Greatest Hits R Mixtape

Max Kuhn, Ph.D

Pfizer Global R&D
Groton, CT
max.kuhn@pfizer.com

Outline

- Three Dots!
- `grid.arrange`
- Interactive graphics

In a World...

Suppose you have some data:

```
> head(dat)
```

```
       RMSE  Rsquared       Date
1 0.6092888 0.9140404 2013-07-13
2 0.5905273 0.9032800 2013-07-14
3 0.5318338 0.9176118 2013-07-15
4 0.5594337 0.9168153 2013-07-16
5 0.5763207 0.9095581 2013-07-17
6 0.5551559 0.9197075 2013-07-18
```

```
> str(dat)
```

```
'data.frame': 323 obs. of  3 variables:
 $ RMSE    : num  0.609 0.591 0.532 0.559 0.576 ...
 $ Rsquared: num  0.914 0.903 0.918 0.917 0.91 ...
 $ Date    : chr  "2013-07-13" "2013-07-14" "2013-07-15" "2013-07-16" ...
```

Analysis

Let's say you want to do some analysis of these data:

- convert the character strings to dates
- smooth the data (using loess)
- take the first derivative of the curve

Some Code

```
> foo <- function(x) {
+   library(lubridate)
+   x$Date <- ymd(x$Date)
+   x$Days <- difftime(x$Date, min(x$Date), units = "days")
+   mod <- loess(RMSE ~ as.numeric(Days), data = x)
+   x$pred <- predict(mod, x)
+   deriv <- c(NA, x$pred[-nrow(x)] - x$pred[-1])
+   x$deriv <- deriv
+   x
+ }
```

Smoothing Options

Right now, the code uses the default `loess` parameters. What do we do if those are not good for these data?

We could add more options to `foo` to handle this:

```
> foo <- function(x, span = .2, degree = 1) {
+   # stuff...
+   mod <- loess(RMSE ~ as.numeric(Days),
+                data = x,
+                span = span,
+                degree = degree)
+   ## other stuff
+ }
```

A Better Method

That's not very appealing, especially if there are a lot of options that might need changing. Another option is to use the three dots or ellipses:

We could add more options to `foo` to handle this:

```
> foo <- function(x, ...) {
+   library(lubridate)
+   x$Date <- ymd(x$Date)
+   x$Days <- difftime(x$Date, min(x$Date), units = "days")
+   mod <- loess(RMSE ~ as.numeric(Days), data = x, ...)
+   x$pred <- predict(mod, x)
+   deriv <- c(NA, x$pred[-nrow(x)] - x$pred[-1])
+   x$deriv <- deriv
+   x
+ }
```

Since `span` and `degree` are not options to `foo`, these get passed to any function(x) that have the ... in their function call.

For Predictive Modeling

This is important to me since I maintain a package called caret, which has wrappers for almost 200 different predictive modeling/machine learning functions.

`train` is a function that can be used to call specific models without dealing with the syntactical differences/minutia of the individual R functions. For example

```r
> mod1 <- train(Class ~ ., data = training, method = "gbm")
>
> mod2 <- train(Class ~ ., data = training, method = "pls",
+               preProc = c("center", "scale"))
```

Two Levels of Arguments

For example, the three dots would allow users to pass arguments for `gbm` in the call to `train`:

```
> mod1 <- train(Class ~ .,
+               data = training,
+               method = "gbm",
+               ## now args to gbm()
+               verbose = FALSE,
+               bag.fraction = 0.5)
```

