Beautiful plotting in R: A ggplot2 cheatsheet

Samuel Chan

14/02/2017

This is a reproduction of Beautiful plotting in R: A ggplot2 cheatsheet (http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3) by Zev Ross. The motivation behind this exercise is to take advantage of the R Notebook format, allowing us to visualize the plotting results after each code chunk. Credits to the original author Zev Ross, first published on August 4, 2014.

Quick Setup: The dataset

We're using data from the National Morbidity and Mortality Air Pollution Study (NMMAPS). To make the plots manageable we're limiting the data to Chicago and 1997-2000. For more detail on this dataset, consult Roger Peng's book Statistical Methods in Environmental Epidemiology with R (http://www.springer.com/statistics/life+sciences,+medicine+%26+health/book/978-0-387-78166-2).

```
# Load the required library
library(ggplot2)

# Load our dataset
nmmaps <- read.csv("chicago-nmmaps.csv", as.is = T)

# Take data only after 1/1/1997
nmmaps$date <- as.Date(nmmaps$date)
nmmaps <- nmmaps[nmmaps$date > as.Date("1996-12-31"),]

# Substring the first four elements [1:4] of the date char so ("1997-01-01") returns "1997"
nmmaps$year <- substring(nmmaps$date, 1,4)
head(nmmaps)
```

```
##      city       date death temp dewpoint      pm10        o3 time season
## 3654 chic 1997-01-01   137 36.0    37.50 13.052268  5.659256 3654 winter
## 3655 chic 1997-01-02   123 45.0    47.25 41.948600  5.525417 3655 winter
## 3656 chic 1997-01-03   127 40.0    38.00 27.041751  6.288548 3656 winter
## 3657 chic 1997-01-04   146 51.5    45.50 25.072573  7.537758 3657 winter
## 3658 chic 1997-01-05   102 27.0    11.25 15.343121 20.760798 3658 winter
## 3659 chic 1997-01-06   127 17.0     5.75  9.364655 14.940874 3659 winter
##      year
## 3654 1997
## 3655 1997
## 3656 1997
## 3657 1997
## 3658 1997
## 3659 1997
```

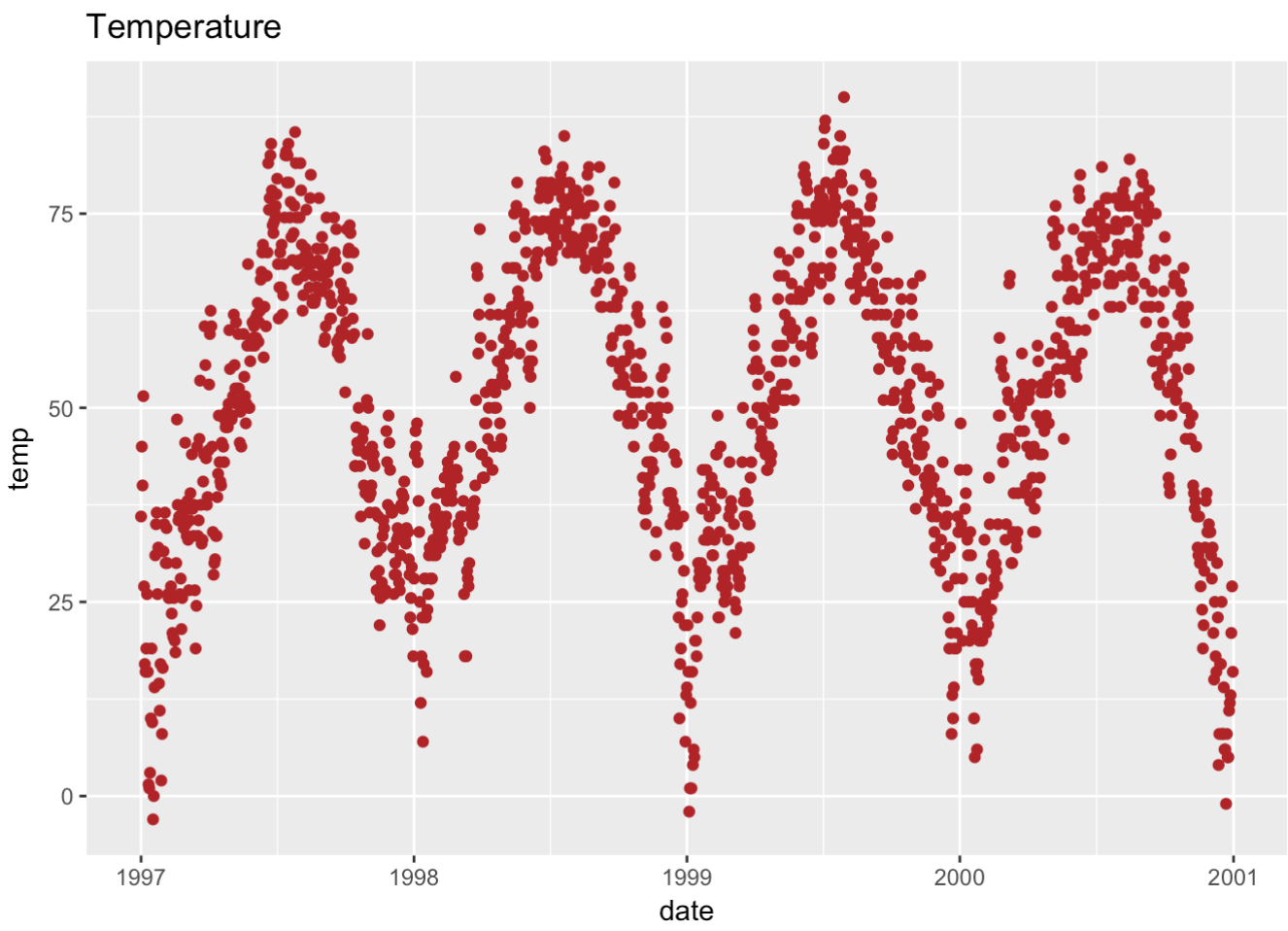
A default plot in ggplot2

```
g <- ggplot(nmmaps, aes(date, temp))+geom_point(color="firebrick")
g
```

Working with the title

Add a title: ggtitle() or labs()

```
g <- g + ggtitle('Temperature')
g
```



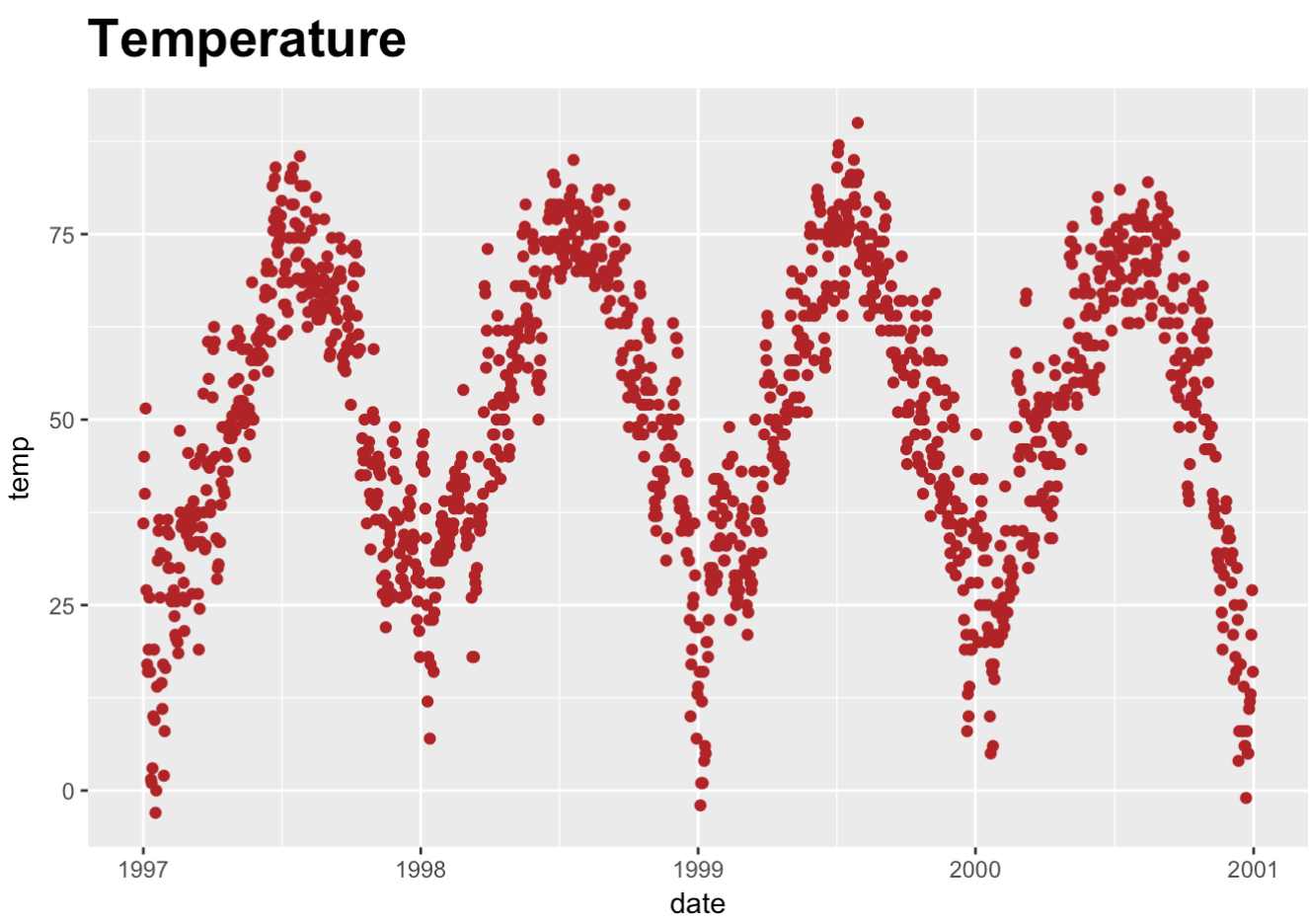
Alternative, we can use labs(): g +

labs(title = 'Title')

Make title bold and add a little space at the baseline: face, margin

Note that the **margin** argument uses the margin function and we provide the top, right, bottom, left margins in that order. The default unit is points.

```
g <- g + theme(plot.title = element_text(size = 20, face = "bold", margin = margin(10,0,10,0)))
g
```

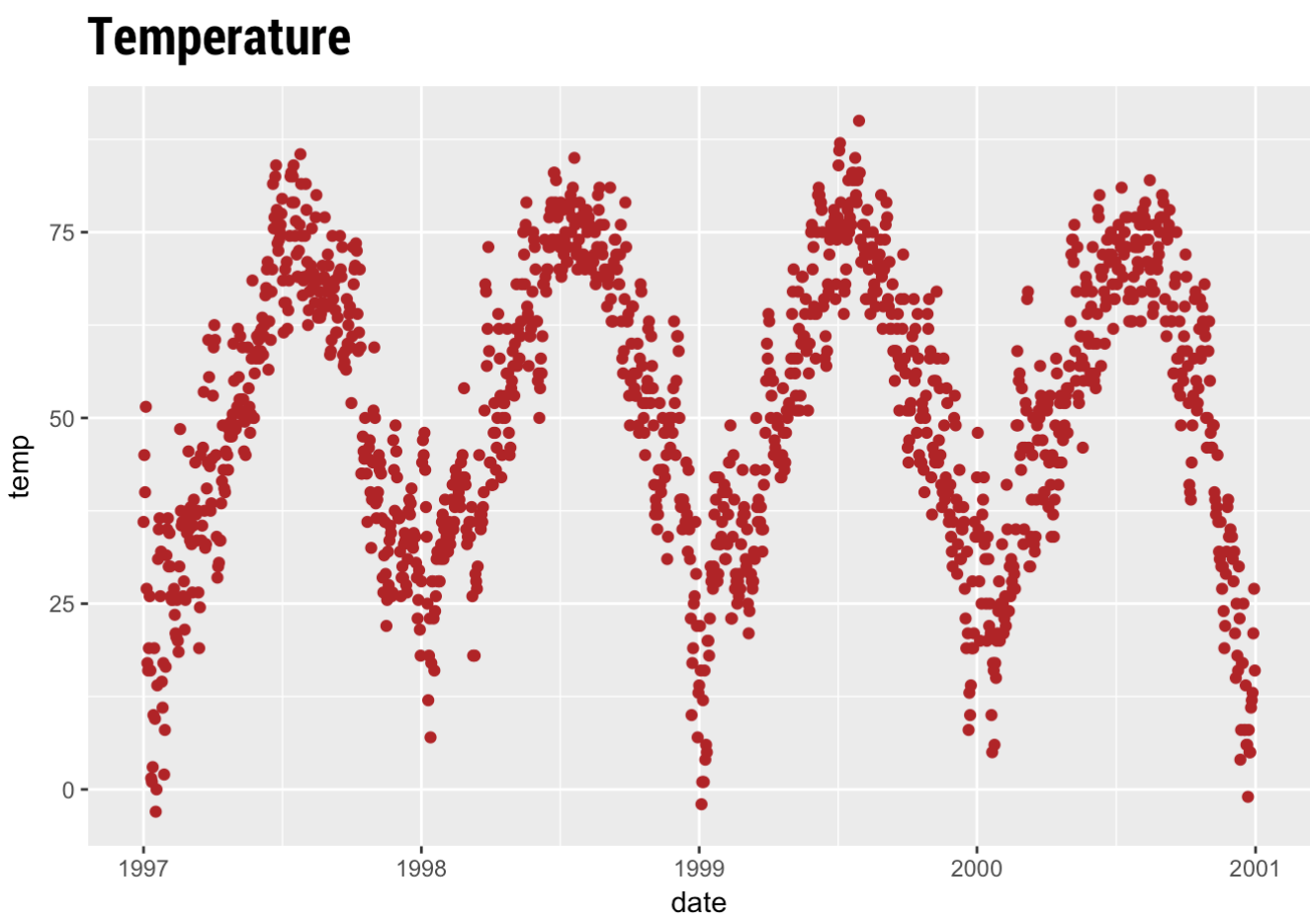


Using a non-traditional font in your title: family

```
library(extrafont)
```

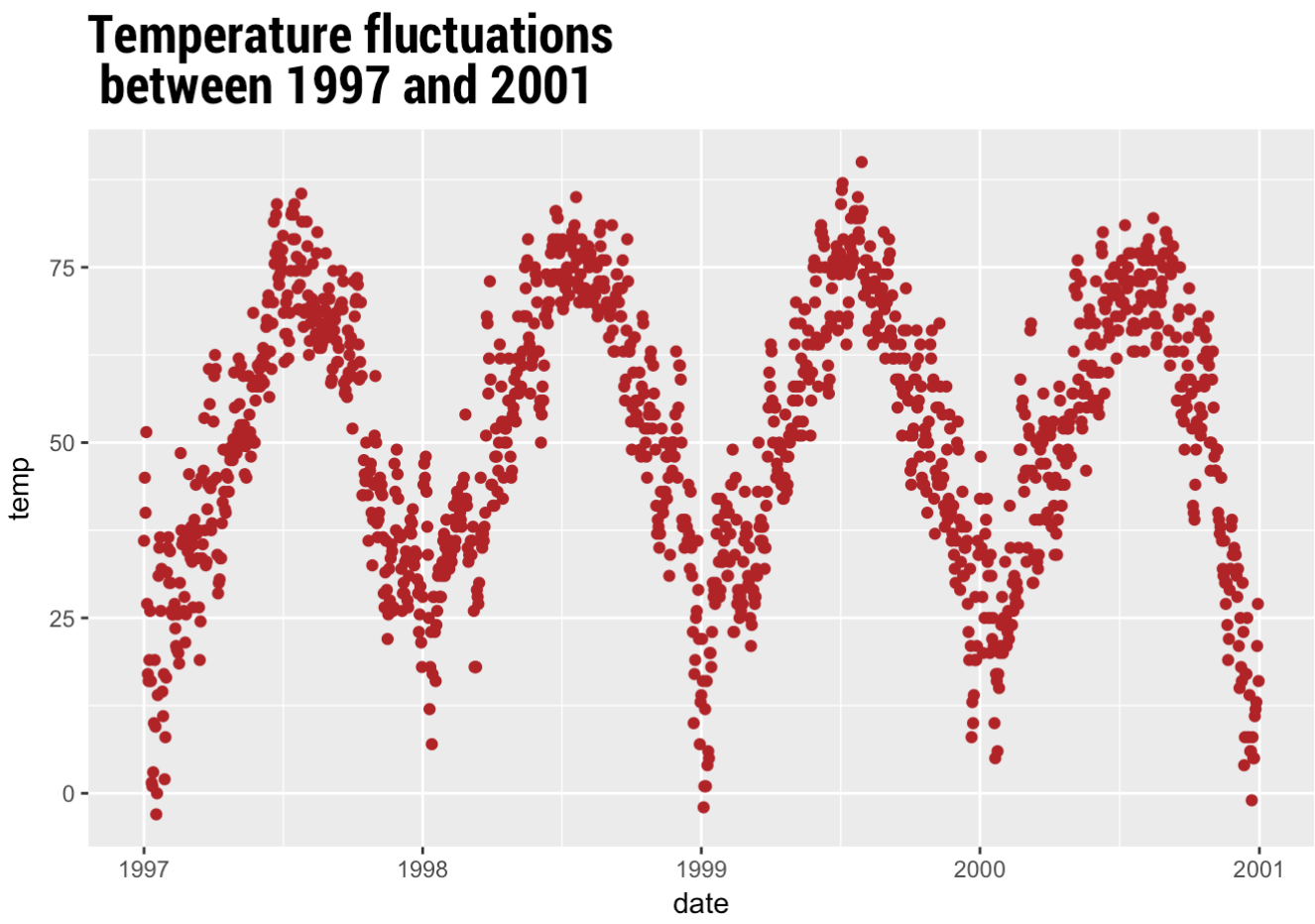
```
## Registering fonts with R
```

```
g <- g + theme(plot.title = element_text(size = 20, lineheight = .8, vjust=1, family = "Roboto Condensed"))
g
```



Change spacing in multi-line text: lineheight

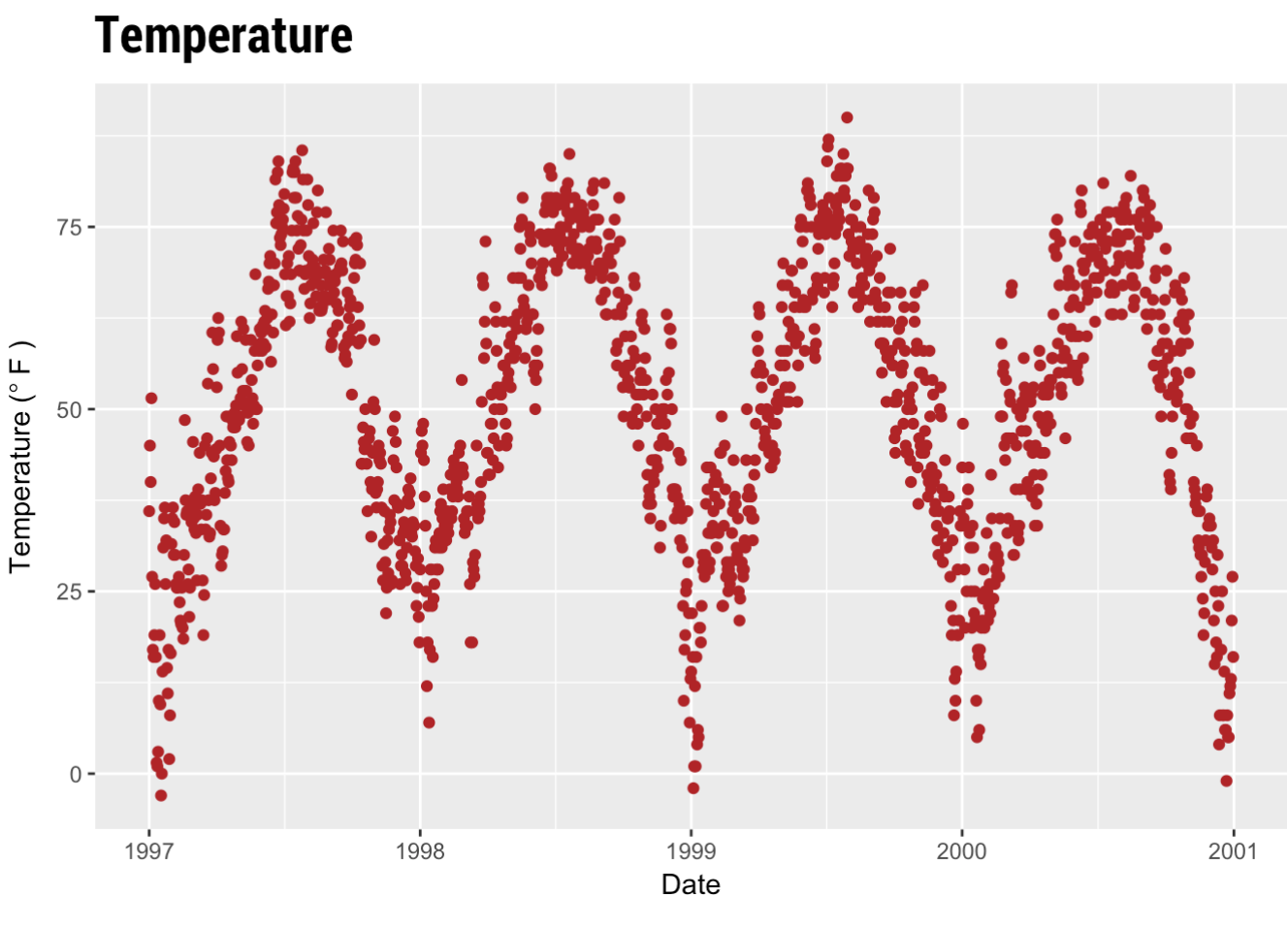
```
g <- g + ggtitle("Temperature fluctuations \n between 1997 and 2001")
g <- g + theme(plot.title = element_text(size=20, face="bold", vjust=1, lineheight = .8))
g
```



Working with axes

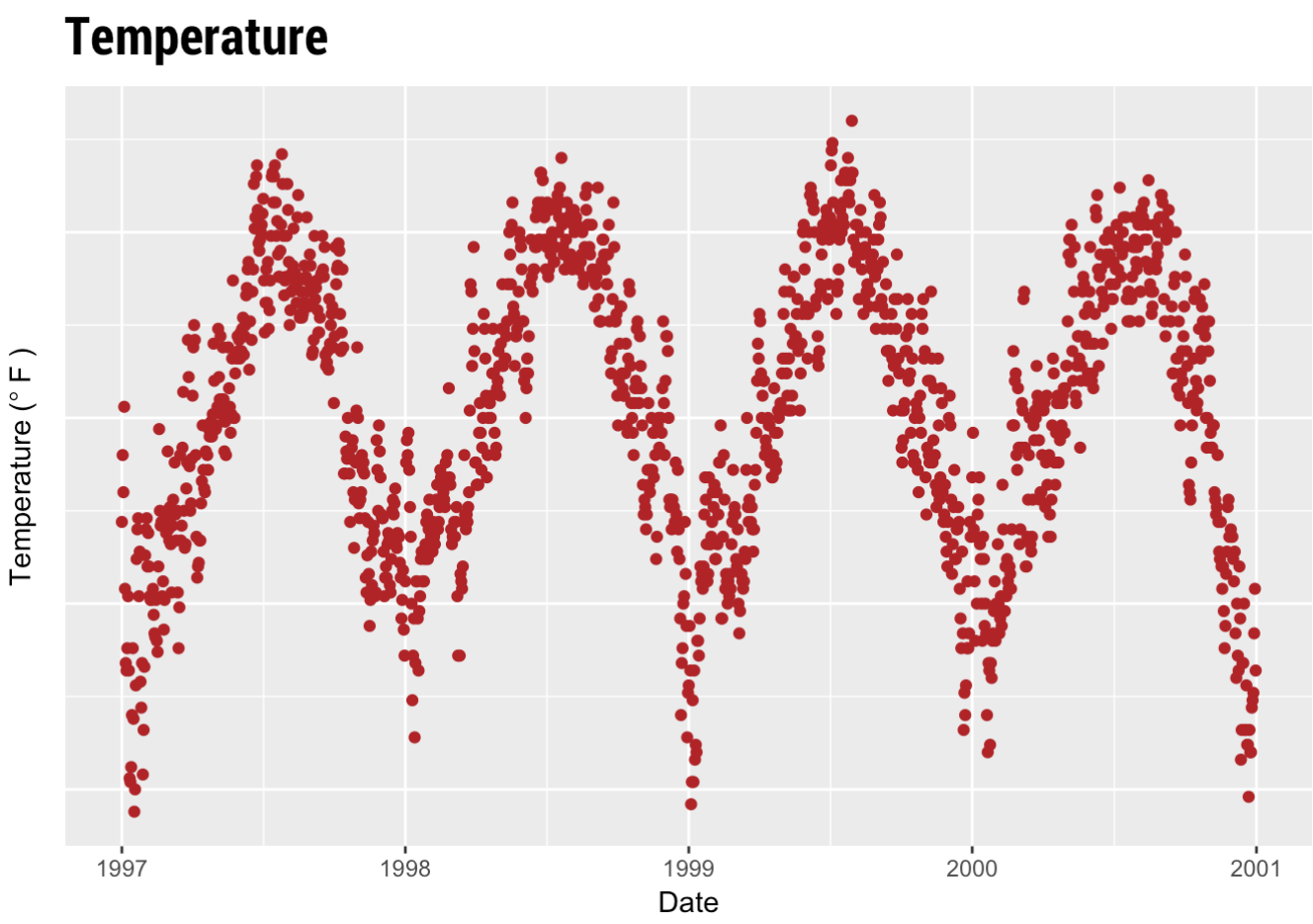
Add x and y axis label: labs(), xlab()

```
g <- g+labs(x="Date", y=expression(paste("Temperature (", degree ~ F, " )")), title = "Temperature")
g
```



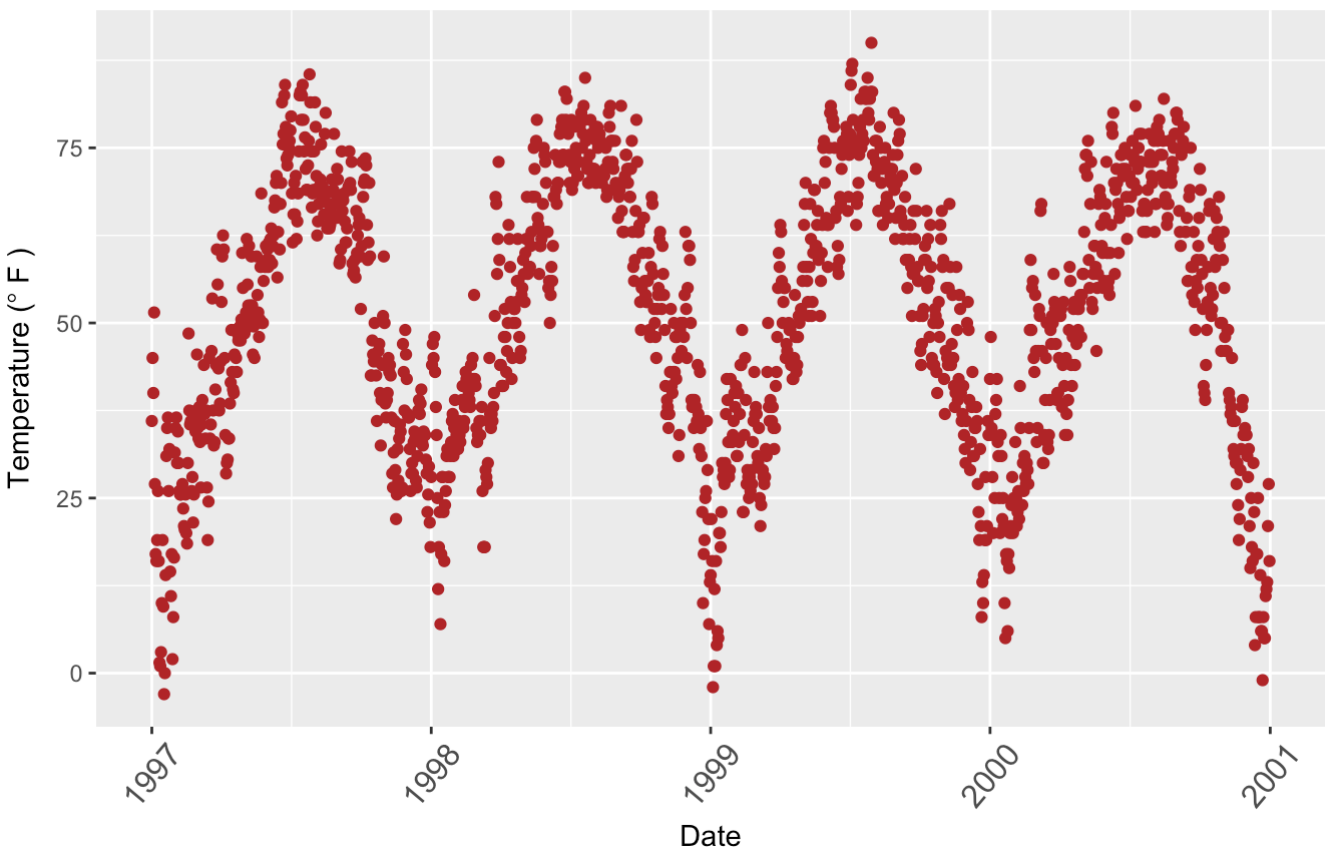
Get rid of axis ticks and tick text: theme(), axis.ticks.y

```
g + theme(axis.ticks.y = element_blank(), axis.text.y = element_blank())
```



