

# Title

*John Muschelli*

*January 9, 2015*

## Contents

Introduction . . . . .	1
Exploratory Analysis . . . . .	1
Data Extraction . . . . .	13

The three “back ticks” (‘) must be followed by curly brackets “{”, and then “r” to tell the computer that you are using R code. This line is then closed off by another curly bracket “}”.

Anything before three more back ticks “```” are then considered R code (a script).

If any code in the document has just a backtick ‘ then nothing, then another backtick, then that word is just printed as if it were code, such as `hey`.

I’m reading in the bike lanes here.

```
# readin is just a "label" for this code chunk
## code chunk is just a "chunk" of code, where this code usually
## does just one thing, aka a module
### comments are still # here
### you can do all your reading in there
### let's say we loaded some packages
library(stringr)
library(plyr)
fname <- "../data/Bike_Lanes.csv"
bike = read.csv(fname, as.is = TRUE)
```

You can write your introduction here.

## Introduction

Bike lanes are in Baltimore. People like them. Why are they so long?

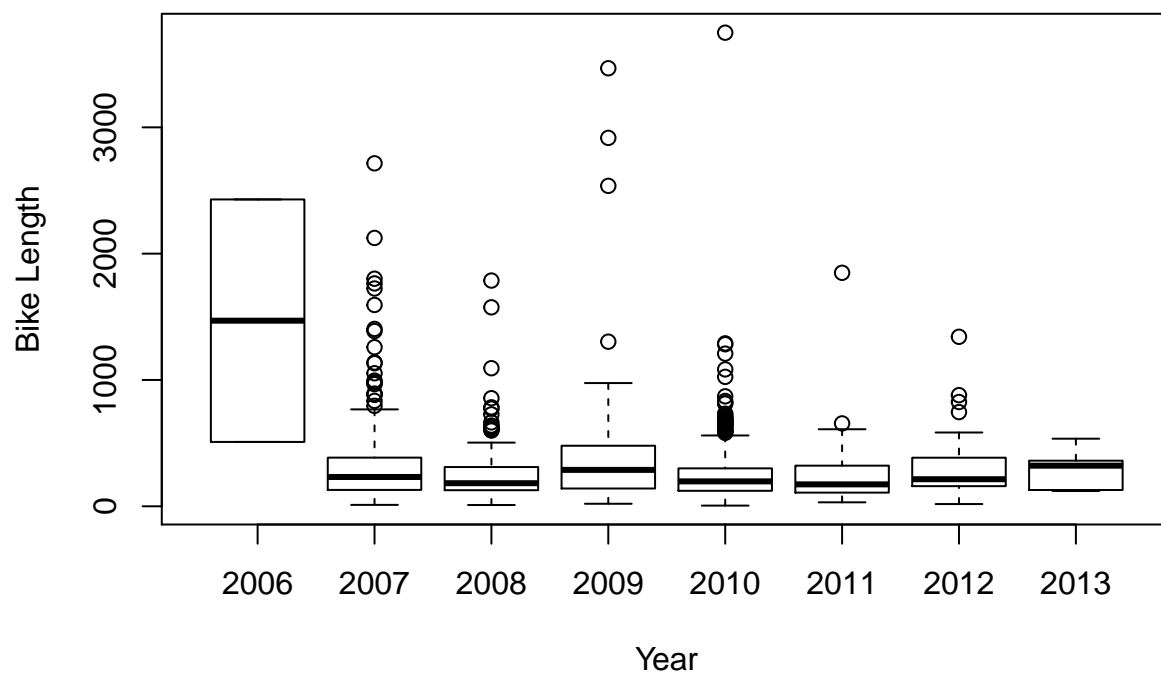
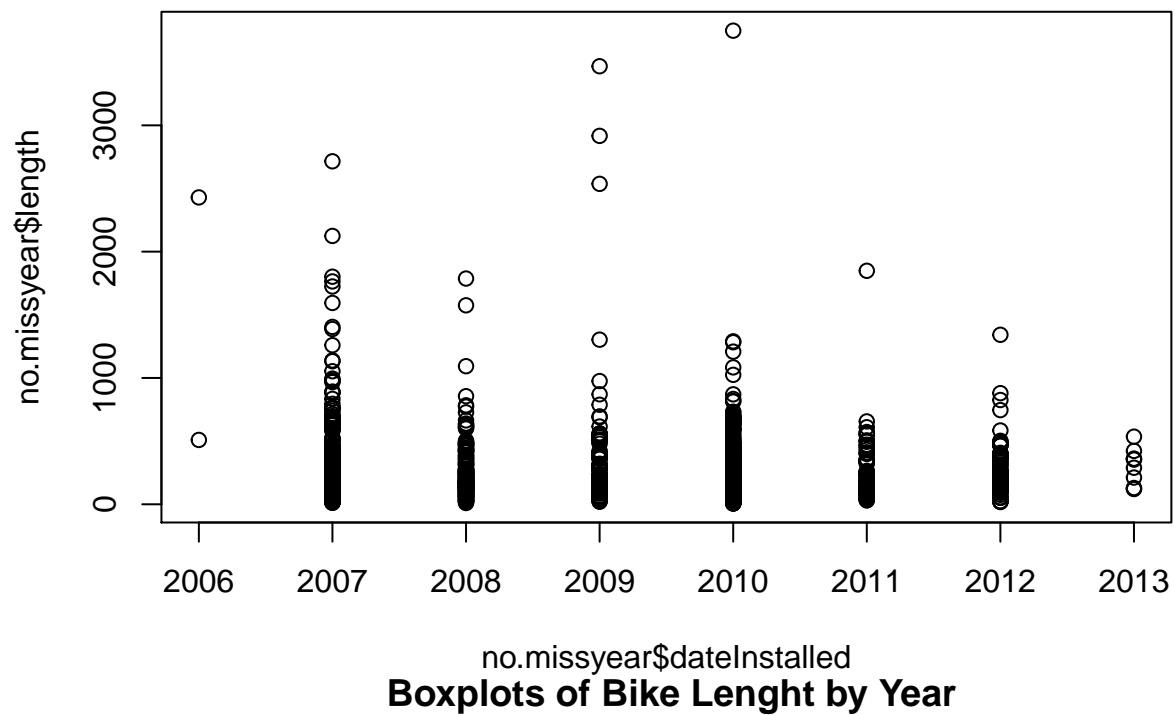
## Exploratory Analysis