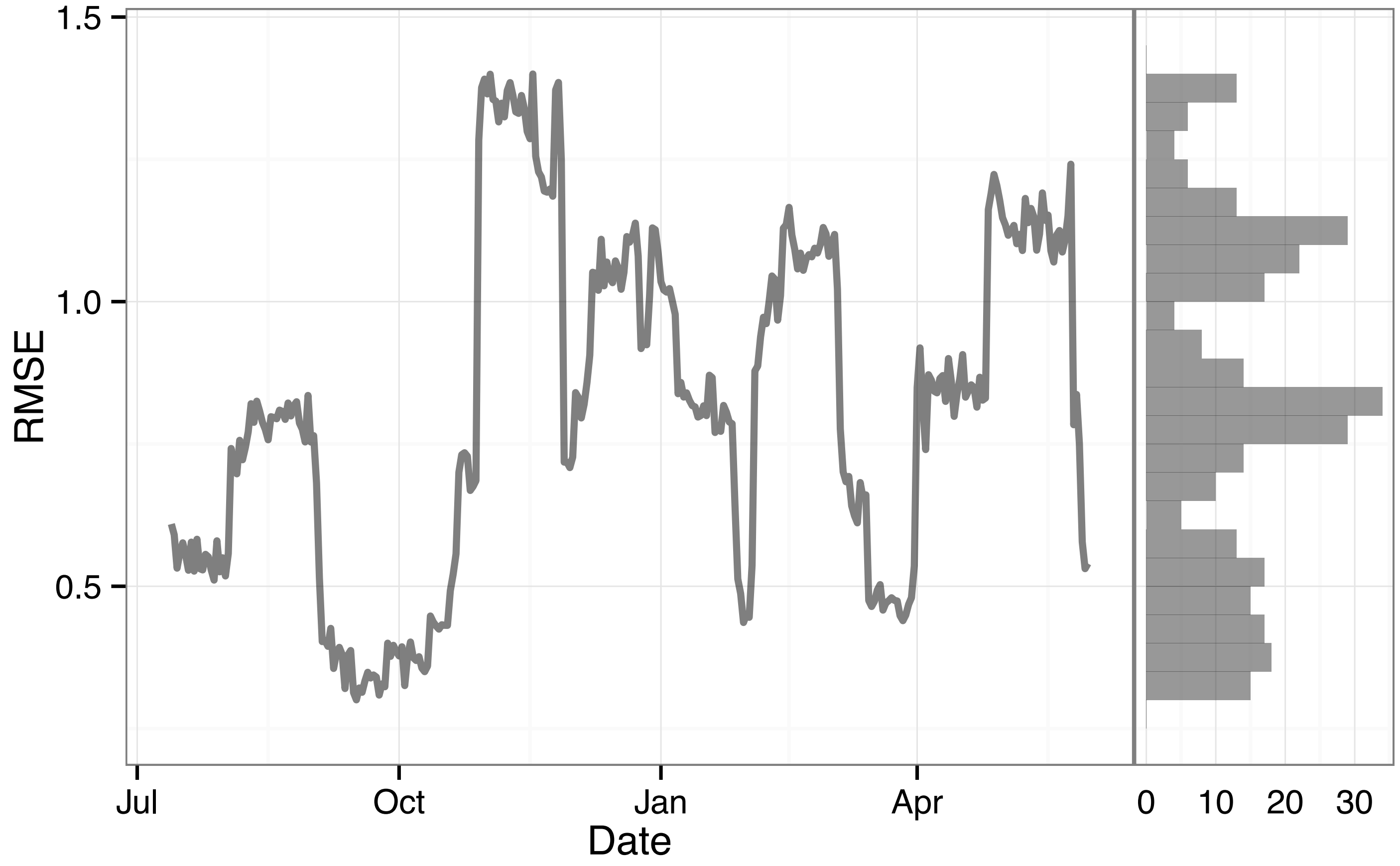
Three Levels of Arguments

caret also has methods for feature selection. For example, gsfs uses genetic algorithms to search for optimal subset of predictors and this can be used to call train.

```
> mod3 <- gafs(x = x, y = y,
+               gafsControl = gafsControl(functions = caretGA),
+               ## now arguments to train()
+               method = 'mda',
+               tuneLength = 5,
+               ## now arguments to mda()
+               start.method = "kmeans")
```

grid.arrange

I would like to make this plot:

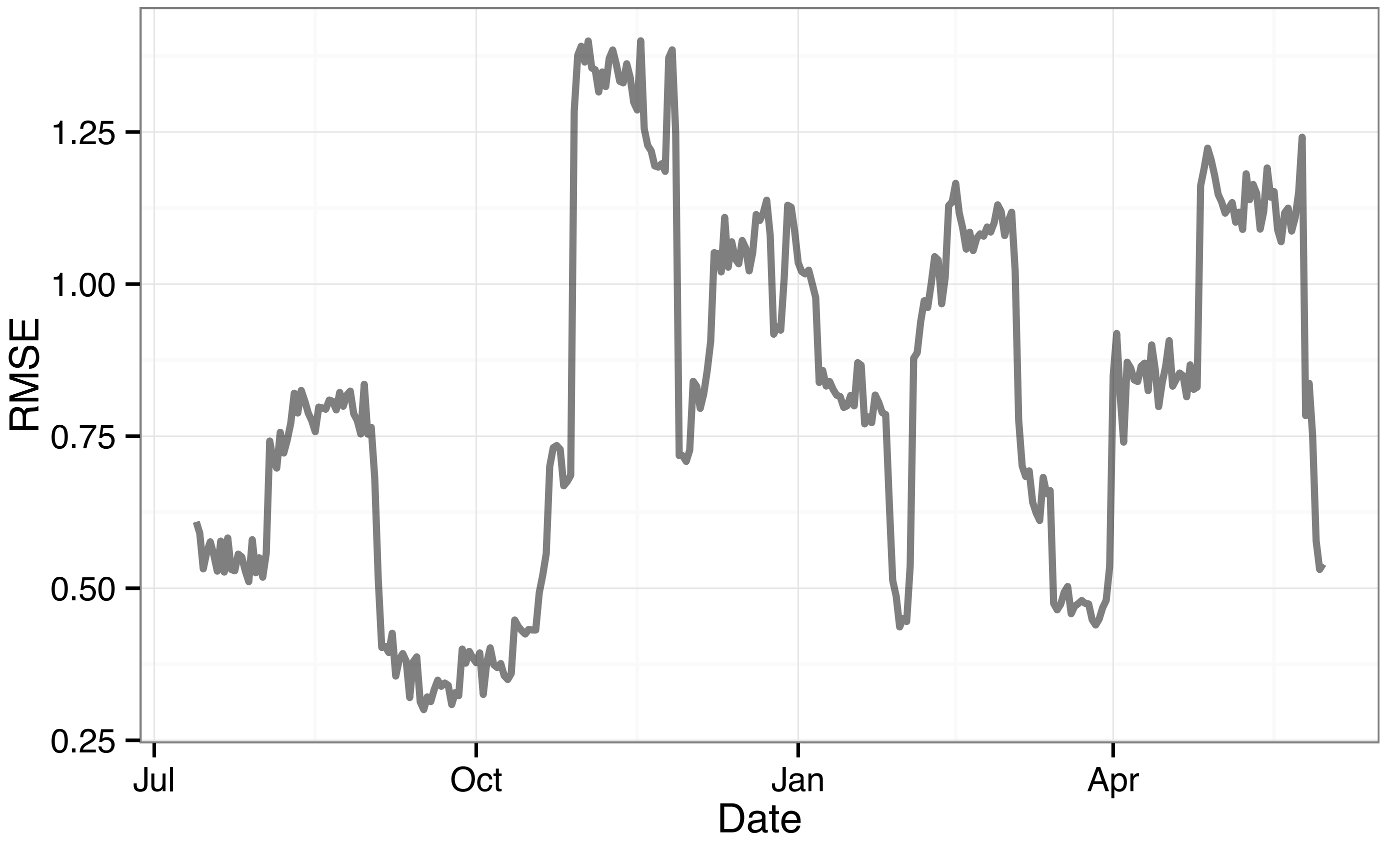


I'll use the grid.arrange function in the ~~ggplot2~~ gridExtra package to do this (thanks to David Winsemius for keeping me honest).