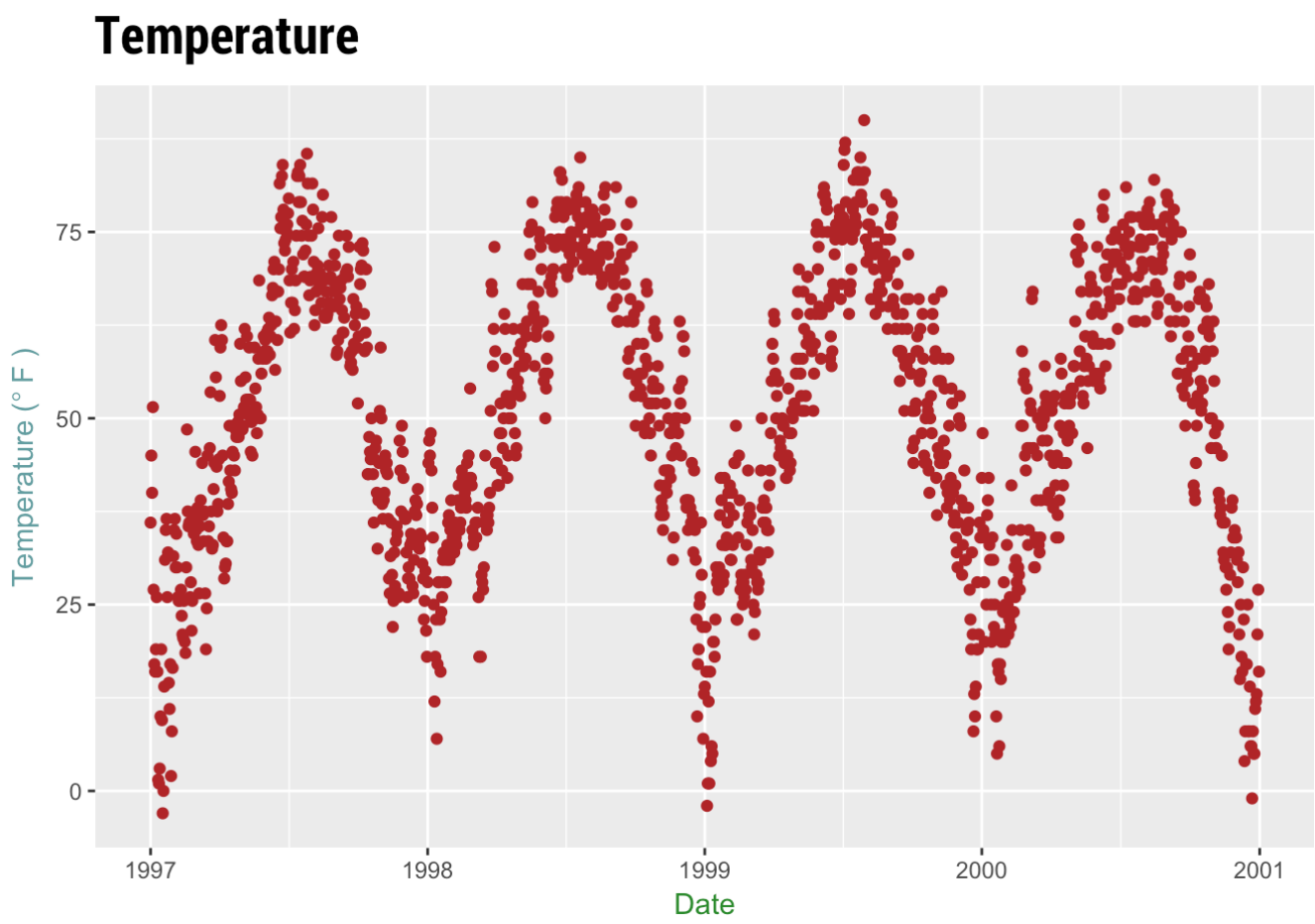
Change size of and rotate tick text: axis.text.x

```
g + theme(axis.text.x = element_text(angle=50, size=12, vjust=0.5))
```



Move the labels away from the plot and add color: axis.title.x, vjust

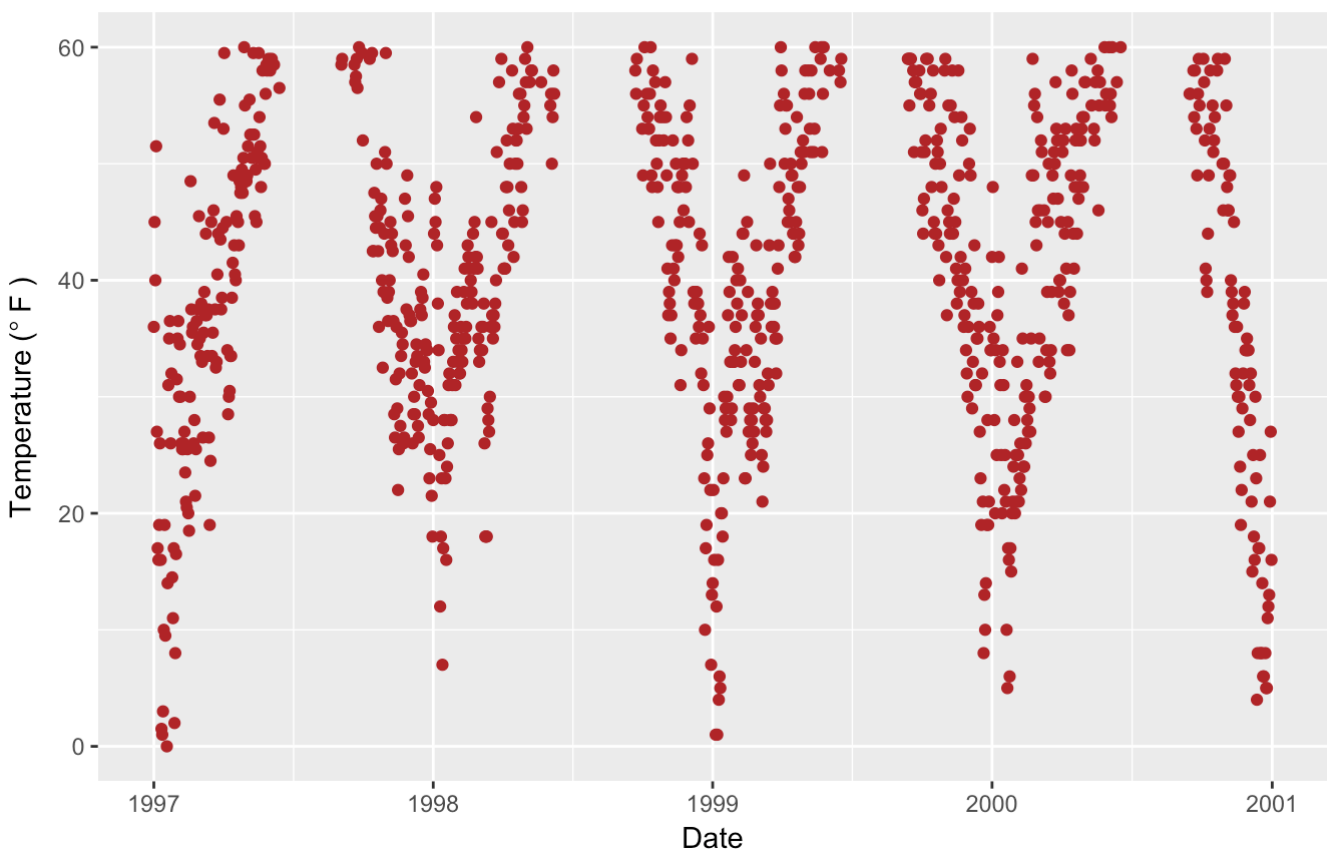
```
g + theme(axis.title.x = element_text(color="forestgreen", vjust=0.35),
          axis.title.y = element_text(color="cadetblue", vjust=0.35))
```



Limit an axis to a range: ylim(), scale_x_continuous(), coord_cartesian()

```
g + ylim(c(0,60))
```

```
## Warning: Removed 550 rows containing missing values (geom_point).
```



Alternatively: g +

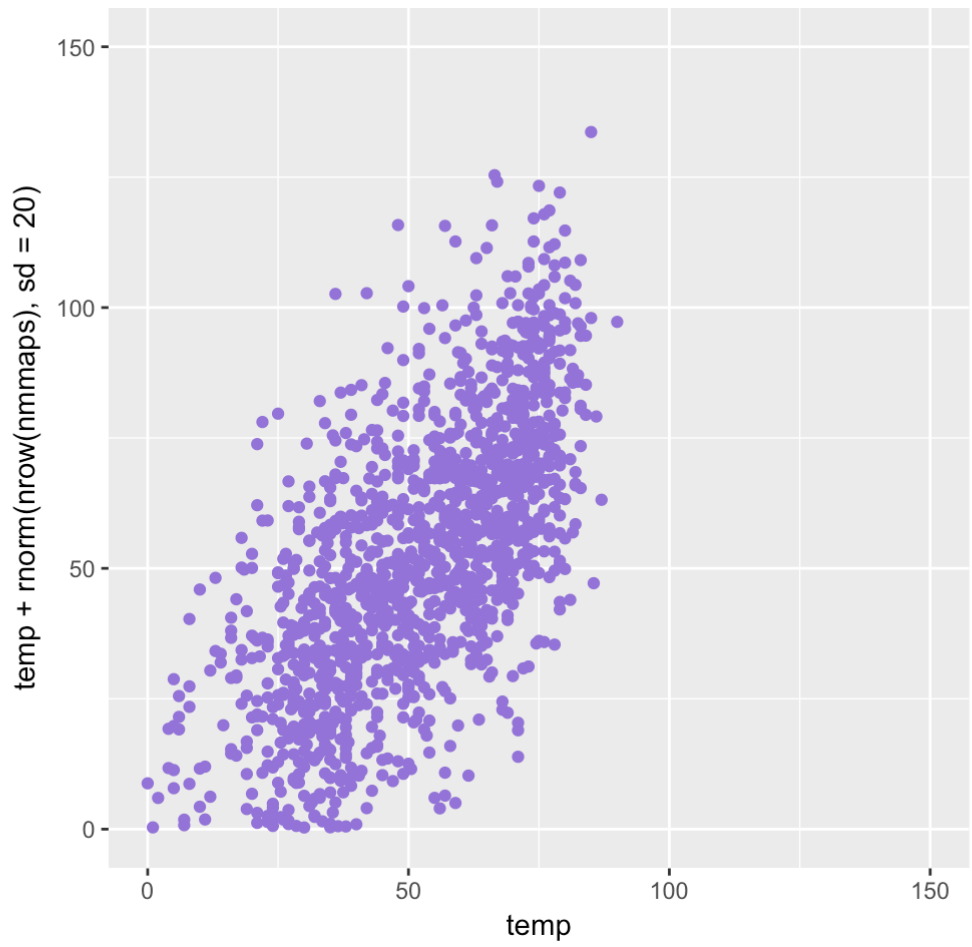
scale_x_continuous(limits=c(0,35)) + g+coord_cartesian(xlim=c(0,35)). The former removes all data points outside the ranger and second one adjusts the visible area.

If you want the axes to be the same: coord_equal()

For demo purposes, we will plot the temperature against the temperature with some random noise. We want both axes to be the same scale / same range.

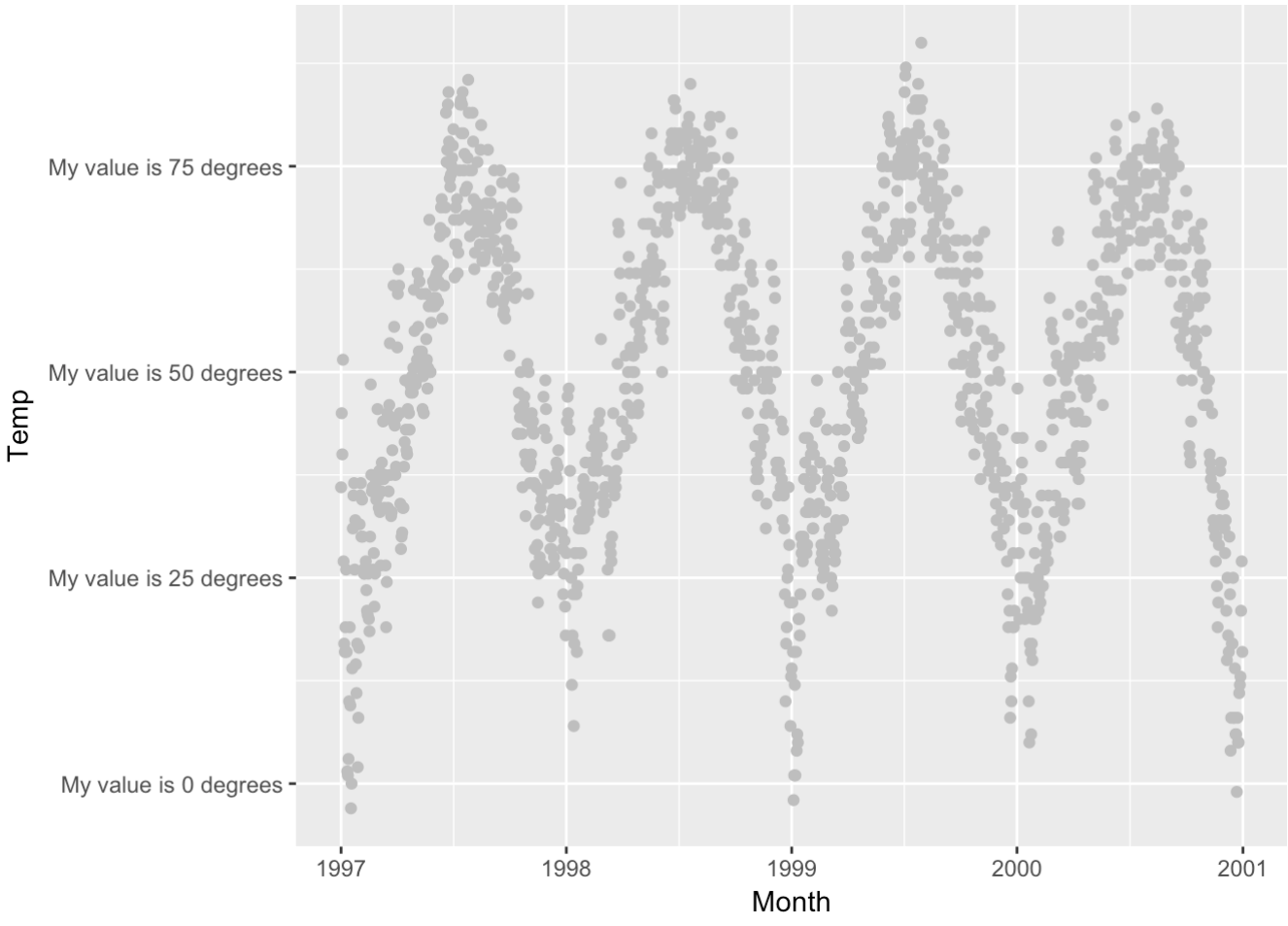
```
# rnorm(1461, mean=0, sd=20) creates 1461 numbers between 0 to 20
ggplot(nmmaps, aes(temp, temp + rnorm(nrow(nmmaps), sd=20))) + geom_point(color="mediumpurple") +xlim(c(0,150))+y
lim(c(0,150))+coord_equal()
```

```
## Warning: Removed 52 rows containing missing values (geom_point).
```



Use a function to alter labels: label=function(x)

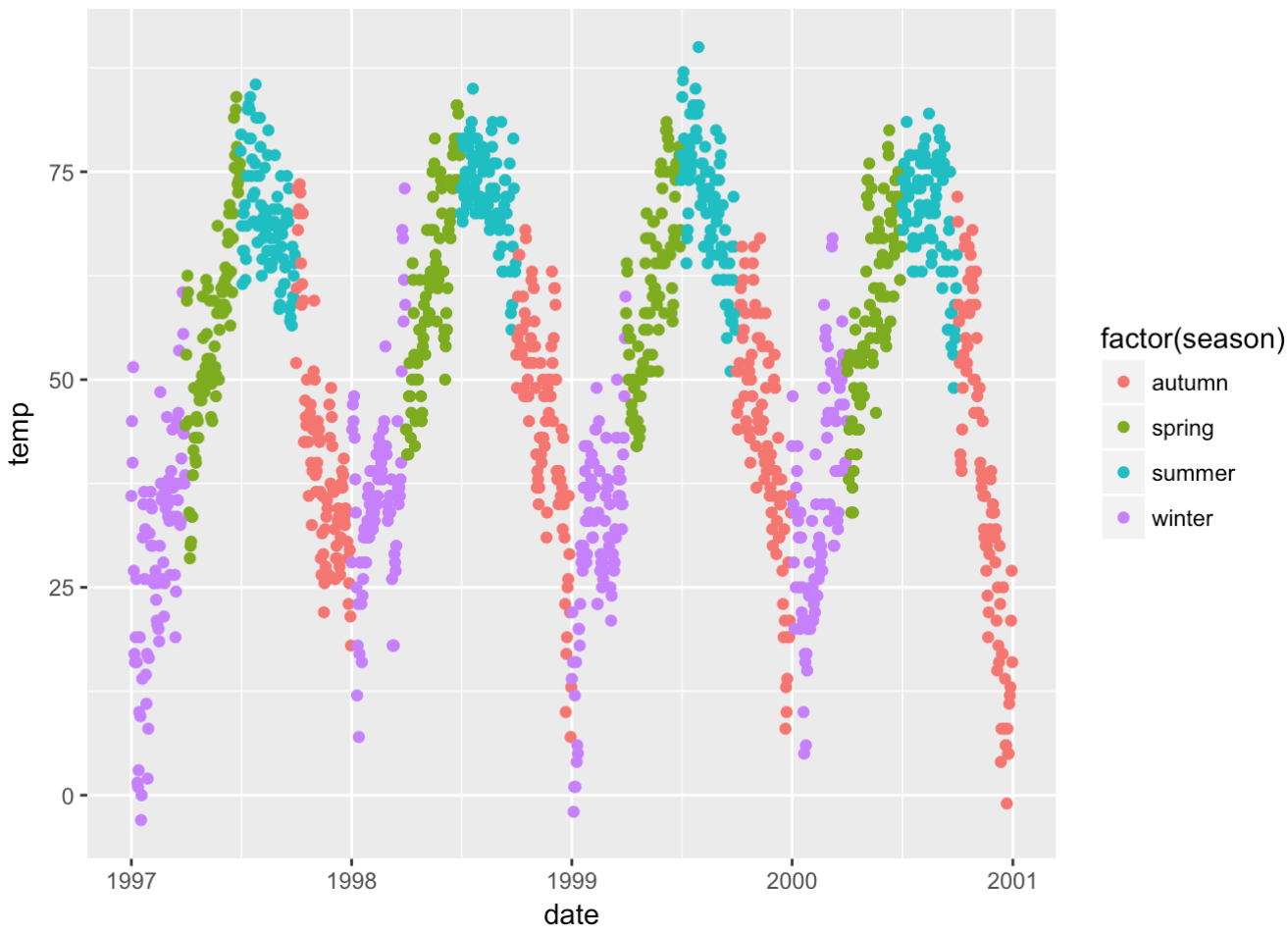
```
ggplot(nmmaps, aes(date, temp)) + geom_point(color="grey") + labs(x="Month", y="Temp") +
scale_y_continuous(label=function(x){return (paste("My value is", x, "degrees"))})
```



Working with the legend

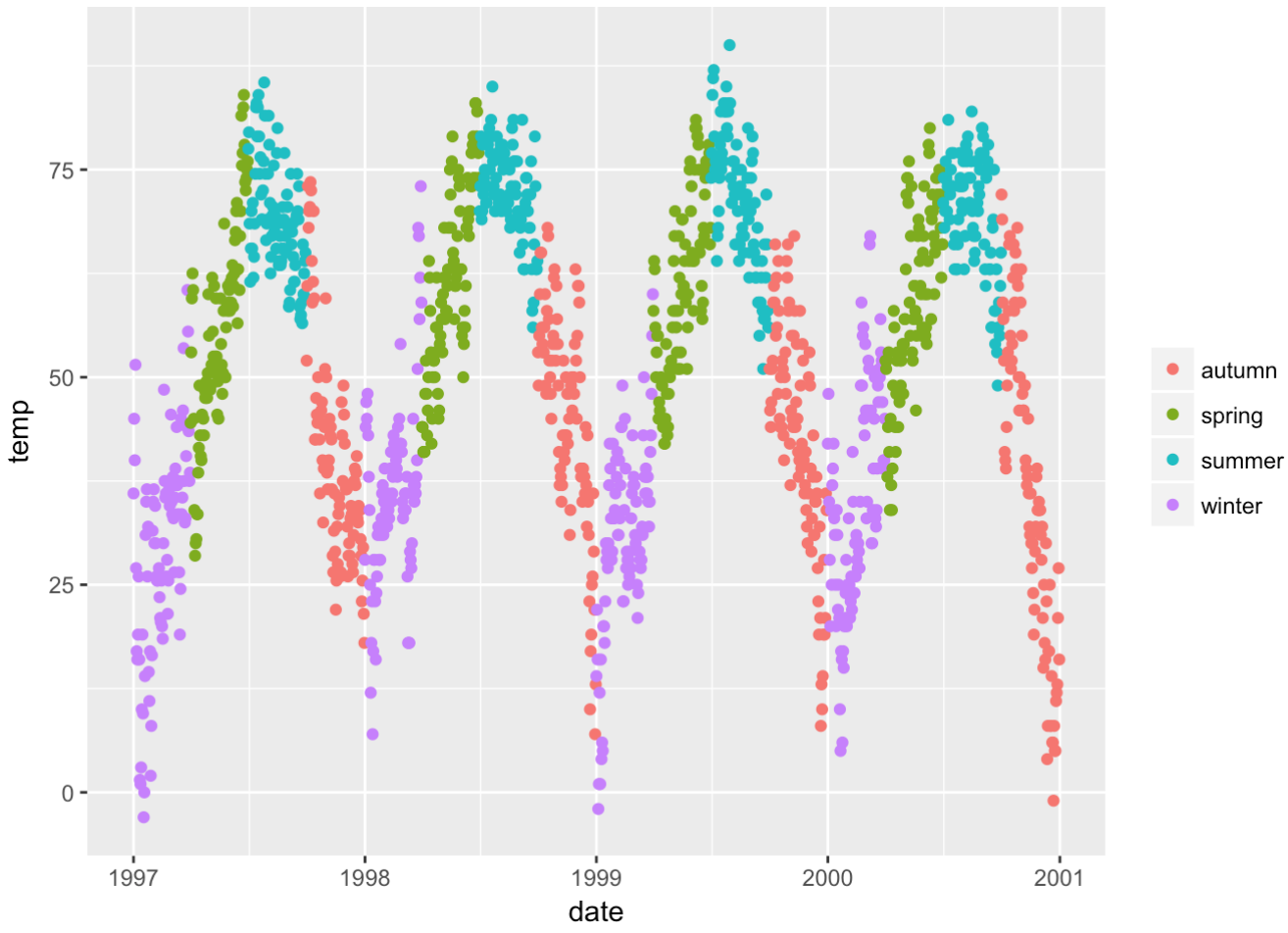
We will color code the plot based on season. We observe that by default the legend title is what we've specified in the color argument.

```
g <- ggplot(nmmaps, aes(date, temp, color=factor(season))) + geom_point()
g
```



Turning off the legend title

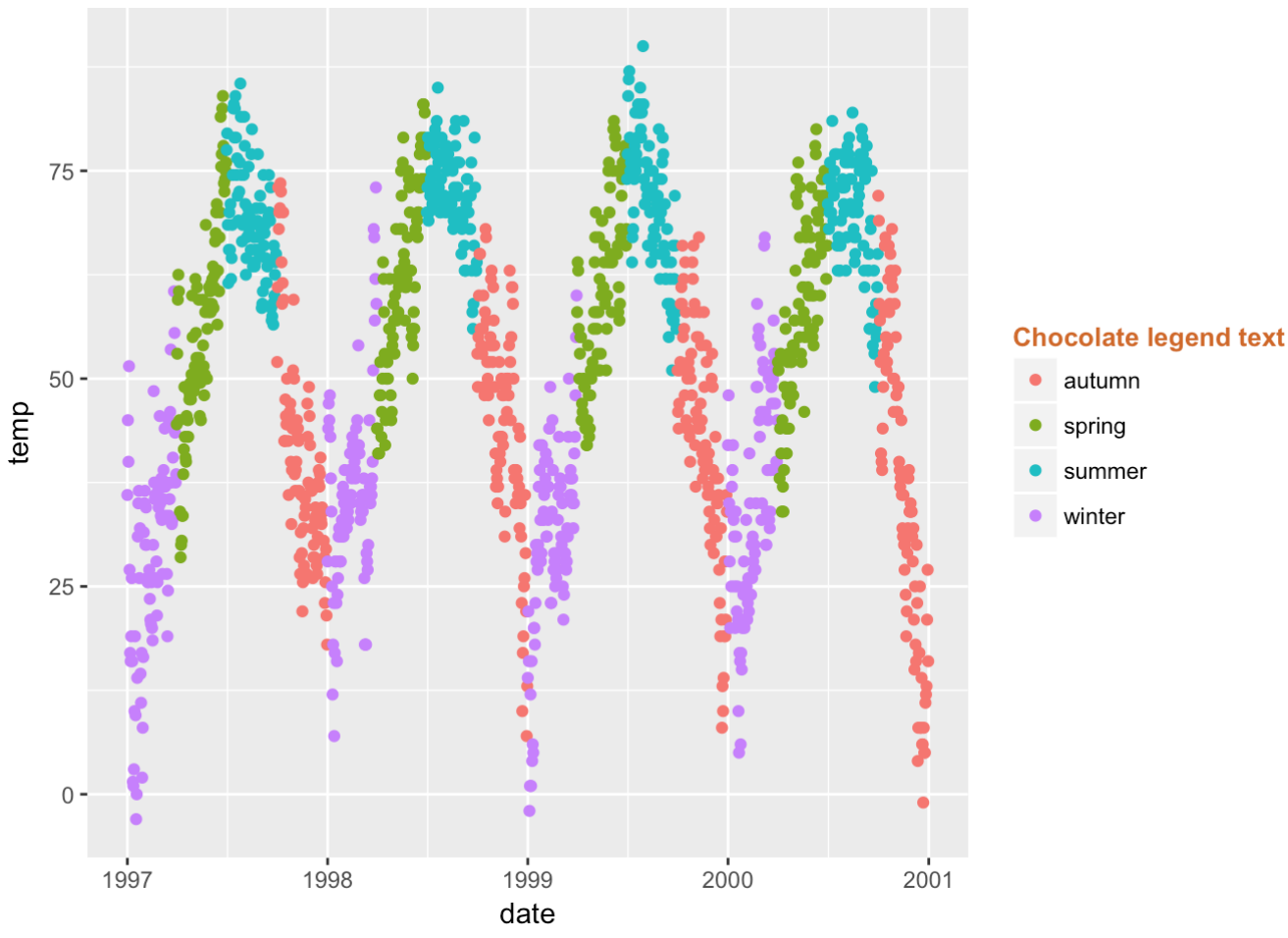
```
g + theme(legend.title=element_blank())
```



Change the title of the legend: scale_color_discrete(name="Title")

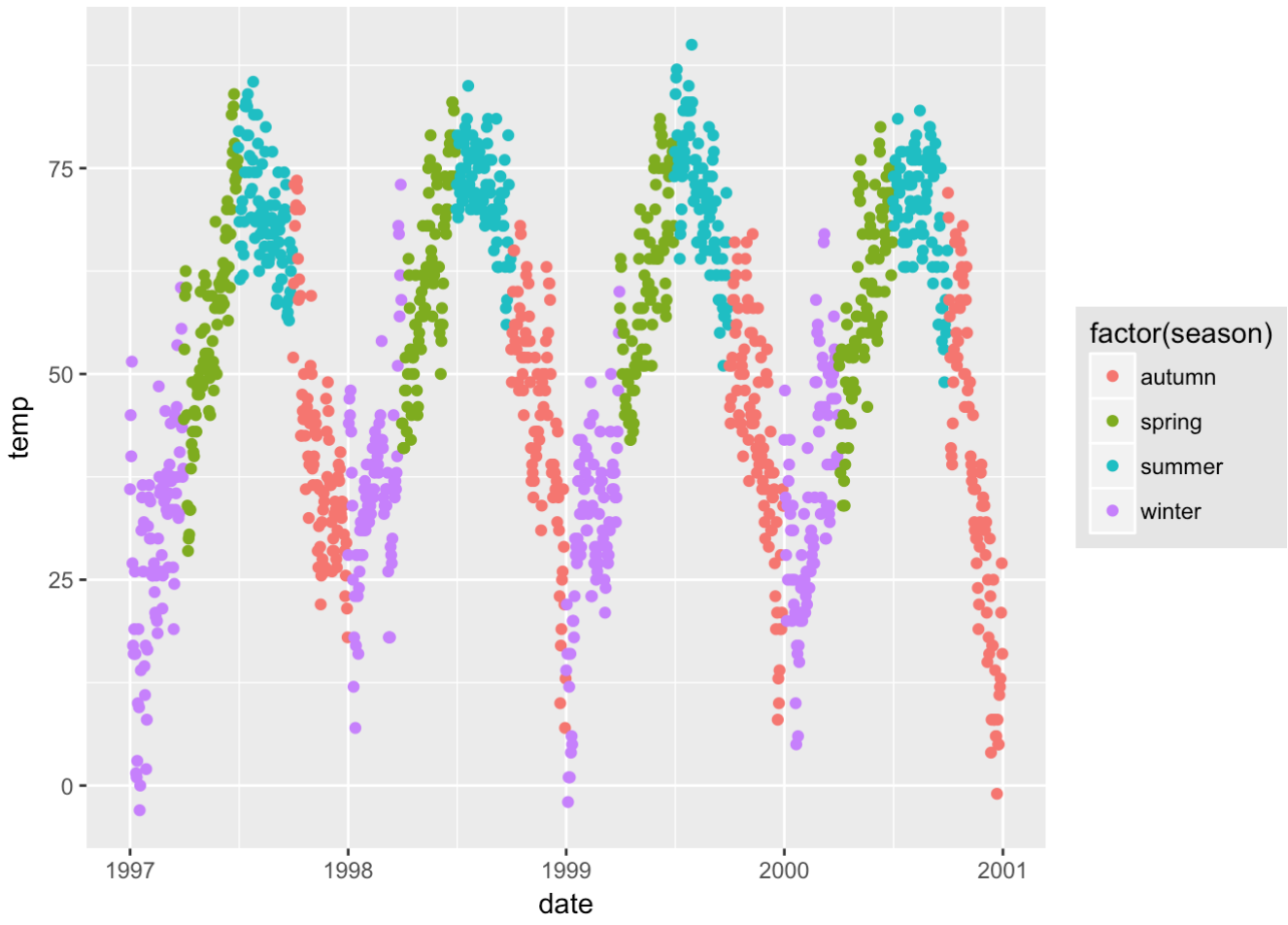
To change the title of the legend we would use the name argument in our scale function. Without the scale function, we will need to change the data itself so it has the right format.

```
g <- ggplot(nmmaps, aes(date, temp, color=factor(season))) + geom_point()
g + theme(legend.title = element_text(color="chocolate", size=10, face="bold")) + scale_color_discrete(name="Choc
olate legend text")
```