Let’s look at some plots of bike length. Let’s say we wanted to look at what affects bike length.

### Plots of bike length

Note we made the subsection by using three “hashes” (pound signs): `###`.

We can turn off R code output by using `echo = FALSE` on the knitr code chunk. s

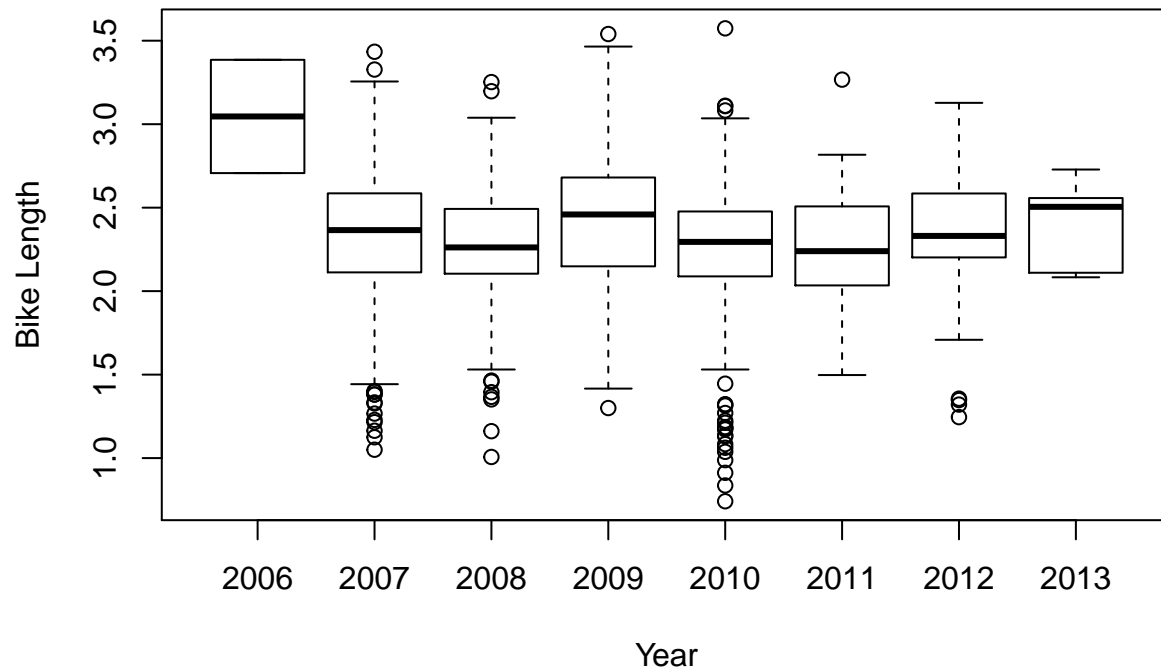


We have a total of 1505 rows.

What does it look like if we took the log (base 10) of the bike length:

```
no.missyear$log.length <- log10(no.missyear$length)
### see here that if you specify the data argument, you don't need to do the $
boxplot(log.length ~ dateInstalled, data=no.missyear, main="Boxplots of Bike Length by Year", xlab="Year")
```

