ggplot

Eric Archer

ggplot is based entirely around constructing a properly formed data.frame of what needs to be plotted and then constructing a plotting statement with the components linked with `+`. The values in the data.frame that are to be plotted are identified with the appropriate "mapping" that is referred to as an "aesthetic". They are specified with the `aes` function with a ggplot object. The first item in a ggplot figure is the `ggplot` function that sets up the data and aesthetic. Here we are setting up a base ggplot object that has the `x` value mapped to `temperature` and the `y` value mapped to `depth`:

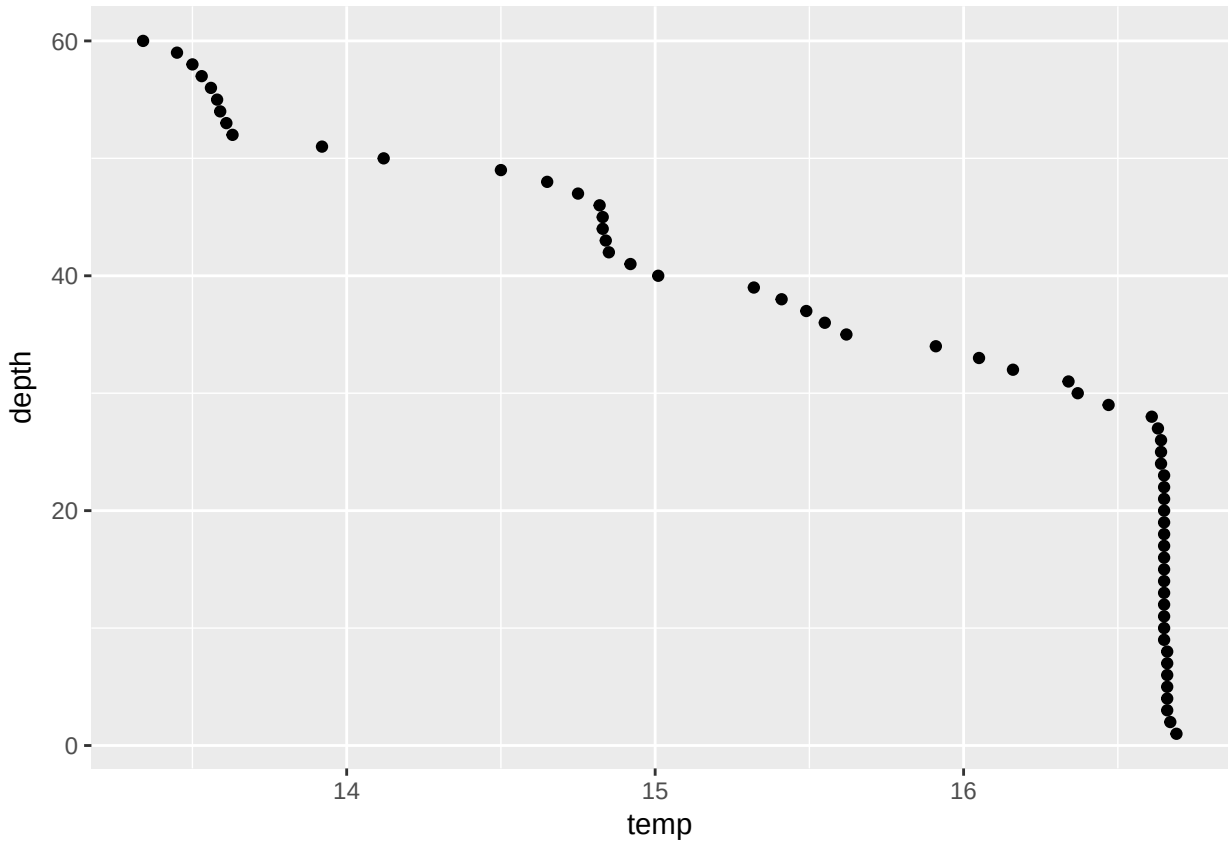
```
# read CTD data and format date columns
ctd <- read.csv("ctd.csv", stringsAsFactors = FALSE)
ctd$date <- as.POSIXct(ctd$sample_date, format = "%Y-%m-%d")
ctd$month <- months(as.Date(ctd$date))
ctd$month <- factor(ctd$month, levels = month.name)
ctd$year <- as.numeric(format(ctd$date, "%Y"))
ctd$quarter <- factor(quarters(as.Date(ctd$date)))

df <- ctd[ctd$station == "Station.1" & grepl("2015", ctd$sample_date) & ctd$month == "February", ]

library(ggplot2)
p <- ggplot(df, mapping = aes(x = temp, y = depth))
```

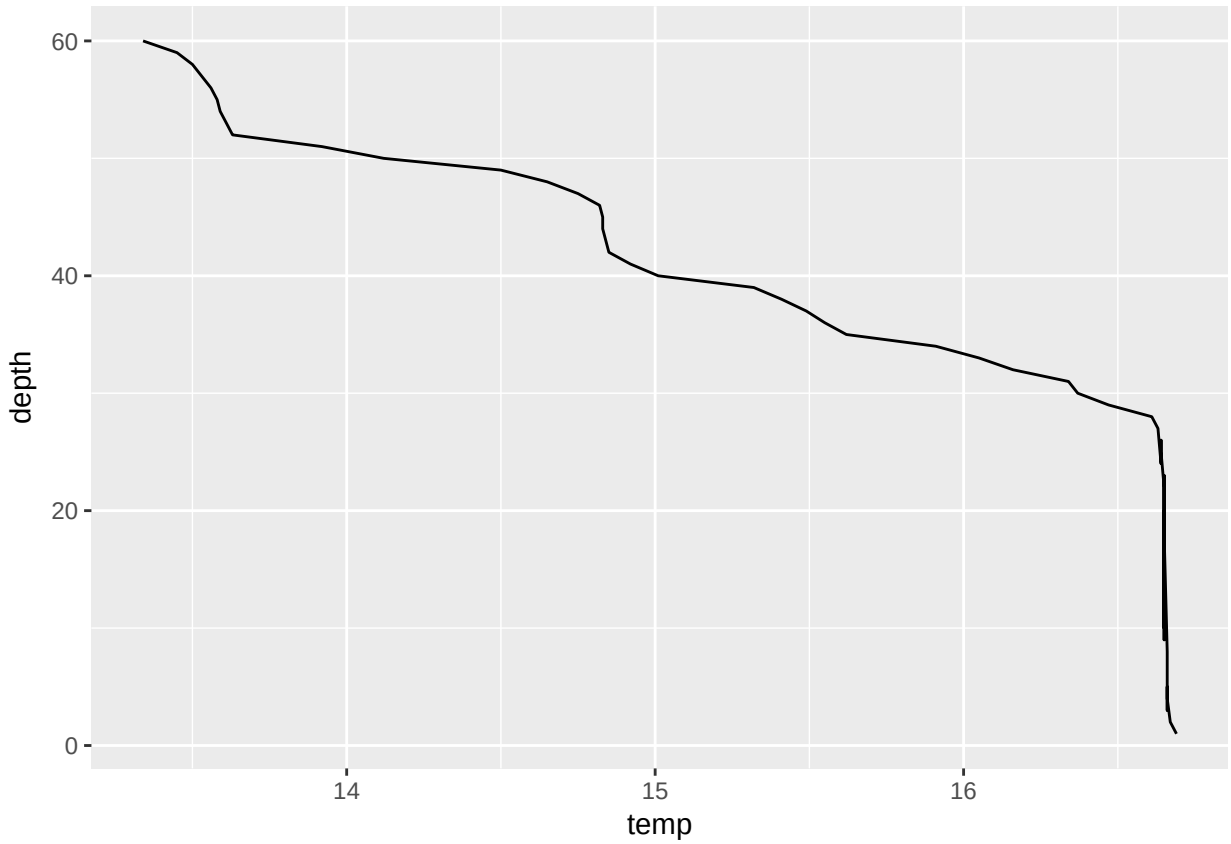
You can see that nothing happens because we haven't specified how to use that mapping. To do that, we have to specify a "geometry" which usually begins with `geom_`. Let's plot some simple points with `geom_point`:

```
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_point()
# we have to use `print` to see the result...
print(p)
```



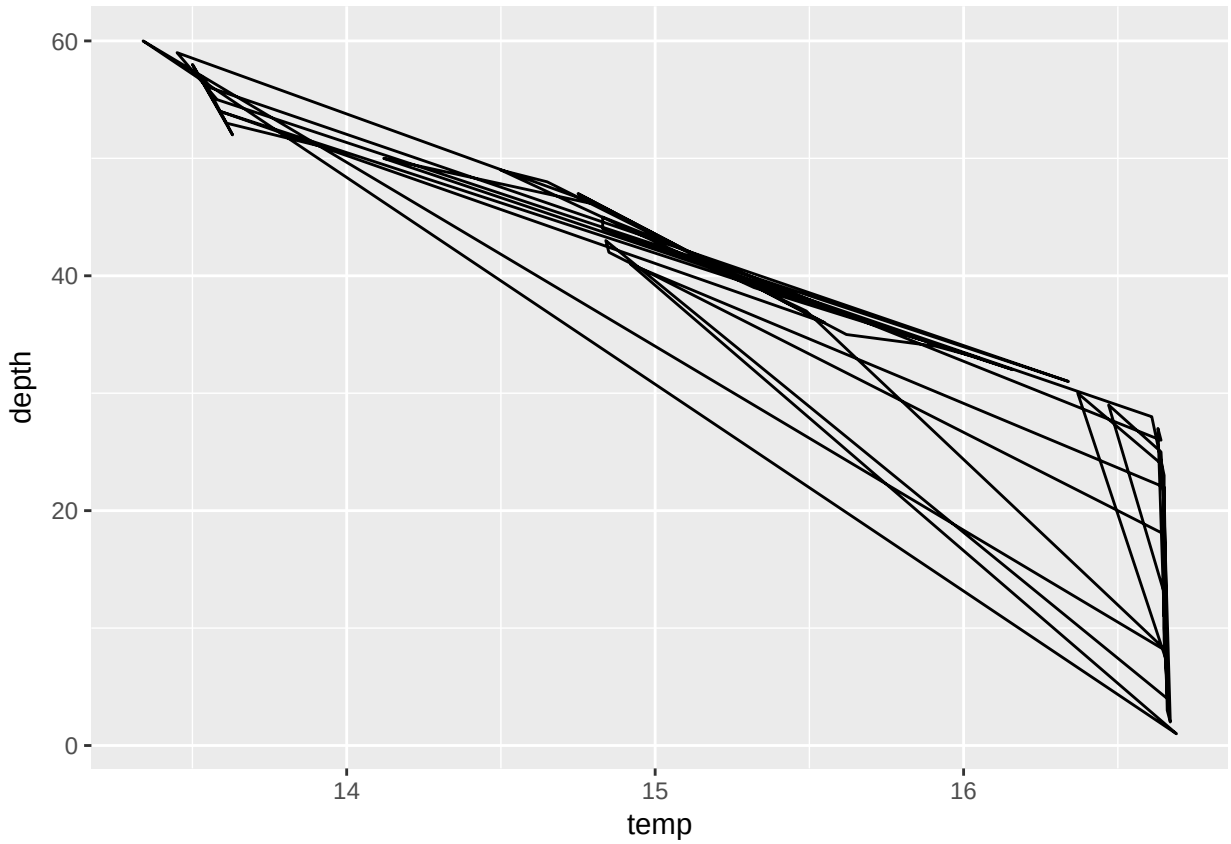
There are two geometries for lines, `geom_line` and `geom_path`. `geom_line` connects the points in the order of the x-axis:

```
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_line()
print(p)
```