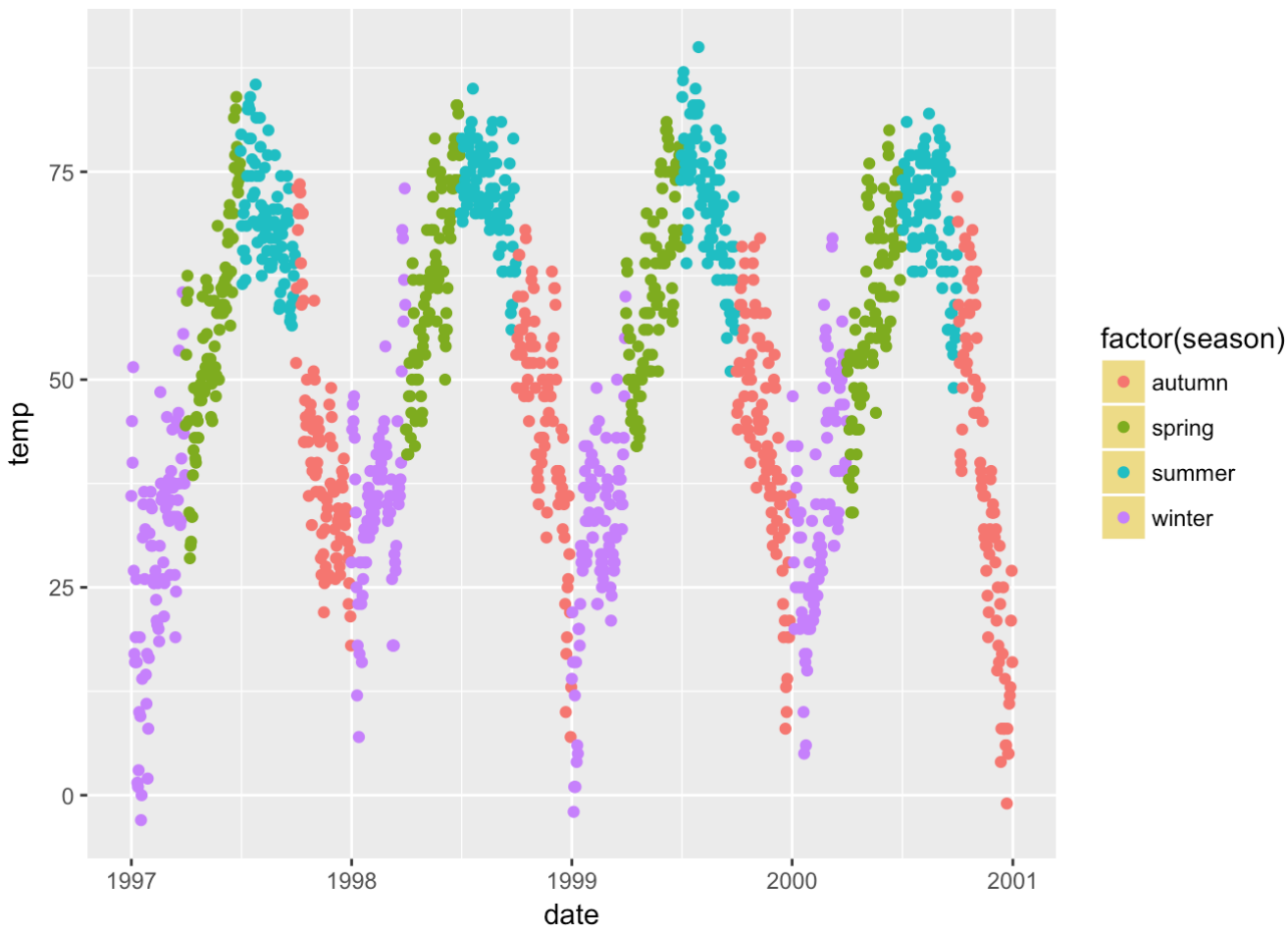
Add a box around legend: legend.background

```
g + theme(legend.background = element_rect(fill="gray90", size=.8))
```



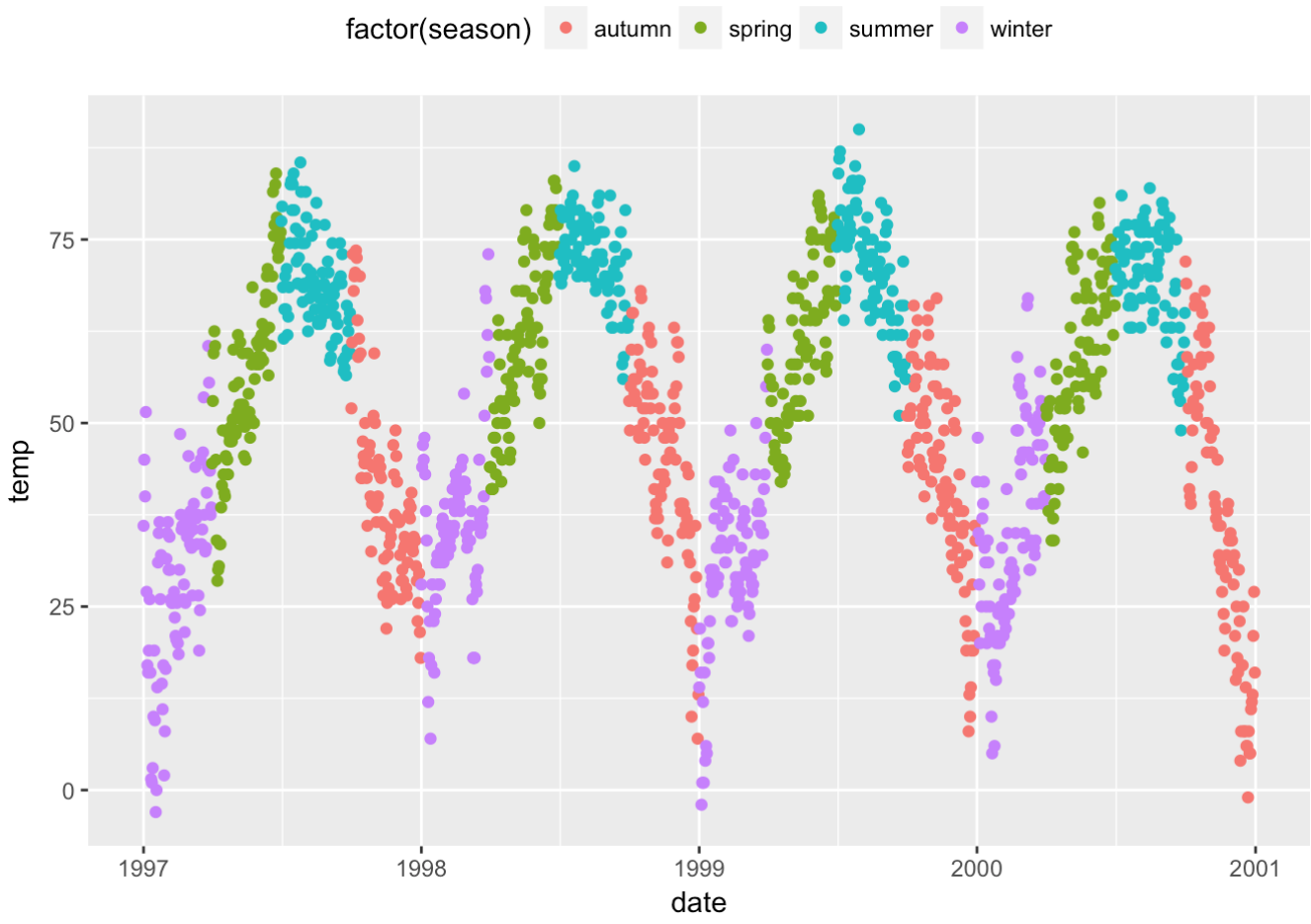
Change the box color in each legend key: legend.key(element_rect(fill="red"))

```
g + theme(legend.key = element_rect(fill="lightgoldenrod2"))
```



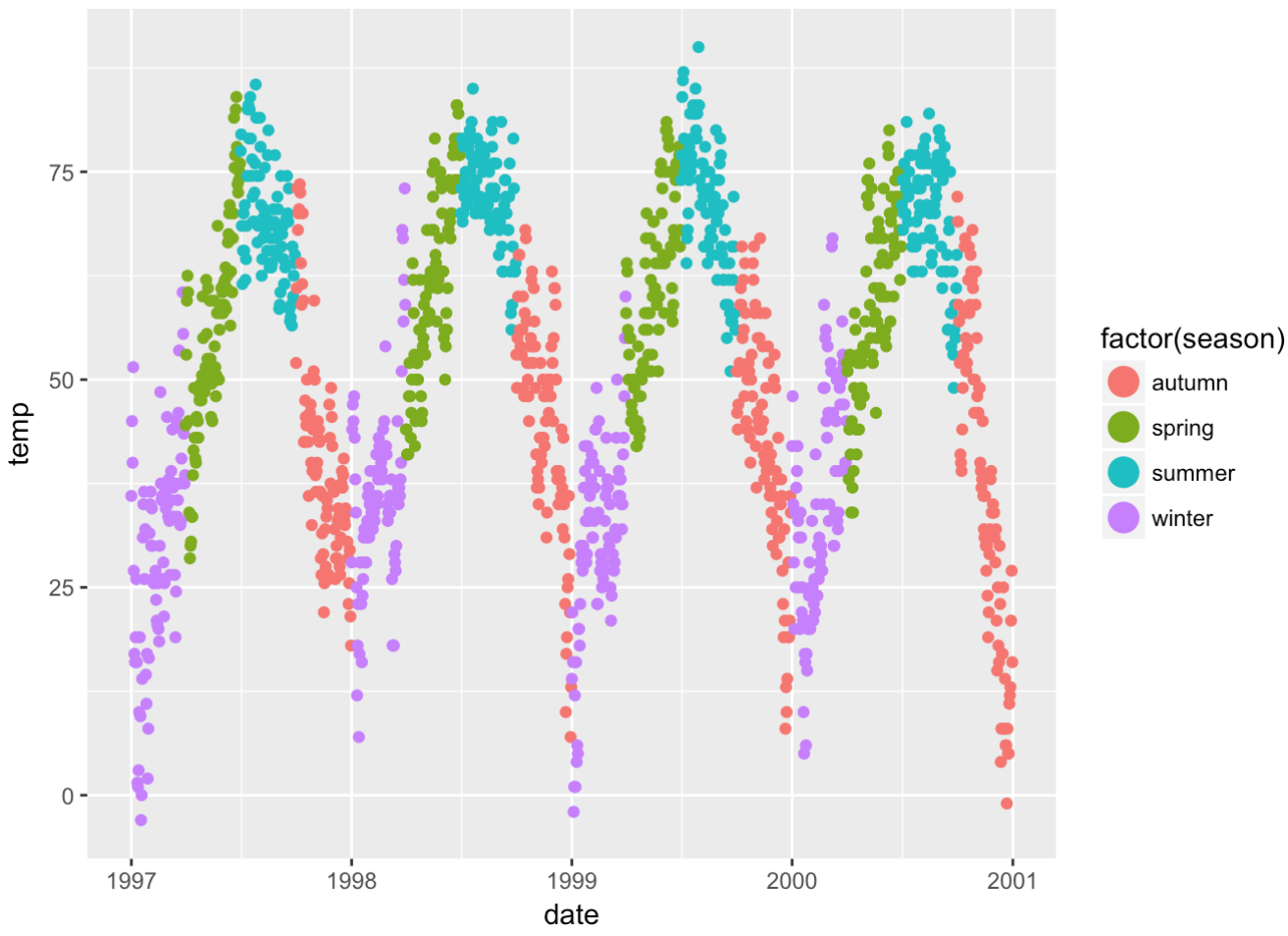
Change the position of the legend: legend.position()

```
g + theme(legend.position = "top")
```



Change the size of the symbols in the legend keys

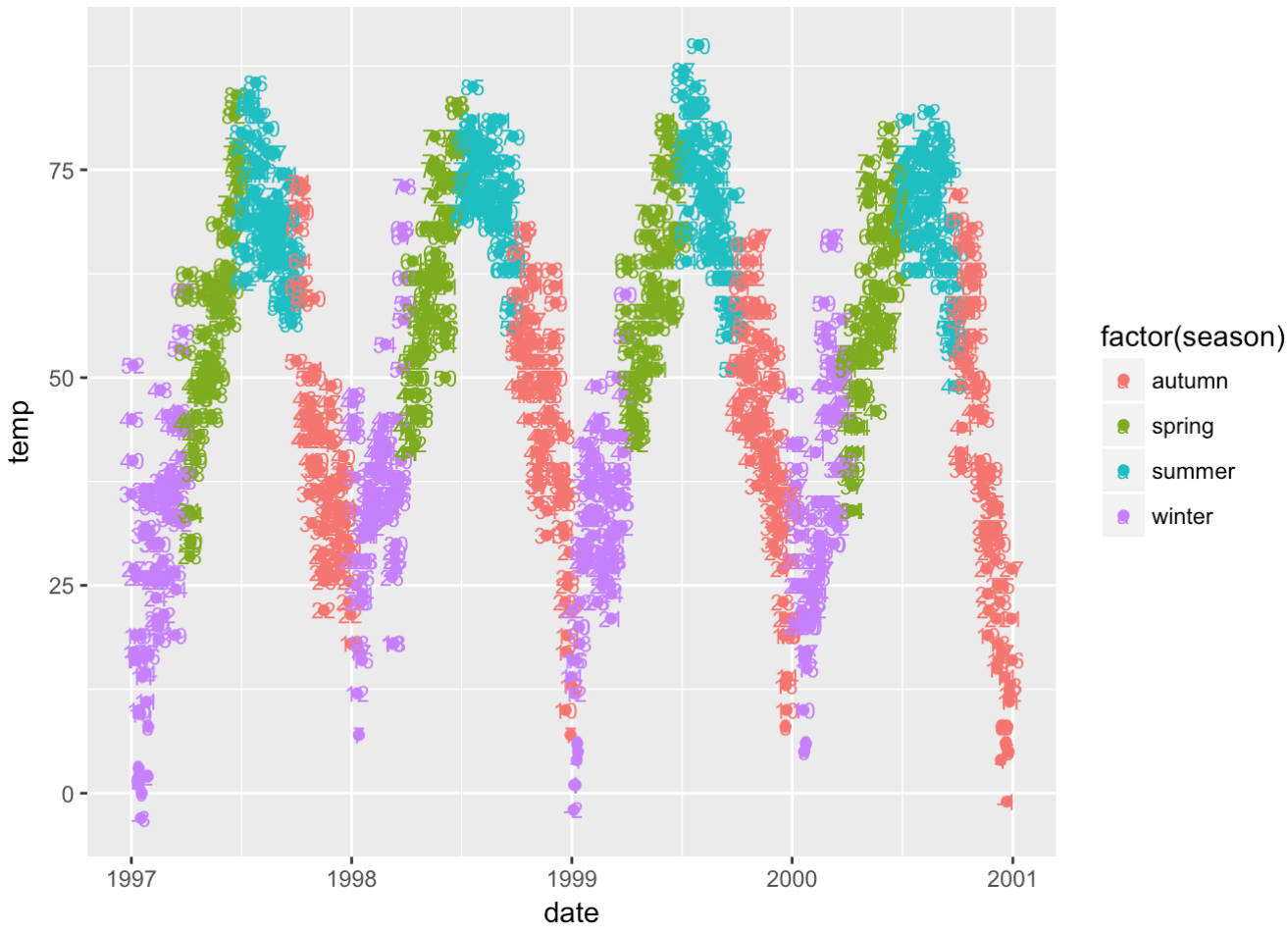
```
g + guides(color = guide_legend(override.aes = list(size=5)))
```



Leave a layer off the legend: show_guide

Let's say we have a point layer and add label text to it using geom_text (now we have two layers), by default both the points and the label text layers (again, two layers) will end up in our legend like this:

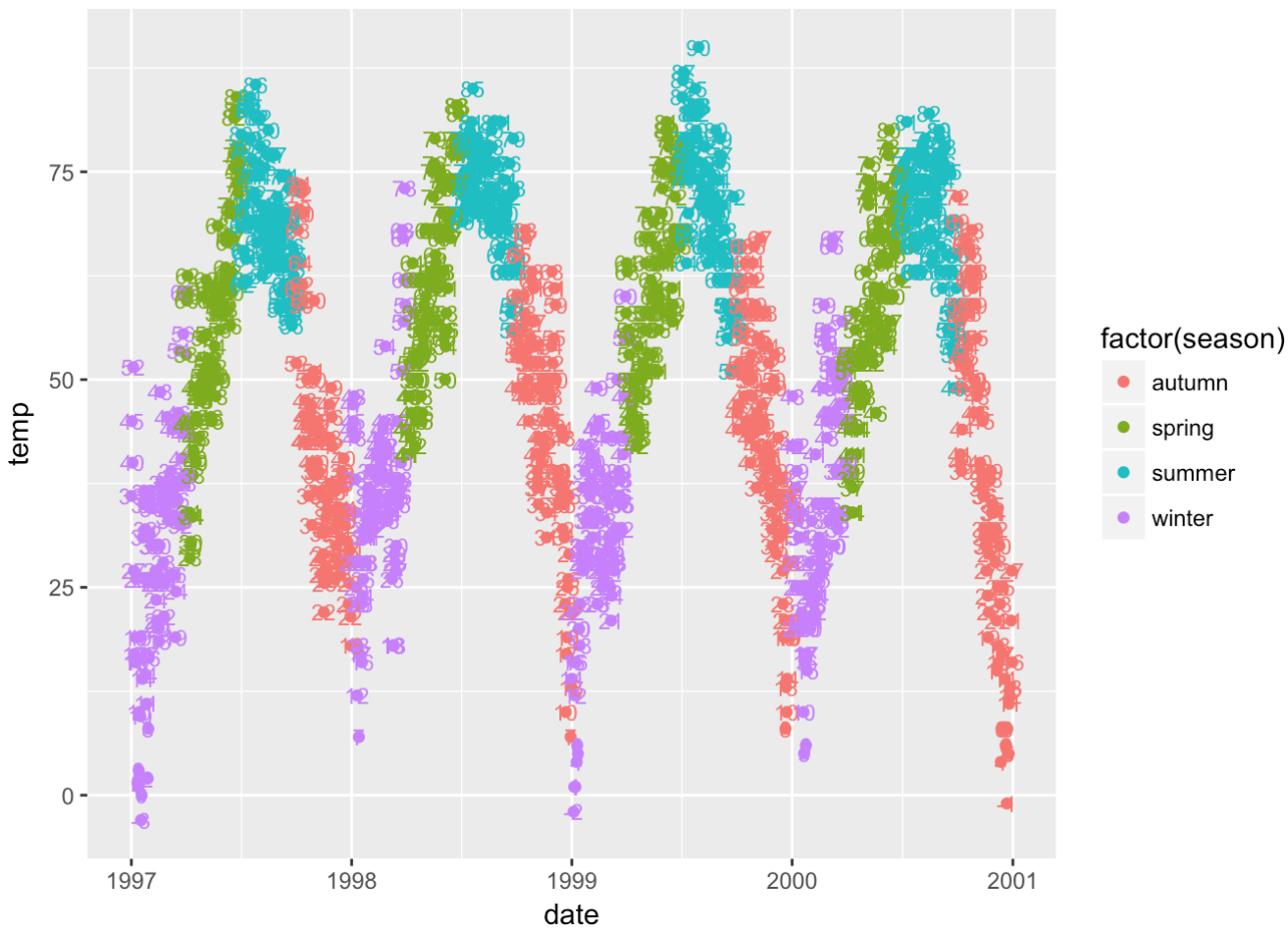
```
g <- ggplot(nmmaps, aes(date, temp, color=factor(season))) + geom_point()
g + geom_text(data=nmmaps, aes(date, temp, label=round(temp)), size=3)
```



Fortunately, we can turn off a layer in the legend using show_guide=F

```
g + geom_text(data=nmmaps, aes(date, temp, label=round(temp)), size=3, show_guide=F)
```

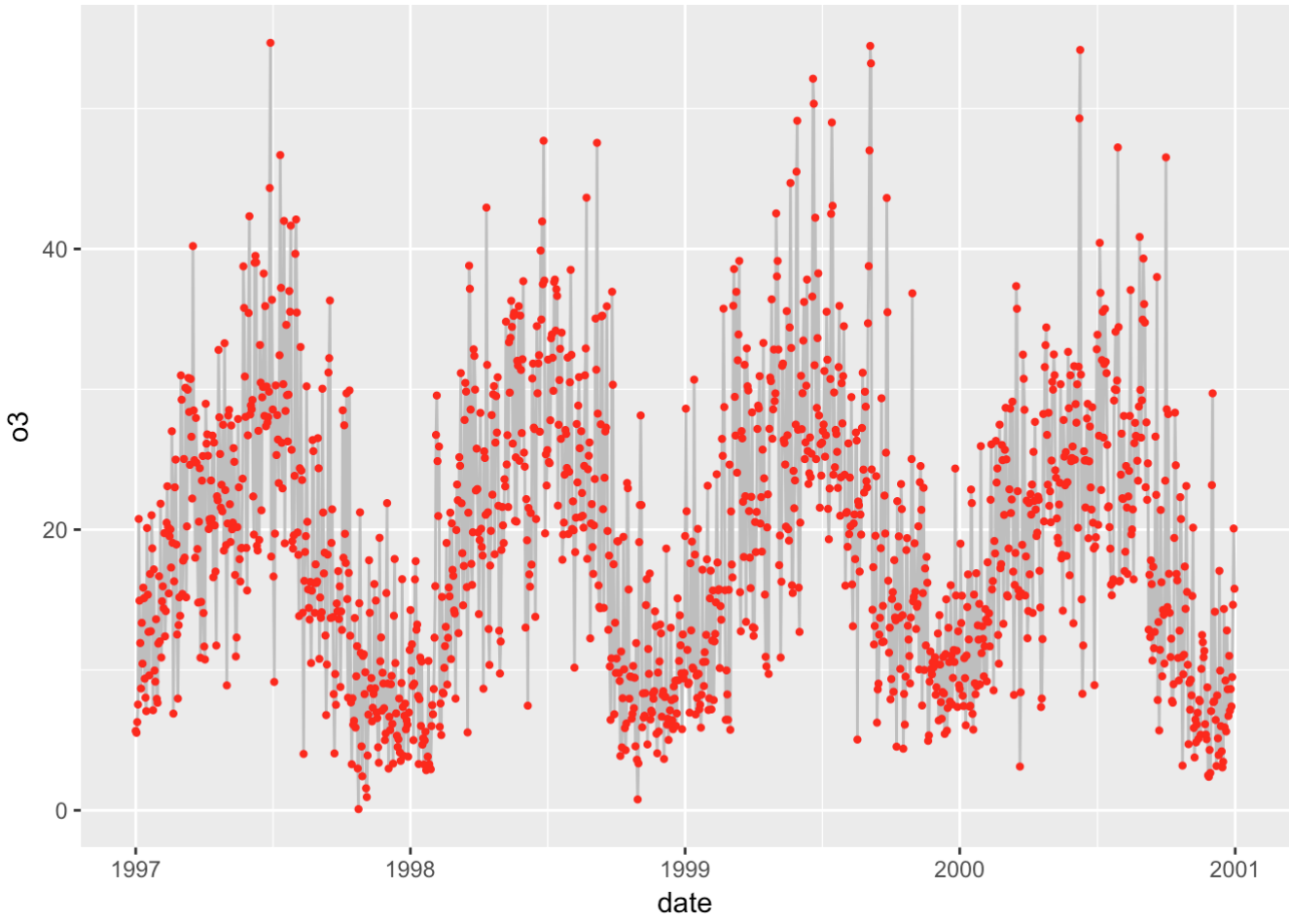
```
## Warning: `show_guide` has been deprecated. Please use `show.legend`
## instead.
```



Manually adding legend items

ggplot2 will not add a legend automatically unless we map aethetics (color, size etc) to a variable. There are times, though, that we want to have a legend so that it's clear what you're plotting. Here is the default:

```
# No legend by default because we didn't map aes to a variable
ggplot(nmmaps, aes(x=date, y=o3)) + geom_line(color="grey") + geom_point(color="red", size=.8)
```

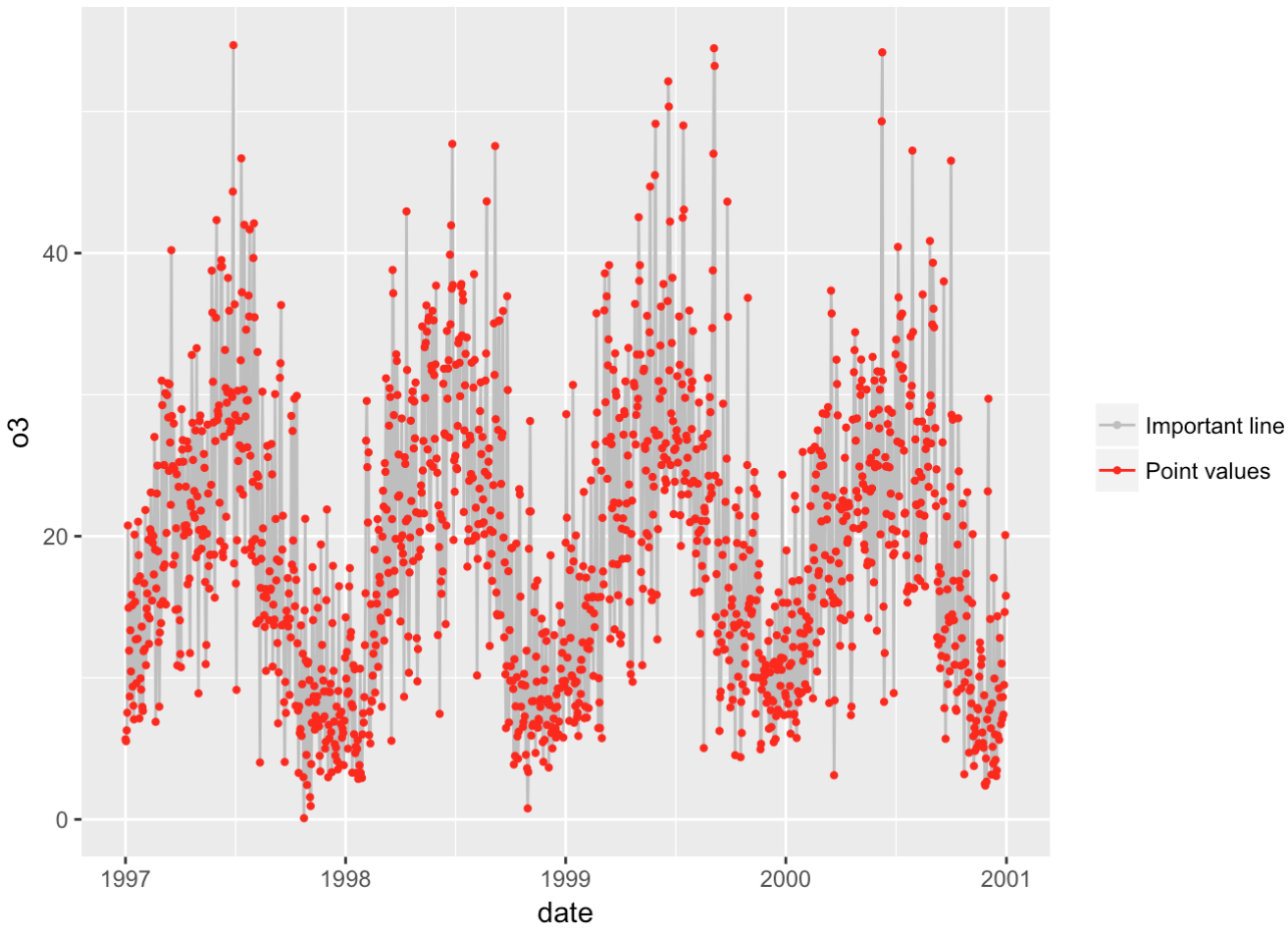


We can force a legend by mapping to a “variable”. We are mapping the lines and the points using aes and we are mapping not to a variable in our dataset but to a single string (so that we get just one color for each).

```
ggplot(nmmaps, aes(x=date, y=o3)) + geom_line(aes(color="Important lines")) + geom_point(aes(color="My points"),
size=.8)
```

We’re getting close but this is not what we want. We wanted grey and red. To change the color, we use scale_color_manual(). Scale_color_manual allows us to create our own discrete scale. The **name** argument will become our legend title and the **values** accepts a set of aesthetic values to map our data values to. If this is a named vector (i.e. ‘Important line’) then the values will be matched based on the names. If unnamed, values will be matched in order.