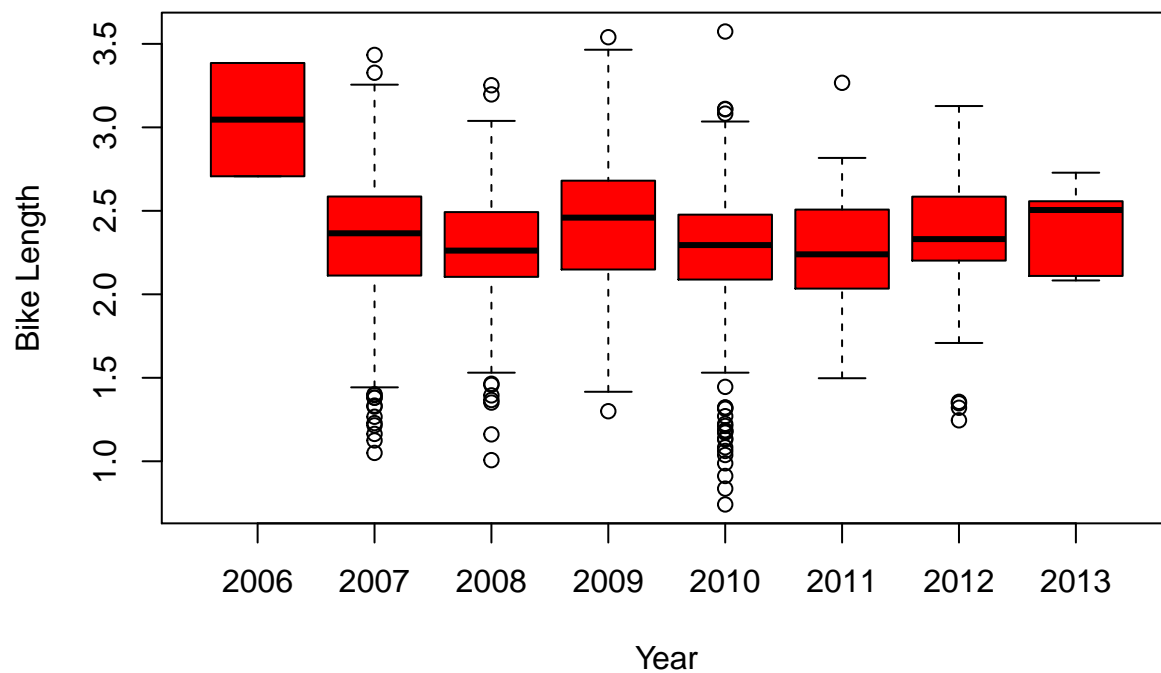
## Boxplots of Bike Lenght by Year



I want my boxplots colored, so I set the `col` argument.

```
boxplot(log.length ~ dateInstalled, data=no.missyyear, main="Boxplots of Bike Lenght by Year", xlab="Year", col="red")
```

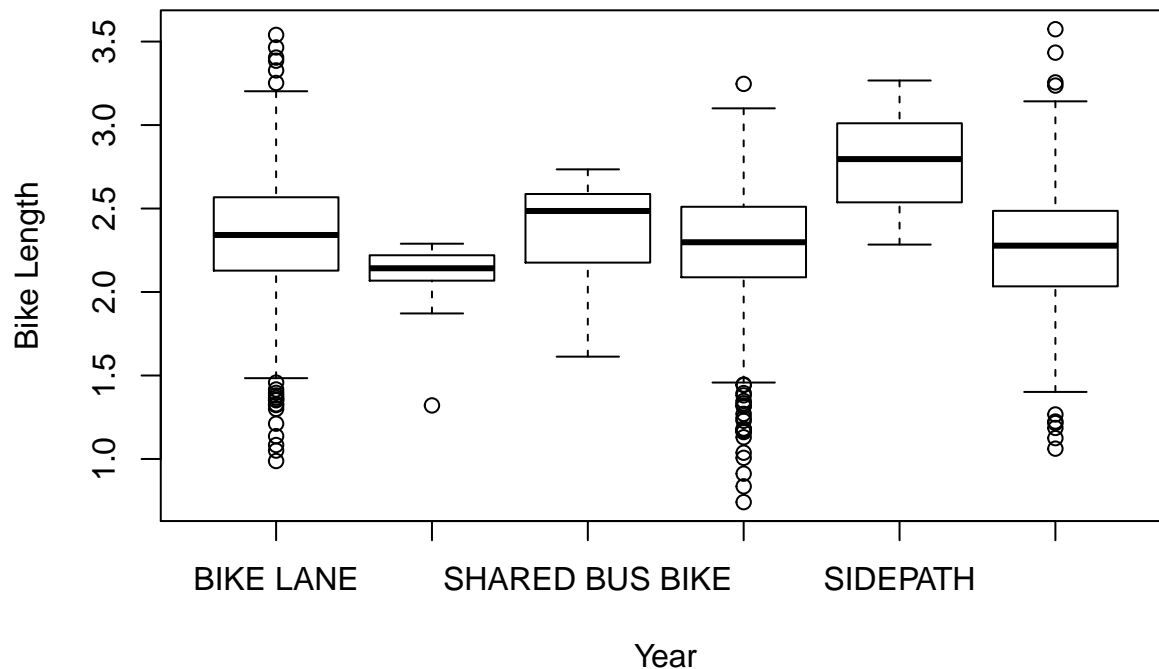
## Boxplots of Bike Lenght by Year



As we can see, 2006 had a much higher bike length. What about for the type of bike path?

```
### type is a character, but when R sees a "character" in a "formula", then it automatically converts it to a factor
### a formula is something that has a y ~ x, which says I want to plot y against x
### or if it were a model you would do y ~ x, which meant regress against y
boxplot(log.length ~ type, data=no.missyear, main="Boxplots of Bike Length by Year", xlab="Year", ylab="Bike Length")
```

## Boxplots of Bike Length by Year



What if we want to extract means by each type?