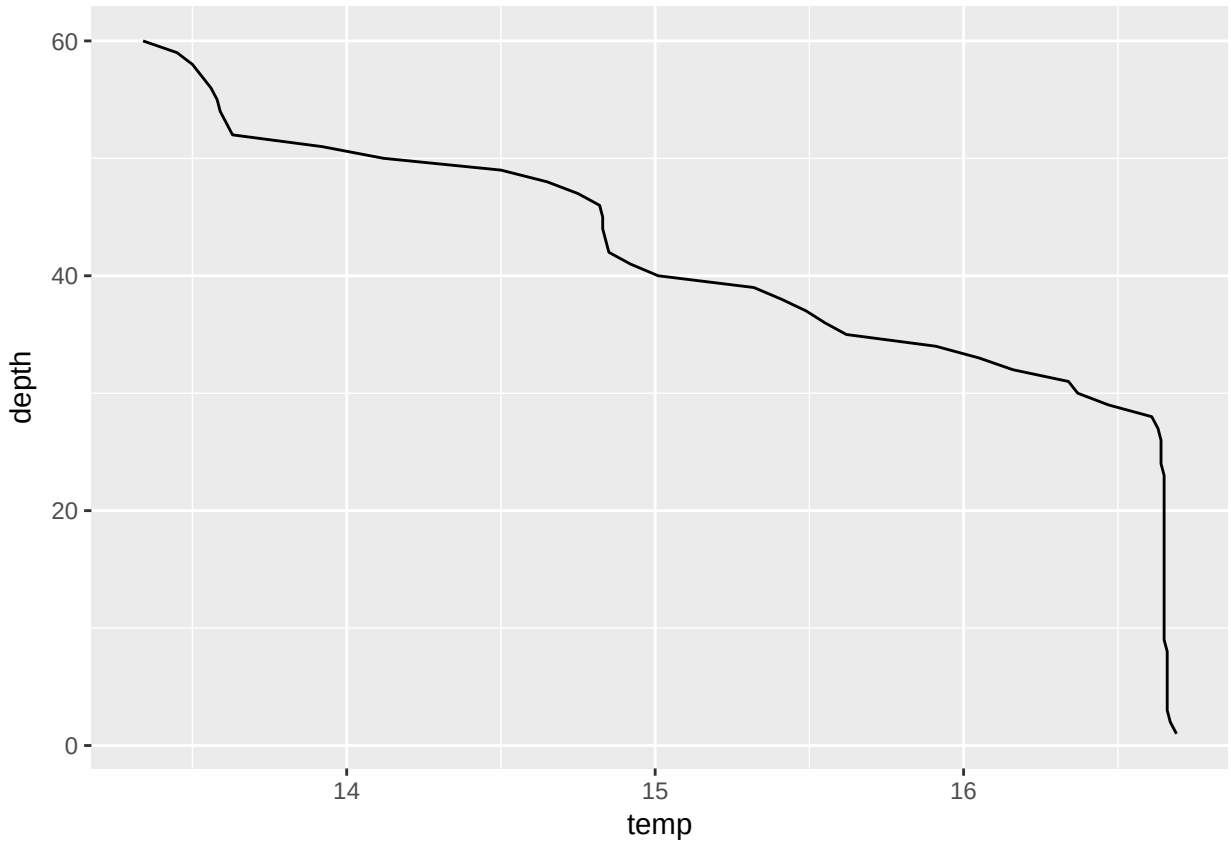
while `geom_path` connects them in order they are found:

```
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path()
print(p)
```



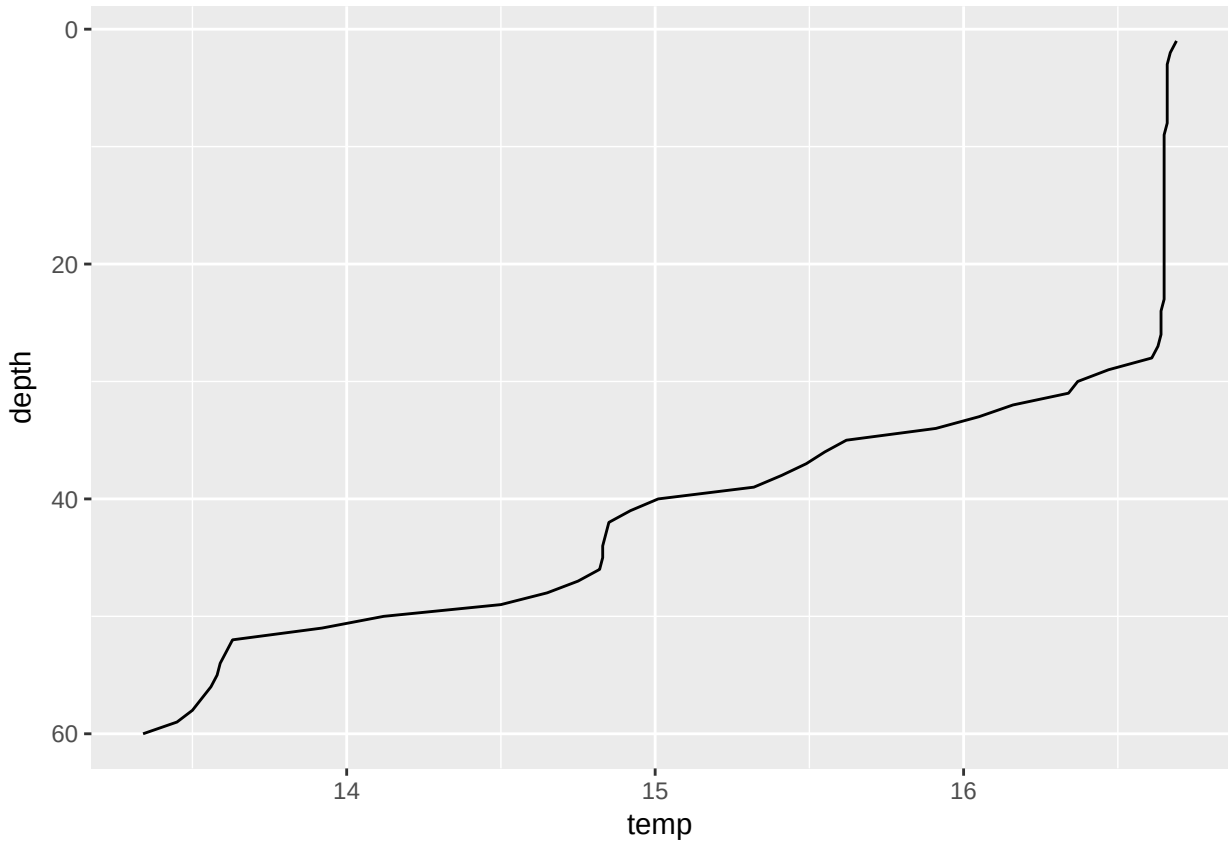
We need to first sort our data in order of depth, then connect them in order with `geom_path`:

```
df <- df[order(df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path()
print(p)
```



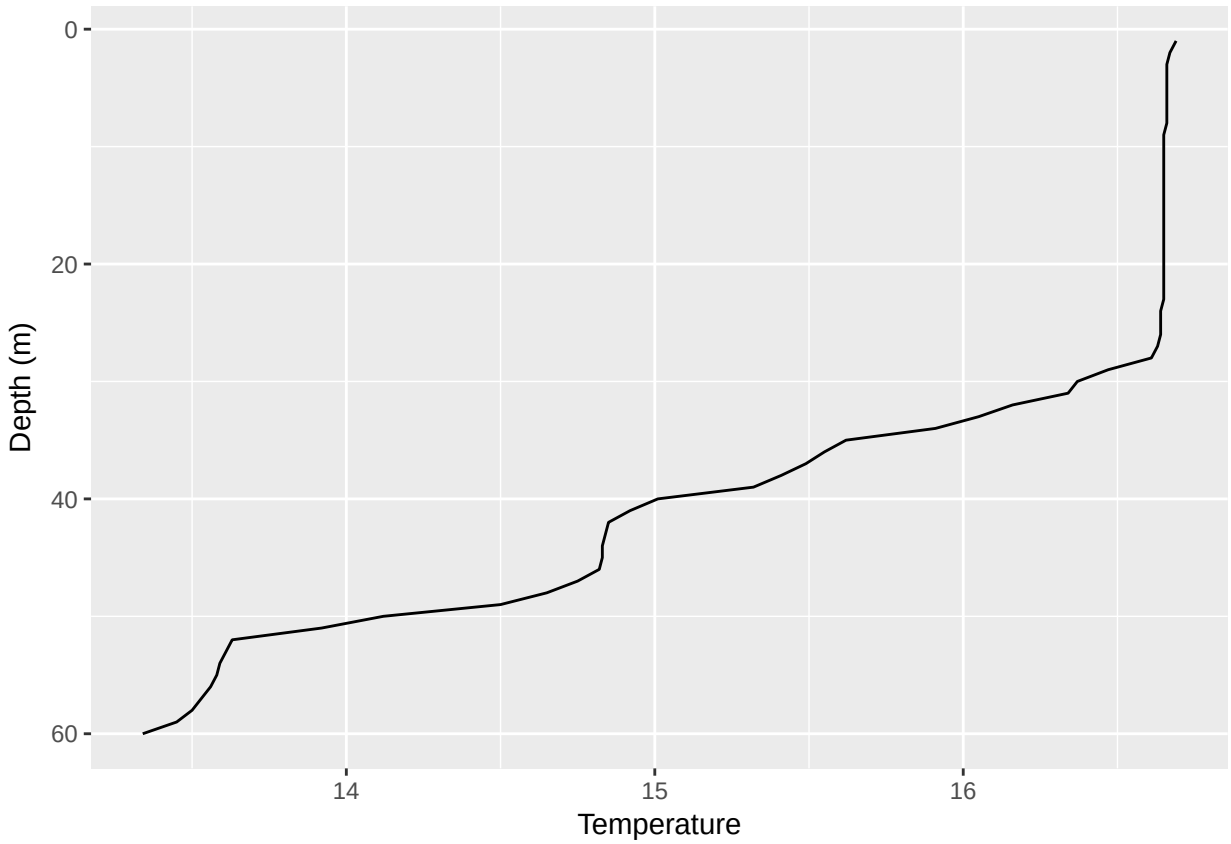
We still have the same problem as before, where we need to reverse the scale of our y-axis to get depth to go from small values at top to high values on the bottom. We can do this by adding 'scale_x_reverse':

```
df <- df[order(df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse()
print(p)
```