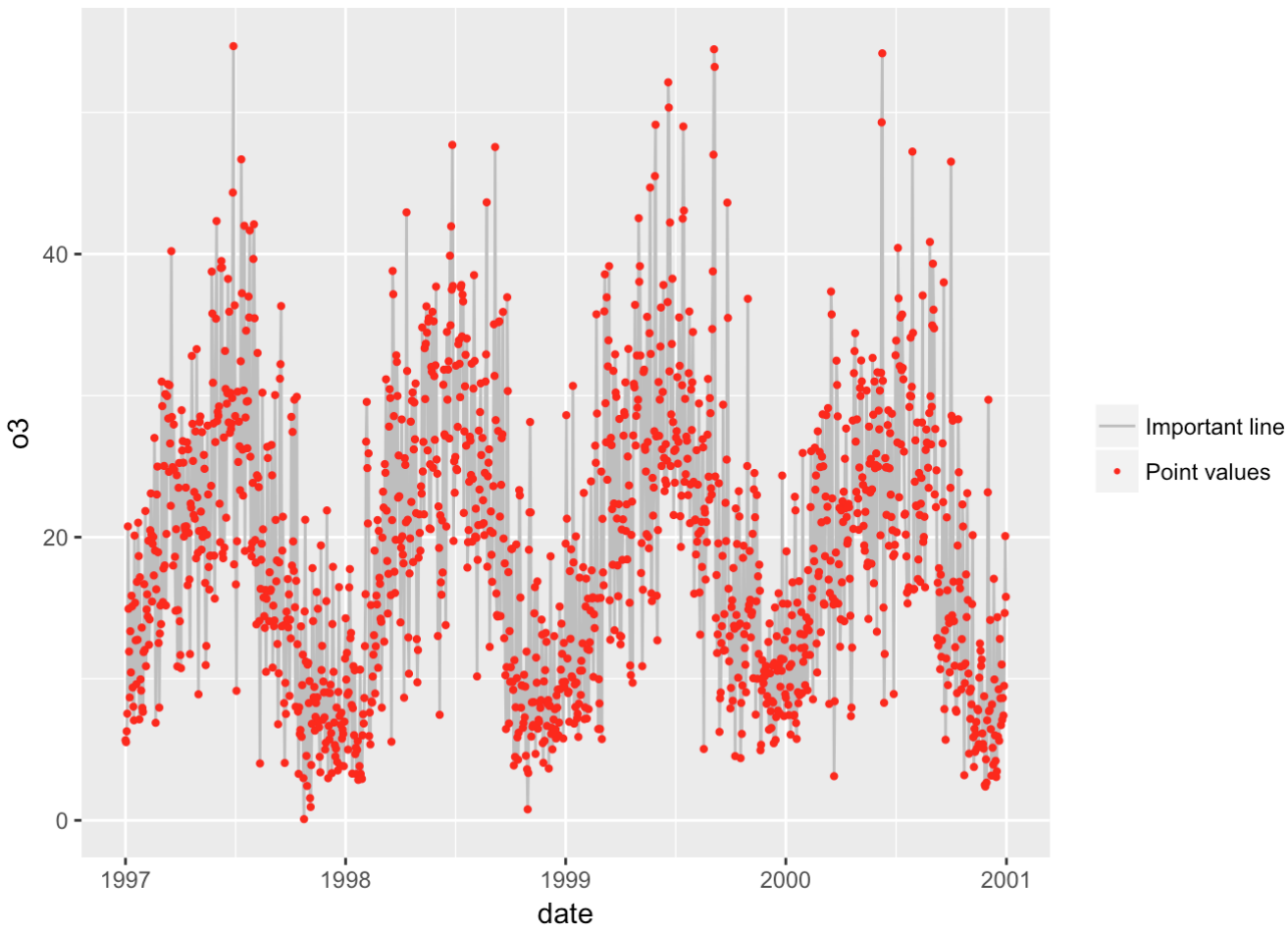
```
# we pass an empty value to name so the legend title will be blank
ggplot(nmmaps, aes(x=date, y=o3)) + geom_line(aes(color="Important line")) + geom_point(aes(color="Point
values"), size=.8) + scale_color_manual(name='', values = c('Important line'='grey', 'Point values' = 'red'))
```



Tantalizingly close! But we don't want a line with a point for both Line=grey and point=red. The final step is to override the aesthetics in the legend. The guide() function allows us to control guides like the legend.

When using guides(), the guides for each scale can be set in our call of scale_color_manual with the argument **guide**. Here we set guide='legend', and then in our call to guides, we specify that we'd like to override the aesthetic of guide_legend

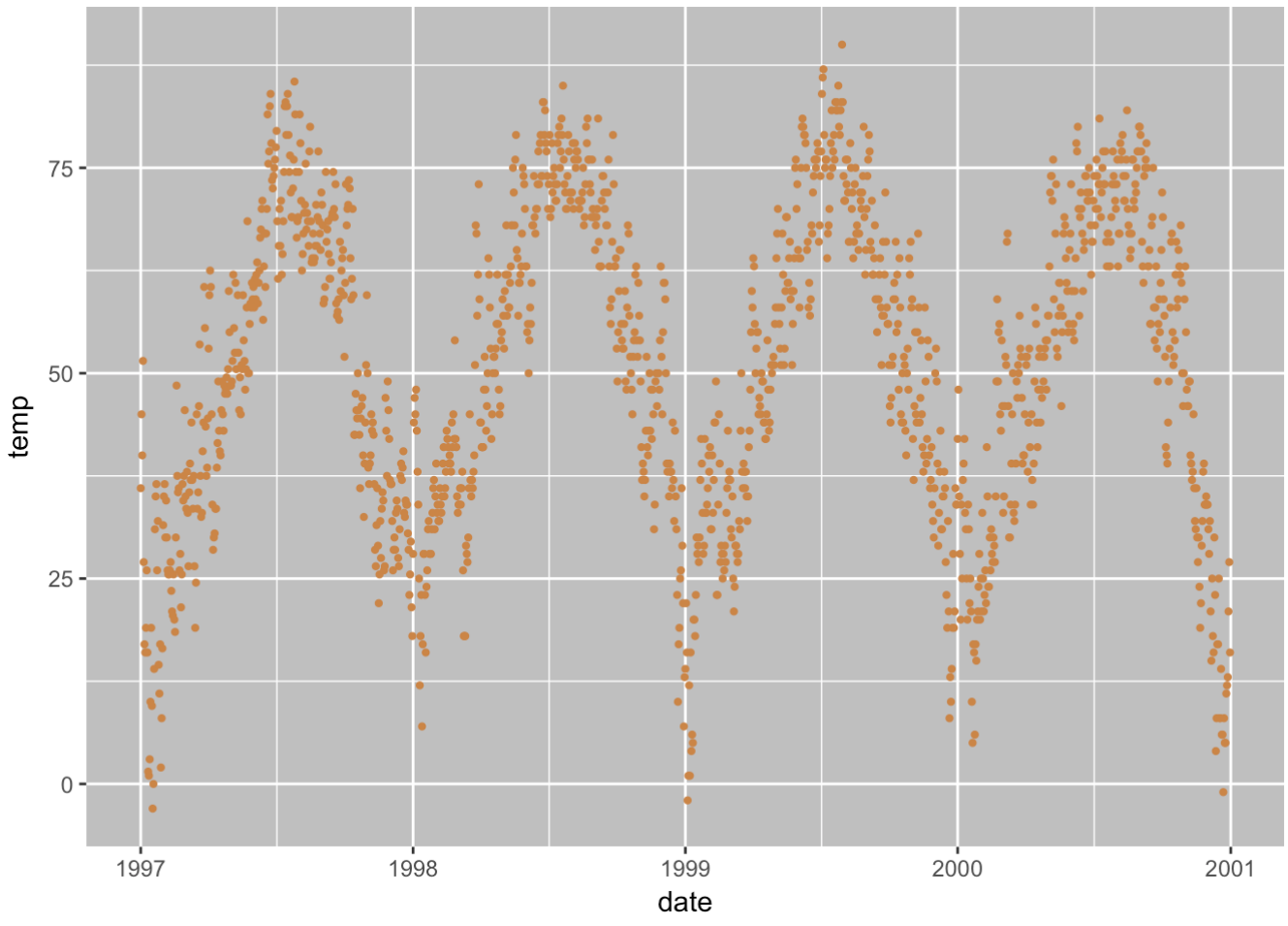
```
# Linetype:1 for Important line; Linetype:0 and shape:16 for Point values
ggplot(nmmaps, aes(x=date, y=o3)) + geom_line(aes(color="Important line")) + geom_point(aes(color="Point
values"), size=.8) + scale_color_manual(name="", values= c('Important line' = 'grey', 'Point values' = 'red'), gu
ide='legend') + guides(color=guide_legend(override.aes = list(linetype=c(1,0), shape=c(NA, 16))))
```



Working with background colors

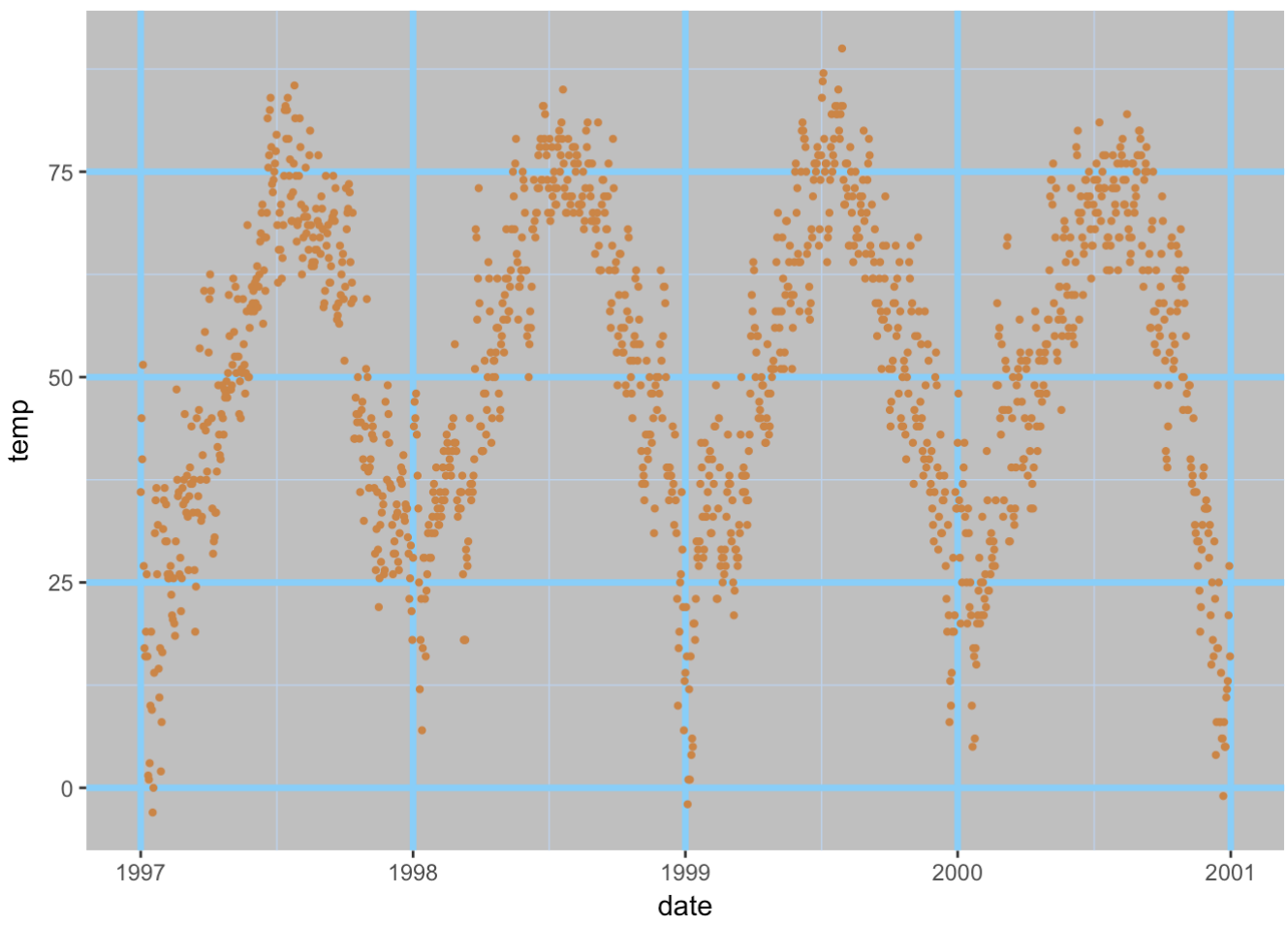
Change the panel color: panel.background

```
ggplot(nmmaps, aes(x=date, y=temp)) + geom_point(color="tan3", size=.8) + theme(panel.background=element_rect(fil
l = "grey75"))
```



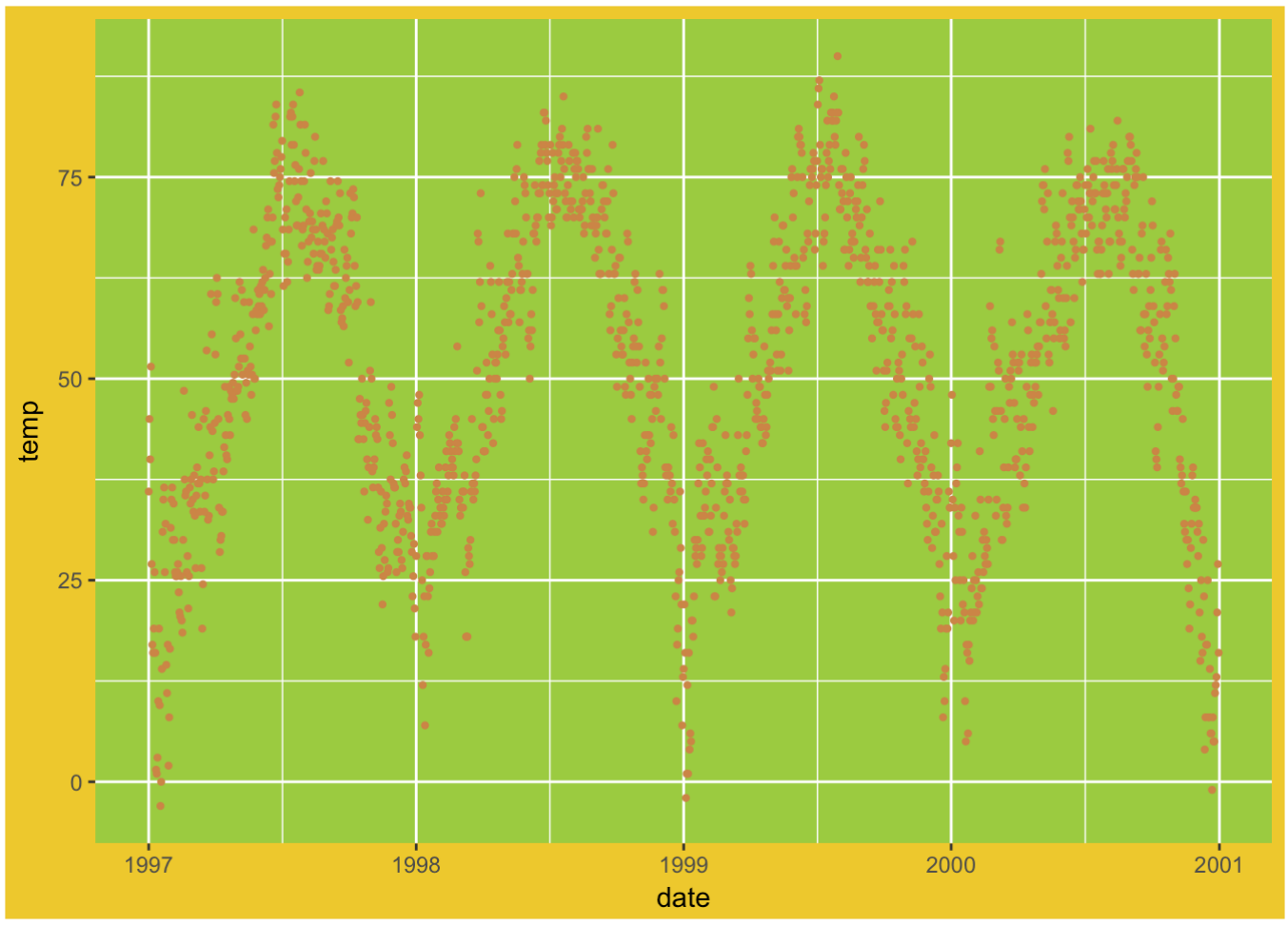
Change the grid lines: panel.grid.major, panel.grid.minor

```
ggplot(nmmaps, aes(x=date, y=temp)) + geom_point(color="tan3", size=.8) + theme(panel.background = element_rect(f
ill = "grey75"),
    panel.grid.major = element_line(color = "lightskyblue", size=1.2),
    panel.grid.minor = element_line(color="lightsteelblue2"))
```



Change the plot background (not the panel) color: plot.background

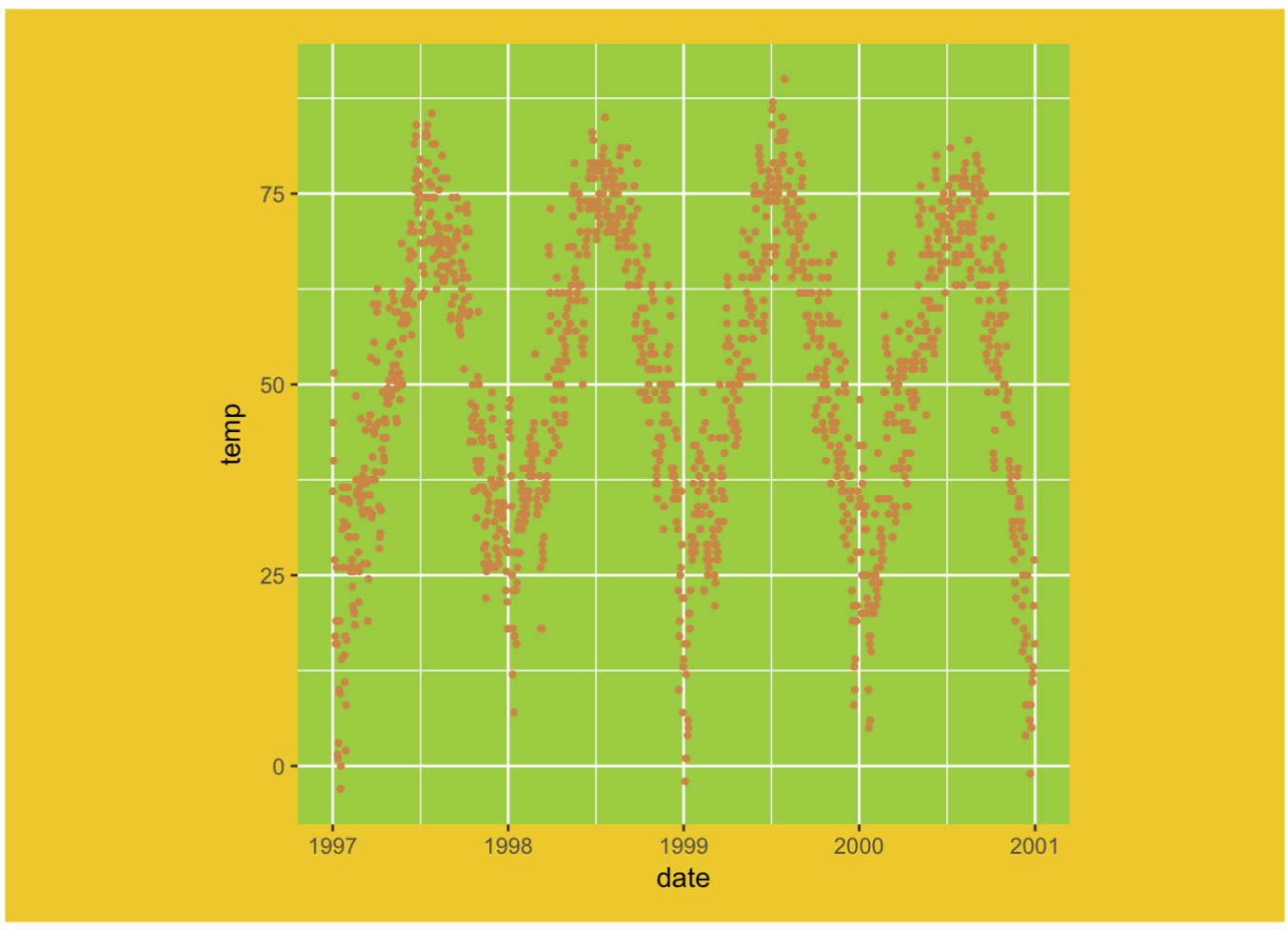
```
ggplot(nmmaps, aes(date, temp))+geom_point(color="tan3", size=.8)+
  theme(plot.background = element_rect(fill = 'gold2'),
        panel.background = element_rect(fill='yellowgreen'))
```



Working with margins: plot.margin

We sometimes find that we need to add a little space to one margin of our plot. Similar to the previous examples we can use an argument to the theme() function. In this case the argument is plot.margin. This argument can handle a variety of different units (cm, inches etc) but it requires the use of the function unit from the package **grid** to specify the units. Here we will use a 6 cm margin on the right and left.

```
# Add extra space to both left and right
library(grid)
ggplot(nmmaps, aes(date, temp))+geom_point(color="tan3", size=.8)+
  theme(plot.background = element_rect(fill = 'gold2'),
        panel.background = element_rect(fill='yellowgreen'),
        plot.margin=unit(c(.5,3,.5,3), "cm")) #Top, right, bottom, left
```

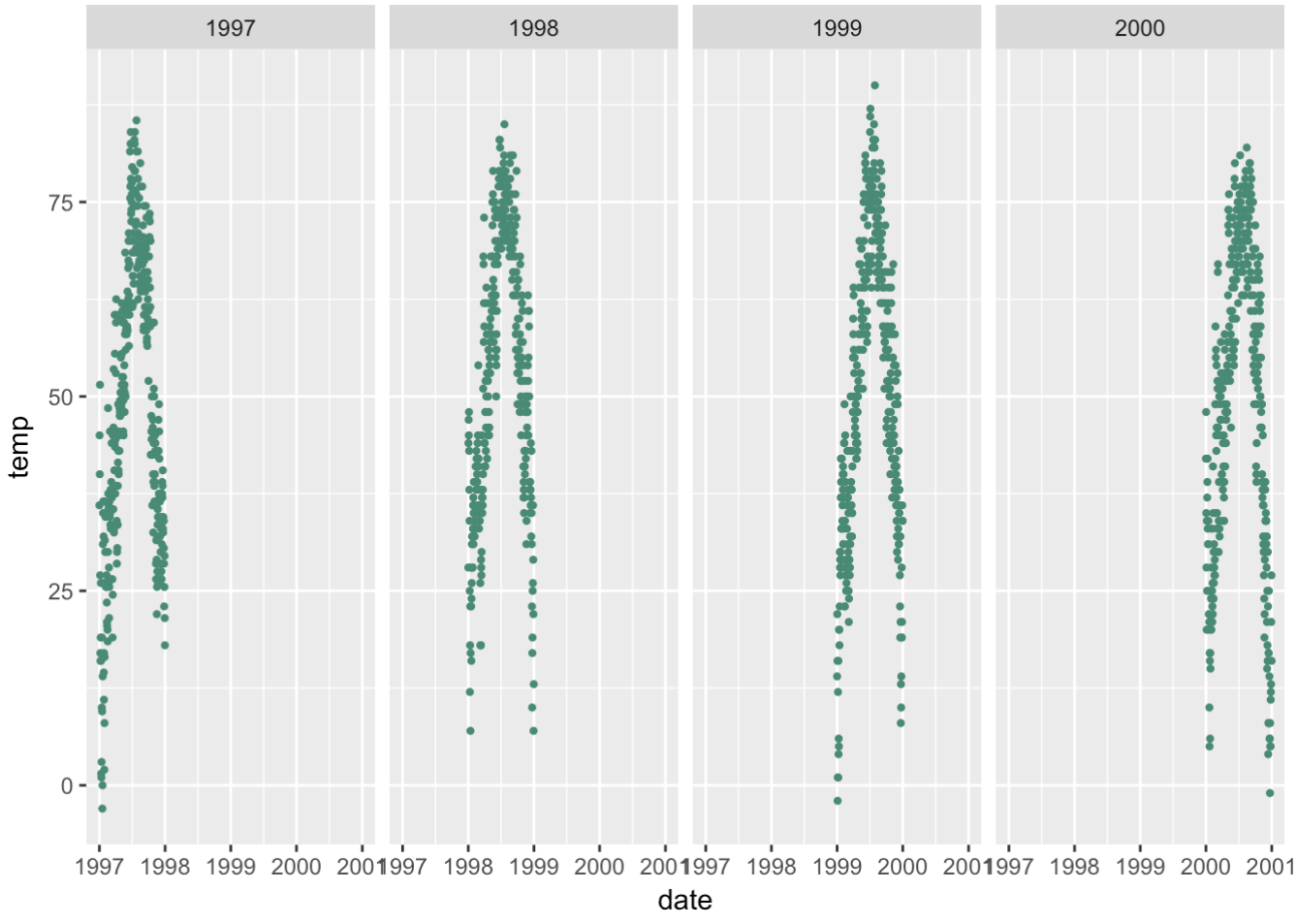


Creating multi-panel plots

The ggplot2 package has two nice functions for creating multi-panel plots. They are related but a little different. **facet_wrap** creates essentially a ribbon of plots based on a single variable while **facet_grid** can take two variables.

Create a single row of plots based on one variable: facet_wrap()

```
# A row of plots based on one variable: year
ggplot(nmmaps, aes(date,temp))+geom_point(color="aquamarine4", size=0.8)+facet_wrap(~year, nrow=1)
```



Create a matrix of plots based on one variable: facet_wrap(nrow=2)

```
ggplot(nmmaps, aes(date, temp)) + geom_point(color="aquamarine4", size=0.5) + facet_wrap(~year, nrow=2)
```

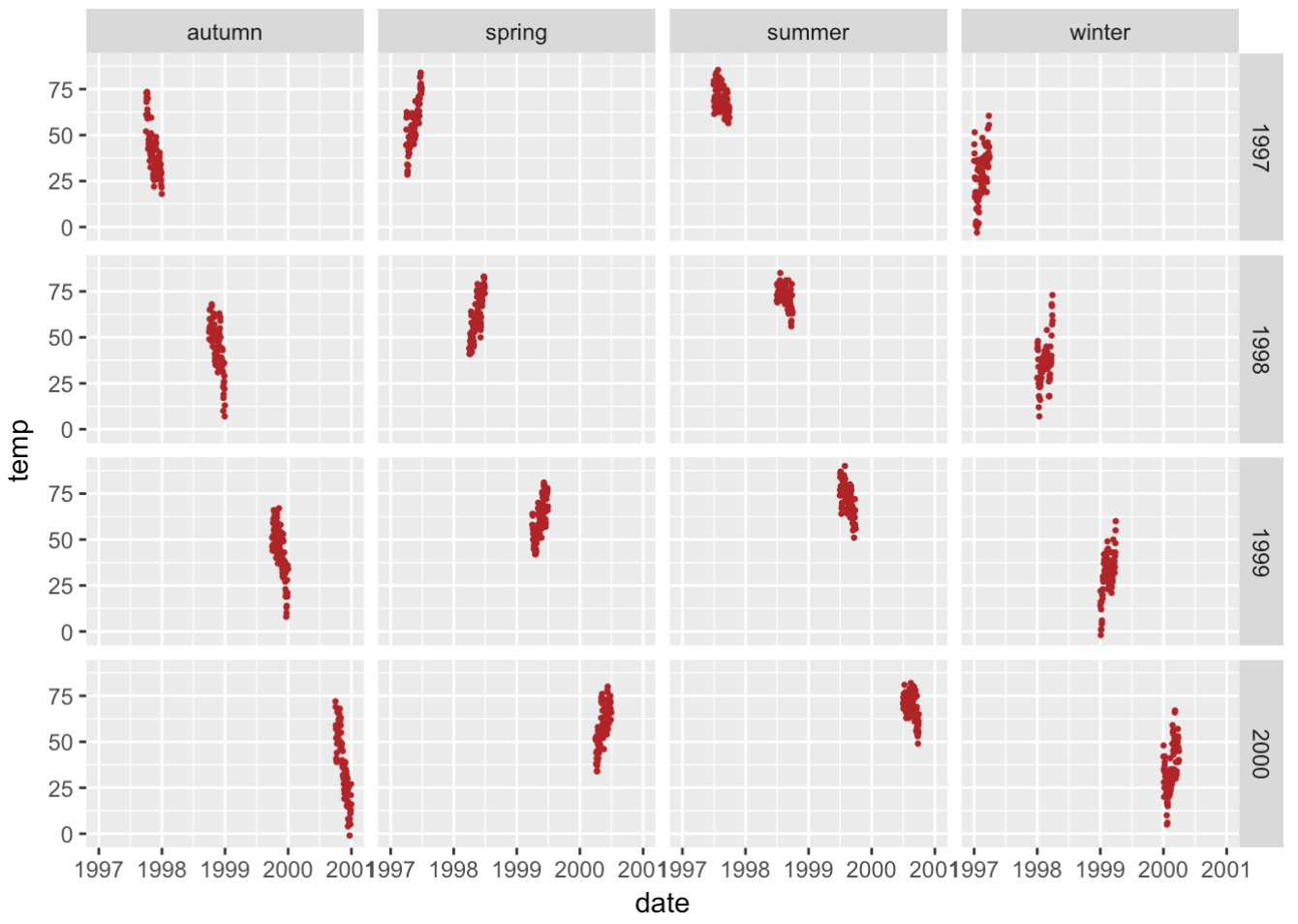
Allow scales to roam free: scales

The default for multi-panel plots in ggplot2 is to use equivalent scales in each panel. But sometimes you want to allow a panel's own data to determine the scale. This is not often a good idea since it may give your user the wrong impression about the data but to do this you can set scales="free" like this:

```
ggplot(nmmaps, aes(date, temp)) + geom_point(color="aquamarine4", size=0.5) + facet_wrap(~year, ncol=2, scales="f
ree")
```

