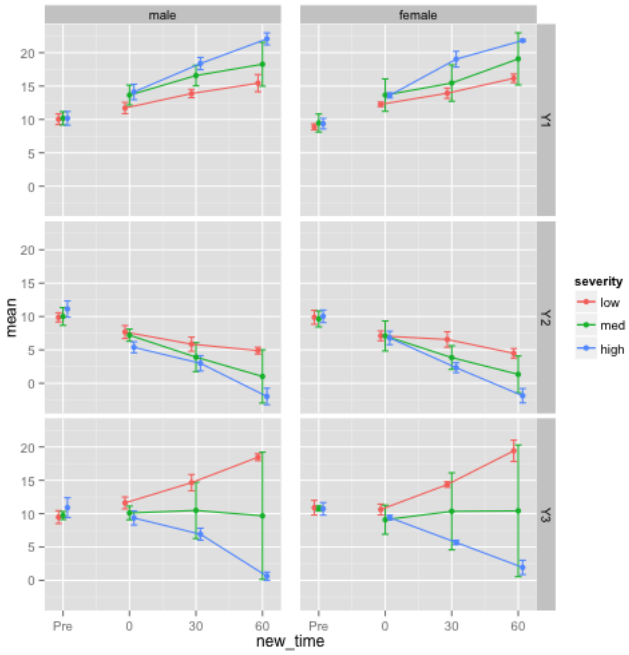
Line plots of longitudinal summary data in R using ggplot2

July 9, 2015

By strictlystat

(This article was first published on **A HopStat and Jump Away » Rbloggers**, and kindly contributed to R-bloggers)

I recently had an email for a colleague asking me to make a figure like this in `ggplot2` or `trellis` in R:



As I know more about how to do things in `ggplot2`, I chose to use that package (if it wasn't obvious from the plot or other posts).

Starting Point

Cookbook R/) has a great starting point for making this graph. The solution there is not sufficient for the desired graph, but that may not be clear why that is. I will go through most of the steps of customization on how to get the desired plot.

Creating Data

To illustrate this, I will create some sample dataset:

trading
ggplot2
animation
time series
finance
latex
excel
ggplot
googlevis
pca
RStudio
quantmod
eclipse
market research
tutorial
how to import image file to R
alt=
rattle
knitr
coplot
rcmdr
gis

RECENT POSTS

RcppGSL 0.3.0
"A 99% TVaR is generally a 99.6% VaR"
Building Wordclouds in R
Bio7 2.3 Released!
Get to know Cortana Analytics: Workshop and webinars
The IQUIT R video series
Legoplots in R (3D barplots in R)
Plotting Time Series in R using Yahoo Finance data
New release: Choroplethr v3.2.0
Coloring (and Drawing) Outside the Lines in ggplot
Measuring business health with Delta LifeCycle Grids and R
Spatio-Temporal Kriging in R
abcfr 0.9-3
R packages with unlimited licenses
New R titles available in Chinese

OTHER SITES

Statistics of Israel
SAS blogs
Jobs for R-users

```
N <- 30
id <- as.character(1:N) # create ids
sexes = c("male", "female")
sex <- sample(sexes, size = N/2, replace = TRUE) #
diseases = c("low", "med", "high")
disease <- rep(diseases, each = N/3) # disease sev
times = c("Pre", "0", "30", "60")
time <- rep(times, times = N) # times measured
t <- 0:3
ntimes = length(t)
y1 <- c(replicate(N/2, rnorm(ntimes, mean = 10+2*t
        replicate(N/2, rnorm(ntimes, mean = 10+4*t
y2 <- c(replicate(N/2, rnorm(ntimes, mean = 10-2*t
        replicate(N/2, rnorm(ntimes, mean = 10-4*t
y3 <- c(replicate(N/2, rnorm(ntimes, mean = 10+t^2
        replicate(N/2, rnorm(ntimes, mean = 10-t^2

data <- data.frame(id=rep(id, each=ntimes), sex=re
                   severity=rep(disease, each=ntim
                   Y1=c(y1), Y2=c(y2), Y3=c(y3)) #
#### factor the variables so in correct order
data$sex = factor(data$sex, levels = sexes)
data$time = factor(data$time, levels = times)
data$severity = factor(data$severity, levels = dis
head(data)
```

```
  id    sex severity time        Y1        Y2
1  1 female      low  Pre  9.262417 11.510636  9.04
2  1 female      low    0 10.223988  8.592833 11.57
3  1 female      low   30 13.650680  5.696405 13.95
4  1 female      low   60 15.528288  5.313968 18.63
5  2 female      low  Pre  9.734716 11.190081 10.08
6  2 female      low    0 12.892207  7.897296  9.79
```