Let's show a few ways:

```
### tapply takes in vector 1, then does a function by vector 2, and then you tell what
### that function is
tapply(no.missyear$log.length, no.missyear$type, mean)
```

```
##      BIKE LANE      CONTRAFLOW SHARED BUS BIKE      SHARROW
##      2.330611      2.087246      2.363005      2.256425
##      SIDEPATH      SIGNED ROUTE
##      2.781829      2.263746
```

```
## aggregate
aggregate(x=no.missyear$log.length, by=list(no.missyear$type), FUN=mean)
```

```
##      Group.1      x
## 1      BIKE LANE 2.330611
## 2      CONTRAFLOW 2.087246
## 3 SHARED BUS BIKE 2.363005
## 4      SHARROW 2.256425
## 5      SIDEPATH 2.781829
## 6      SIGNED ROUTE 2.263746
```

```
### now let's specify the data argument and use a "formula" - much easier to read and
## more "intuitive"
aggregate(log.length ~ type, data=no.missyear, FUN=mean)
```

```
##           type log.length
## 1      BIKE LANE  2.330611
## 2    CONTRAFLOW  2.087246
## 3 SHARED BUS BIKE  2.363005
## 4        SHARROW  2.256425
## 5      SIDEPATH  2.781829
## 6    SIGNED ROUTE  2.263746
```

```
## ddply is from the plyr package
##takes in a data frame, (the first d refers to data.frame)
## splits it up by some variables (let's say type)
## then we'll use summarise to summarize whatever we want
## then returns a data.frame (the second d) - hence why it's ddply
## if we wanted to do it on a "list" thne return data.frame, it'd be ldply
ddply(no.missyear, .(type), summarise,
      mean=mean(log.length)
    )
```

```
##           type      mean
## 1      BIKE LANE 2.330611
## 2    CONTRAFLOW 2.087246
## 3 SHARED BUS BIKE 2.363005
## 4        SHARROW 2.256425
## 5      SIDEPATH 2.781829
## 6    SIGNED ROUTE 2.263746
```

ddply (and other functions in the plyr package) is cool because you can do multiple functions really easy.

Let's show a what if we wanted to go over type and dateInstalled:

```
### For going over 2 variables, we need to do it over a "list" of vectors
tapply(no.missyear$log.length,
      list(no.missyear$type, no.missyear$dateInstalled),
      mean)
```

```
##           2006      2007      2008      2009      2010      2011
## BIKE LANE  3.046261 2.351256 2.365728 2.381418 2.306994 2.242132
## CONTRAFLOW      NA      NA      NA      NA 2.087246      NA
## SHARED BUS BIKE      NA      NA      NA 2.350759 2.403824      NA
## SHARROW      NA 2.300954 2.220850 2.691814 2.247131      NA
## SIDEPATH      NA      NA 2.625486      NA 2.773850 3.266816
## SIGNED ROUTE      NA 2.287593      NA      NA 2.239475 2.210112
##           2012      2013
## BIKE LANE  2.36151 2.408306
## CONTRAFLOW      NA      NA
## SHARED BUS BIKE      NA      NA
## SHARROW      2.23636      NA
## SIDEPATH      NA      NA
## SIGNED ROUTE      NA      NA
```

```
tapply(no.missyear$log.length,
       list(no.missyear$type, no.missyear$dateInstalled),
       mean, na.rm=TRUE)
```

```
##           2006      2007      2008      2009      2010      2011
## BIKE LANE    3.046261 2.351256 2.365728 2.381418 2.306994 2.242132
## CONTRAFLOW      NA      NA      NA      NA 2.087246      NA
## SHARED BUS BIKE      NA      NA      NA 2.350759 2.403824      NA
## SHARROW      NA 2.300954 2.220850 2.691814 2.247131      NA
## SIDEPATH      NA      NA 2.625486      NA 2.773850 3.266816
## SIGNED ROUTE      NA 2.287593      NA      NA 2.239475 2.210112
##           2012      2013
## BIKE LANE    2.36151 2.408306
## CONTRAFLOW      NA      NA
## SHARED BUS BIKE      NA      NA
## SHARROW    2.23636      NA
## SIDEPATH      NA      NA
## SIGNED ROUTE      NA      NA
```

```
## aggregate - looks better
aggregate(log.length ~ type + dateInstalled, data=no.missyear, FUN=mean)
```

```
##           type dateInstalled log.length
## 1      BIKE LANE      2006    3.046261
## 2      BIKE LANE      2007    2.351256
## 3      SHARROW      2007    2.300954
## 4    SIGNED ROUTE      2007    2.287593
## 5      BIKE LANE      2008    2.365728
## 6      SHARROW      2008    2.220850
## 7      SIDEPATH      2008    2.625486
## 8      BIKE LANE      2009    2.381418
## 9    SHARED BUS BIKE      2009    2.350759
## 10     SHARROW      2009    2.691814
## 11     BIKE LANE      2010    2.306994
## 12    CONTRAFLOW      2010    2.087246
## 13    SHARED BUS BIKE      2010    2.403824
## 14     SHARROW      2010    2.247131
## 15     SIDEPATH      2010    2.773850
## 16    SIGNED ROUTE      2010    2.239475
## 17     BIKE LANE      2011    2.242132
## 18     SIDEPATH      2011    3.266816
## 19    SIGNED ROUTE      2011    2.210112
## 20     BIKE LANE      2012    2.361510
## 21     SHARROW      2012    2.236360
## 22     BIKE LANE      2013    2.408306
```

```
## ddply is from the plyr package
ddply(no.missyear, .(type, dateInstalled), summarise,
      mean=mean(log.length),
      median=median(log.length),
      Mode=mode(log.length),
      Std.Dev=sd(log.length)
    )
```