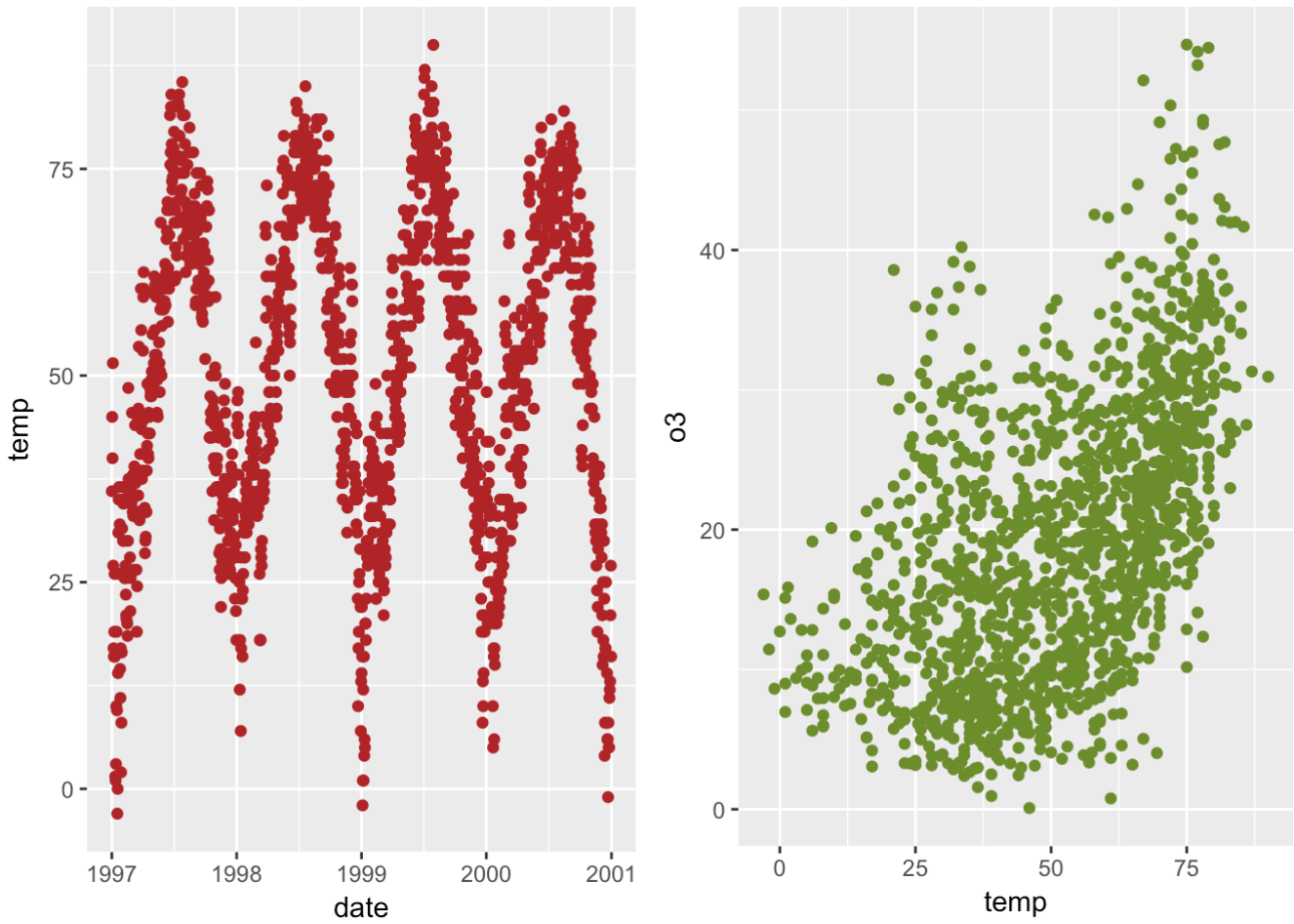
Create a grid of plots using two variables: facet_grid()

```
ggplot(nmmaps, aes(date, temp)) + geom_point(color="firebrick", size=0.5) + facet_grid(year~season)
```

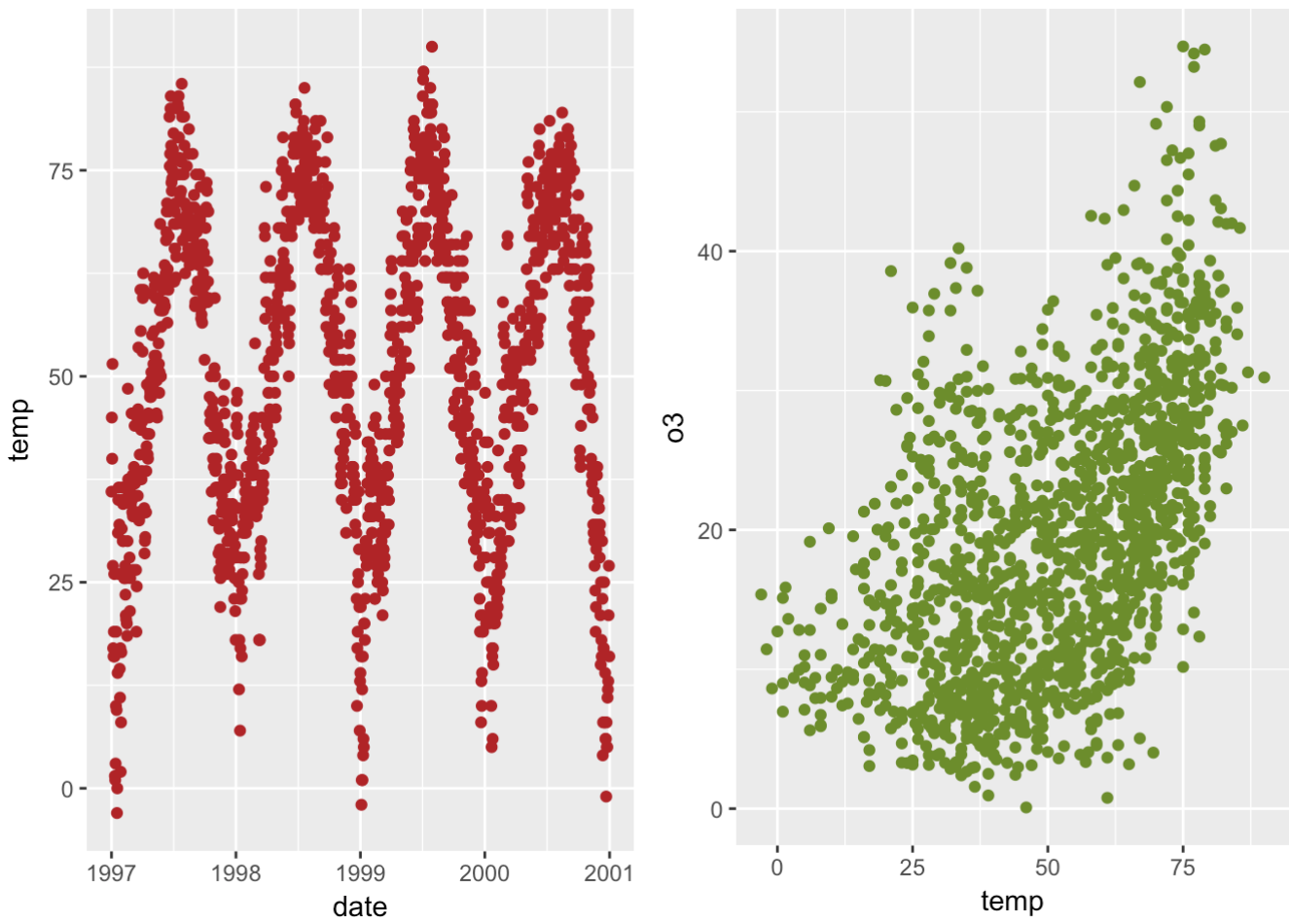


Put two potentially unrelated plots side by side: pushViewport(), grid.arrange()

```
myplot1<-ggplot(nmmaps, aes(date, temp))+geom_point(color="firebrick")
myplot2<-ggplot(nmmaps, aes(temp, o3))+geom_point(color="olivedrab")

library(grid)
pushViewport(viewport(layout = grid.layout(1,2)))
print(myplot1, vp = viewport(layout.pos.row = 1, layout.pos.col = 1))
print(myplot2, vp = viewport(layout.pos.row = 1, layout.pos.col = 2))
```



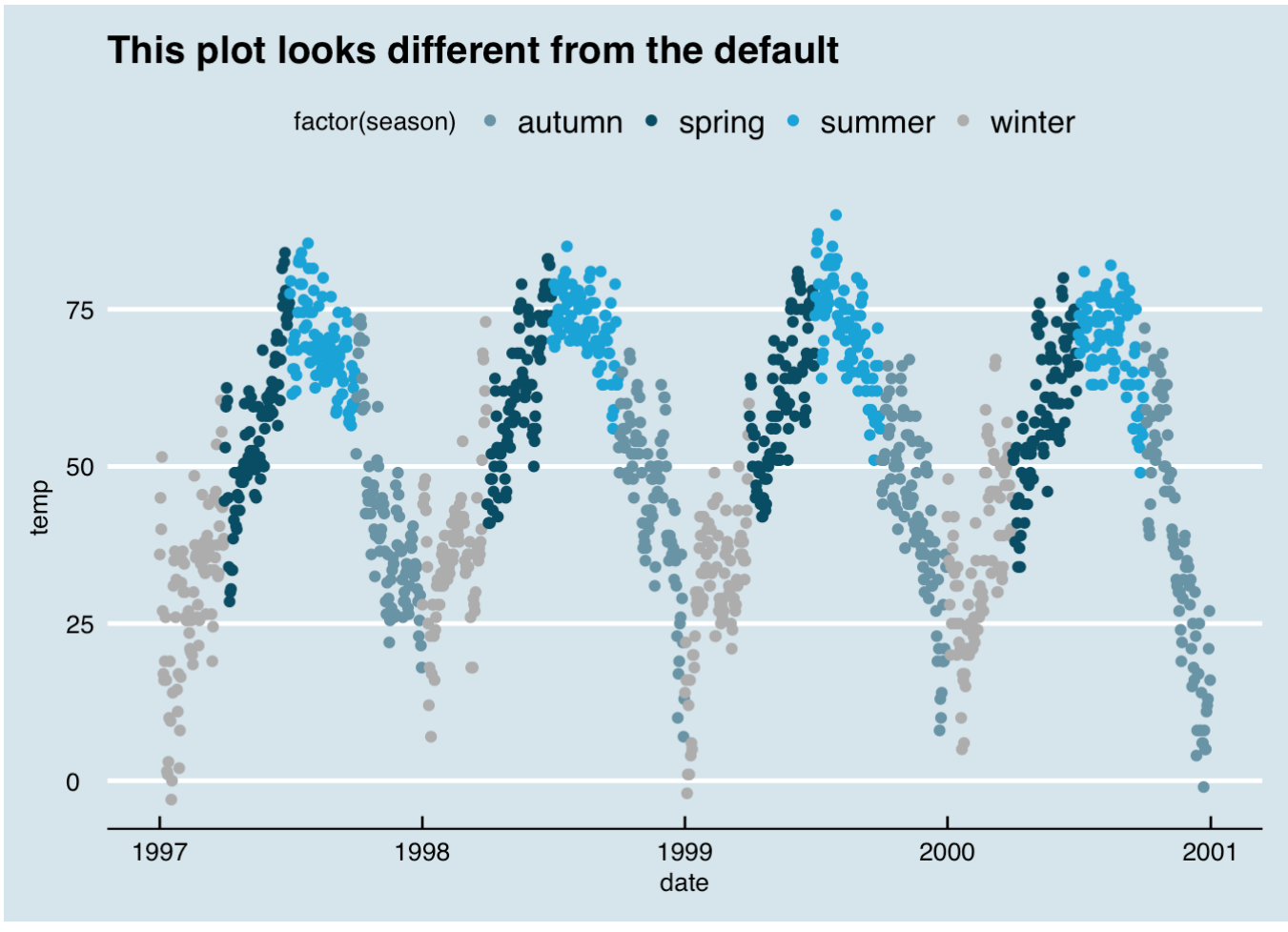

```
# alternatively, a little easier
library(gridExtra)
grid.arrange(myplot1, myplot2, ncol=2)
```



Working with themes

Use a new theme: load ggthemes, theme_xx()

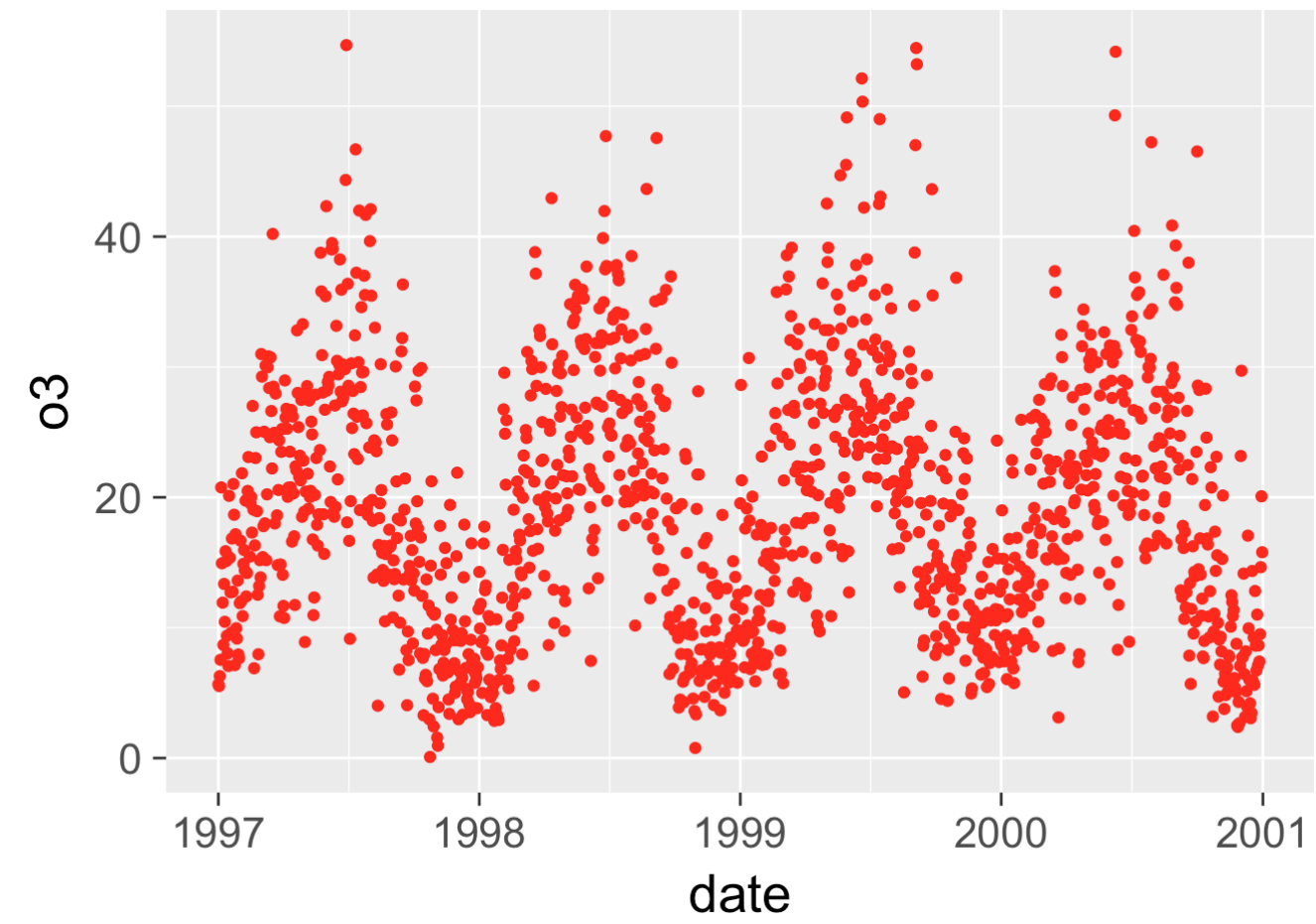
```
library(ggthemes)
ggplot(nmmaps, aes(date, temp, color=factor(season))) + geom_point() + ggtitle("This plot looks different from th
e default") + theme_economist() + scale_colour_economist()
```



Change the size of all plot text elements: theme_set()

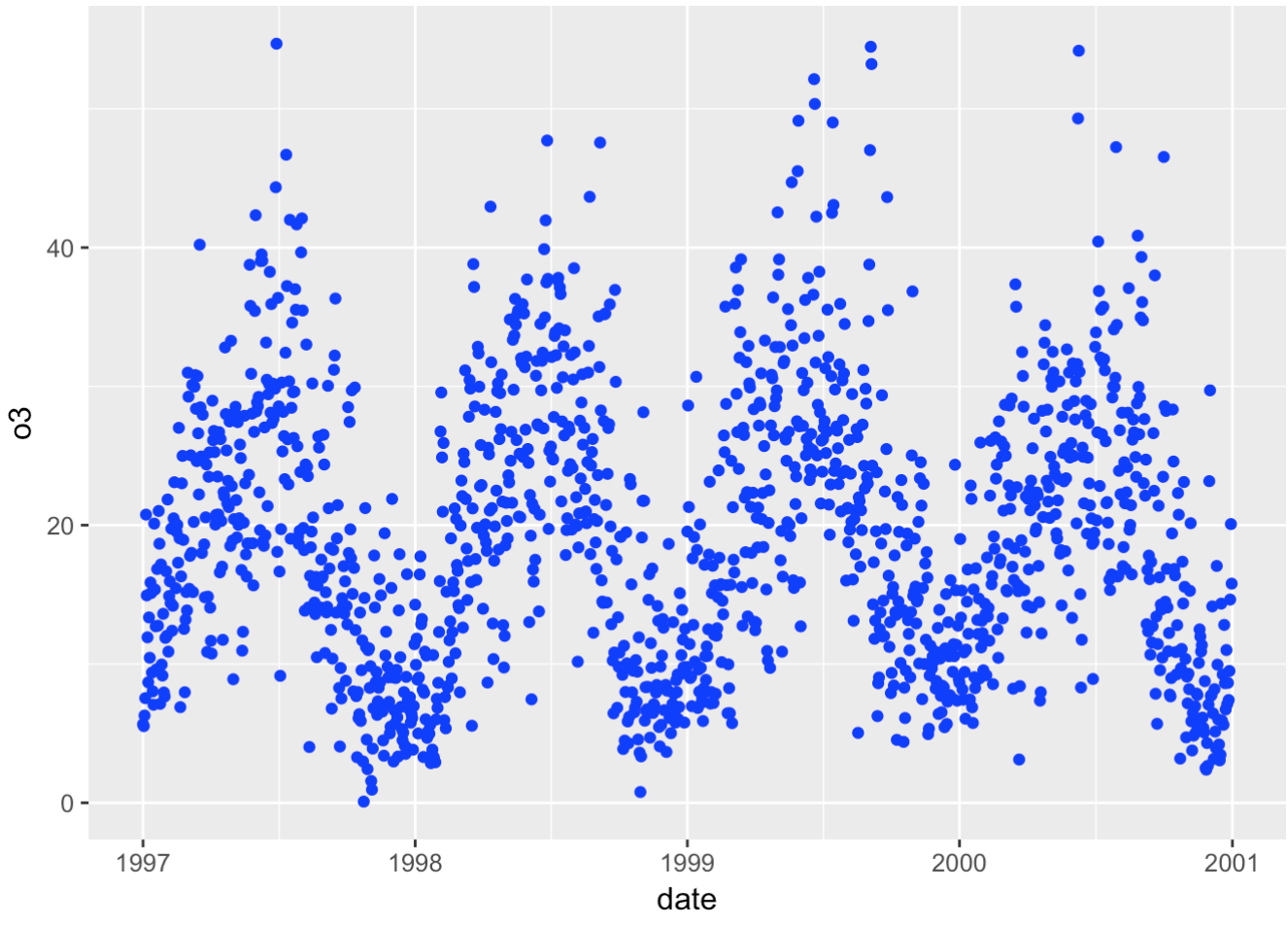
Personally, I find default size of the tick text, legends and other elements to be a little too small. Luckily it's incredibly easy to change the size of all the text elements at once. If you look below at the section on creating a custom theme you'll notice that the sizes of all the elements are relative (rel()) to the base_size. As a result, you can simply change the base_size and you're done.

```
theme_set(theme_gray(base_size=20))
ggplot(nmmaps, aes(x=date, y=o3)) + geom_point(color="red")
```



Changing the default font size to something more sensible

```
theme_set(theme_gray(base_size=12))
ggplot(nmmaps, aes(x=date, y=o3)) + geom_point(color="blue")
```



Tips on creating a custom theme

If you want to change the theme for an entire session you can use theme_set as in theme_set(theme_bw()). The default is called theme_gray. If you wanted to create your own custom theme, you could extract the code directly from the gray theme and modify. Type the following into the console:

```
theme_gray
```

```
## function (base_size = 11, base_family = "")
## {
##     half_line <- base_size/2
##     theme(line = element_line(colour = "black", size = 0.5, linetype = 1,
##         lineend = "butt"), rect = element_rect(fill = "white",
##         colour = "black", size = 0.5, linetype = 1), text = element_text(family = base_family,
##         face = "plain", colour = "black", size = base_size, lineheight = 0.9,
##         hjust = 0.5, vjust = 0.5, angle = 0, margin = margin(),
##         debug = FALSE), axis.line = element_blank(), axis.line.x = NULL,
##         axis.line.y = NULL, axis.text = element_text(size = rel(0.8),
##             colour = "grey30"), axis.text.x = element_text(margin = margin(t = 0.8 *
##             half_line/2), vjust = 1), axis.text.x.top = element_text(margin = margin(b = 0.8 *
##             half_line/2), vjust = 0), axis.text.y = element_text(margin = margin(r = 0.8 *
##             half_line/2), hjust = 1), axis.text.y.right = element_text(margin = margin(l = 0.8 *
##             half_line/2), hjust = 0), axis.ticks = element_line(colour = "grey20"),
##         axis.ticks.length = unit(half_line/2, "pt"), axis.title.x = element_text(margin = margin(t = half_lin
e),
##             vjust = 1), axis.title.x.top = element_text(margin = margin(b = half_line),
##             vjust = 0), axis.title.y = element_text(angle = 90,
##             margin = margin(r = half_line), vjust = 1), axis.title.y.right = element_text(angle = -90,
##             margin = margin(l = half_line), vjust = 0), legend.background = element_rect(colour = NA),
##         legend.spacing = unit(0.4, "cm"), legend.spacing.x = NULL,
##         legend.spacing.y = NULL, legend.margin = margin(0.2,
##             0.2, 0.2, "cm"), legend.key = element_rect(fill = "grey95",
##             colour = "white"), legend.key.size = unit(1.2, "lines"),
##         legend.key.height = NULL, legend.key.width = NULL, legend.text = element_text(size = rel(0.8)),
##         legend.text.align = NULL, legend.title = element_text(hjust = 0),
##         legend.title.align = NULL, legend.position = "right",
##         legend.direction = NULL, legend.justification = "center",
##         legend.box = NULL, legend.box.margin = margin(0, 0, 0,
##             0, "cm"), legend.box.background = element_blank(),
##         legend.box.spacing = unit(0.4, "cm"), panel.background = element_rect(fill = "grey92",
##             colour = NA), panel.border = element_blank(), panel.grid.major = element_line(colour = "white"),
##         panel.grid.minor = element_line(colour = "white", size = 0.25),
##         panel.spacing = unit(half_line, "pt"), panel.spacing.x = NULL,
##         panel.spacing.y = NULL, panel.ontop = FALSE, strip.background = element_rect(fill = "grey85",
##             colour = NA), strip.text = element_text(colour = "grey10",
##             size = rel(0.8)), strip.text.x = element_text(margin = margin(t = half_line,
##             b = half_line)), strip.text.y = element_text(angle = -90,
##             margin = margin(l = half_line, r = half_line)), strip.placement = "inside",
##         strip.placement.x = NULL, strip.placement.y = NULL, strip.switch.pad.grid = unit(0.1,
##             "cm"), strip.switch.pad.wrap = unit(0.1, "cm"), plot.background = element_rect(colour = "white"),
##         plot.title = element_text(size = rel(1.2), hjust = 0,
##             vjust = 1, margin = margin(b = half_line * 1.2)),
##         plot.subtitle = element_text(size = rel(0.9), hjust = 0,
##             vjust = 1, margin = margin(b = half_line * 0.9)),
##         plot.caption = element_text(size = rel(0.9), hjust = 1,
##             vjust = 1, margin = margin(t = half_line * 0.9)),
##         plot.margin = margin(half_line, half_line, half_line,
##             half_line), complete = TRUE)
## }
## <environment: namespace:ggplot2>
```

And we can copy + edit values according to the aesthetic needs of our theme.

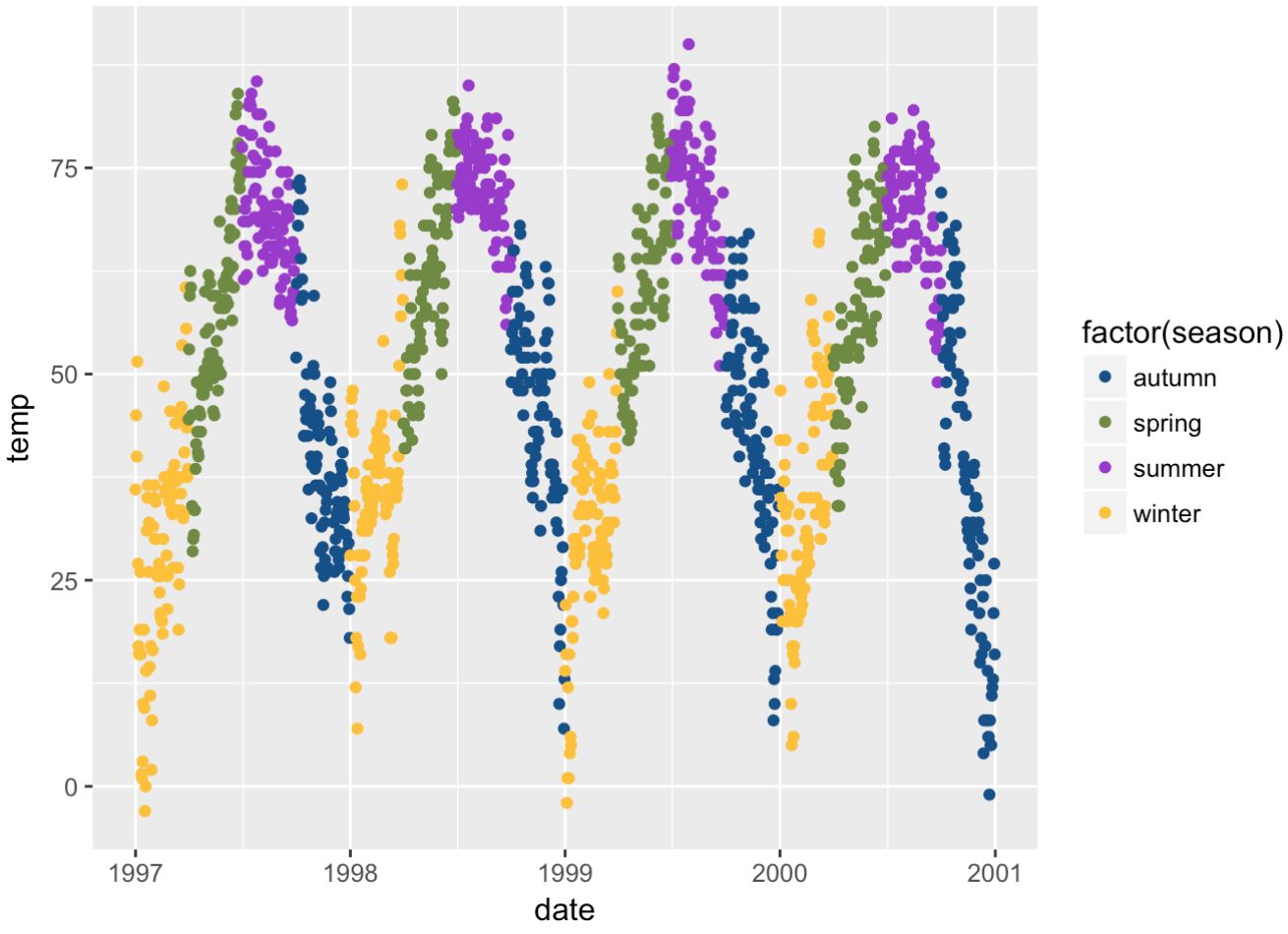
Working with colors

For simple applications working with colors is straightforward in ggplot2 but when you have more advanced needs it can be a challenge. For a more advaned treatment of the topic you should probably get your hands on Hadley's book (http://www.springer.com/statistics/computational+statistics/book/978-0-387-98140-6) which has nice coverage. There are a few other good sources including the R Cookbook (http://www.cookbook-r.com/Graphs/Colors_(ggplot2)) and the ggplot2 online docs (http://docs.ggplot2.org/current/). Tian Zheng at Columbia has created a useful PDF of R colors (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf).

In order to use color with your data, most importantly, you need to know if you're dealing with a categorical or continuous variable.