We have a longitudinal dataset with 30 different people/units with different ID. Each ID has a single sex and disease severity. Each ID has 4 replicates, measuring 3 separate variables (`Y1`, `Y2`, and `Y3`) at each time point. The 4 time points are previous (`Pre`)/baseline, time 0, 30, and 60, which represent follow-up.

Reformatting Data

In `ggplot2`, if you want to plot all 3 `Y` variables, you must have them in the same column, with another column indicating which variable you want plot. Essentially, I need to make the data "longer". For this, I will reshape the data using the `reshape2` package and the function `melt`.

```
library(reshape2)
long = melt(data, measure.vars = c("Y1", "Y2", "Y3"
head(long)
```

```
  id    sex severity time variable     value
1  1 female      low  Pre       Y1  9.262417
2  1 female      low    0       Y1 10.223988
3  1 female      low   30       Y1 13.650680
4  1 female      low   60       Y1 15.528288
5  2 female      low  Pre       Y1  9.734716
6  2 female      low    0       Y1 12.892207
```

It may not be clear what has been reshaped, but reordering the `data.frame` can illustrate that each `Y` variable is now a separate row:

```
head(long[ order(long$id, long$time, long$variable)
```

```
    id    sex severity time variable     value
1    1 female      low  Pre       Y1  9.262417
121  1 female      low  Pre       Y2 11.510636
241  1 female      low  Pre       Y3  9.047127
2    1 female      low    0       Y1 10.223988
122  1 female      low    0       Y2  8.592833
242  1 female      low    0       Y3 11.570381
3    1 female      low   30       Y1 13.650680
123  1 female      low   30       Y2  5.696405
243  1 female      low   30       Y3 13.954316
4    1 female      low   60       Y1 15.528288
```

Creating Summarized data frame

We will make a `data.frame` with the means and standard deviations for each group, for each sex, for each `Y` variable, for separate time points. I will use `plyr` to create this `data.frame`, using `ddply` (first `d` representing I'm putting in a `data.frame`, and the second `d` representing I want `data.frame` out):

```
library(plyr)
agg = ddply(long, .(severity, sex, variable, time),
  c(mean=mean(x$value), sd = sd(x$value))
})
head(agg)
```

```
  severity  sex variable time      mean        sd
1      low male       Y1  Pre  9.691420 1.1268324
2      low male       Y1    0 12.145178 1.1218897
3      low male       Y1   30 14.304611 0.3342055
4      low male       Y1   60 15.885740 1.7616423
5      low male       Y2  Pre  9.653853 0.7404102
6      low male       Y2    0  7.652401 0.7751223
```

There is nothing special about means/standard deviations. It could be any summary measures you are interested in visualizing.

We will also create the Mean + 1 standard deviation. We could have done standard error or a confidence interval, etc.

```
agg$lower = agg$mean + agg$sd
agg$upper = agg$mean - agg$sd
```

Now, `agg` contains the data we wish to plot.