We can specify our axis labels individually with `xlab` and `ylab`, or together with `labs`:

```
df <- df[order(df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```

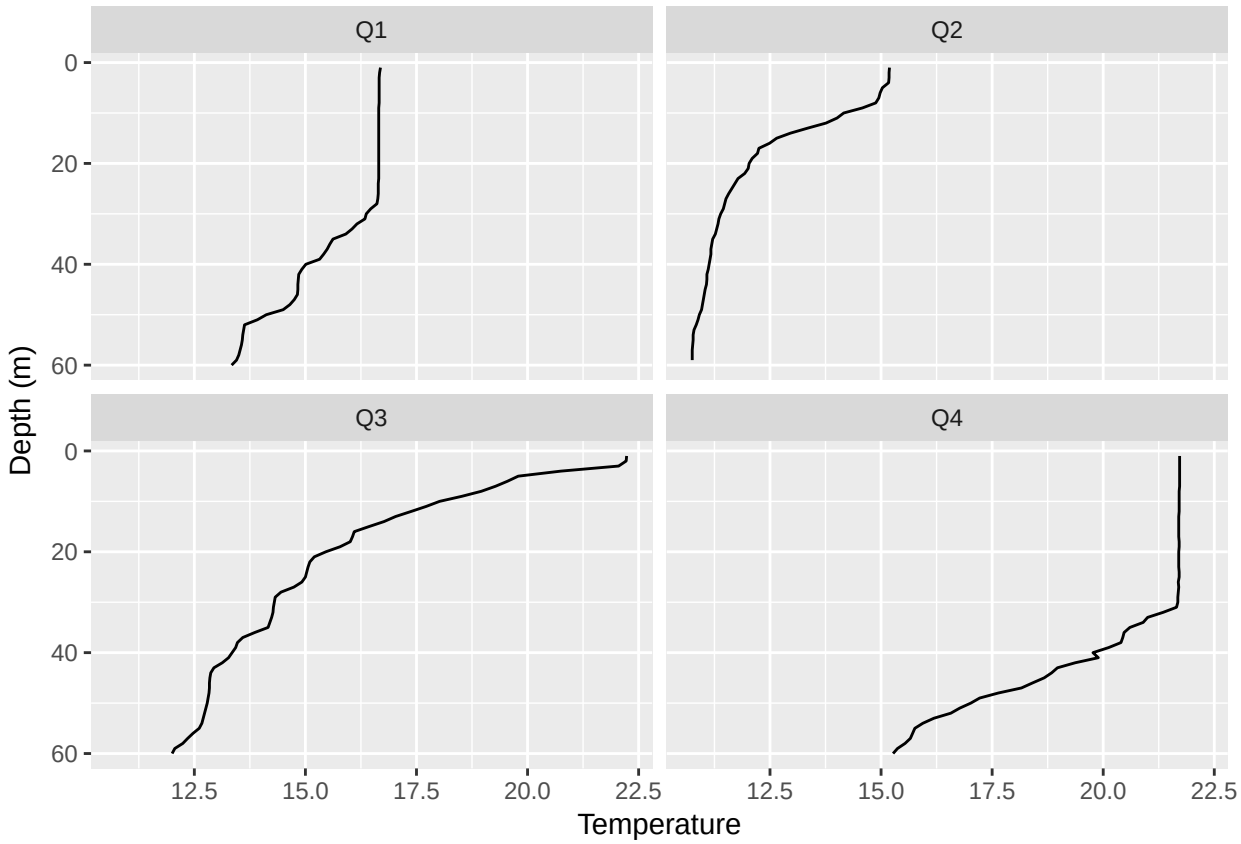


Facetting

Multiple panels in ggplot are created using "facets". There are two primary ways to do this: creating a facet for sequential levels of a factor, where the panels are placed in a specified number of rows and/or columns (`facet_wrap`), or two-dimensional facets where one factor is represented by rows and the other by the columns (`facet_grid`).

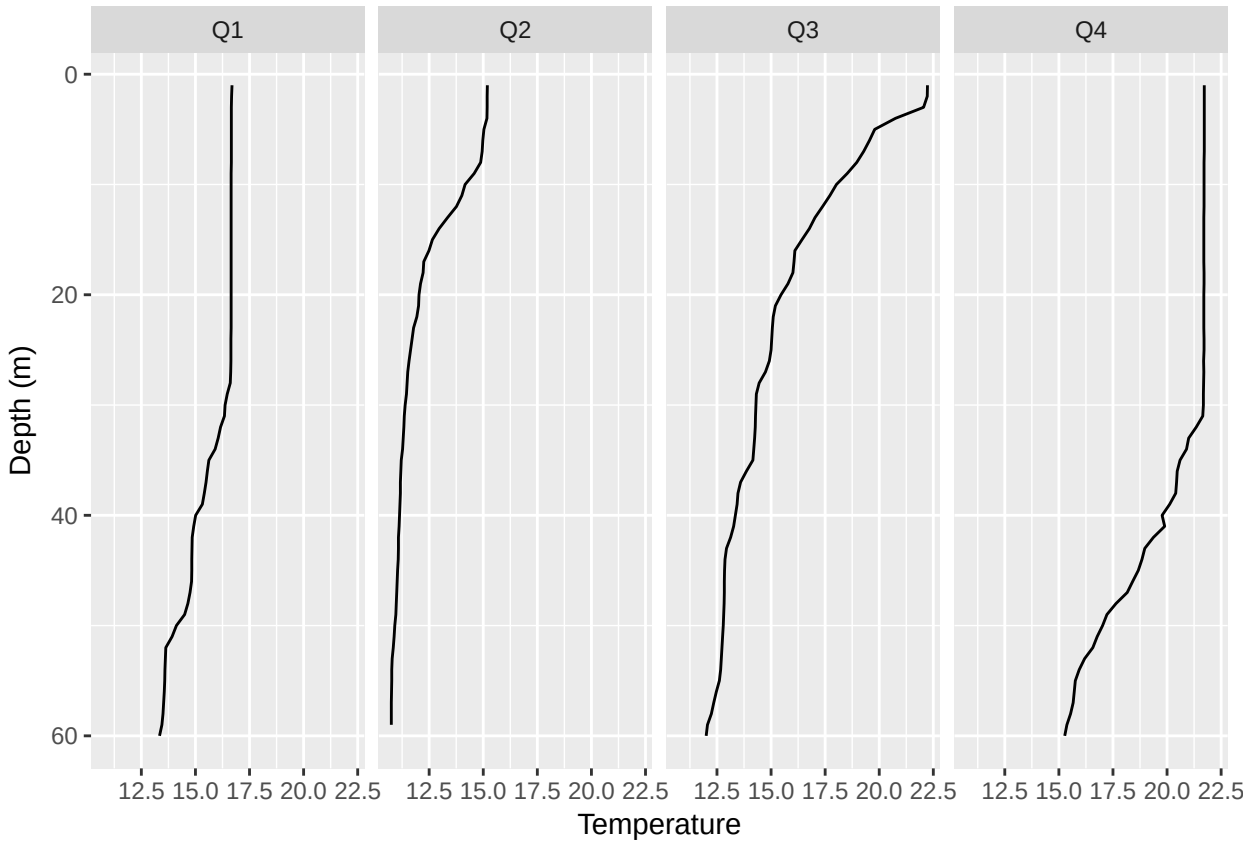
Here is an example of `facet_wrap` to plot the temperature profile for every quarter at Station.1 in 2015:

```
df <- subset(ctd, station == "Station.1" & year == 2015)
df <- df[order(df$quarter, df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  facet_wrap(~ quarter) +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```



