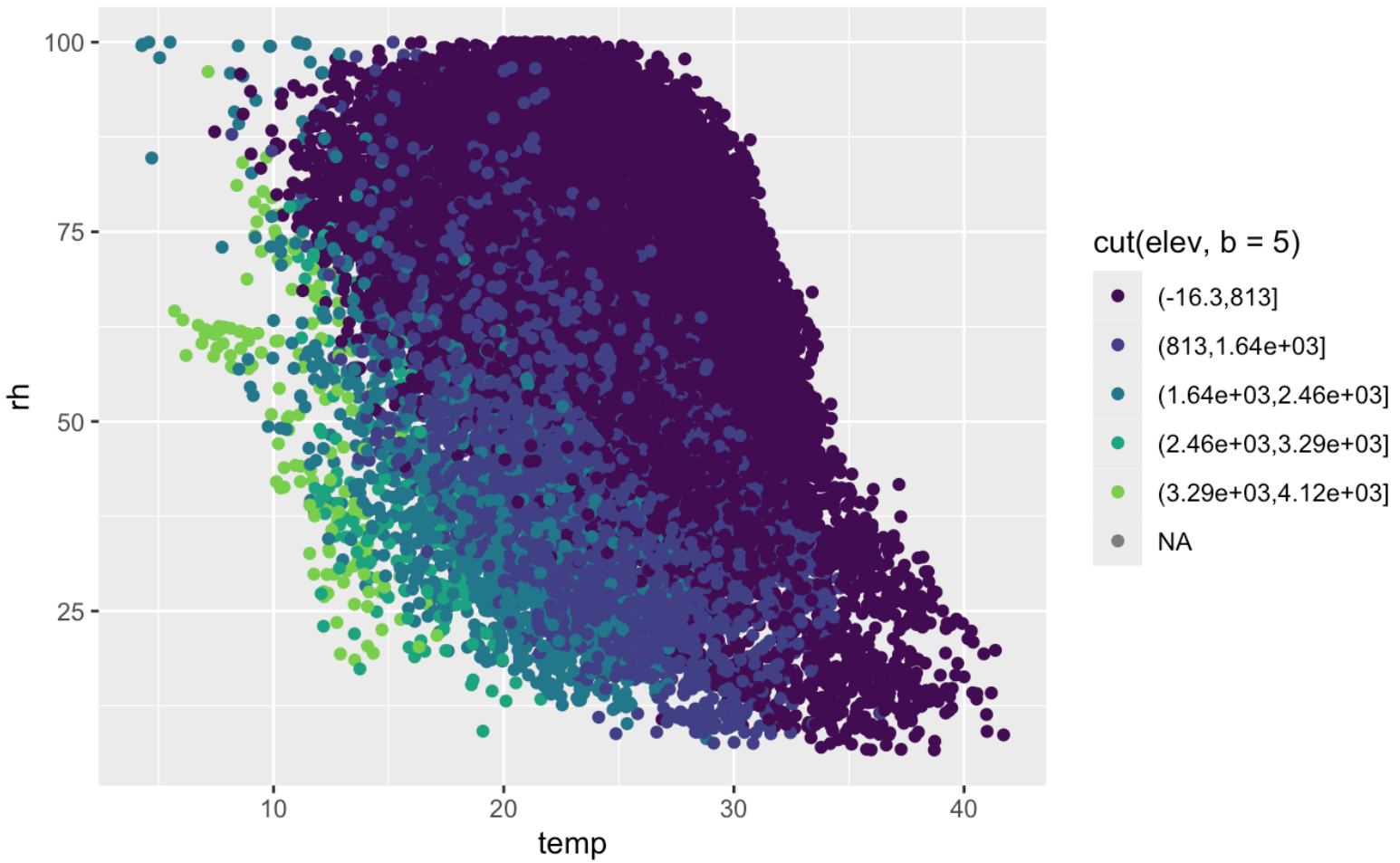
Changing Colorscales

```
ggplot(data = met_avg) +
  geom_point(mapping=aes(x=temp, y=rh, color=elev))+
  scale_color_gradient(low="blue", high="red")
```



Changing Colorscales

```
1 ggplot(data=met_avg) +
2   geom_point(mapping= aes(x=temp, y=rh, color = cut(elev, b=5))) +
3   scale_color_manual(values = viridis::viridis(6))
```



A Great reference

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

https://r-graphics.org/

Reminder - the ggplot2 cheatsheet

A briefer summary can be found here:

https://github.com/rstudio/cheatsheets/blob/main/data-visualization-2.1.pdf

Rstudio has a variety of other great Cheatsheets.

Maps with leaflet

Let's create a map of monthly average temperatures at each of the weather stations and color the points by a temperature gradient. We need to create a color palette and we can add a legend.

```
library(leaflet)
met_avg2 <- summarize(met,
                      temp =     mean(temp, na.rm = TRUE),
                      lat =      mean(lat, na.rm = TRUE),
                      lon =      mean(lon, na.rm = TRUE),
                      .by = c(USAFID))
met_avg2 <- met_avg2[!is.na(met_avg2$temp), ]

# Generating a color palette
temp.pal <- colorNumeric(c('darkgreen','goldenrod','brown'), domain=met_avg2$temp)

temp.pal(20)
```

```
[1] "#B69A19"
```

```
temp.pal
```

```
function (x)
{
    if (length(x) == 0 || all(is.na(x))) {
        return(pf(x))
    }
    if (is.null(rng))
        rng <- range(x, na.rm = TRUE)
    rescaled <- scales::rescale(x, from = rng)
    if (any(rescaled < 0 | rescaled > 1, na.rm = TRUE))
```

```
        warning("Some values were outside the color scale and will be treated as NA")
    if (reverse) {
        rescaled <- 1 - rescaled
    }
    pf(rescaled)
}
<bytecode: 0x14fa3f2a8>
<environment: 0x14fa11bd8>
attr(,"colorType")
[1] "numeric"
```

Maps with leaflet

For the tile providers, take a look at this site: https://leaflet-extras.github.io/leaflet-providers/preview/

```
tempmap <- leaflet(met_avg2) %>%
  # The looks of the Map
  addProviderTiles('CartoDB.Positron') %>%
  # Some circles
  addCircles(
    lat = ~lat, lng=~lon,
                                                  # HERE IS OUR PAL!
    label = ~paste0(round(temp,2), ' C'), color = ~ temp.pal(temp),
    opacity = 1, fillOpacity = 1, radius = 500
    ) %>%
  # And a pretty legend
  addLegend('bottomleft', pal=temp.pal, values=met_avg2$temp,
            title='Temperature, C', opacity=1)
```

Maps with leaflet

```
1 tempmap
```