Time is not on your side

Time as a factor

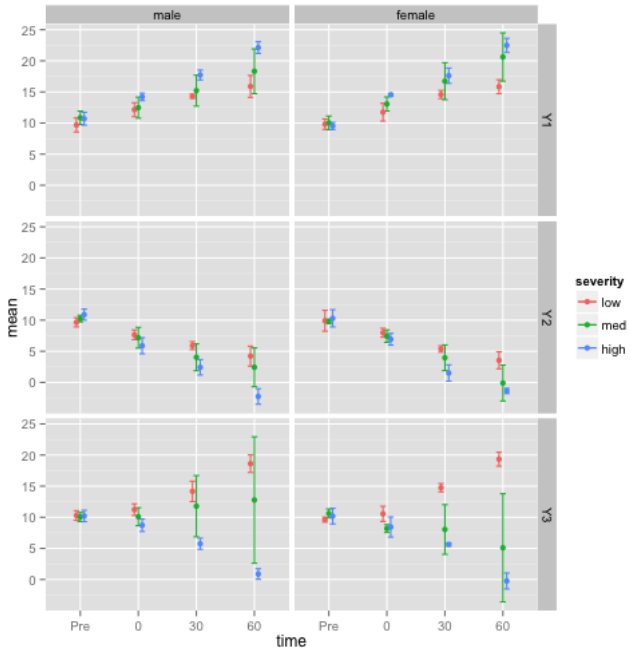
If you look at the plot we wish to make, we want the lines to be connected for times 0, 30, 60, but not for the previous data. Let's try using the `time` variable, which is a `factor`. We create `pd`, which will be a `ggplot2` object, which tells that I wish to plot the means + error bars slightly next to each other.

```
class(agg$time)
```

```
[1] "factor"
```

```
pd <- position_dodge(width = 0.2) # move them .2 to

gbase  = ggplot(agg, aes(y=mean, colour=severity)) +
  geom_errorbar(aes(ymin=lower, ymax=upper), width=
  geom_point(position=pd) + facet_grid(variable ~ s
gline = gbase + geom_line(position=pd)
print(gline + aes(x=time))
```



None of the lines are connected! This is because `time` is a `factor`. We will use `gbase` and `gline` with different times to show how the end result can be achieved.

Time as a numeric

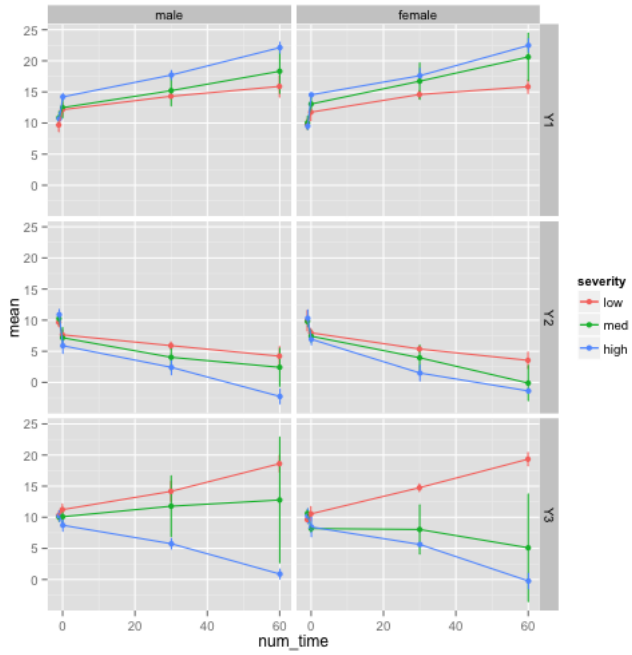
We can make `time` a numeric variable, and simply replace `Pre` with `-1` so that it can be plotted as well.

```
agg$num_time = as.numeric(as.character(agg$time))
agg$num_time[ is.na(agg$num_time) ] = -1
unique(agg$num_time)
```

```
[1] -1  0 30 60
```


In a previous post, I have discussed as an aside of creating a plot in `ggplot2` and then creating adding data to the `data.frame`. You must use the `%+%` to update the data in the object.