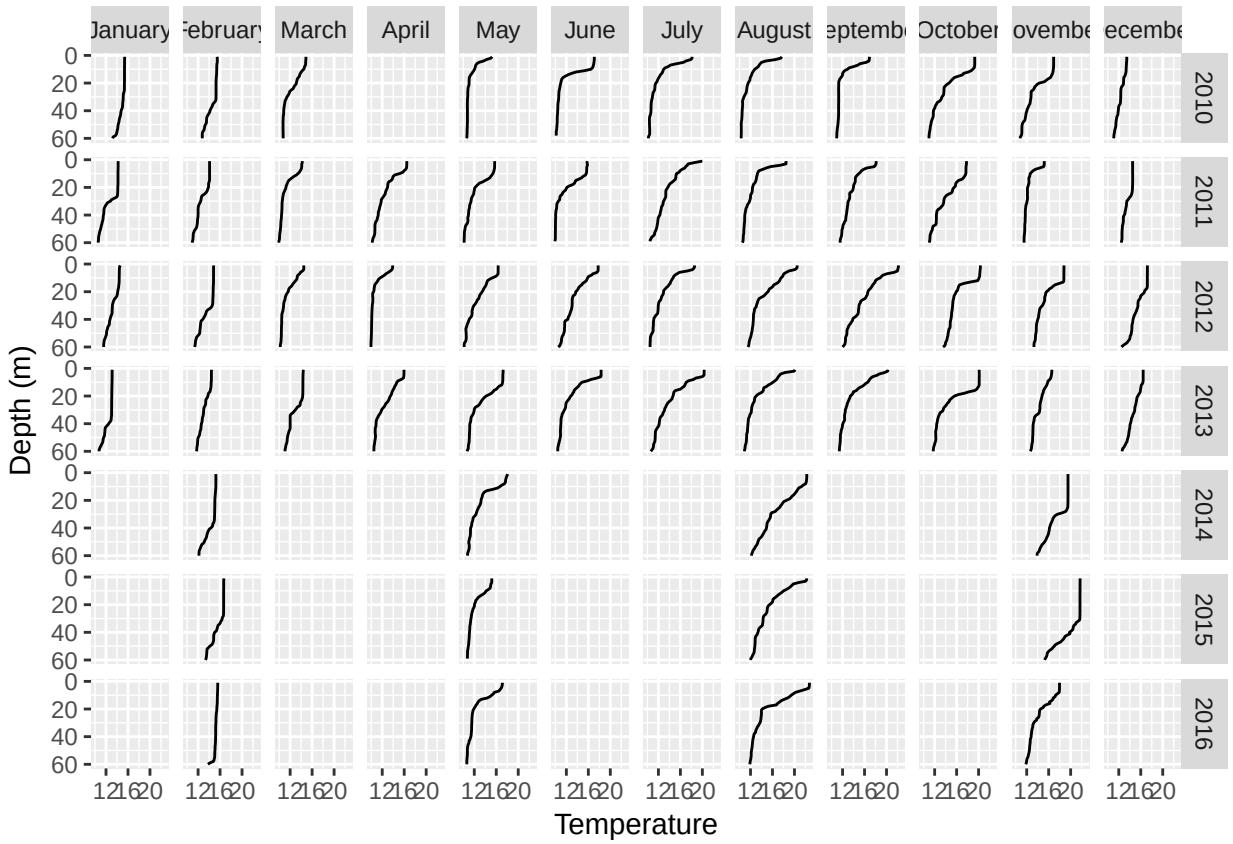
We can make this a single row or single column:

```
df <- subset(ctd, station == "Station.1" & year == 2015)
df <- df[order(df$quarter, df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  facet_wrap(~ quarter, nrow = 1) +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```



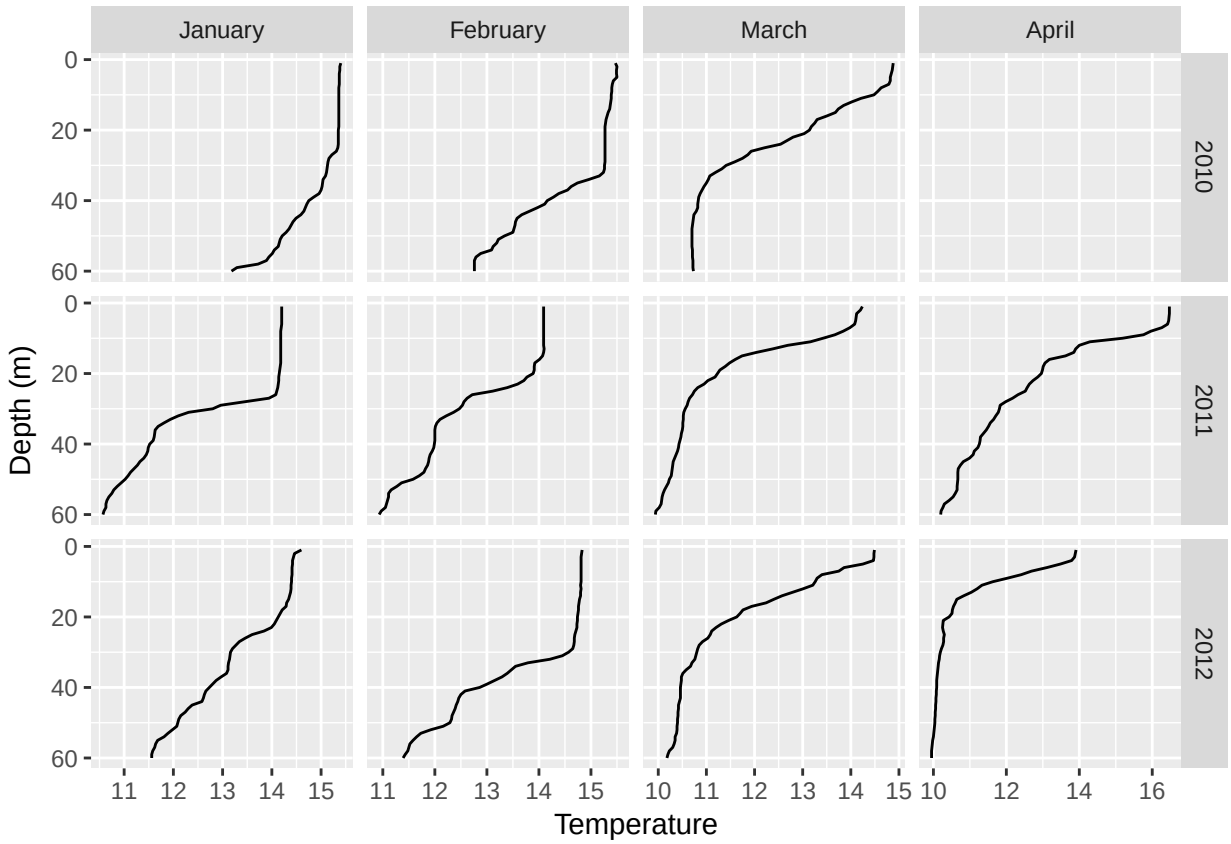
To demonstrate `facet_grid`, let's look at the temperature profiles for Station.39 across months(rows) and years (columns):

```
df <- subset(ctd, station == "Station.1")
df <- df[order(df$year, df$quarter, df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  facet_grid(year ~ month) +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```

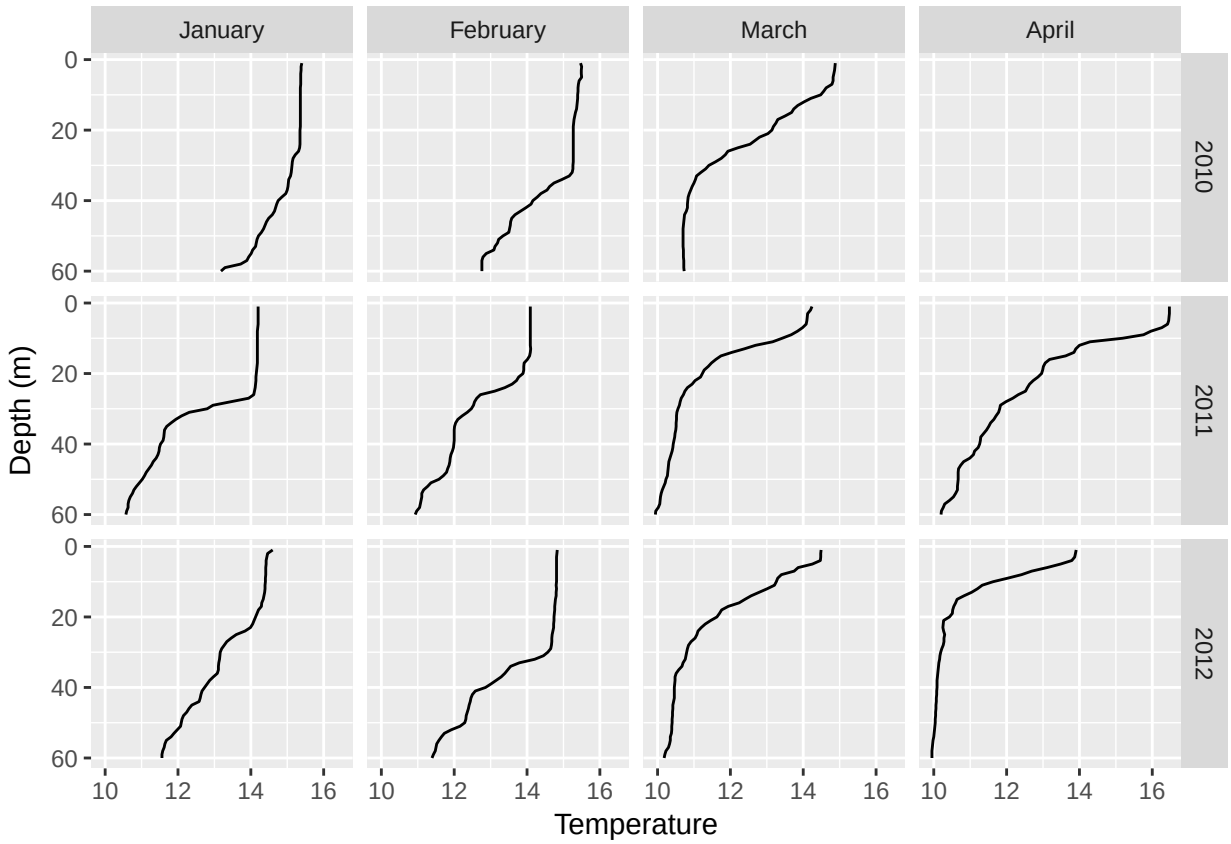


You can control how the axis scales are set in the facets. By default all facets share the same axis ranges. You can let axes have their own ranges for rows or columns by specifying the `scales` argument as either `free_x`, `free_y`, or `free`:

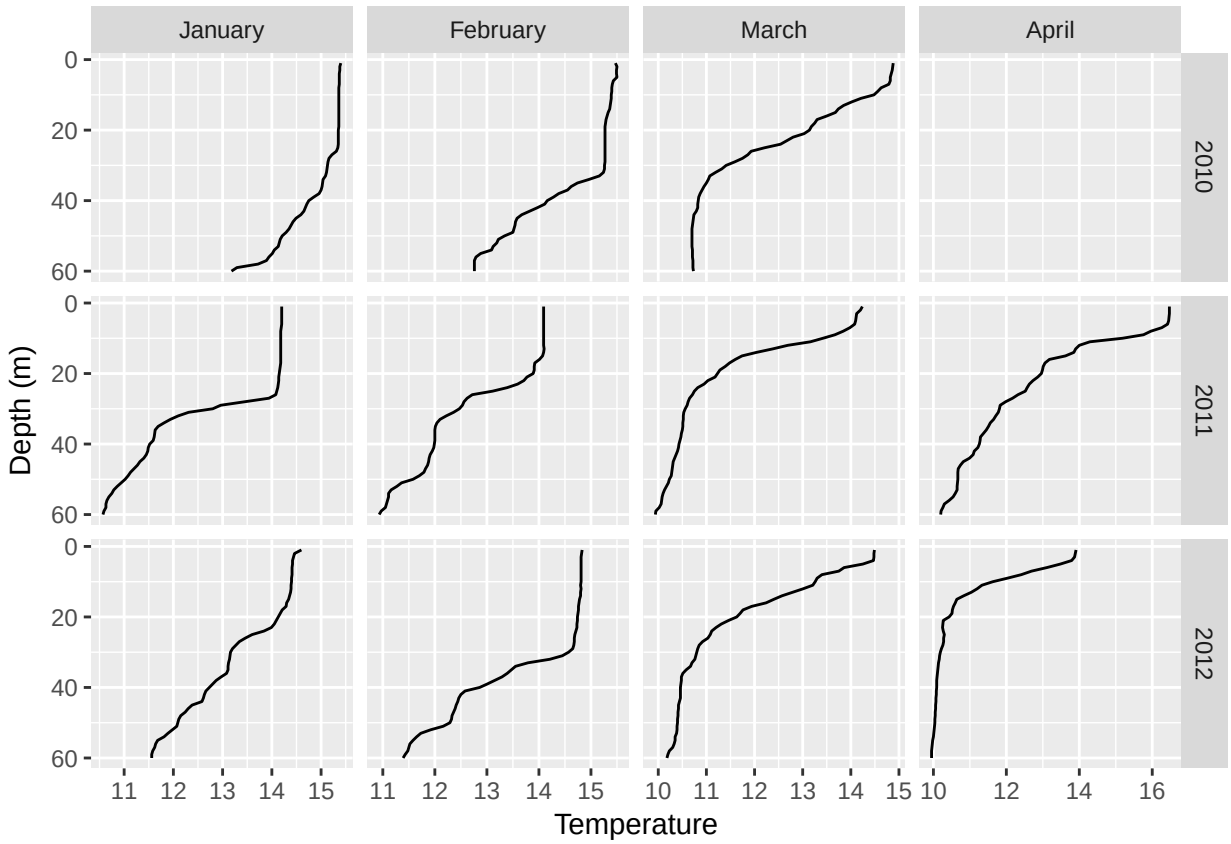
```
df <- subset(ctd, year %in% 2010:2012 & month %in% c("January", "February", "March", "April") & station
df <- df[order(df$year, df$month, df$depth), ]
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  facet_grid(year ~ month, scales = "free_x") +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```