	type	dateInstalled	mean	median	Mode	Std.Dev
## 1	BIKE LANE	2006	3.046261	3.046261	numeric	0.47973544
## 2	BIKE LANE	2007	2.351256	2.444042	numeric	0.40662247
## 3	BIKE LANE	2008	2.365728	2.354641	numeric	0.38916236
## 4	BIKE LANE	2009	2.381418	2.311393	numeric	0.49447436
## 5	BIKE LANE	2010	2.306994	2.328486	numeric	0.32075915
## 6	BIKE LANE	2011	2.242132	2.235462	numeric	0.33397773
## 7	BIKE LANE	2012	2.361510	2.323863	numeric	0.28528097
## 8	BIKE LANE	2013	2.408306	2.505012	numeric	0.24040604
## 9	CONTRAFLOW	2010	2.087246	2.142250	numeric	0.25655109
## 10	SHARED BUS BIKE	2009	2.350759	2.463997	numeric	0.30609512
## 11	SHARED BUS BIKE	2010	2.403824	2.586681	numeric	0.27379952
## 12	SHARROW	2007	2.300954	2.363596	numeric	0.42192796
## 13	SHARROW	2008	2.220850	2.238021	numeric	0.32664161
## 14	SHARROW	2009	2.691814	2.707891	numeric	0.06945133
## 15	SHARROW	2010	2.247131	2.298322	numeric	0.35904709
## 16	SHARROW	2012	2.236360	2.338508	numeric	0.42924259
## 17	SIDEPATH	2008	2.625486	2.786834	numeric	0.29583110
## 18	SIDEPATH	2010	2.773850	2.773850	numeric	0.33479504
## 19	SIDEPATH	2011	3.266816	3.266816	numeric	NA
## 20	SIGNED ROUTE	2007	2.287593	2.331816	numeric	0.41825297
## 21	SIGNED ROUTE	2010	2.239475	2.255658	numeric	0.39200947
## 22	SIGNED ROUTE	2011	2.210112	2.207824	numeric	0.20880213

OK let's do an linear model

```
### type is a character, but when R sees a "character" in a "formula", then it automatically converts it to a factor
### a formula is something that has a y ~ x, which says I want to plot y against x
### or if it were a model you would do y ~ x, which meant regress against y
mod.type = lm(log.length ~ type, data=no.missyear)
mod.yr = lm(log.length ~ factor(dateInstalled), data=no.missyear)
mod.yrtype = lm(log.length ~ type + factor(dateInstalled), data=no.missyear)
summary(mod.type)
```

```
##
## Call:
## lm(formula = log.length ~ type, data = no.missyear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.51498 -0.19062  0.02915  0.23220  1.31021
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      2.33061     0.01487 156.703 < 2e-16 ***
## typeCONTRAFLOW    -0.24337     0.10288  -2.366 0.018127 *
## typeSHARED BUS BIKE  0.03239     0.06062   0.534 0.593194
## typeSHARROW        -0.07419     0.02129  -3.484 0.000509 ***
## typeSIDEPATH        0.45122     0.15058   2.997 0.002775 **
## typeSIGNED ROUTE   -0.06687     0.02726  -2.453 0.014300 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.367 on 1499 degrees of freedom
## Multiple R-squared:  0.01956,    Adjusted R-squared:  0.01629
## F-statistic:  5.98 on 5 and 1499 DF,  p-value: 1.74e-05
```

That's rather UGLY, so let's use a package called `xtable` and then make this model into an `xtable` object and then print it out nicely.

```
### DON'T DO THIS.  YOU SHOULD ALWAYS DO library() statements in the FIRST code chunk.
### this is just to show you the logic of a report/analysis.
require(xtable)
# smod <- summary(mod.yr)
xtab <- xtable(mod.yr)
```

Well `xtable` can make html tables, so let's print this. We must tell R that the results is actually an html output, so we say the results should be embedded in the html "asis" (aka just print out whatever R spits out).

```
print.xtable(xtab, type="html")
```

```
Estimate
Std. Error
t value
Pr(>|t|)
(Intercept)
3.0463
0.2600
11.71
0.0000
factor(dateInstalled)2007
-0.7332
0.2608
-2.81
0.0050
factor(dateInstalled)2008
-0.7808
0.2613
-2.99
0.0029
factor(dateInstalled)2009
-0.6394
0.2631
-2.43
0.0152
```



```

factor(dateInstalled)2010
-0.7791
0.2605
-2.99
0.0028
factor(dateInstalled)2011
-0.8022
0.2626
-3.05
0.0023
factor(dateInstalled)2012
-0.7152
0.2625
-2.72
0.0065
factor(dateInstalled)2013
-0.6380
0.2849
-2.24
0.0253