The Left–Hand Side

The line plot on the left isn't that hard to create:

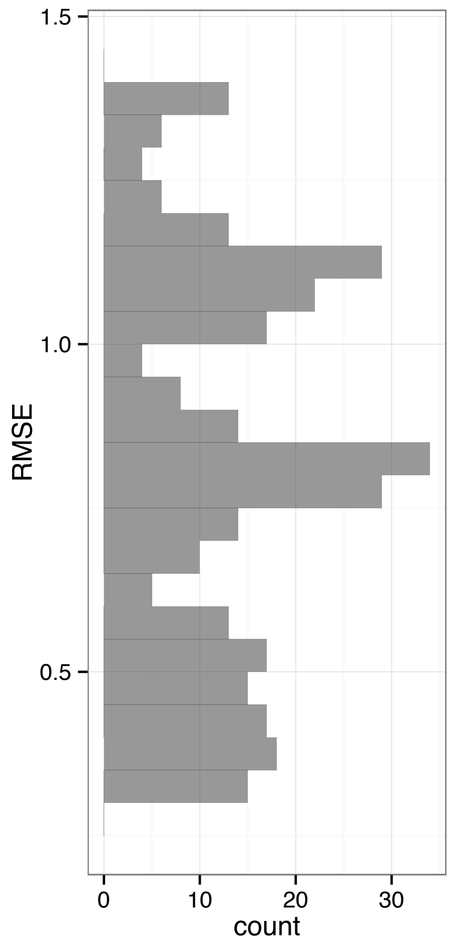
```
> ggplot(dat) +
+   geom_line(aes(x = Date, y = RMSE), lwd = 1, alpha = .5) +
+   xlab("Date")
```



The Right–Hand Side

Neither is the histogram

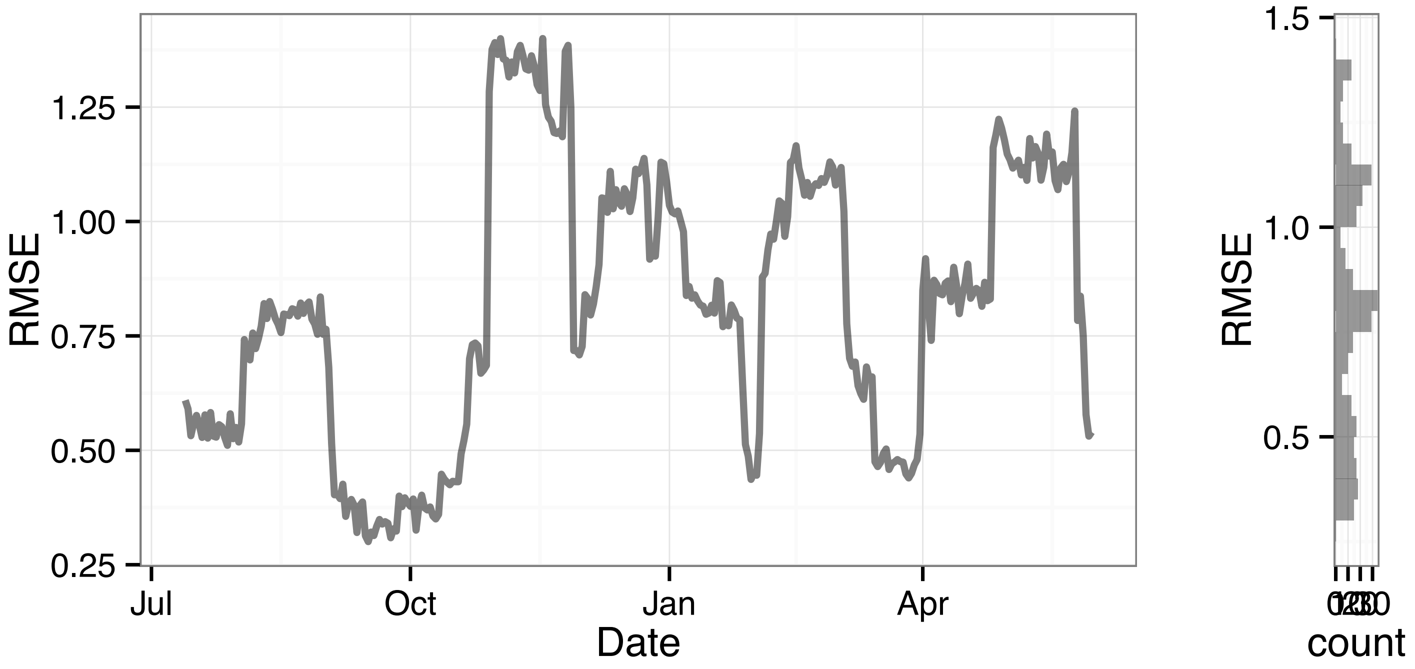
```
> ggplot(dat) +
+   geom_histogram(aes(x = RMSE), binwidth = .05, alpha = .5) +
+   coord_flip()
```