```r
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  facet_grid(year ~ month, scales = "free_y") +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```



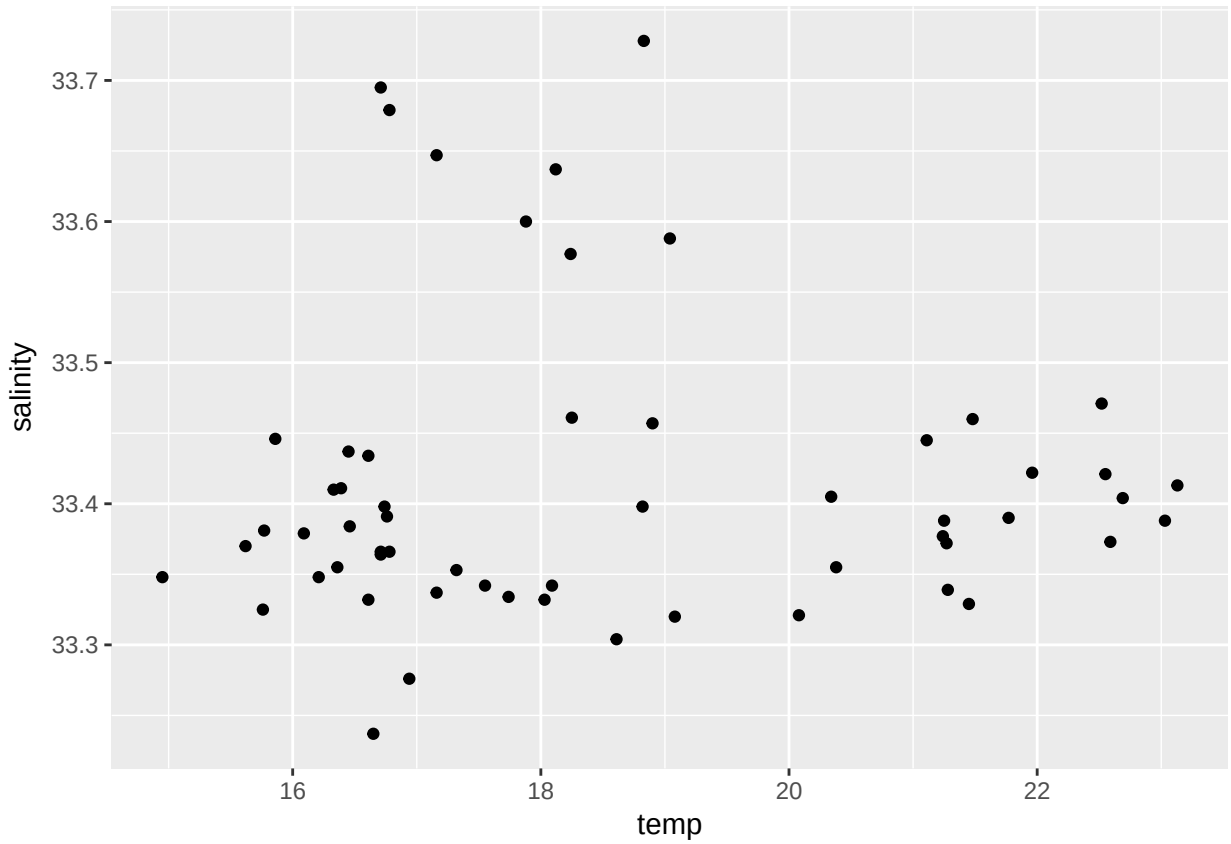

```
p <- ggplot(df, mapping = aes(x = temp, y = depth)) +
  geom_path() +
  scale_y_reverse() +
  facet_grid(year ~ month, scales = "free") +
  labs(x = "Temperature", y = "Depth (m)")
print(p)
```



Grouping

Within a single plot, groups can be specified with the `group`, `color`, or `fill` arguments. Which one is used depends on the type of plot being created. For example, let's look at the surface temperatures and salinities for Station.39 in 2015:

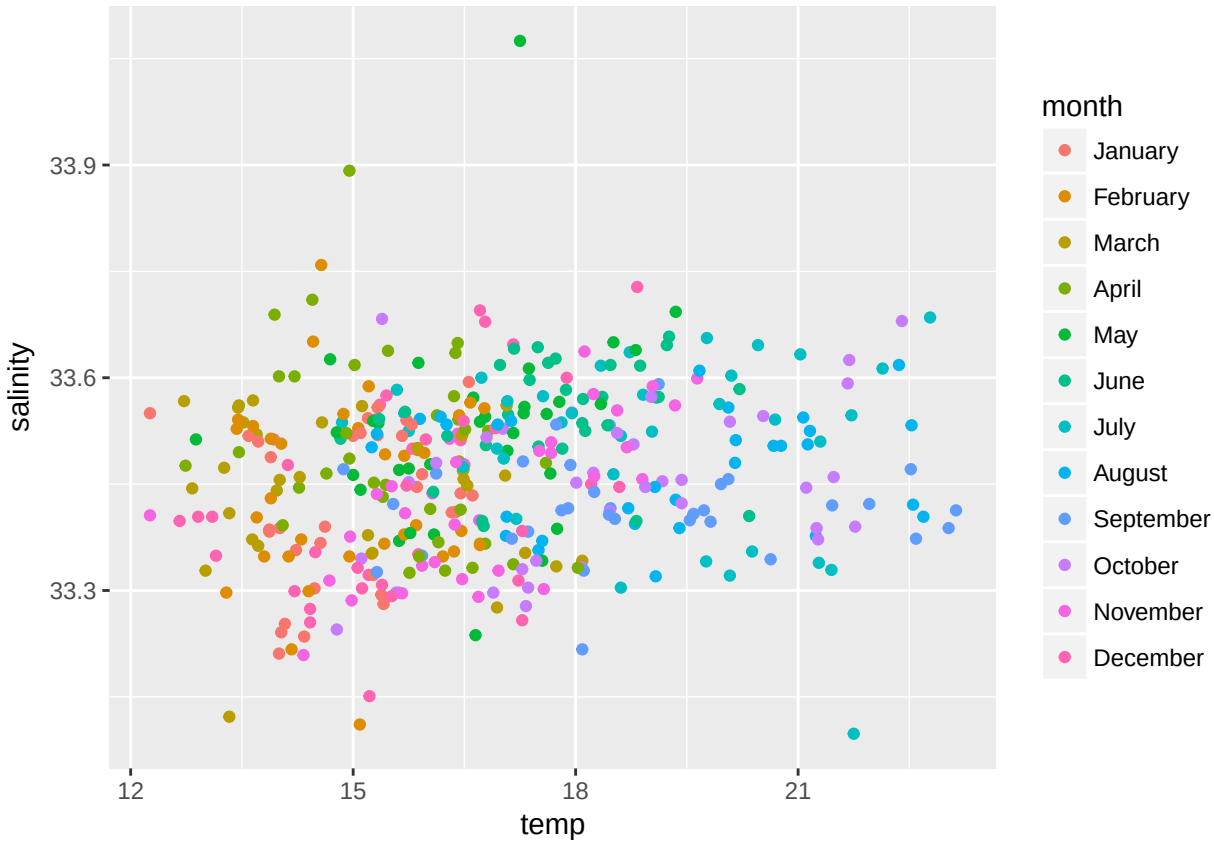
```
df <- subset(ctd, station == "Station.39" & depth == 1 & year == 2015)
p <- ggplot(df, aes(temp, salinity)) +
  geom_point()
print(p)
```



Let's color these by month:

```
df <- subset(ctd, station == "Station.39" & depth == 1)
p <- ggplot(df, aes(temp, salinity, color = month)) +
  geom_point()
print(p)
```

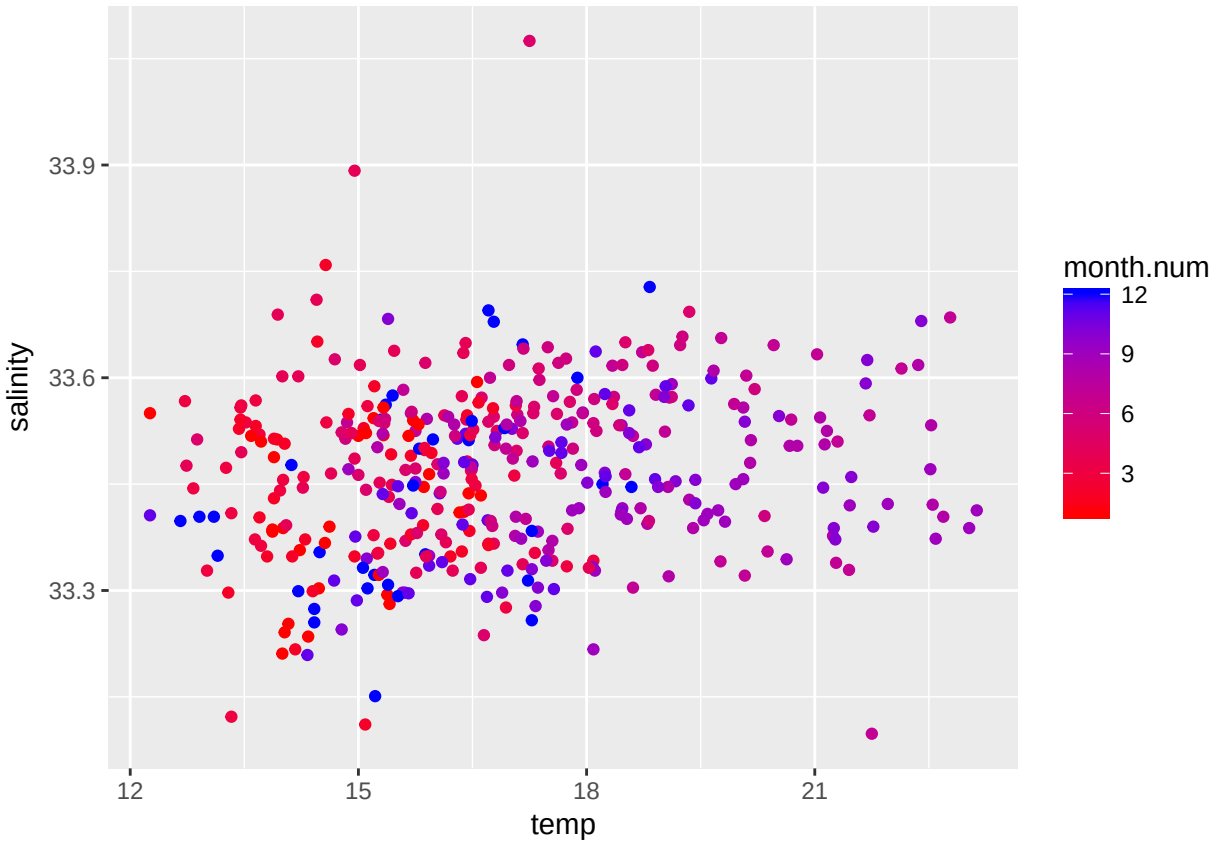
```
Warning: Removed 4 rows containing missing values (geom_point).
```