```

OK, that's pretty good, but let's say we have all three models. Another package called **stargazer** can put models together easily and print them out. So **xtable** is really good when you are trying to print out a table (in html, otherwise make the table and use **write.csv** to get it in Excel and then format) really quickly and in a report. But it doesn't work so well with *many* models together. So let's use stargazer. Again, you need to use **install.packages("stargazer")** if you don't have function.

```
require(stargazer)
```

OK, so what's the difference here? First off, we said results are "markup", so that it will not try to reformat the output. Also, I didn't want those # for comments, so I just made comment an empty string "".

```
stargazer(mod.yr, mod.type, mod.yrtype, type="text")
```

```

=====
                        Dependent variable:
-----
                                log.length
(1)                                (2)                                (3)
-----
factor(dateInstalled)2007      -0.733***                                -0.690***
                                (0.261)                                (0.259)

```

factor(dateInstalled)2008	-0.781*** (0.261)		-0.742*** (0.260)
factor(dateInstalled)2009	-0.639** (0.263)		-0.619** (0.262)
factor(dateInstalled)2010	-0.779*** (0.260)		-0.736*** (0.259)
factor(dateInstalled)2011	-0.802*** (0.263)		-0.790*** (0.261)
factor(dateInstalled)2012	-0.715*** (0.262)		-0.700*** (0.261)
factor(dateInstalled)2013	-0.638** (0.285)		-0.638** (0.283)
typeCONTRAFLOW		-0.243** (0.103)	-0.224** (0.103)
typeSHARED BUS BIKE		0.032 (0.061)	-0.037 (0.069)
typeSHARROW		-0.074*** (0.021)	-0.064*** (0.023)
typeSIDEPATH		0.451*** (0.151)	0.483*** (0.150)
typeSIGNED ROUTE		-0.067** (0.027)	-0.067** (0.029)
Constant	3.046*** (0.260)	2.331*** (0.015)	3.046*** (0.258)

Observations	1,505	1,505	1,505
R2	0.017	0.020	0.033
Adjusted R2	0.012	0.016	0.026
Residual Std. Error	0.368 (df = 1497)	0.367 (df = 1499)	0.365 (df = 1492)
F Statistic	3.691*** (df = 7; 1497)	5.980*** (df = 5; 1499)	4.285*** (df = 12; 1492)

=====

Note: \*p<0.1; \*\*p<0.05; \*\*\*p<0.01

If we use

```
stargazer(mod.yr, mod.type, mod.yrtype, type="html")
```

Dependent variable:

log.length

(1)

(2)

(3)

factor(dateInstalled)2007

-0.733\*\*\*

-0.690\*\*\*

(0.261)

(0.259)

factor(dateInstalled)2008

-0.781\*\*\*

-0.742\*\*\*

(0.261)

(0.260)

factor(dateInstalled)2009

-0.639\*\*

-0.619\*\*

(0.263)

(0.262)

factor(dateInstalled)2010

-0.779\*\*\*

-0.736\*\*\*

(0.260)

(0.259)

factor(dateInstalled)2011

-0.802\*\*\*

-0.790\*\*\*

(0.263)

(0.261)

factor(dateInstalled)2012

-0.715\*\*\*

-0.700\*\*\*

(0.262)

(0.261)

factor(dateInstalled)2013

-0.638\*\*

-0.638\*\*

(0.285)

(0.283)  
 typeCONTRAFLOW  
 -0.243\*\*  
 -0.224\*\*  
 (0.103)  
 (0.103)  
 typeSHARED BUS BIKE  
 0.032  
 -0.037  
 (0.061)  
 (0.069)  
 typeSHARROW  
 -0.074\*\*\*  
 -0.064\*\*\*  
 (0.021)  
 (0.023)  
 typeSIDEPATH  
 0.451\*\*\*  
 0.483\*\*\*  
 (0.151)  
 (0.150)  
 typeSIGNED ROUTE  
 -0.067\*\*  
 -0.067\*\*  
 (0.027)  
 (0.029)  
 Constant  
 3.046\*\*\*  
 2.331\*\*\*  
 3.046\*\*\*  
 (0.260)  
 (0.015)  
 (0.258)  
 Observations  
 1,505  
 1,505

```

1,505
R2
0.017
0.020
0.033
Adjusted R2
0.012
0.016
0.026
Residual Std. Error
0.368 (df = 1497)
0.367 (df = 1499)
0.365 (df = 1492)
F Statistic
3.691*** (df = 7; 1497)
5.980*** (df = 5; 1499)
4.285*** (df = 12; 1492)
Note:
 $p < 0.1$ ;  $p < 0.05$ ;  $p < 0.01$ 