Putting them together

Neither is the histogram

```
> grid.arrange(scatter_obj, hist_obj,
+              ncol=2, nrow=1,
+              widths=c(5, 1), heights=1)
```

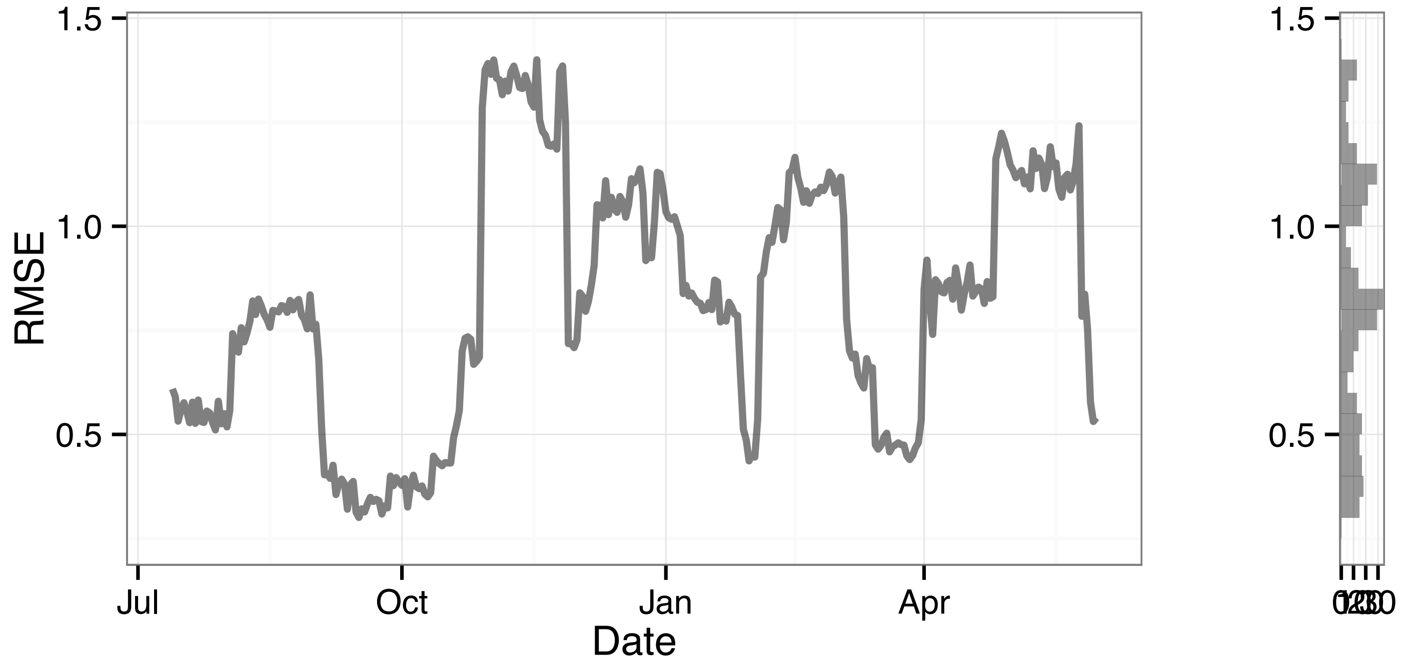


Oh No

Besides the obvious whitespace problem, note that the y–axis range is not the same in both plots.