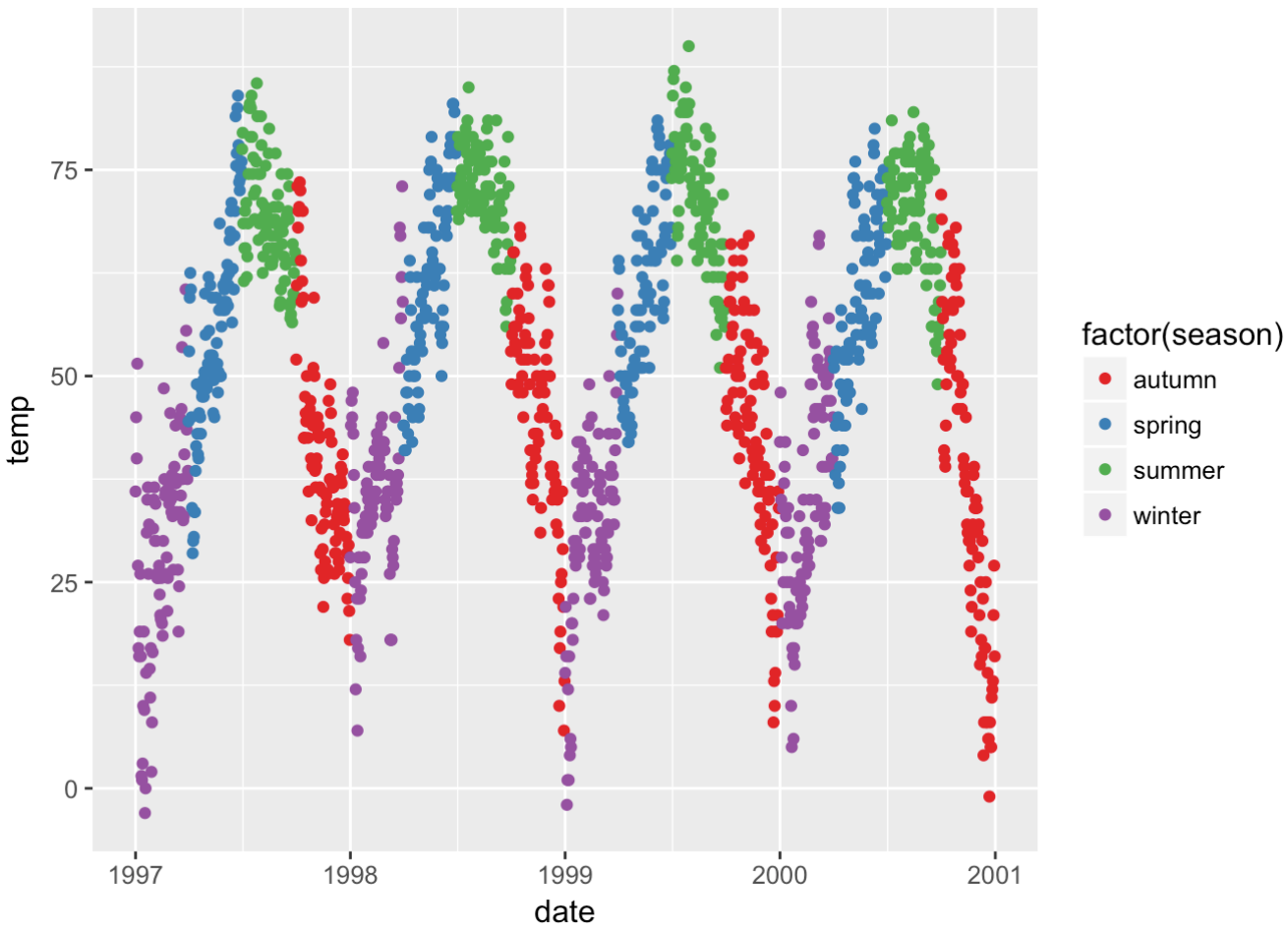
Categorial variables - manually select the colors: scale_color_manual()

```
ggplot(nmmaps, aes(date, temp, color=factor(season))) + geom_point() + scale_color_manual(values = c("dodgerblue
4", "darkolivegreen4",
                              "darkorchid3", "goldenrod1"))
```



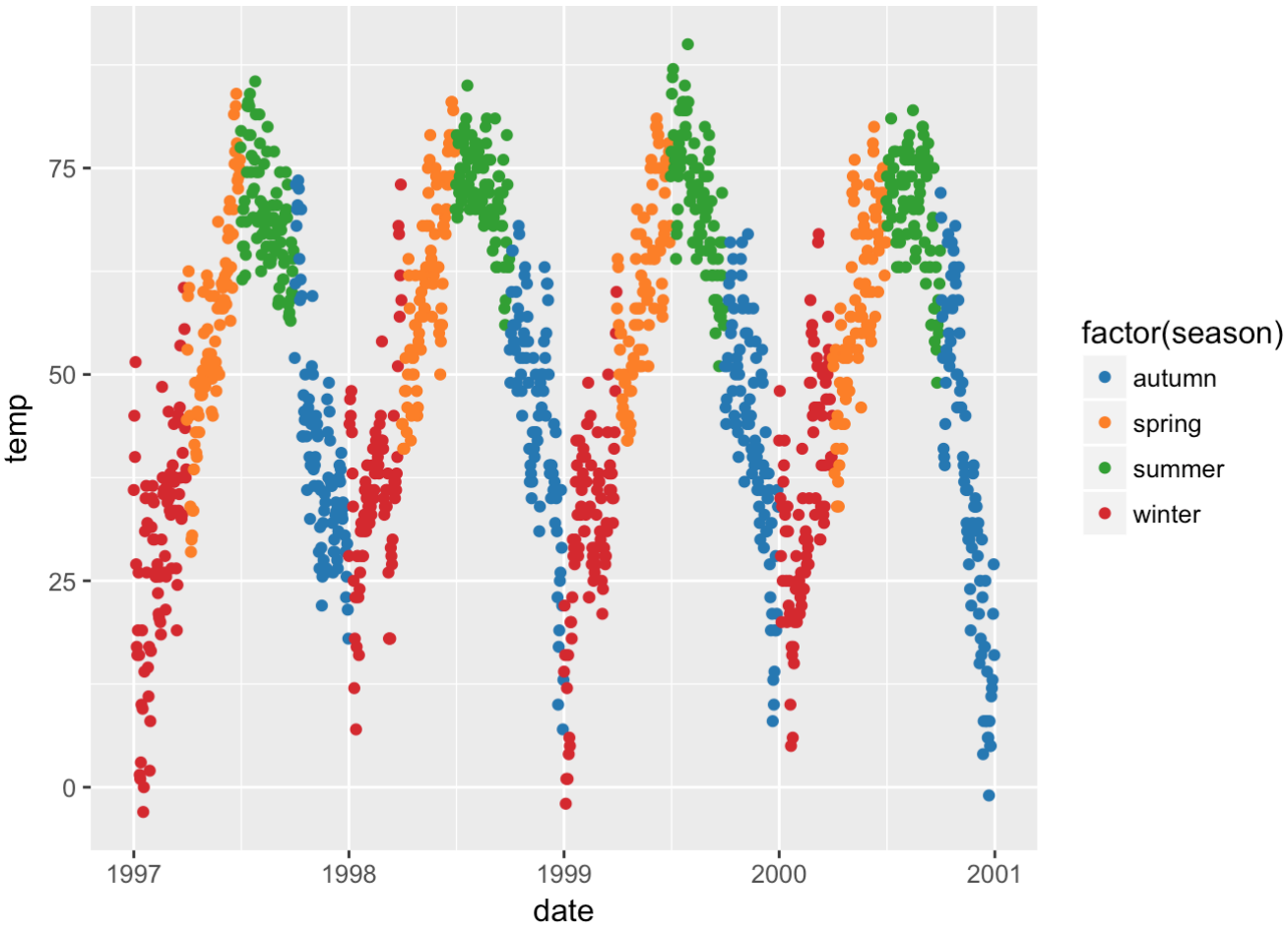
Categorial variables - using a built-in palette: scale_color_brewer()

```
ggplot(nmmaps, aes(date, temp, color=factor(season))) + geom_point() + scale_color_brewer(palette="Set1")
```



We can also use the Tableau colors if we have ggthemes: scale_color_tableau

```
library(ggthemes)
ggplot(nmmaps, aes(x=date, y=temp, color=factor(season)))+geom_point() + scale_color_tableau()
```

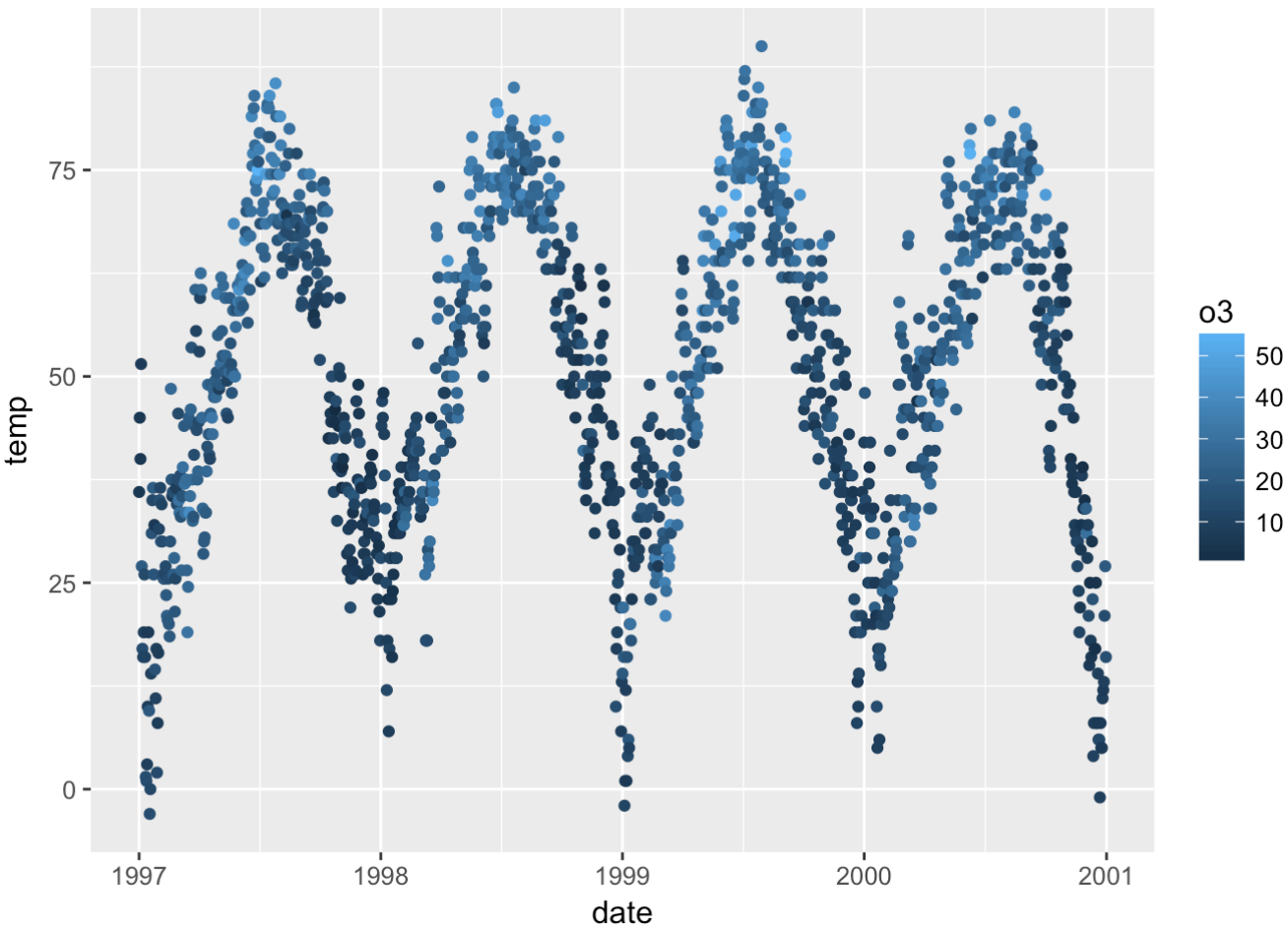


Color choice with continuous variables: scale_color_gradient(), scale_color_gradient2()

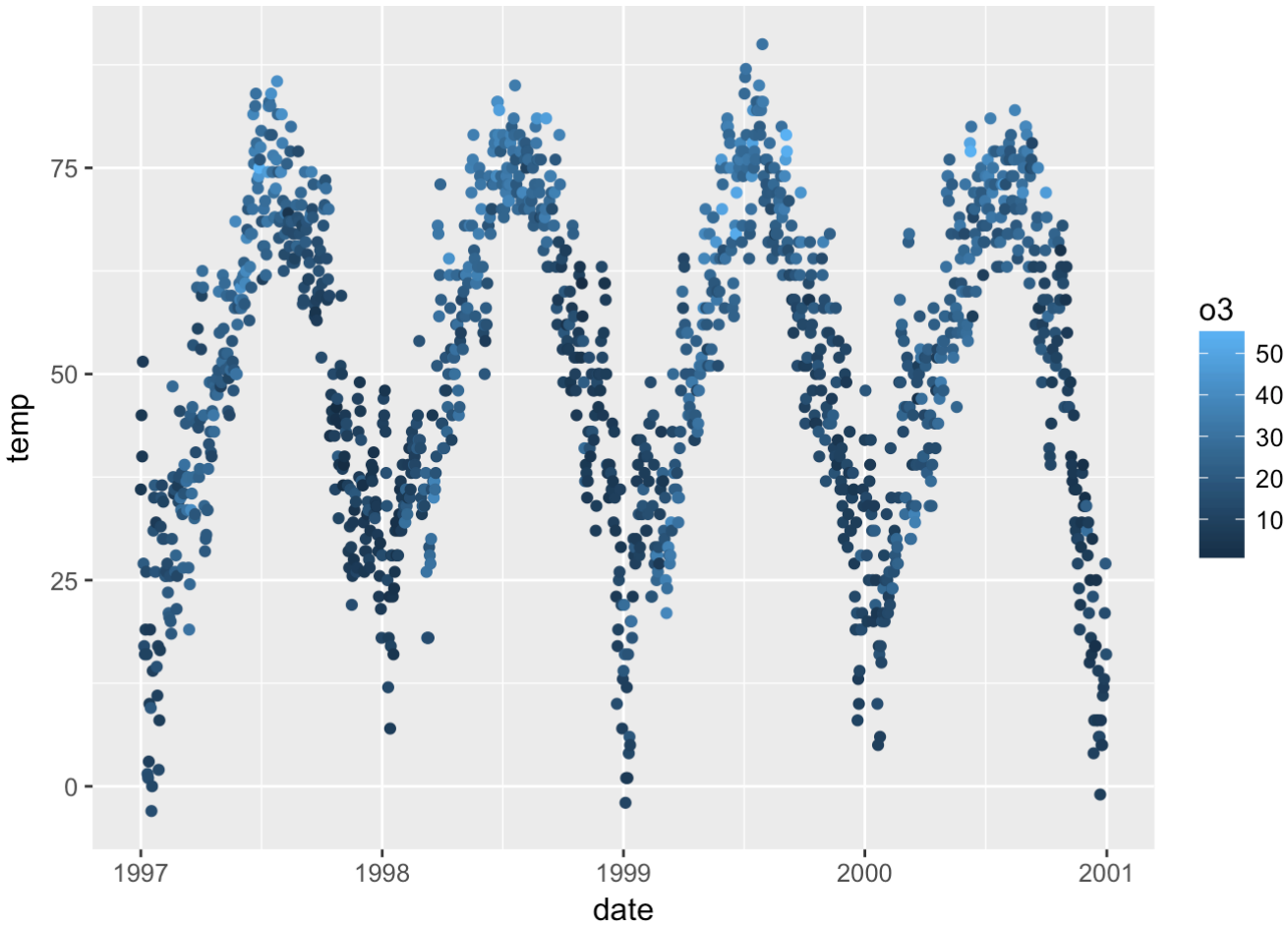
In our example we will change the color variable to ozone (o3), a continuous variable that is strongly related to temperature (higher temperature = higher ozone). The function scale_color_gradient() is a sequential gradient while scale_color_gradient2() is diverging.

Here is a default continuous color scheme (sequential color scheme):

```
ggplot(nmmaps, aes(x=date, y=temp, color=o3))+geom_point()
```

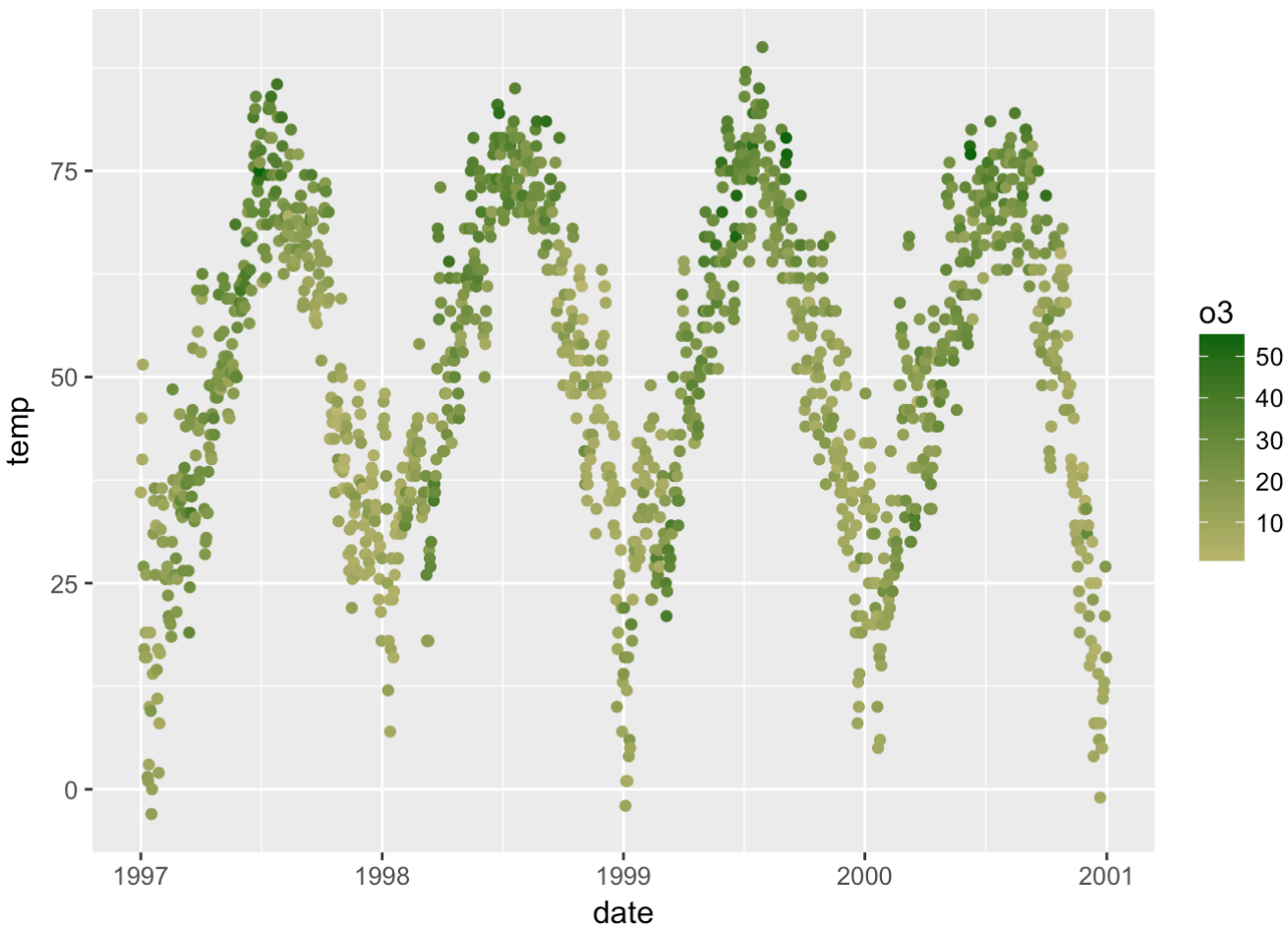



```
# this code produces an identical plot
ggplot(nmmaps, aes(date, temp, color=o3))+geom_point()+scale_color_gradient()
```



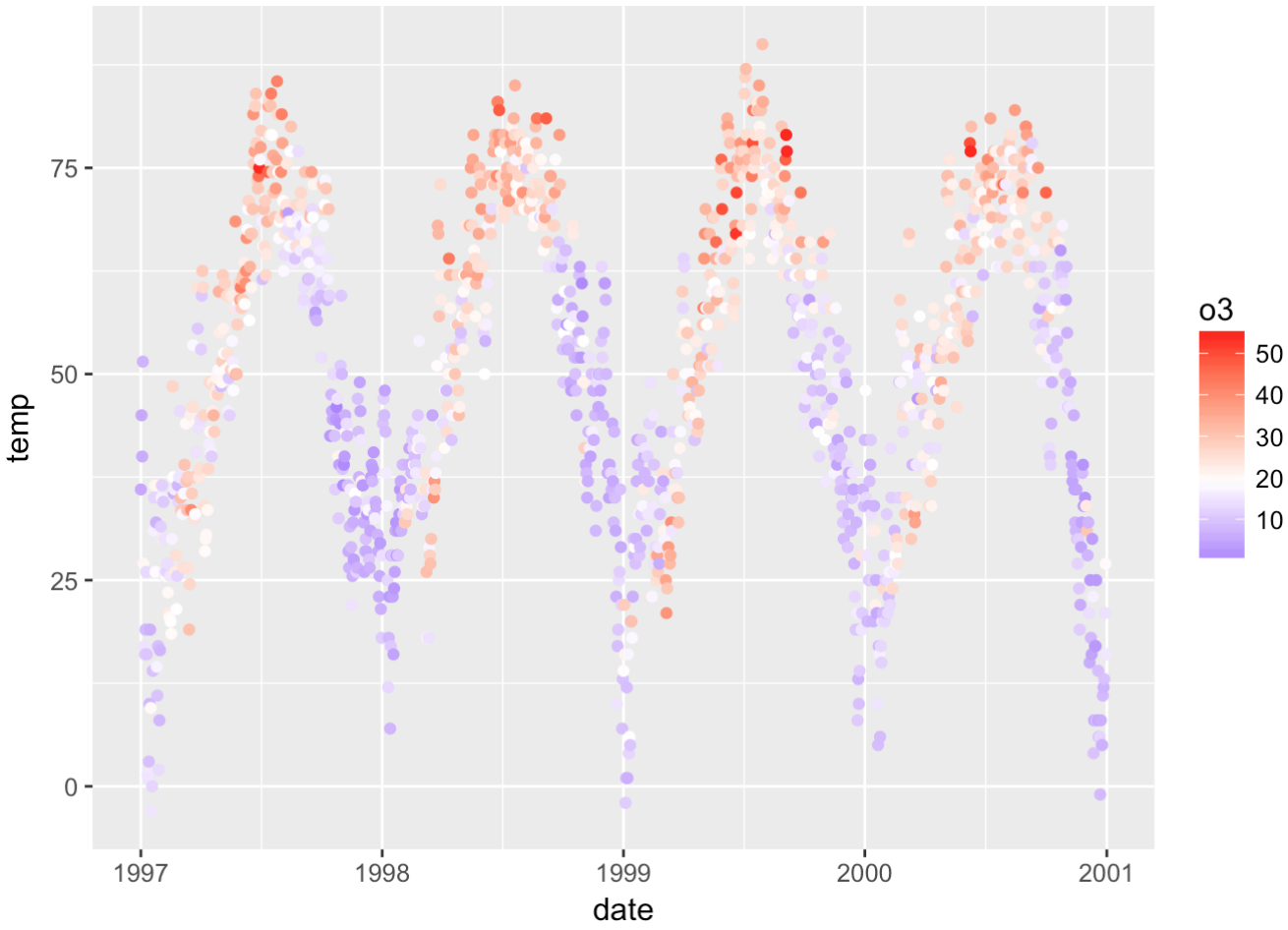
To manually change the low and high colors (sequential color scheme):

```
ggplot(nmmaps, aes(x=date, y=temp, color=o3)) + geom_point() + scale_color_gradient(low = "darkkhaki", high= "dar
kgreen")
```



The temperature data is normally distributed so how about a diverging color scheme (rather than sequential). For diverging color we can use the scale_color_gradient2 function with a midpoint parameter.

```
mid <- mean(nmmaps$o3)
ggplot(nmmaps, aes(date, temp, color=o3))+geom_point()+scale_color_gradient2(midpoint = mid, low="blue", mid="whi
te", high="red")
```

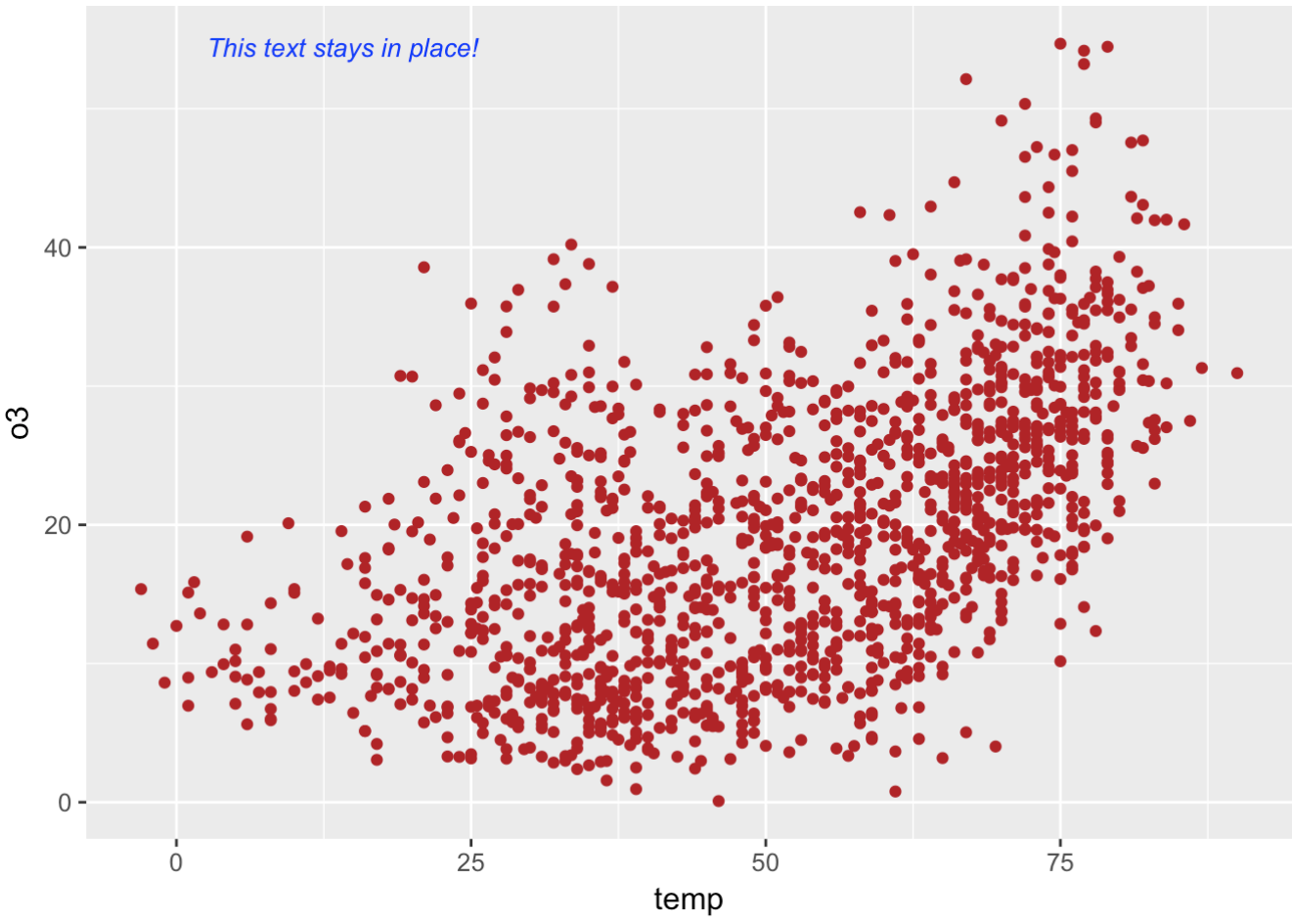


Working with annotation

Add text annotation in the top-right, top-left etc: annotation_custom(), textGrob()

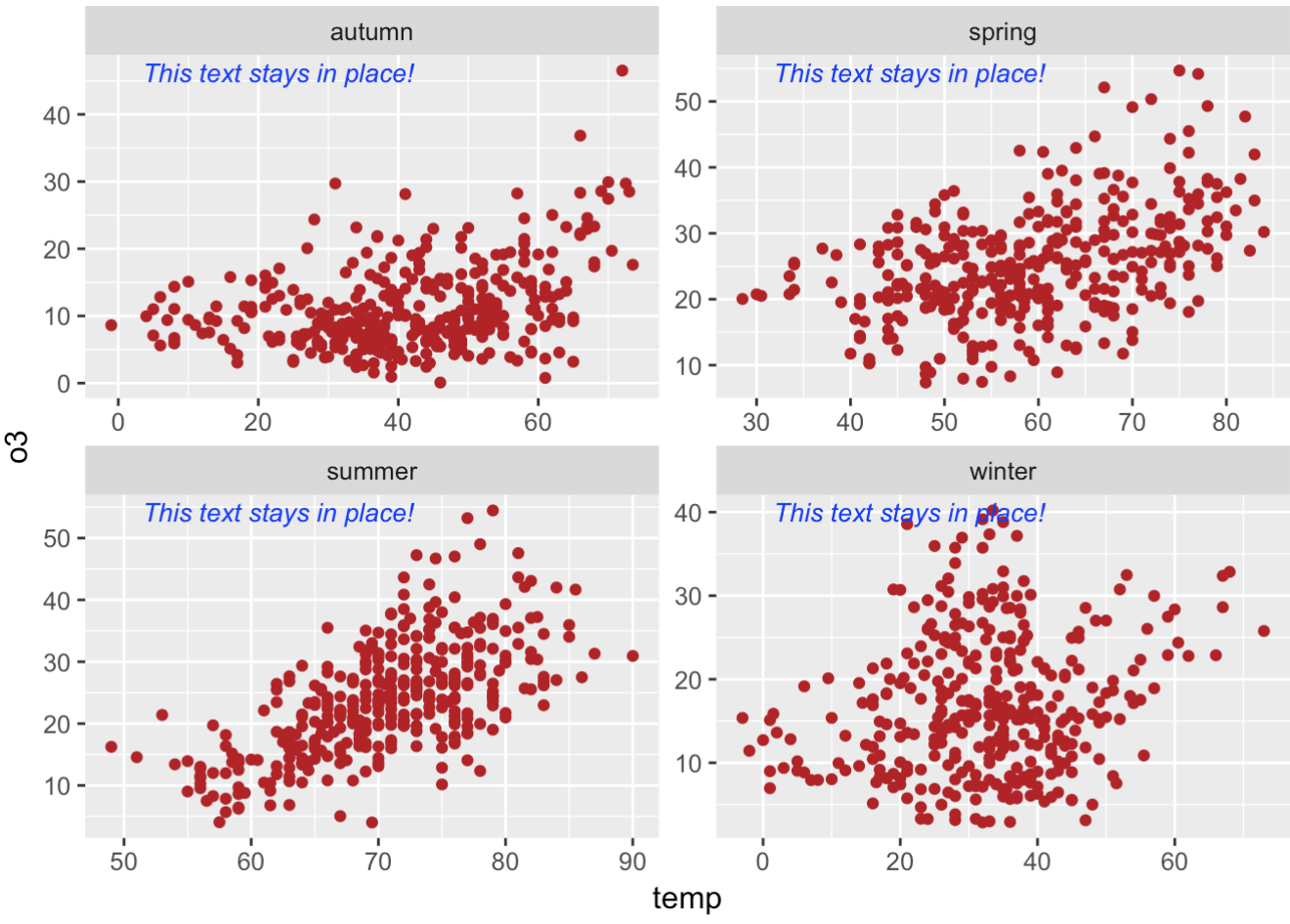
The grobTree function (from grid) creates a grid graphical object and textGrob creates the text graphical object. The annotation_custom() function comes from ggplot2 and is designed to use a grob as input.

```
library(grid)

my_grob <- grobTree(textGrob("This text stays in place!", x=0.1, y=0.95, hjust=0, gp = gpar(col="blue",
fontsize=10, fontface="italic")))

ggplot(nmmaps, aes(temp, o3)) + geom_point(color="firebrick")+annotation_custom(my_grob)
```



'Big deal' you say!? It is a big deal. The value here is particularly evident when you have multiple plots with different scales. In the plot below you see that the axis scales vary yet the same code as above can be used to put the annotation is the same place on each facet. Nice!