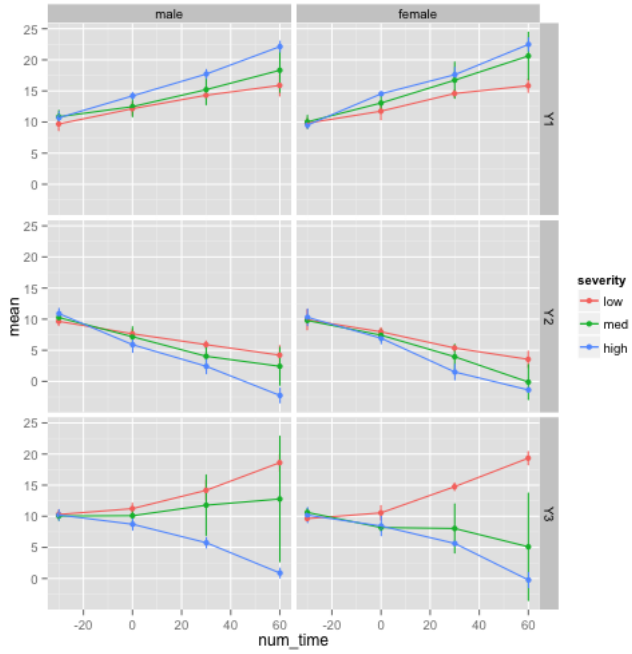
```
gline = gline %+% agg
print(gline + aes(x=num_time))
```



If you look closely, you can see that `Pre` and time `0` are very close and not labeled, but also connected. As the scale on the x-axis has changed, the width of the error bar (set to `0.3`), now is too small and should be changed if using this solution.

Although there can be a discussion if the `Pre` data should be even on the same plot or the same timeframe, I will leave that for you to dispute. I don't think it's a terrible idea, and I think the plot works because the `Pre` and `0` time point data are not connected. There was nothign special about `-1`, and here we use `-30` to make it evenly spaced:

```
agg$num_time[ agg$num_time == -1 ] = -30
gline = gline %+% agg
print(gline + aes(x=num_time))
```



That looks similar to what we want. Again, `Pre` is connected to the data, but we also now have a labeling problem with the x-axis somewhat. We still must change the width of the error bar in this scenario as well.

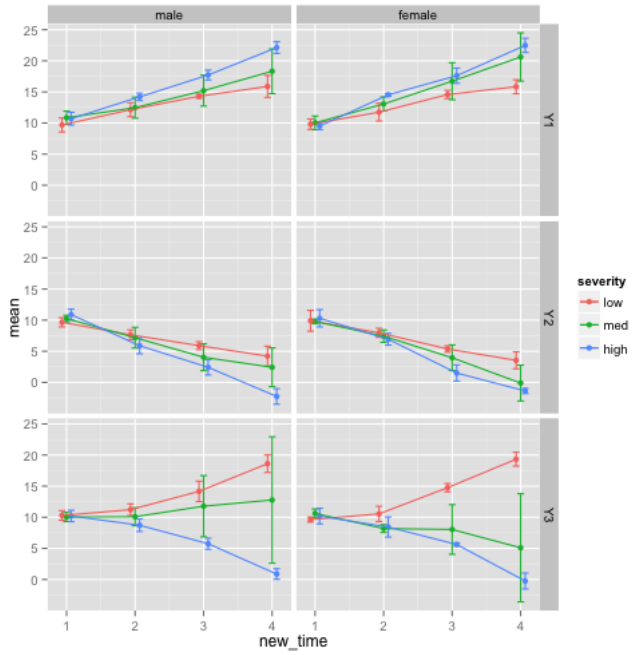
Time as a numeric, but not the actual time point

In the next case, we simply use `as.numeric` to the factor to create a variable `new_time` that will be `1` for the first level of `time` (in this case `Pre`) to the number of time points, in this case `4`.

```
agg$new_time = as.numeric(agg$time)
unique(agg$new_time)
```

```
[1] 1 2 3 4
```

```
gline = gline %+% agg
print(gline + aes(x = new_time))
```



Here we have something similar with the spacing, but now the labels are not what we want. Also, `Pre` is still connected. The width of the error bars is now on a scale from 1-4, so they look appropriate.