Because month is a number, a continuous scale is used. We can chane the nature of this scale using `scale_color_gradient`:

```
df$month.num <- as.numeric(df$month)
p <- ggplot(df, aes(temp, salinity, color = month.num)) +
  geom_point() +
  scale_color_gradient(low = "red", high = "blue")
print(p)
```

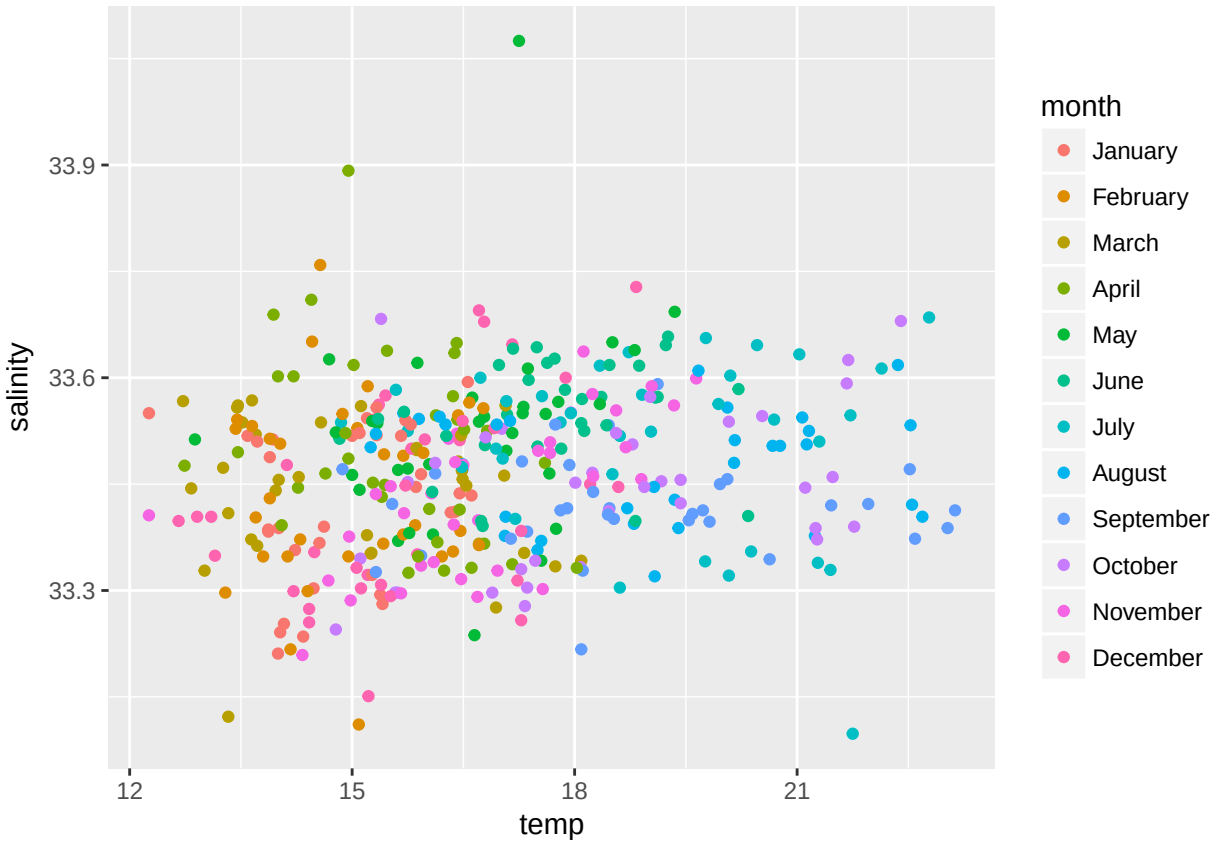
```
Warning: Removed 4 rows containing missing values (geom_point).
```



If month is a factor, then the default plot looks like this, where each month gets its own color:

```
p <- ggplot(df, aes(temp, salinity, color = month)) +
  geom_point()
print(p)
```

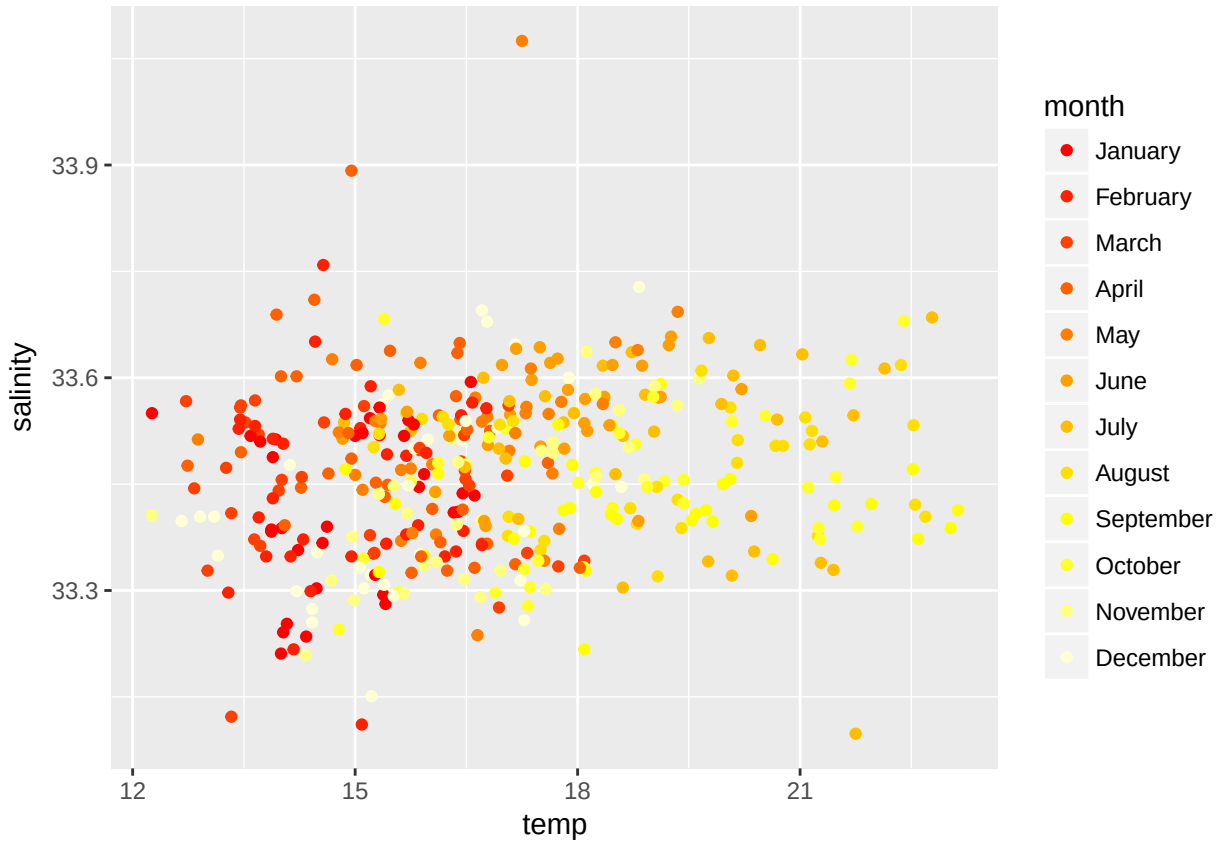
```
Warning: Removed 4 rows containing missing values (geom_point).
```



To change the colors of a factor, I recommend using the RColorBrewer palettes (see http://colorbrewer2.org) and `scale_color_brewer`. However, for 12 months, let's use the built-in `heat.colors` palette and specify it with `scale_color_manual`:

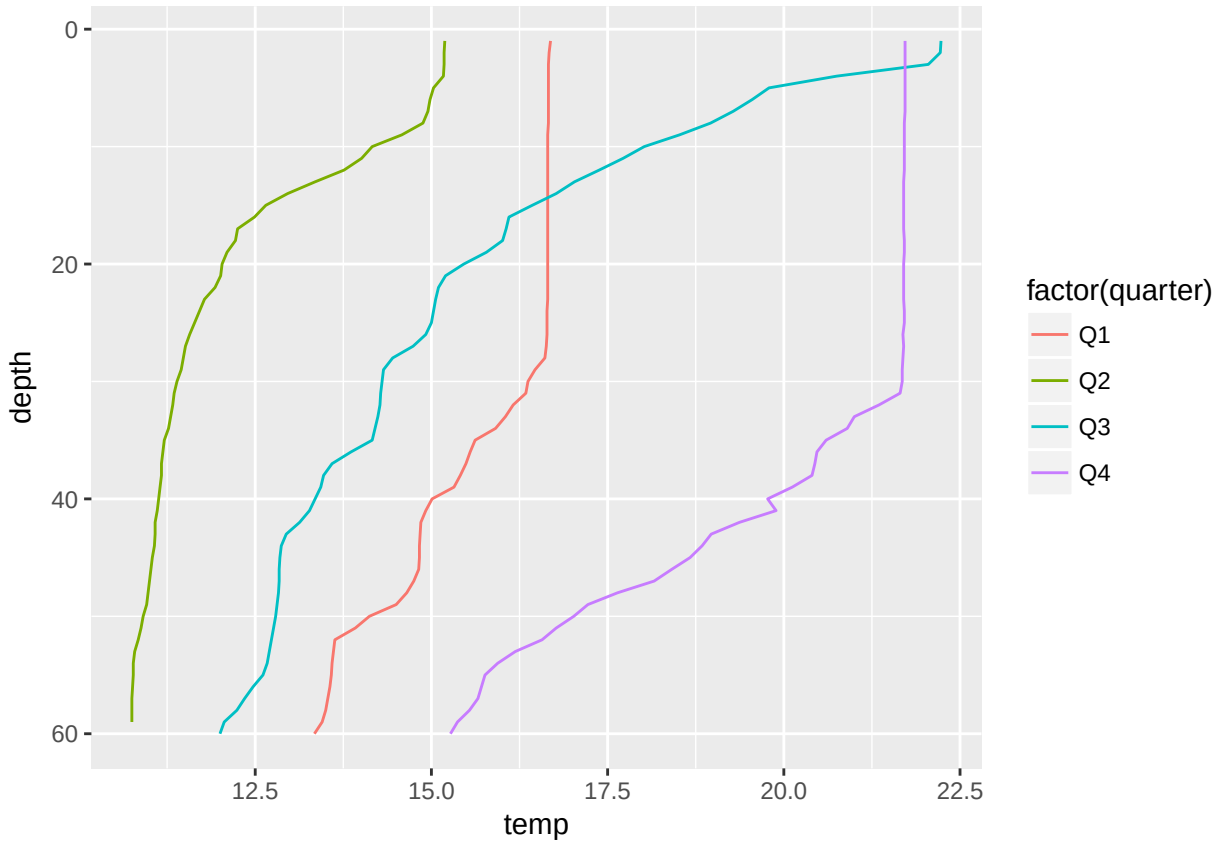
```
p <- ggplot(df, aes(temp, salinity, color = month)) +
  geom_point() +
  scale_color_manual(values = heat.colors(12))
print(p)
```

```
Warning: Removed 4 rows containing missing values (geom_point).
```



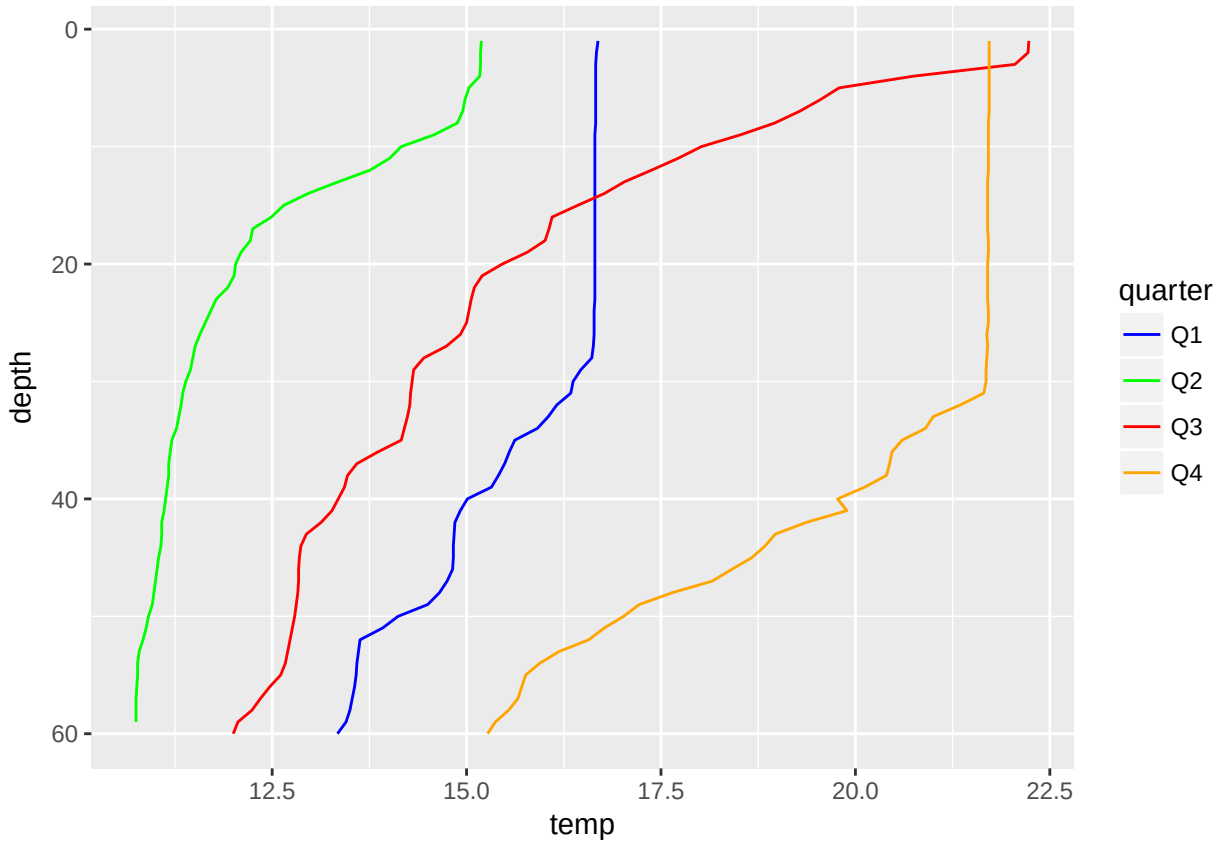
Colors can also be specified by category. For instance, let's plot temperature profiles for Station.39 in 2015 for each quarter:

```
df <- subset(ctd, station == "Station.1" & year == 2015)
df <- df[order(df$month, df$depth), ]
p <- ggplot(df, aes(temp, depth, color = factor(quarter))) +
  geom_path() +
  scale_y_reverse()
print(p)
```



Let's make each quarter a factor and manually set the colors:

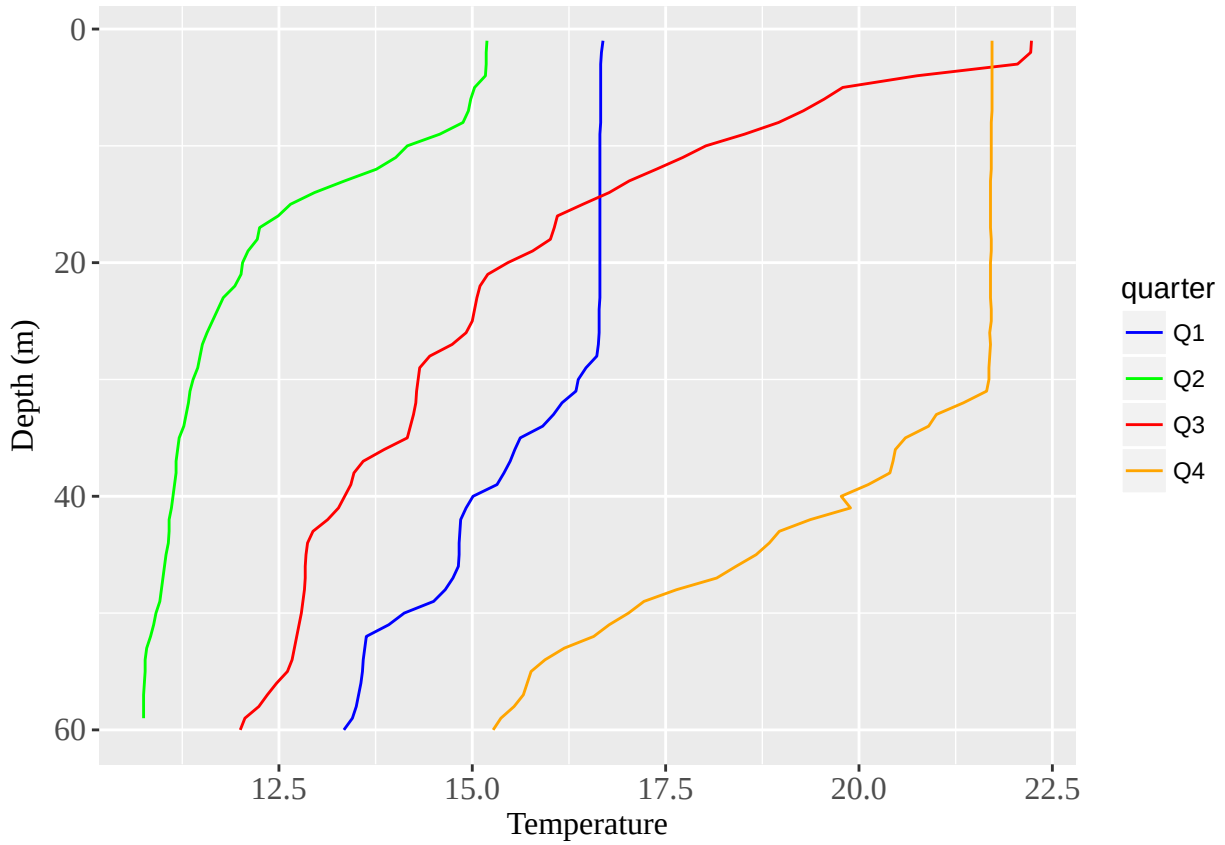
```
q.colors <- c(Q3 = "red", Q4 = "orange", Q1 = "blue", Q2 = "green")
p <- ggplot(df, aes(temp, depth, color = quarter)) +
  geom_path() +
  scale_y_reverse() +
  scale_color_manual(values = q.colors)
print(p)
```