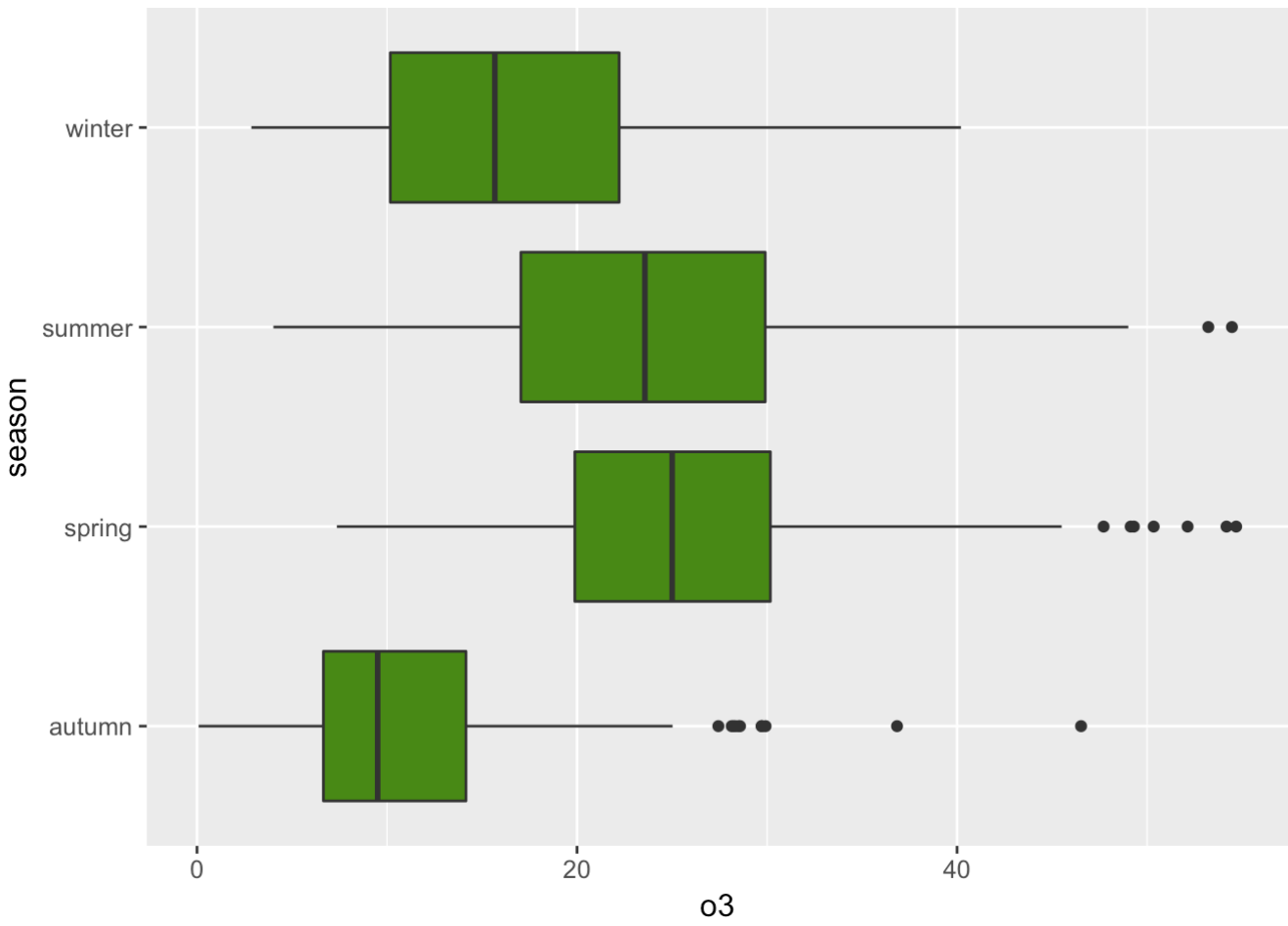
```
library(grid)

my_grob <- grobTree(textGrob("This text stays in place!", x=0.1, y=0.95, hjust=0, gp = gpar(col="blue",
fontsize=10, fontface="italic")))

ggplot(nmmaps, aes(temp, o3)) + geom_point(color="firebrick") + facet_wrap(~season, scales="free") + annotation_c
ustom(my_grob)
```



Working with coordinates

Flip a plot on its side: coord_flip()

```
ggplot(nmmaps, aes(x=season, y=o3))+geom_boxplot(fill="chartreuse4")+coord_flip()
```

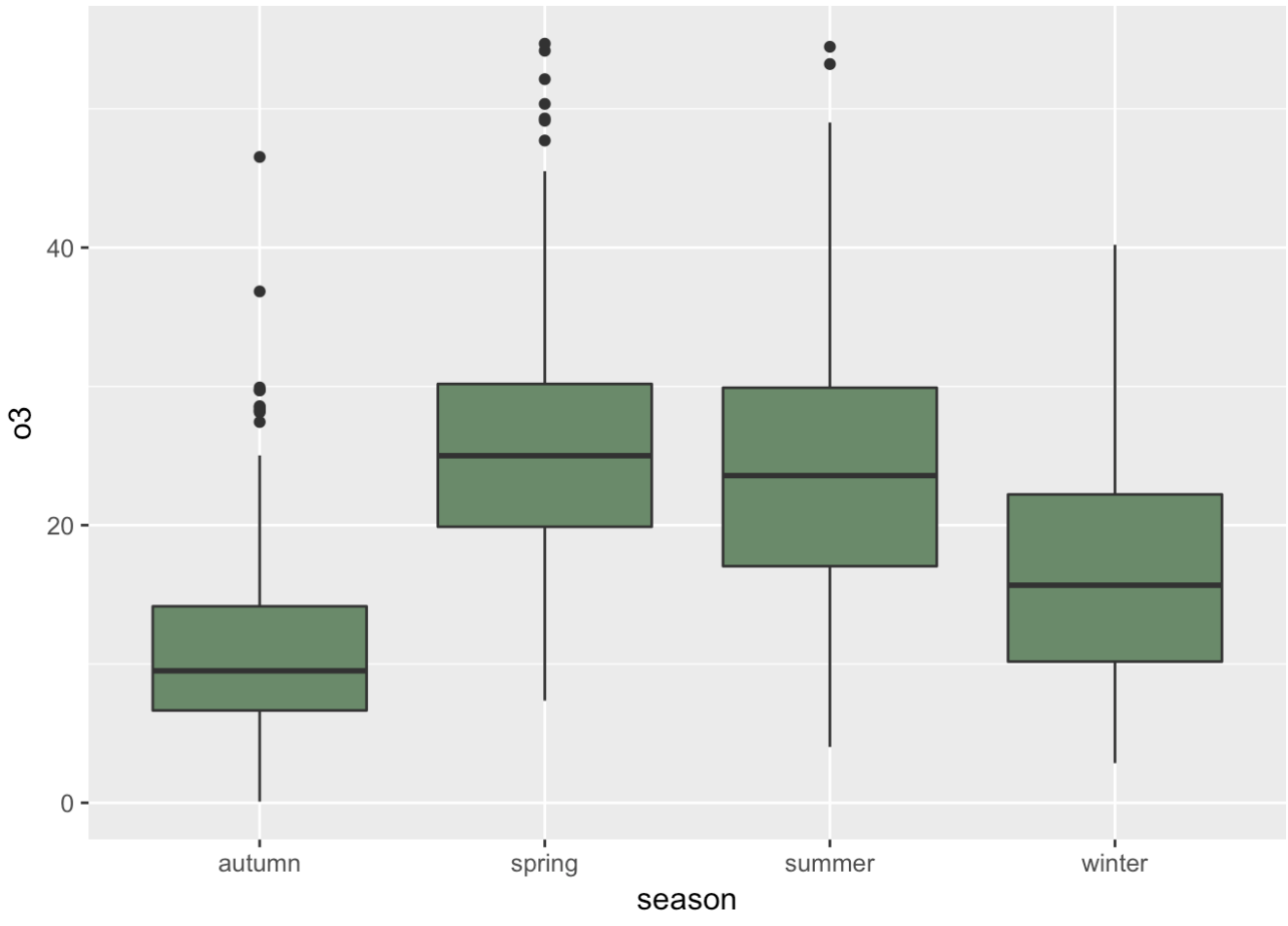


Working with plottypes

Alternatives to geom_point: geom_jitter()

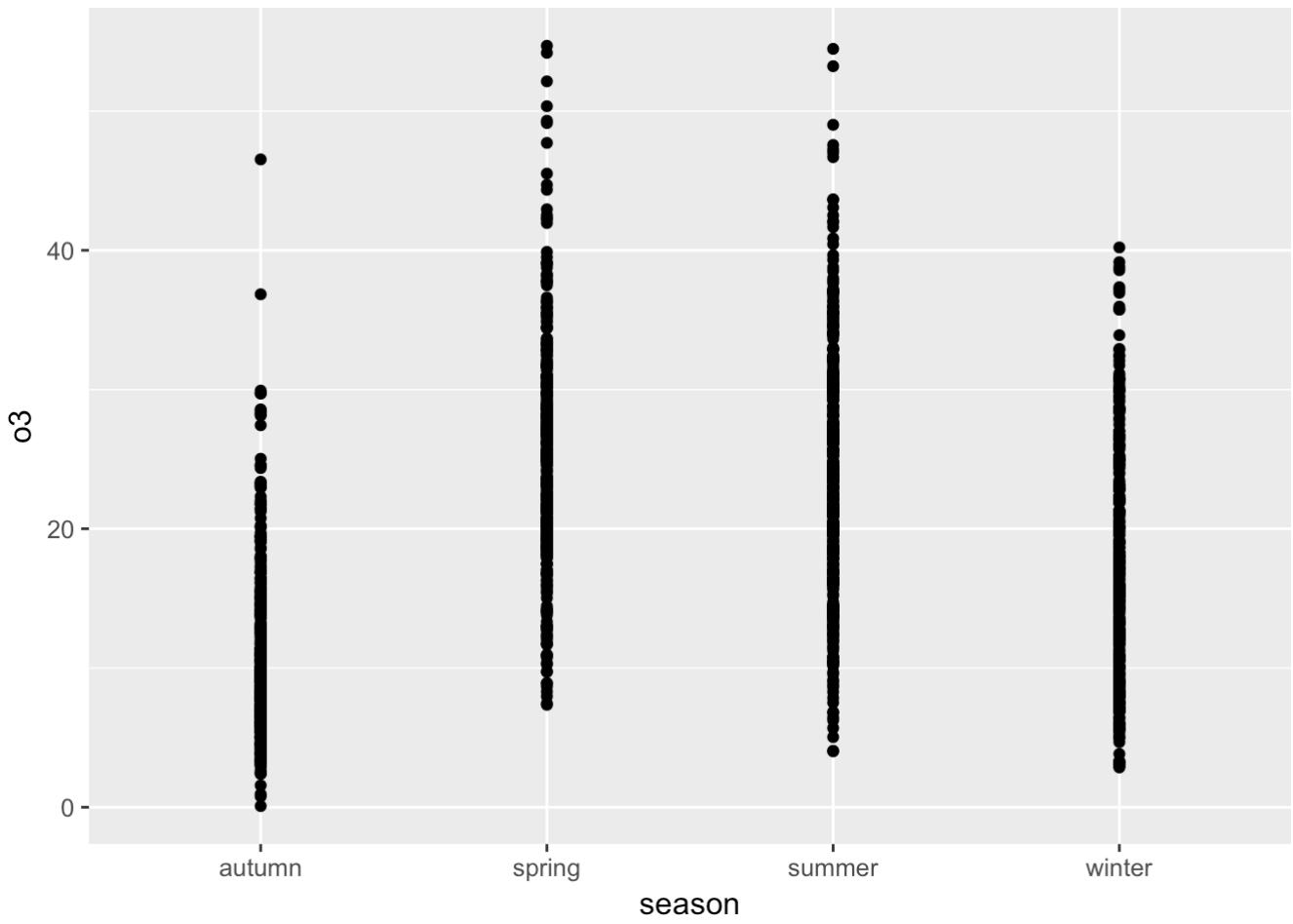
Box plots are great, but they can be so incredibly boring. There are alternatives, first – a box plot:

```
g<-ggplot(nmmaps, aes(x=season, y=o3))
g+geom_boxplot(fill="darkseagreen4")
```



Effective, yes. Interesting, no. What if we plot the points themselves?

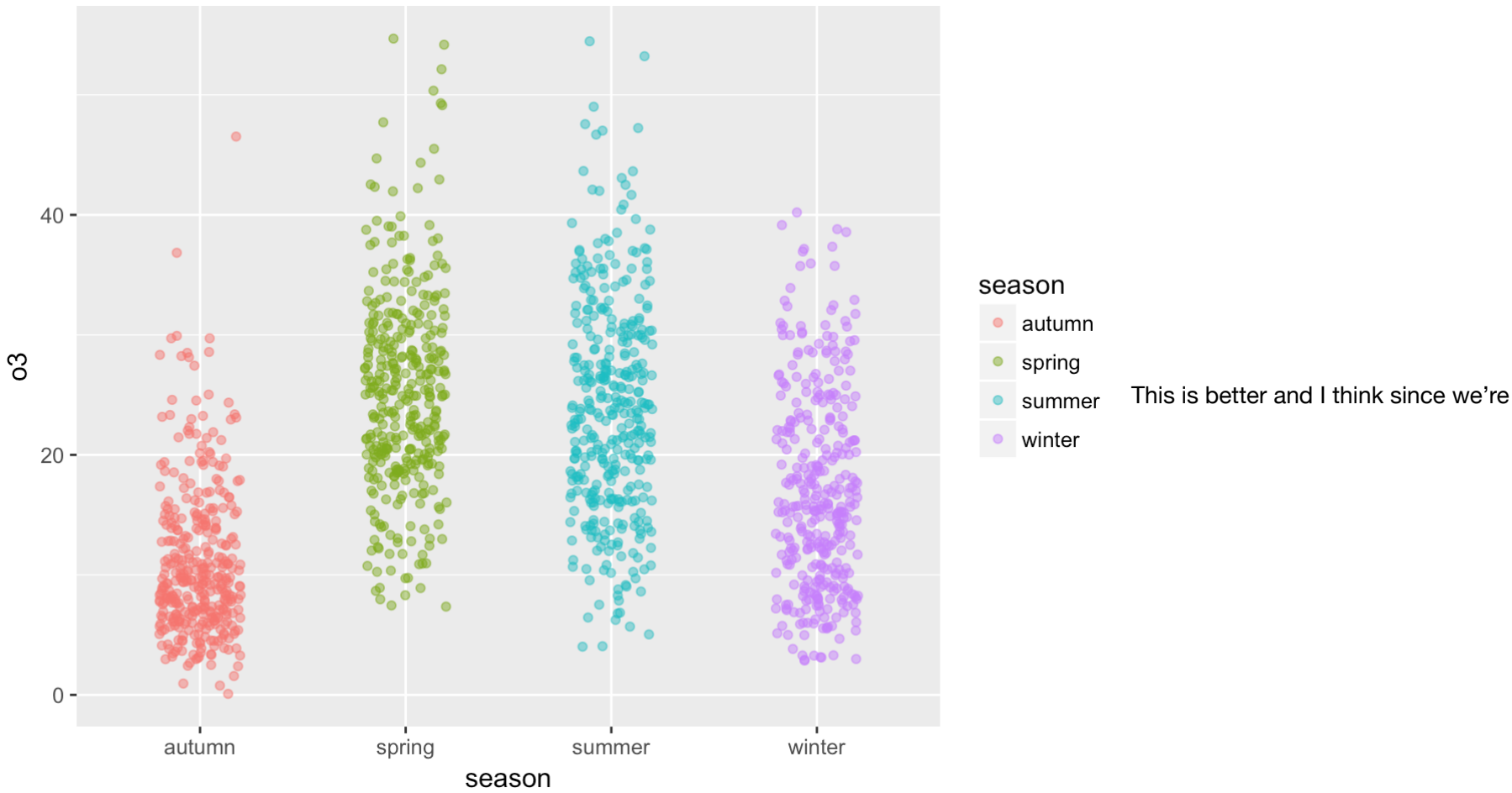
```
g+geom_point()
```



Not only boring but uninformative, you could add transparency to deal with overplotting, but this is not good either. Let's try something else.

Try adding a little jitter to the data. I like this for in-house visualization but be careful using jittering because youre purposely adding noise to your data and this can result in misinterpretation of your data.

```
g+geom_jitter(alpha=0.5, aes(color=season), position=position_jitter(width=.2))
```

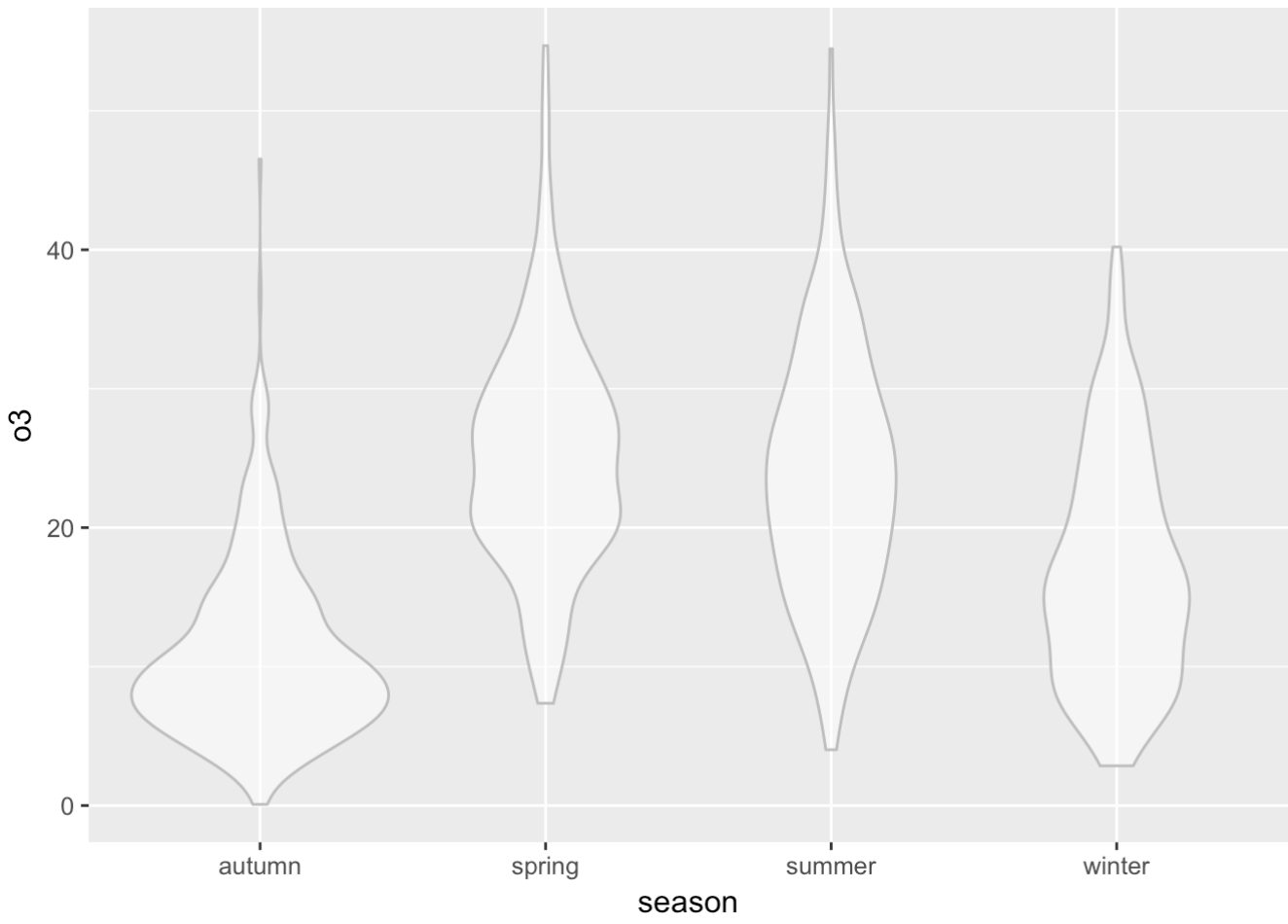


This is better and I think since we're working with season, a variable everyone will be familiar with, the extra noise will not likely lead to confusion.

Alternatives to geom_boxplot: geom_violin()

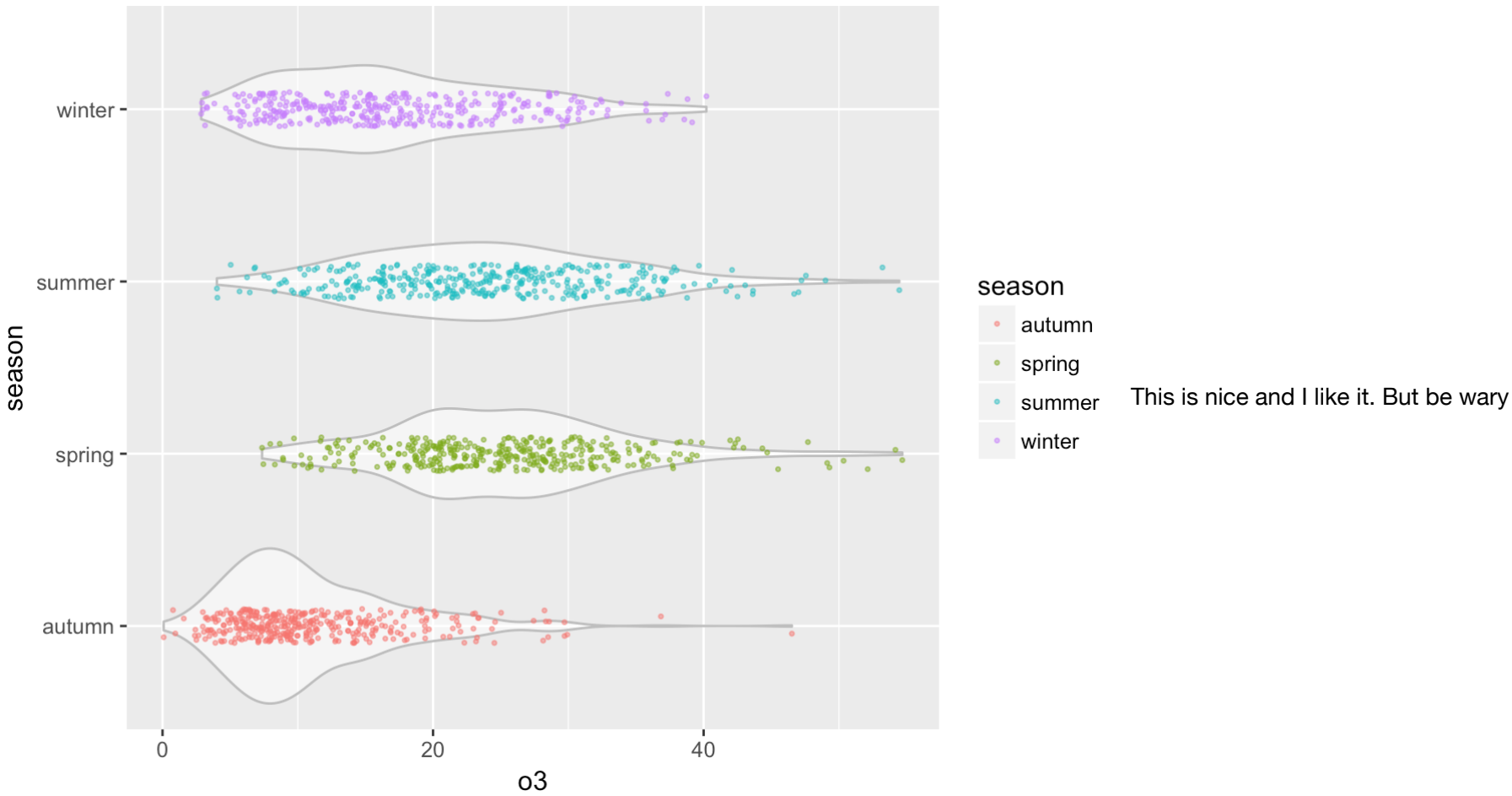
Violin plots, similar to box plots except you're using a kernel density to show where you have the most data, are a useful visualization.

```
g+geom_violin(alpha=0.5, color="gray")
```



What if we rotated and added the jittered points:

```
g+geom_violin(alpha=0.5, color="gray")+geom_jitter(alpha=0.5, aes(color=season), position =
position_jitter(width=0.1), size=0.6)+coord_flip()
```



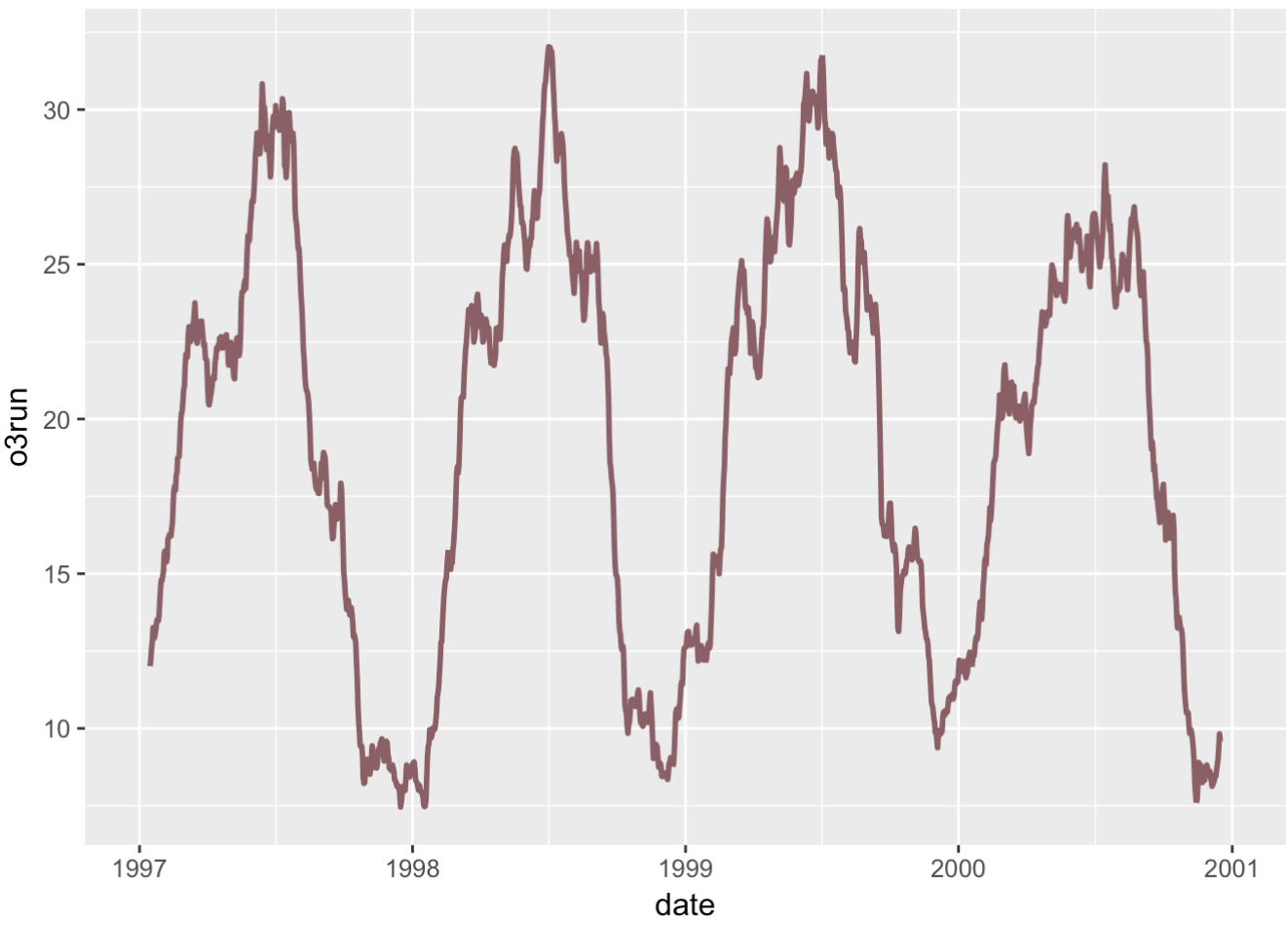
This is nice and I like it. But be wary of using unusual plot types, they take more time for your users to understand. Sometimes the simplest and most conventional plot type is your best bet when sharing with others. Box plots may be boring but people know how to interpret them immediately.