Creating a Separate data.frame

Here, we will create a separate `data.frame` for the data that we want to connect the points. We want the times 0-60 to be connected and the `Pre` time point to be separate.

```
sub_no_pre = agg[ agg$time != "Pre",]
```

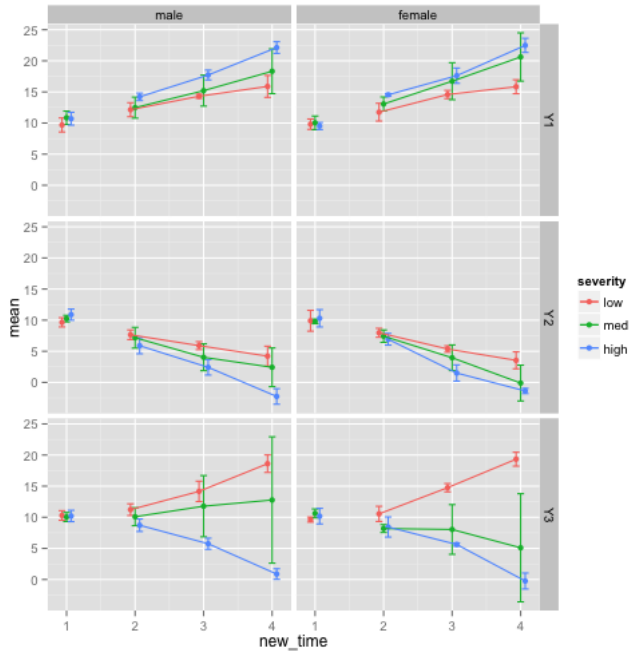
Mulitple data sets in plot function

Note, previously we did:

```
gline = gbase + geom_line(position=pd)
```

This assumes that `geom_line` uses the same `data.frame` as the rest of the plot (`agg`). We can fully specify the arguments in `geom_line` so that the line is only for the non-Pre data:

```
gbase = gbase %+% agg
gline = gbase + geom_line(data = sub_no_pre, positi
                          aes(x = new_time, y = mea
print(gline + aes(x = new_time))
```



Note, the arguments in `aes` should match the rest of the plot for this to work smoothly and correctly.

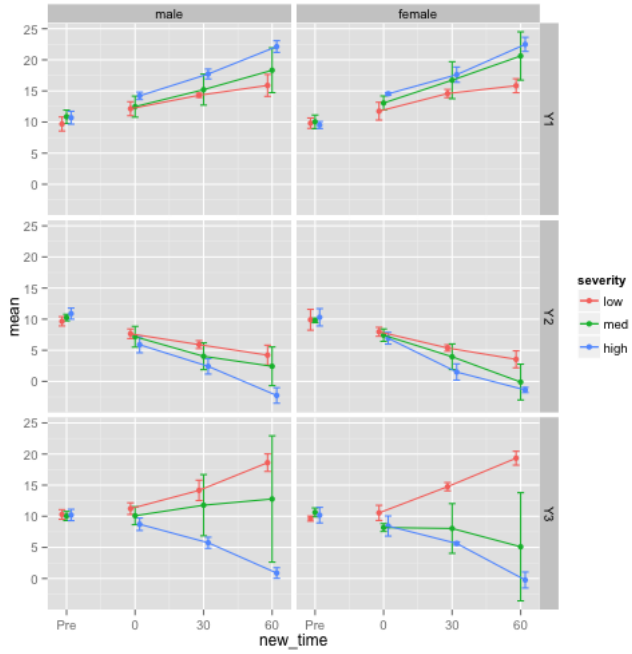
Relabeling the axes

Now, we simply need to re-label the x-axis so that it corresponds to the correct times:

```
g_final = gline + aes(x=new_time) +
  scale_x_continuous(breaks=c(1:4), labels=c("Pre",
```

We could be more robust in this code, using the levels of the factor:

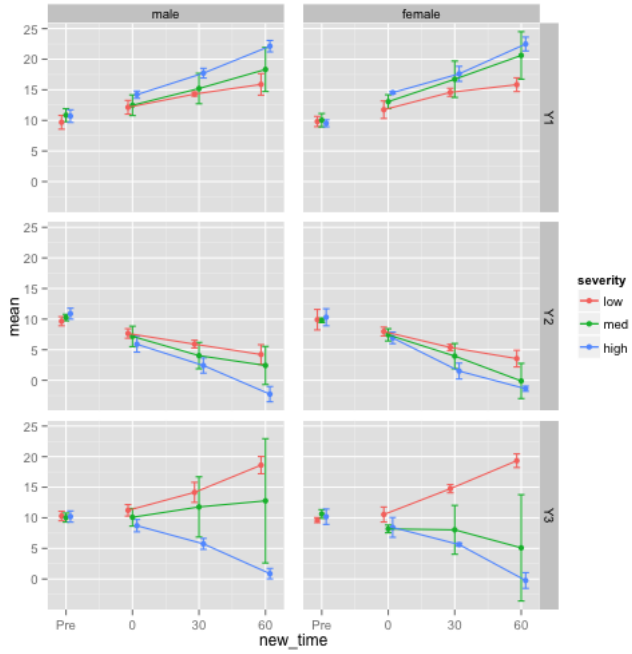
```
time_levs = levels(agg$time)
g_final = gline + aes(x=new_time) +
  scale_x_continuous(
    breaks= 1:length(time_levs),
    labels = time_levs)
print(g_final)
```



Give me a break

My colleague also wanted to separate the panels a bit. We will use the `panel.margin` arguments and use the `unit` function from the `grid` package to define how far apart we want the axes.

```
library(grid)
g_final = g_final + theme(panel.margin.x = unit(1,
                          panel.margin.y = unit(0.5
print(g_final)
```



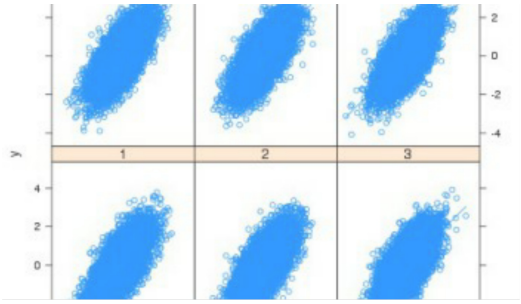
Additional options and conclusoin

I believe legends should be inside a plot for many reasons (I may write about that). Colors can be changed (see `scale_colour_manual`). Axis labels should be changed, and the `Y` should be labeled to what they are (this is a toy example).

Overall, this plot seems to be what they wanted and the default options work okay. I hope this illustrates how to customize a `ggplot` to your needs and how you may need to use multiple `data.frames` to achieve your desired result.

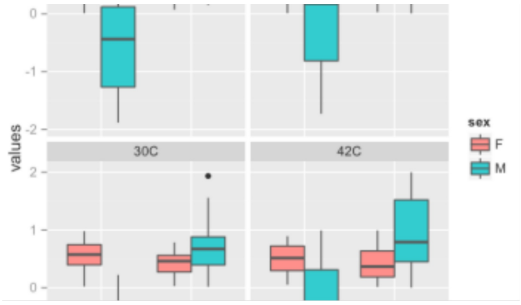
Comments: 5

Related



Lattice when modeling, ggplot when publishing
In "R bloggers"

Normality and Testing for Normality
In "R bloggers"



ggplot2 multiple boxplots with metadata
In "R bloggers"

To **leave a comment** for the author, please follow the link and comment on his blog: **A HopStat and Jump Away » Rbloggers**.

R-bloggers.com offers **daily e-mail updates** about R news and tutorials on topics such as: visualization (ggplot2, Boxplots, maps, animation), programming (RStudio, Sweave, LaTeX, SQL, Eclipse, git, hadoop, Web Scraping) statistics (regression, PCA, time series, trading) and more...

Comments are closed.