```

## Data Extraction

Let's say I want to get data INTO my text. Like there are N number of bike lanes with a date installed that isn't zero. There are 1505 bike lanes with a date installed after 2006. So you use one backtick ' and then you say "r" to tell that it's R code. And then you run R code that gets evaluated and then returns the value. Let's say you want to compute a bunch of things:

```

### let's get number of bike lanes installed by year
n.lanes = ddply(no.missyear, .(dateInstalled), nrow)
names(n.lanes) <- c("date", "nlanes")
n2009 <- n.lanes$nlanes[ n.lanes$date == 2009]
n2010 <- n.lanes$nlanes[ n.lanes$date == 2010]
getwd()

```

```
## [1] "/Users/johnmuschelli/Dropbox/Classes/winterR_2016/Knitr/lecture"
```

Now I can just say there are 86 lanes in 2009 and 625 in 2010.

```

fname <- "../data/Charm_City_Circulator_Ridership.csv"
# fname <- file.path(data.dir, "Charm_City_Circulator_Ridership.csv")
## file.path takes a directory and makes a full name with a full file path
charm = read.csv(fname, as.is=TRUE)

```

```
library(chron)
days = levels(weekdays(1, abbreviate=FALSE))
charm$day <- factor(charm$day, levels=days)
charm$date <- as.Date(charm$date, format="%m/%d/%Y")
cn <- colnames(charm)
daily <- charm[, c("day", "date", "daily")]
```

```
charm$daily <- NULL
require(reshape2)
long.charm <- melt(charm, id.vars = c("day", "date"))
long.charm$type <- "Boardings"
long.charm$type[ grepl("Alightings", long.charm$variable)] <- "Alightings"
long.charm$type[ grepl("Average", long.charm$variable)] <- "Average"

long.charm$line <- "orange"
long.charm$line[ grepl("purple", long.charm$variable)] <- "purple"
long.charm$line[ grepl("green", long.charm$variable)] <- "green"
long.charm$line[ grepl("banner", long.charm$variable)] <- "banner"
long.charm$variable <- NULL

long.charm$line <-factor(long.charm$line, levels=c("orange", "purple",
                                                  "green", "banner"))

head(long.charm)
```

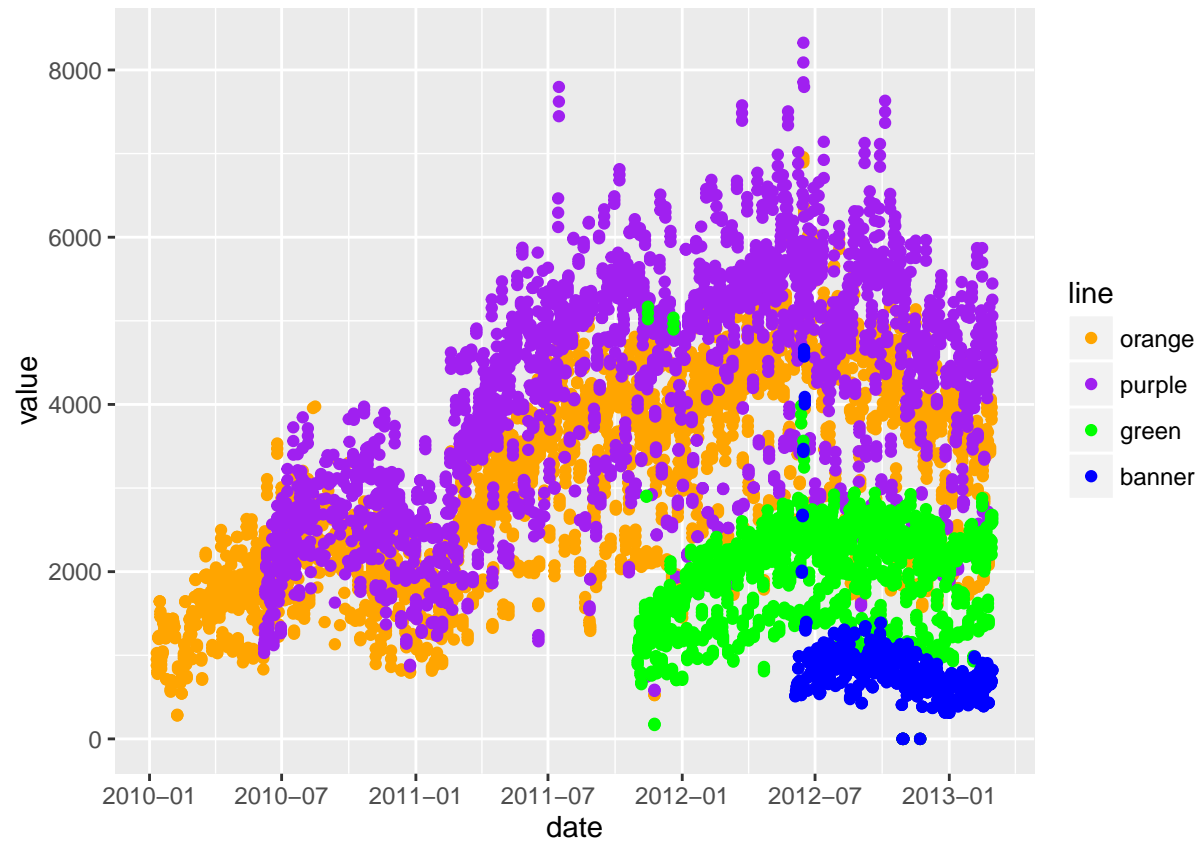
```
##      day      date value      type  line
## 1  Monday 2010-01-11   877 Boardings orange
## 2  Tuesday 2010-01-12   777 Boardings orange
## 3 Wednesday 2010-01-13  1203 Boardings orange
## 4 Thursday 2010-01-14  1194 Boardings orange
## 5   Friday 2010-01-15  1645 Boardings orange
## 6 Saturday 2010-01-16  1457 Boardings orange
```

```
### NOW R has a column of day, the date, a "value", the type of value and the
### circulator line that corresponds to it
### value is now either the Alightings, Boardings, or Average from the charm dataset
```