Let's fix the range issue first by presetting the values and get rid of the histogram labels:

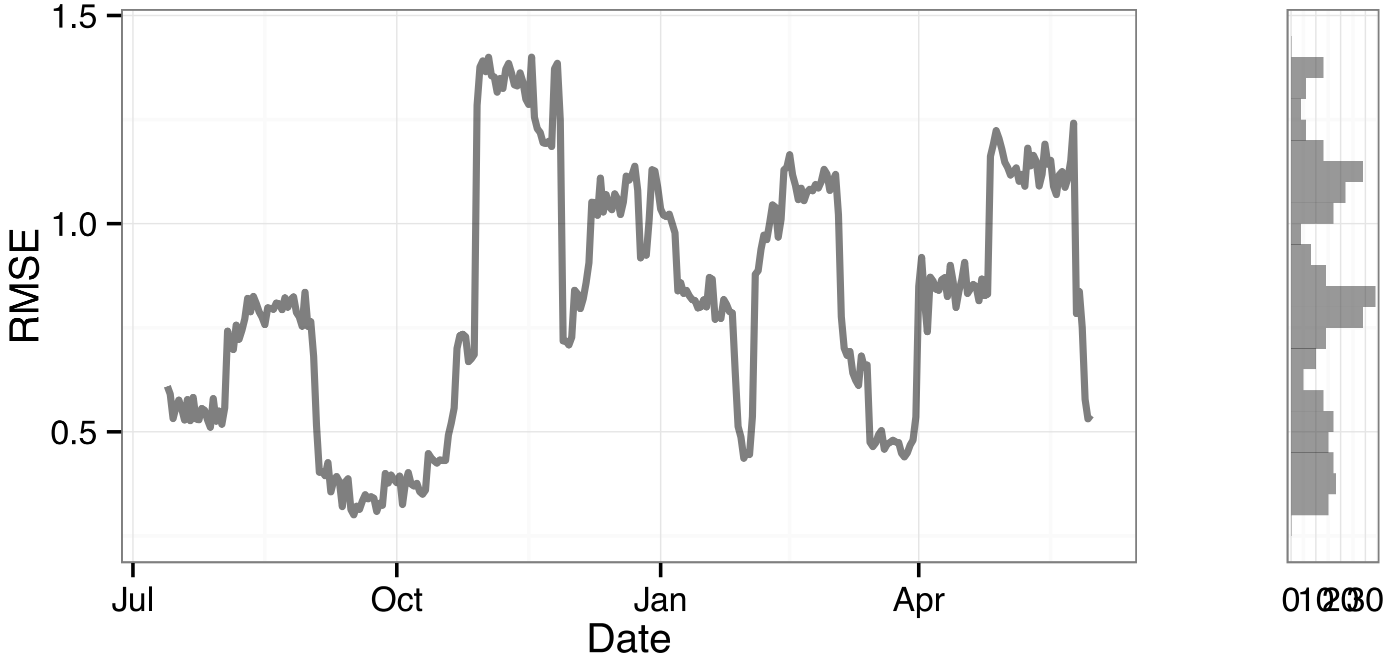
```
> rmse_range <- extendrange(dat$RMSE)
> scatter_obj <- scatter_obj + ylim(rmse_range)
> hist_obj <- hist_obj+ xlim(rmse_range) + xlab("") + ylab("")
```



Ticks

Now let's get rid of the y-axis tick marks on the histogram:

```
> hist_obj <- hist_obj +
+   theme(axis.ticks = element_blank(),
+         axis.text.y = element_blank())
```