Themes

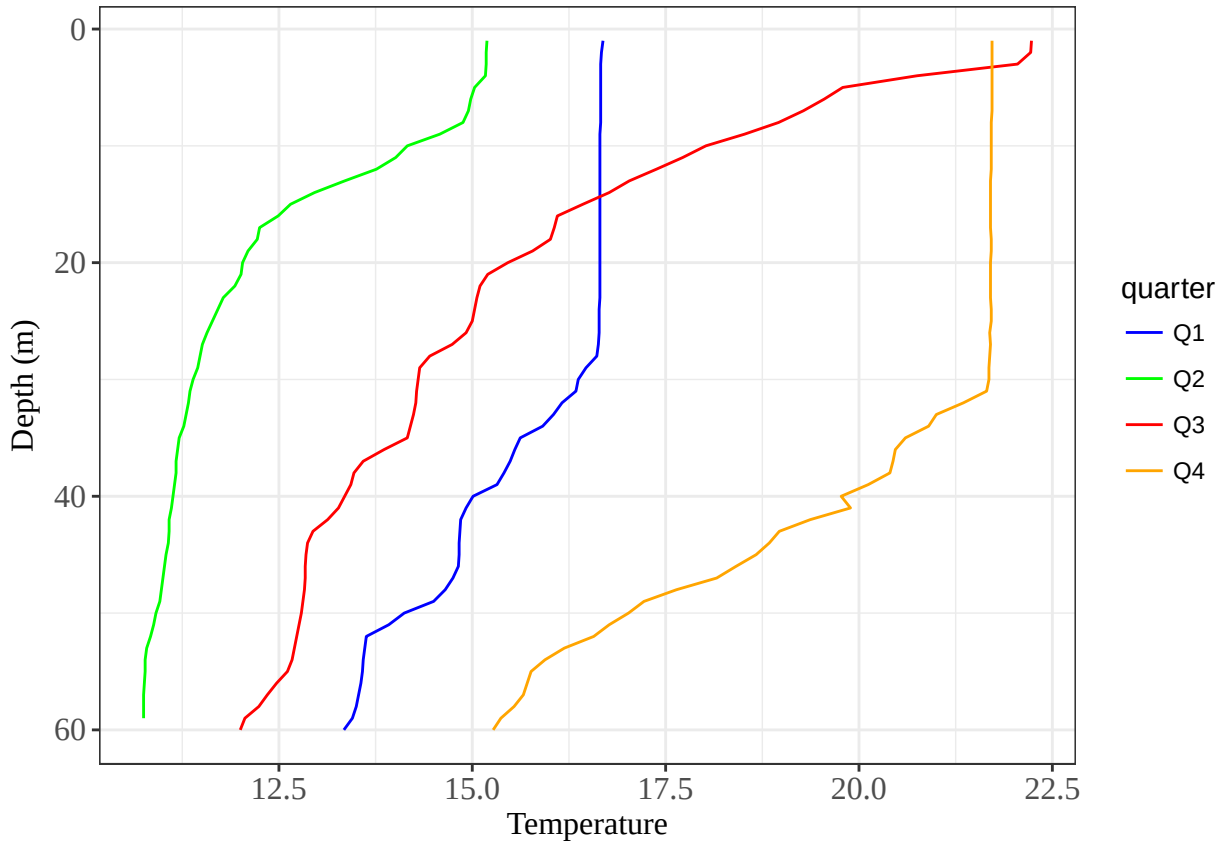
Features of the plot like font sizes, legend position, etc. can be specified with `theme`. For example, we'll set the tick labels and axis labels to Times New Roman 12 point:

```
p <- ggplot(df, aes(temp, depth, color = quarter)) +
  geom_path() +
  scale_y_reverse() +
  labs(x = "Temperature", y = "Depth (m)") +
  scale_color_manual(values = q.colors) +
  theme(
    axis.title = element_text(family = "Times", size = 12),
    axis.text = element_text(family = "Times", size = 12)
  )
print(p)
```

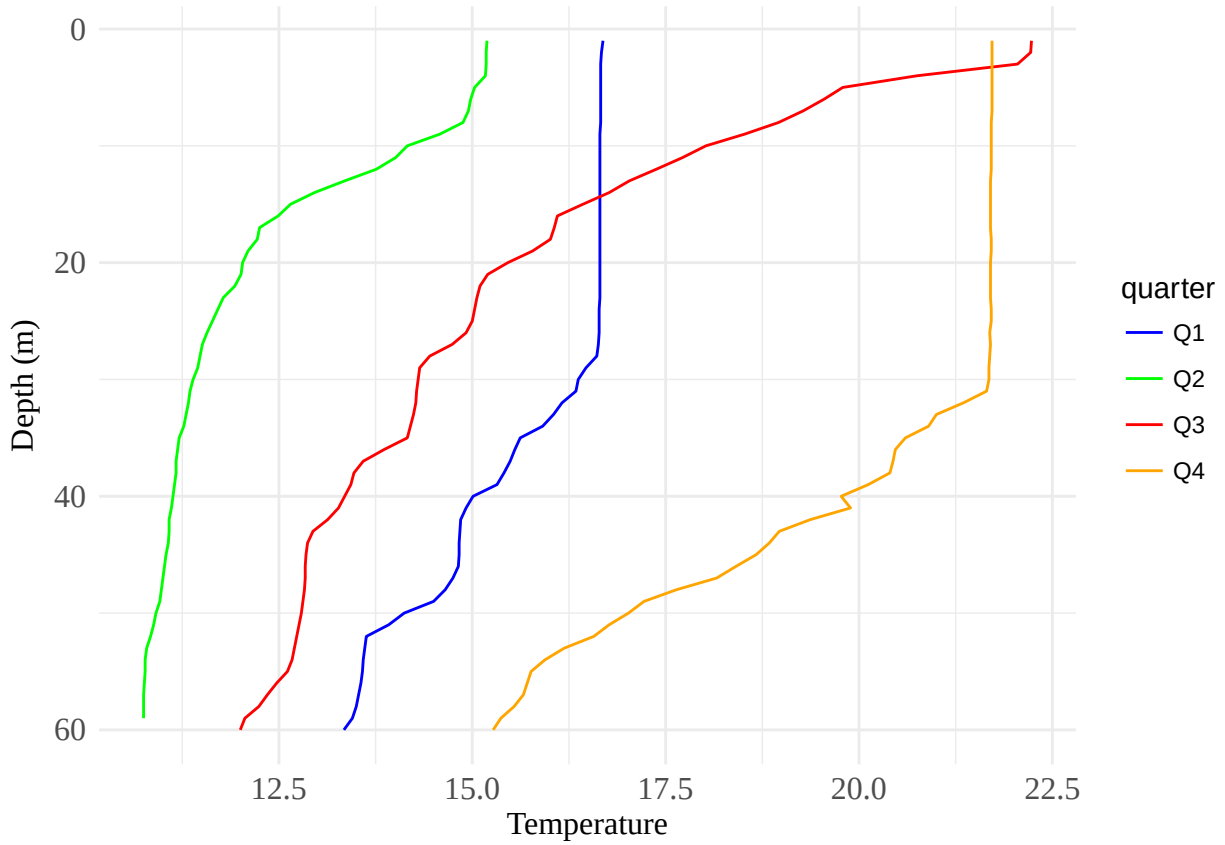


If you don't like the default background shadng and lines and you don't want to hand specify all components, you can try a few of the alternate themes:

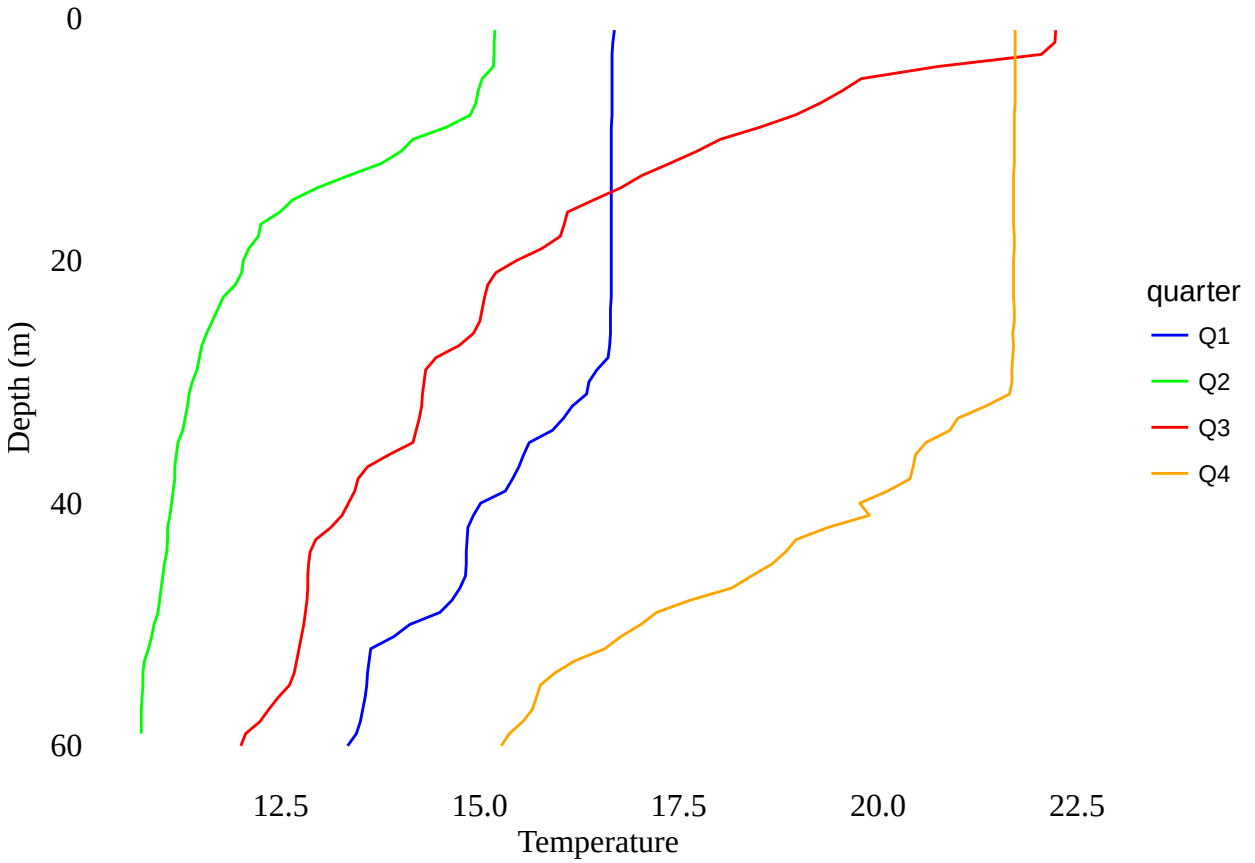
```
p <- ggplot(df, aes(temp, depth, color = quarter)) +
  geom_path() +
  scale_y_reverse() +
  scale_color_manual(values = q.colors) +
  labs(x = "Temperature", y = "Depth (m)") +
  theme_bw() +
  theme(
    axis.title = element_text(family = "Times", size = 12),
    axis.text = element_text(family = "Times", size = 12)
  )
print(p)
```




```
p <- ggplot(df, aes(temp, depth, color = quarter)) +
  geom_path() +
  scale_y_reverse() +
  scale_color_manual(values = q.colors) +
  labs(x = "Temperature", y = "Depth (m)") +
  theme_minimal() +
  theme(
    axis.title = element_text(family = "Times", size = 12),
    axis.text = element_text(family = "Times", size = 12)
  )
print(p)
```




```
p <- ggplot(df, aes(temp, depth, color = quarter)) +
  geom_path() +
  scale_y_reverse() +
  scale_color_manual(values = q.colors) +
  labs(x = "Temperature", y = "Depth (m)") +
  theme_void() +
  theme(
    axis.title = element_text(family = "Times", size = 12),
    axis.text = element_text(family = "Times", size = 12)
  )
print(p)
```


0
20
40
60
Depth (m)
12.5
15.0
17.5
20.0
22.5
Temperature
quarter
Q1
Q2
Q3
Q4