Add a ribbon to your plot: geom_ribbon()

This is not the perfect dataset for this, but using ribbon can be useful. In this example we will create a 30-day running average using the filter() function so that our ribbon is not too noisy.

```
# Add a filter
nmmaps$o3run<-as.numeric(filter(nmmaps$o3, rep(1/30,30), sides=2))

ggplot(nmmaps, aes(date, o3run))+geom_line(color="lightpink4", lwd=1)
```

```
## Warning: Removed 29 rows containing missing values (geom_path).
```

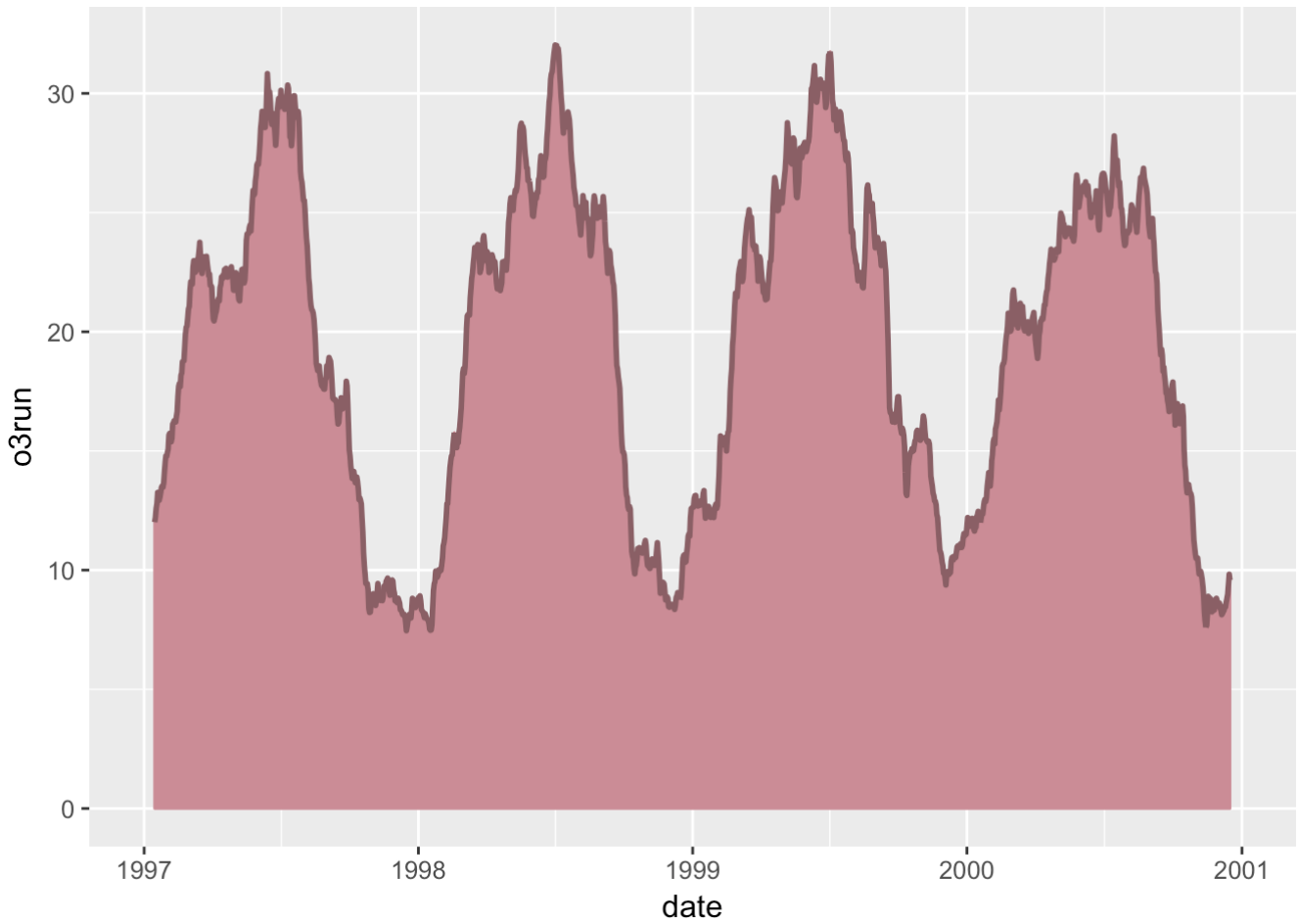


Note: what filter() does with filter size = 30 is explained in more details on this StackOverflow thread (http://stackoverflow.com/questions/14372880/simple-examples-of-filter-function-recursive-option-specifically). Simply put, we create a running average and use both sides of the value (sides=2) instead of only using past values (sides=1). So on the 17th April, the o3run value would be the running average of 1st April to 30th April. If sides=1, it would instead have been the running average of 1st April to 17th April. Sides can only be either 1 or 2. Contrast this to the o3 value of 17th April which is just the value of 17th April alone.

How does it look if we fill in the area below the curve using the geom_ribbon() function?

```
ggplot(nmmaps, aes(date, o3run)) + geom_ribbon(aes(ymin=0, ymax=o3run), fill="lightpink3", color="lightpink3") +
geom_line(color="lightpink4", lwd=1)
```

```
## Warning: Removed 29 rows containing missing values (geom_path).
```



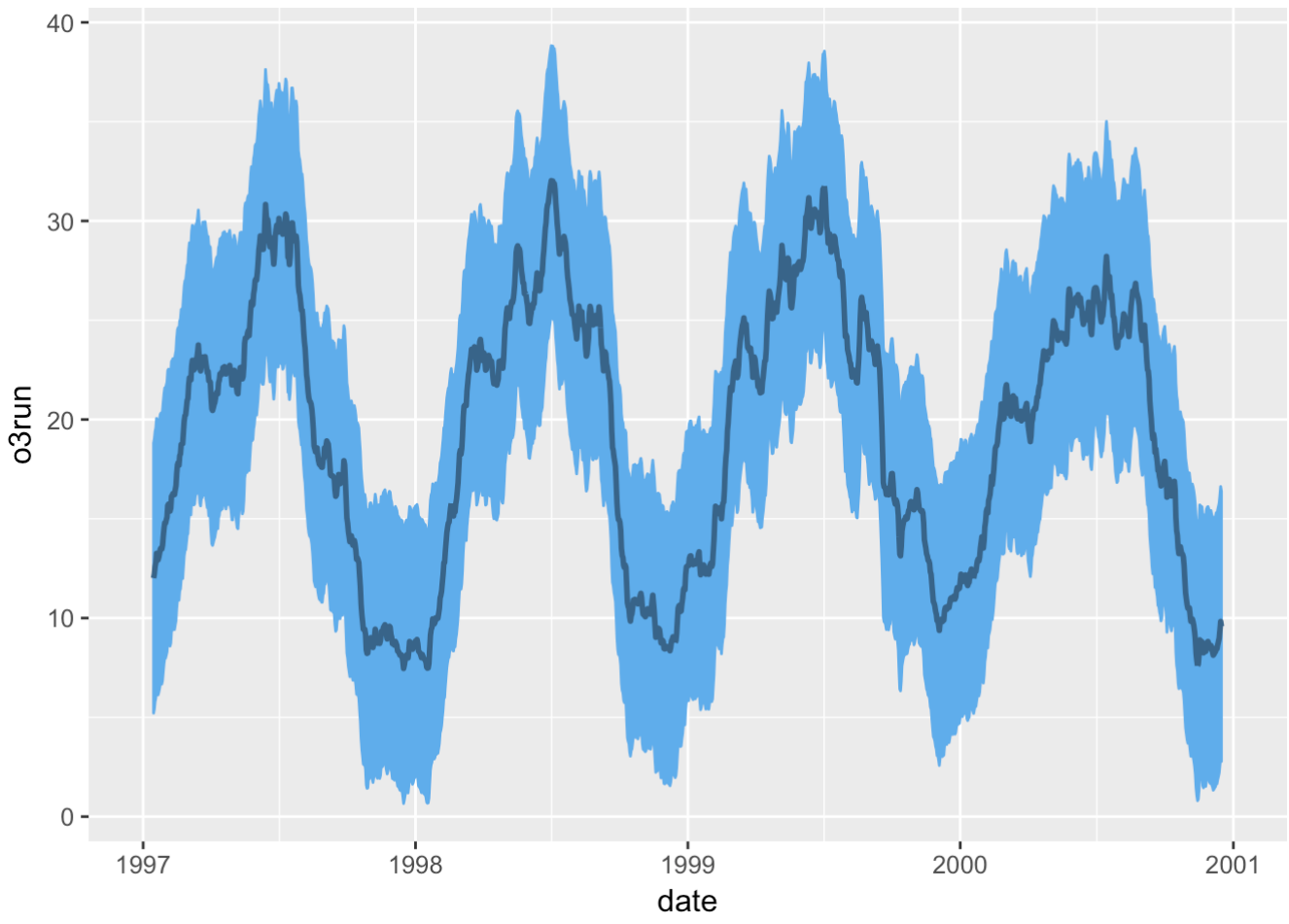
For each x value, geom_ribbon

displays a y interval defined by ymin and ymax. **geom_area** is a special case of geom_ribbon, where the ymin is fixed to 0.

Above is not really the conventional way to use geom_ribbon(), we would have used geom_area instead. Instead, why don't we draw a ribbon that gives us one standard deviation above and below our data:

```
nmmaps$mino3 <- nmmaps$o3run-sd(nmmaps$o3run, na.rm=T)
nmmaps$maxo3 <- nmmaps$o3run+sd(nmmaps$o3run, na.rm=T)

ggplot(nmmaps, aes(x=date, y=o3run))+geom_ribbon(aes(ymin = mino3, ymax = maxo3), fill="steelblue2", color="steel
blue2") + geom_line(color="steelblue4", lwd=1)
```

```
## Warning: Removed 29 rows containing missing values (geom_path).
```



Create a tiled correlation plot: geom_tile()

First step in creating a tiled correlation plot is to create the correlation matrix. We use Pearson because all the variables are fairly normally distributed in our nmmaps dataset. We may want to consider Spearman if our variables follow a different pattern. Note that since a correlation matrix has redundant information we will be setting half of it to NA.

```
# careful! We're sorting the field names so that the ordering in the final plot is correct. We call cor() to crea
te the correlation matrix

thecor <- round(cor(nmmaps[,sort(c("death", "temp", "dewpoint", "pm10", "o3"))],
                    method = "pearson", use = "pairwise.complete.obs"),2)
thecor[lower.tri(thecor)] <- NA
thecor
```

```
##          death dewpoint    o3 pm10  temp
## death        1    -0.47 -0.24 0.00 -0.49
## dewpoint    NA     1.00  0.45 0.33  0.96
## o3          NA       NA  1.00 0.21  0.53
## pm10        NA       NA    NA 1.00  0.37
## temp        NA       NA    NA   NA  1.00
```

Now we will put it in a “long” format using the melt function from the reshape2 package and drop the records with NA values. Putting it in a long format makes it possible to map the aesthetic: aes(Var1, Var2)

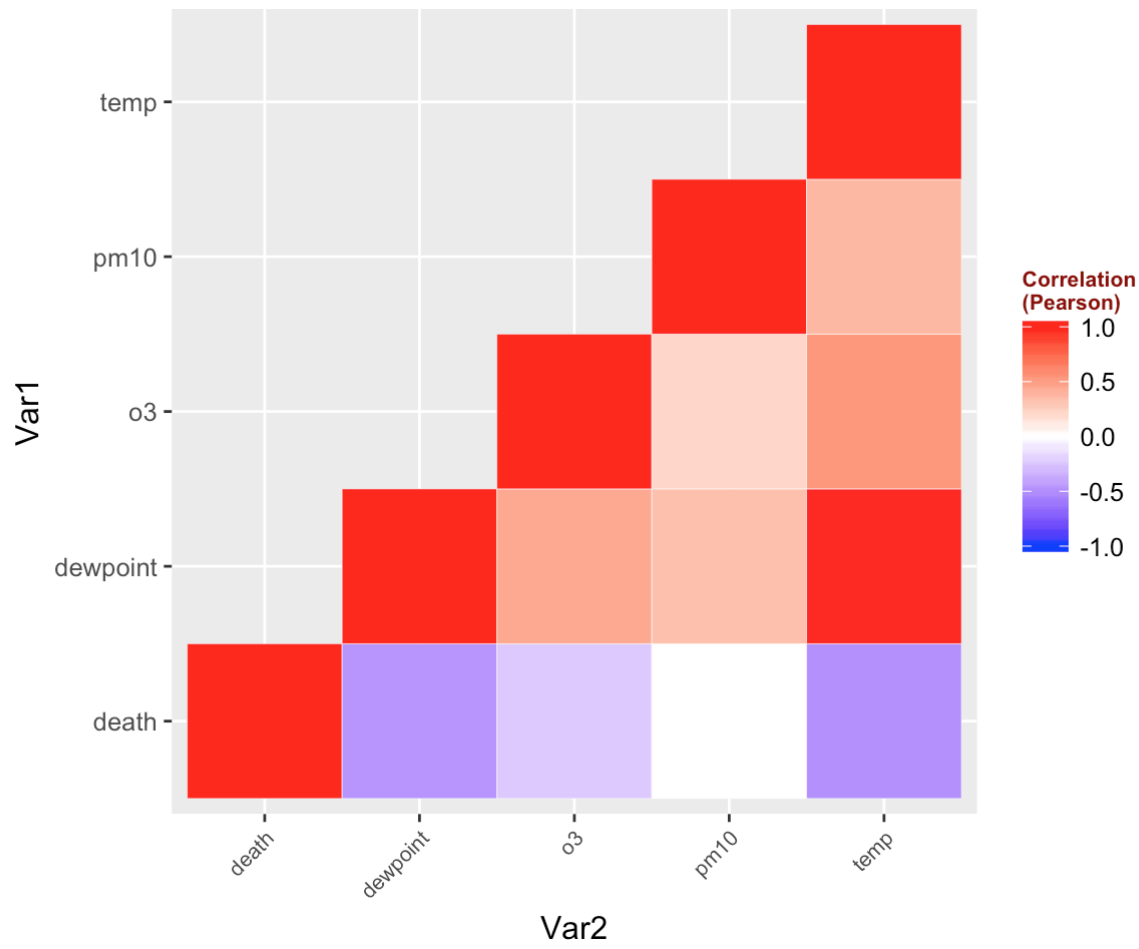
```
library(reshape2)
thecor <- melt(thecor)
thecor$Var1<-as.character(thecor$Var1)
thecor$Var2<-as.character(thecor$Var2)
thecor<-na.omit(thecor)
head(thecor)
```

```
##        Var1     Var2 value
## 1     death    death  1.00
## 6     death dewpoint -0.47
## 7  dewpoint dewpoint  1.00
## 11    death       o3 -0.24
## 12 dewpoint       o3  0.45
## 13       o3       o3  1.00
```

We only printed the head but thecor actually has 25 rows (5 variables ^2)

Now for the plot. We are using geom_tile but if you have a lot of data you might consider geom_raster which can be much faster.

```
ggplot(thecor, aes(Var2, Var1)) + geom_tile(data=thecor, aes(fill=value), color="white") + scale_fill_gradient2(l
ow="blue", high="red", mid="white", midpoint=0, limit=c(-1,1), name="Correlation\n(Pearson)") + theme(axis.text.x
 = element_text(angle=45, vjust=1, size=8, hjust=1), legend.title = element_text(color="darkred", size=8, face="b
old")) + coord_equal()
```



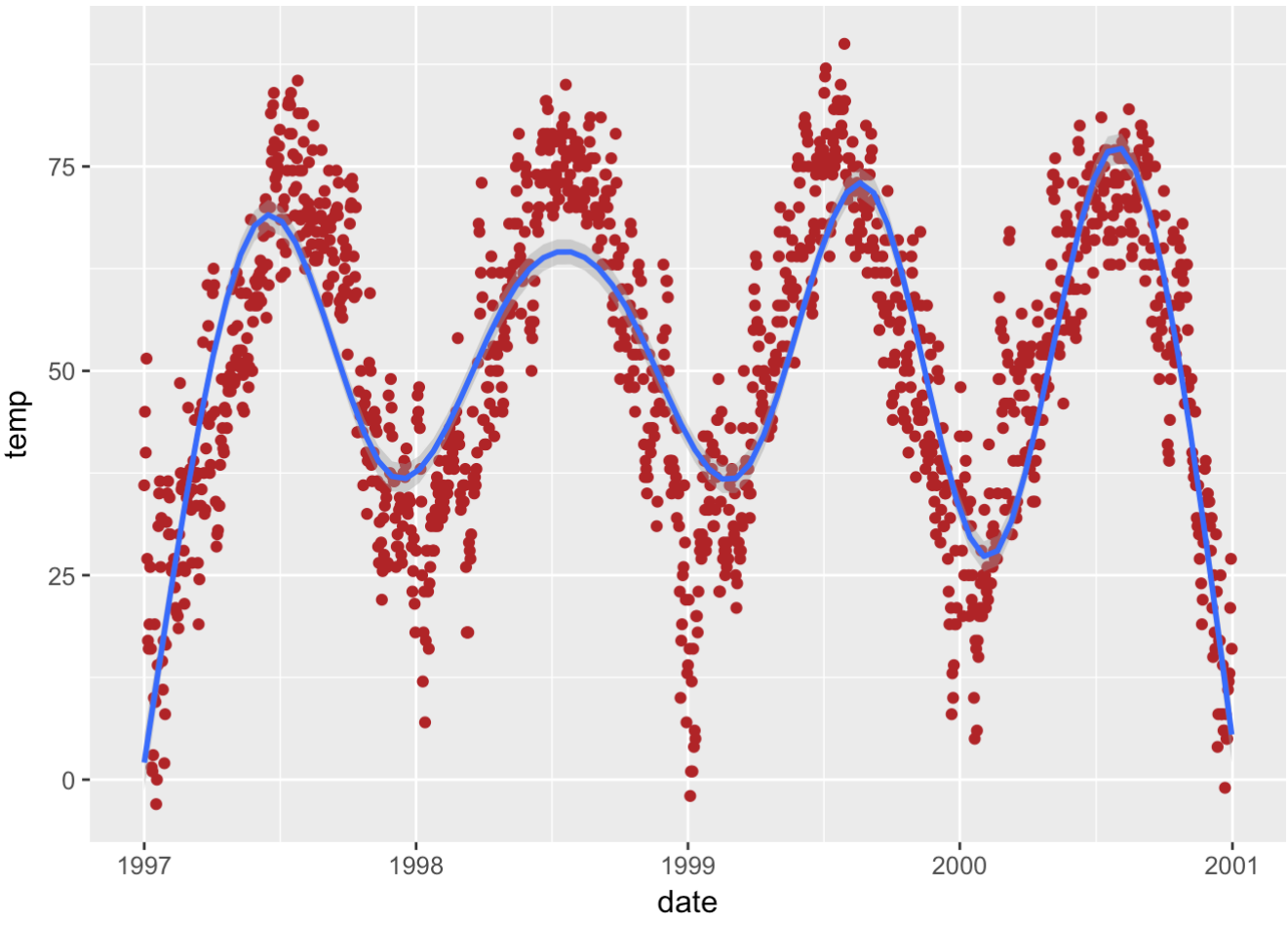
Working with smooths

You’ve likely already learned how amazingly easy it is to add a smooth to your data using ggplot2. You can simply use stat_smooth() which will add a LOESS smooth if you have fewer than 1000 points or a GAM otherwise. Since we have more than 1000 points the smooth is a GAM.

Default - adding LOESS or GAM: stat_smooth()

```
# Without specifying any formula
ggplot(nmmaps, aes(date, temp))+geom_point(color="firebrick")+stat_smooth()
```

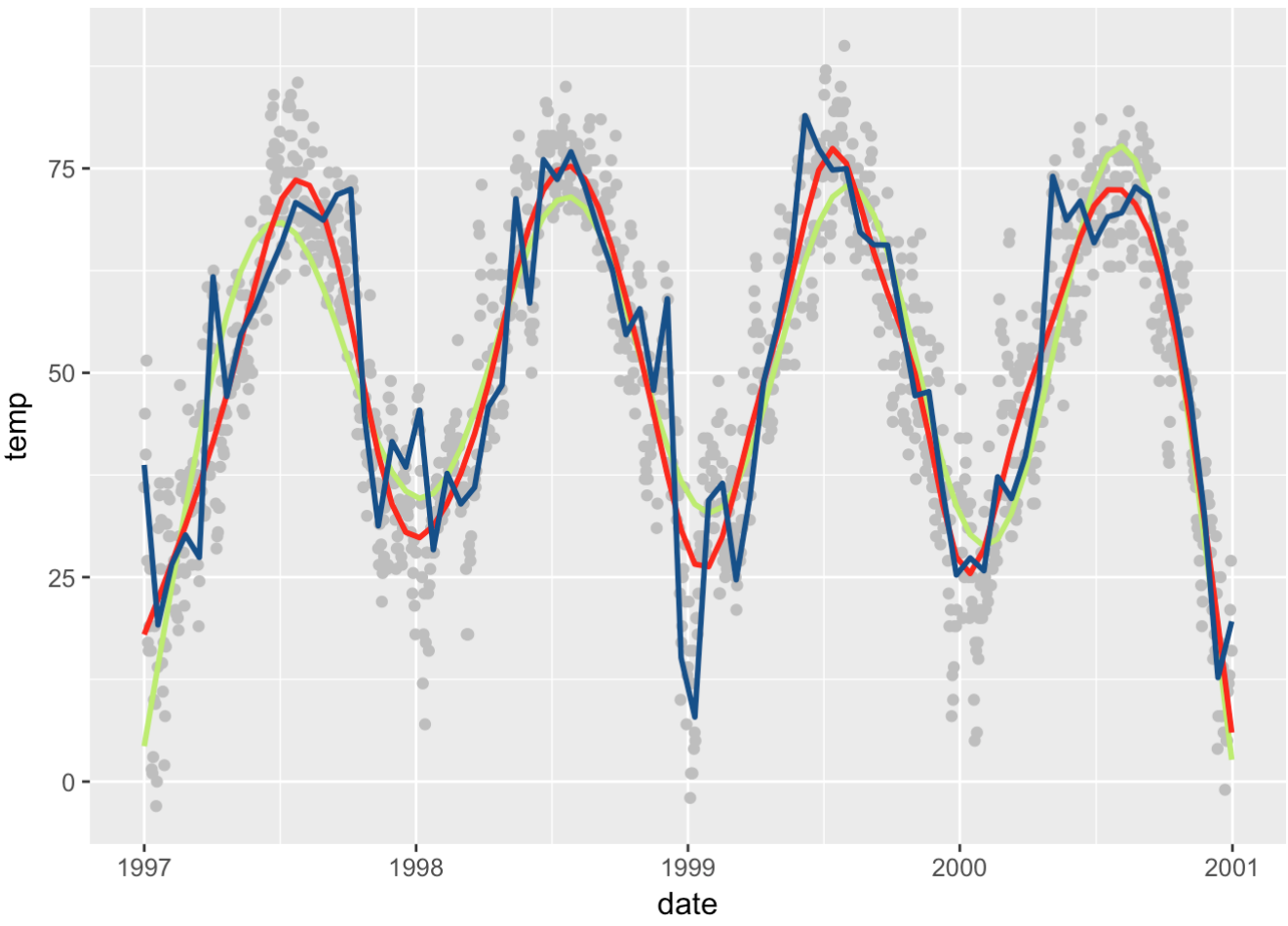
```
## `geom_smooth()` using method = 'gam'
```



Specifying the formula: stat_smooth(formula=)

But ggplot2 allows us to specify the model you want it to use. For example, let's say we want to increase the GAM dimension (add some additional wiggles to the smooth):

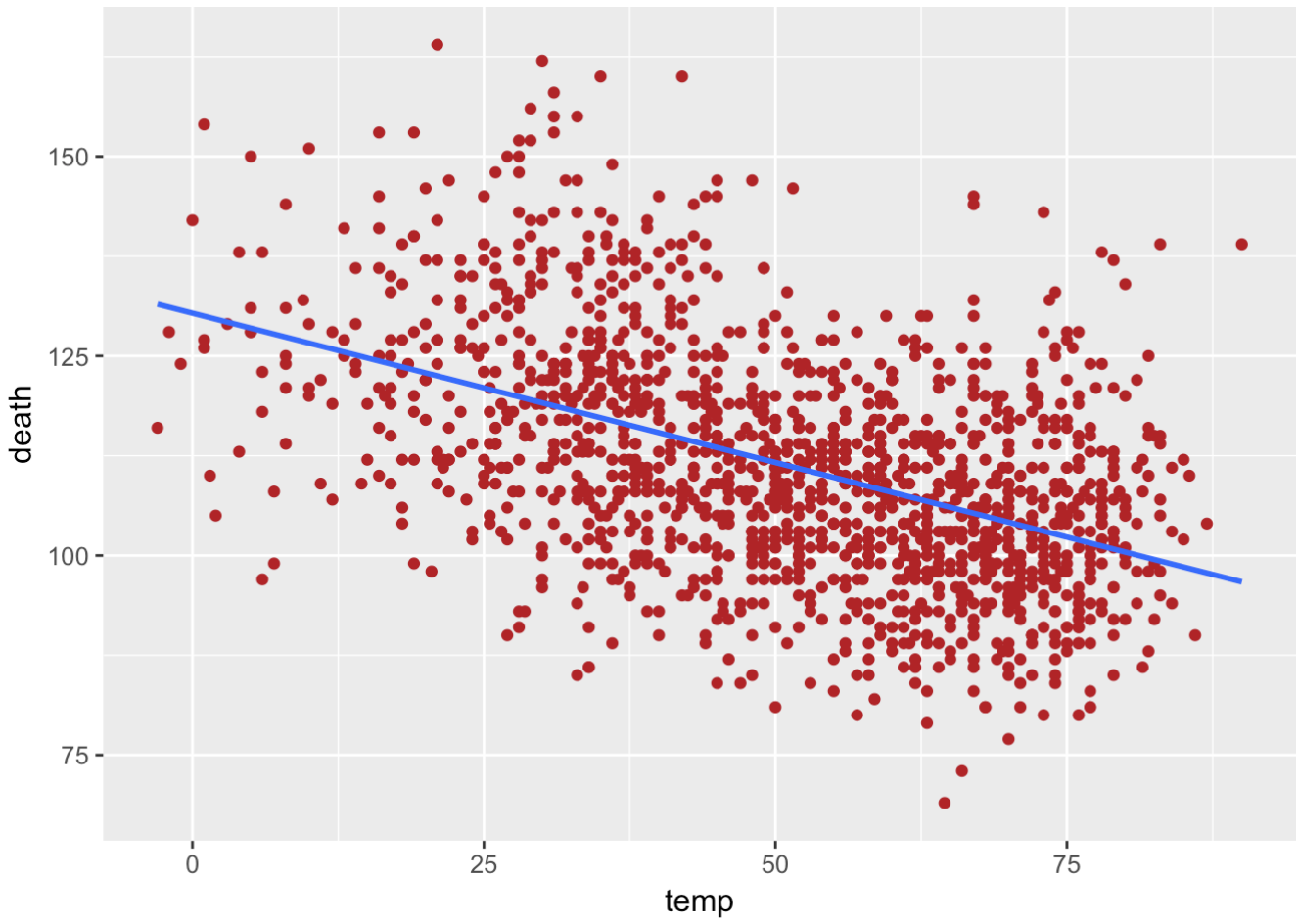
```
ggplot(nmmaps, aes(date, temp)) + geom_point(color="grey") +   stat_smooth(method="gam", formula = y~s(x, k=10),
col="darkolivegreen2", se=FALSE, size=1) + stat_smooth(method="gam", formula = y~s(x, k=30), col="red", se=FALSE,
 size=1) + stat_smooth(method="gam", formula = y~s(x, k=500), col="dodgerblue4", se=FALSE, size=1)
```



Adding a linear fit: stat_smooth(method="lm")

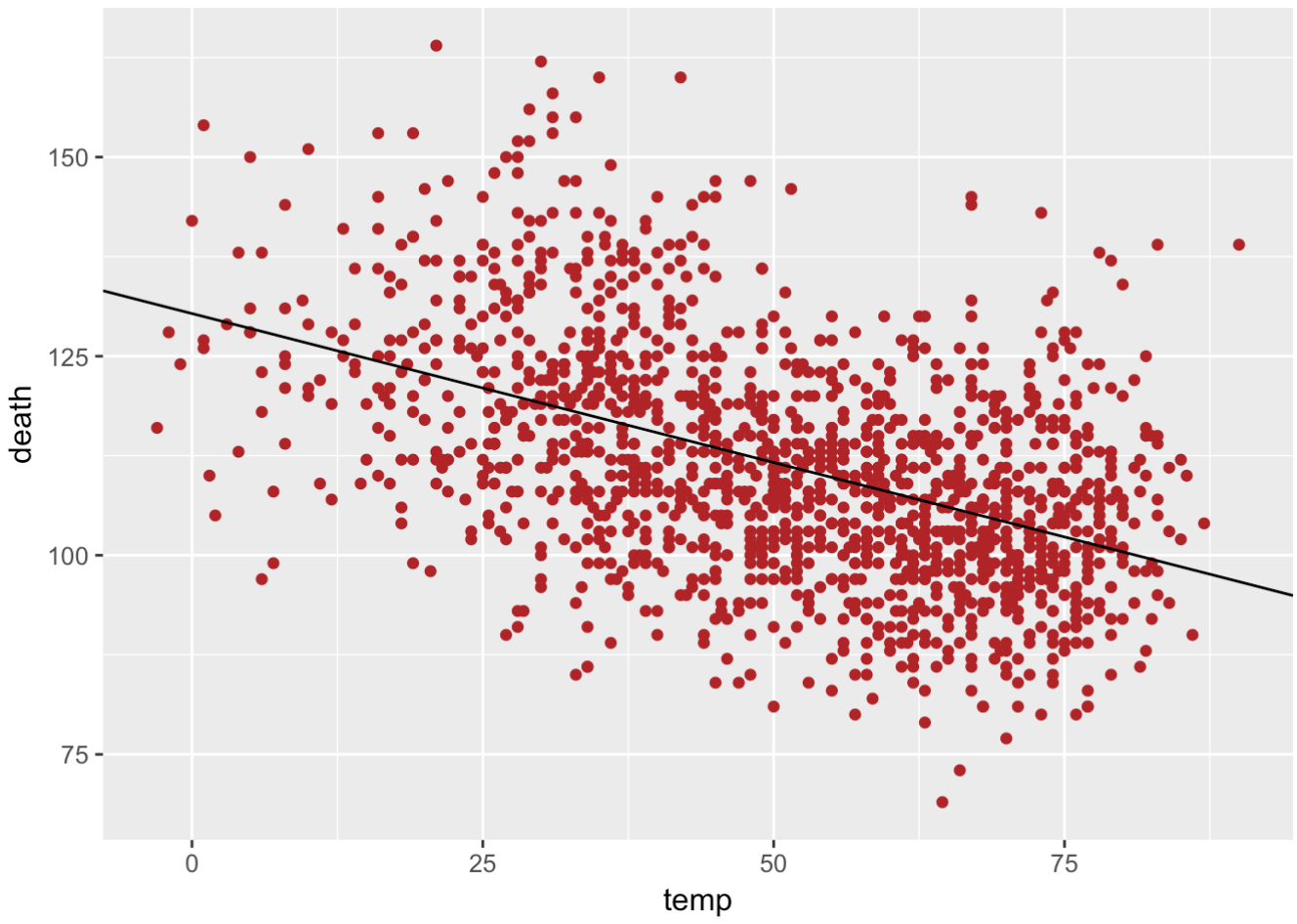
Although the default is smooth, it is also easy to add a standard linear fit

```
#se=TRUE displays confidence interval around smooth. FALSE to override this default behavior
ggplot(nmmaps, aes(temp, death)) + geom_point(color="firebrick") + stat_smooth(method="lm", se=FALSE)
```



Note that the same could be achieved using the more cumbersome:

```
lmTemp <- lm(death~temp, data=nmmaps)
ggplot(nmmaps, aes(temp, death))+geom_point(col="firebrick")+geom_abline(intercept = lmTemp$coef[1], slope = lmTe
mp$coef[2])
```



The project is accomplished using R version 3.2.2 and ggplot2 version 2.2.1.

```
packageVersion("ggplot2")
```

```
## [1] '2.2.1'
```

```
packageVersion("ggthemes")
```

```
## [1] '3.3.0'
```

```
getRversion()
```

```
## [1] '3.2.2'
```