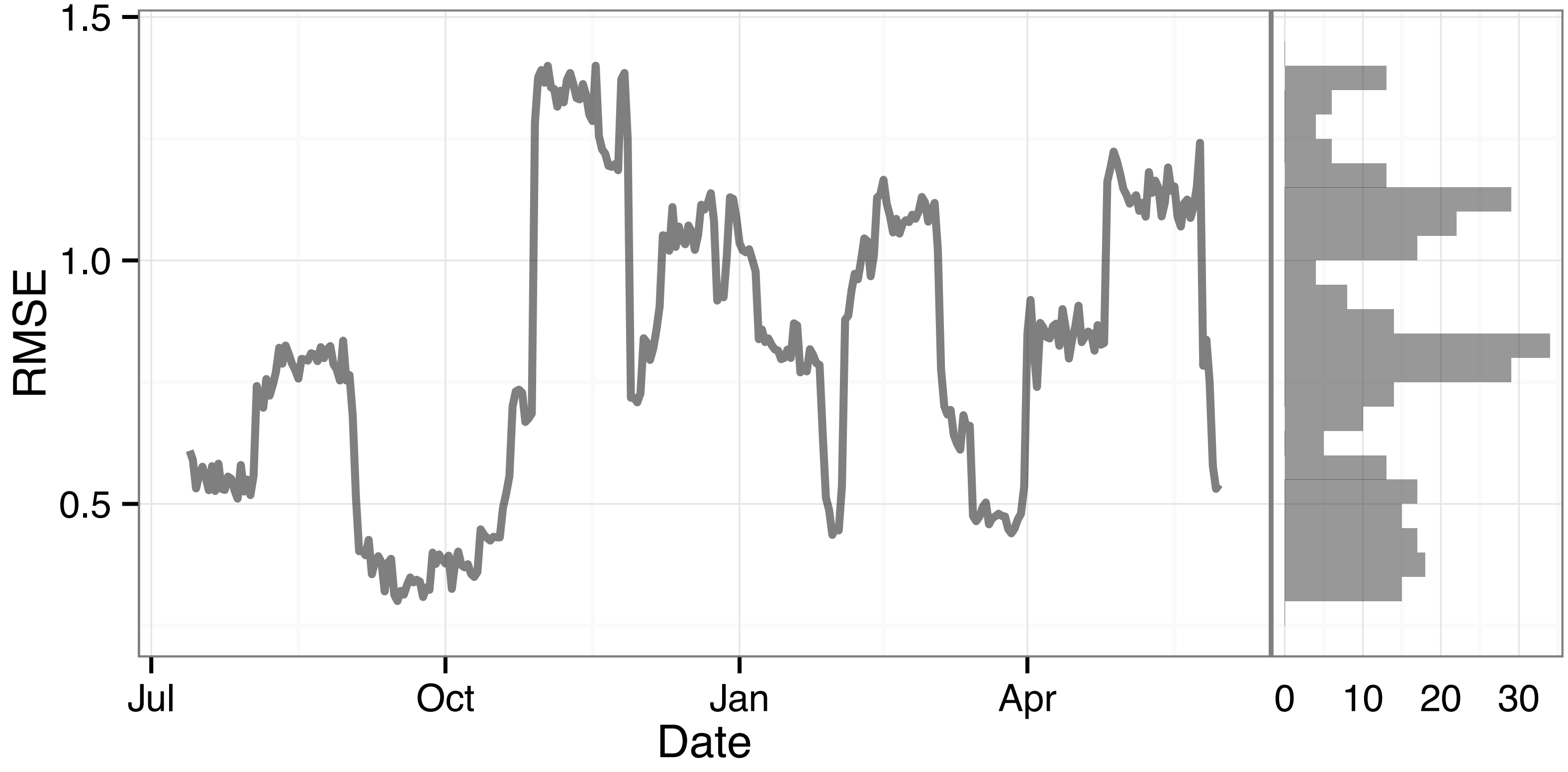


Almost There!

For the last part, we need to set the plot margins for both figures. This took a lot of experimentation:

```
> scatter_margins <- c(top = 3, right = 6, bottom = 2, left = 2)
> hist_margins <- c(top = 3, right = 4, bottom = 2, left = -14)
>
> hist_obj <- hist_obj +
+   theme(plot.margin = unit(hist_margins, "mm"),
+         axis.ticks = element_blank(),
+         axis.text.y = element_blank())
>
> scatter_obj <- scatter_obj +
+   theme(plot.margin = unit(scatter_margins, "mm"))
```

Finally!



Interactive Graphics

What I'd *like* to have is the ability to create visualizations that have interactivity in the sense that you can zoom axes and get more information from individual data points.

For the previous line plot, that isn't hard to do with rCharts and Morris plots:

```
> library(rCharts)  ## on github
> m1 <- mPlot(x = "Date", y = "RMSE", type = "Line", data = dat)
> m1$set(pointSize = 0, lineWidth = 1)
> m1
```

In general though, this isn't really solved (yet) and rCharts is really rough at this time.

Shiny!

We use RStudio's shiny package and server a good deal and love it.

We use it to give our clients semi–interactive access to the results of analyses that we have put together.

For me, I would rather put more biological CPU cycles into effective ways to communicate results than into new ways to generate p–values and FDR corrections.