Let's do some plotting now!

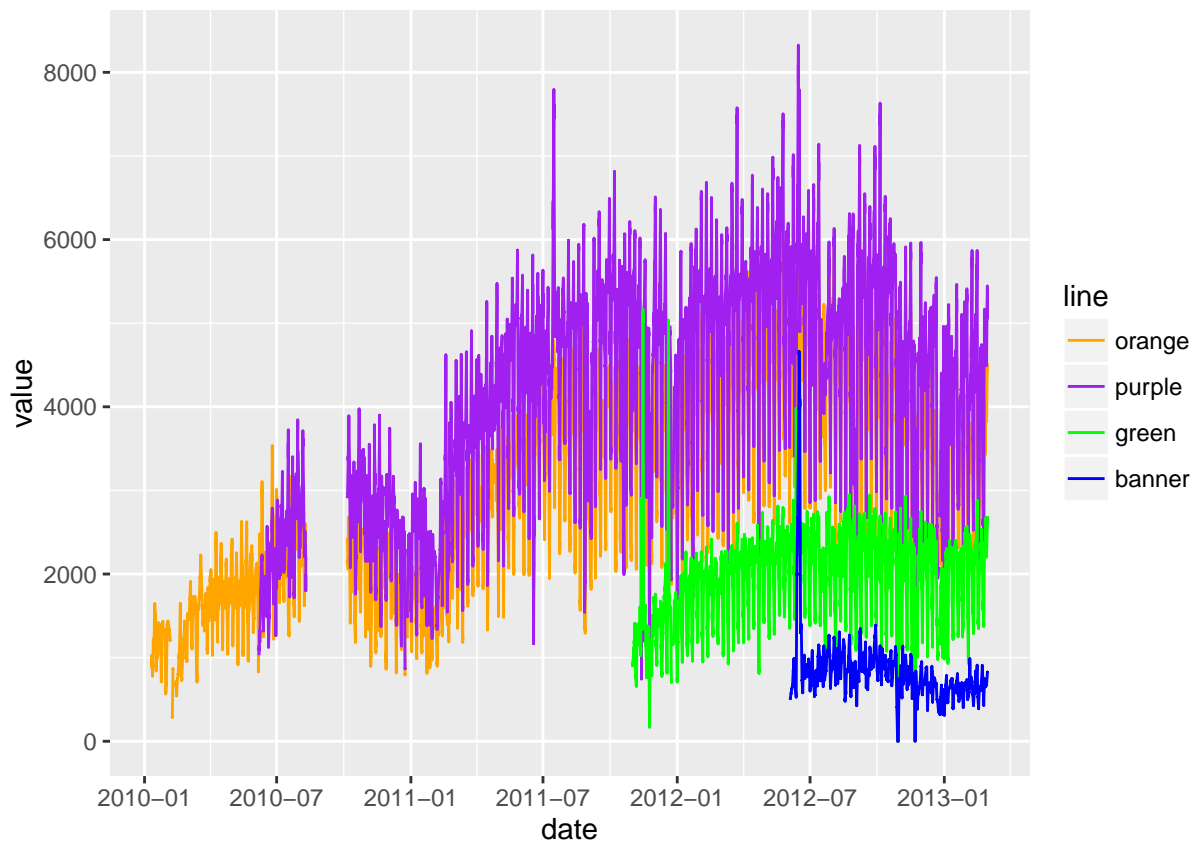
```
require(ggplot2)
### let's make a "ggplot"
### the format is ggplot(dataframe, aes(x=COLNAME, y=COLNAME))
### where COLNAME are colnames of the dataframe
### you can also set color to a different factor
### other options in AES (fill, alpha level -which is the "transparency" of points)
g <- ggplot(long.charm, aes(x=date, y=value, color=line))
### let's change the colors to what we want- doing this manually, not letting it choose
### for me
g <- g + scale_color_manual(values=c("orange", "purple", "green", "blue"))
### plotting points
g + geom_point()
```

```
## Warning: Removed 5328 rows containing missing values (geom_point).
```



```
### Let's make Lines!  
g + geom_line()
```

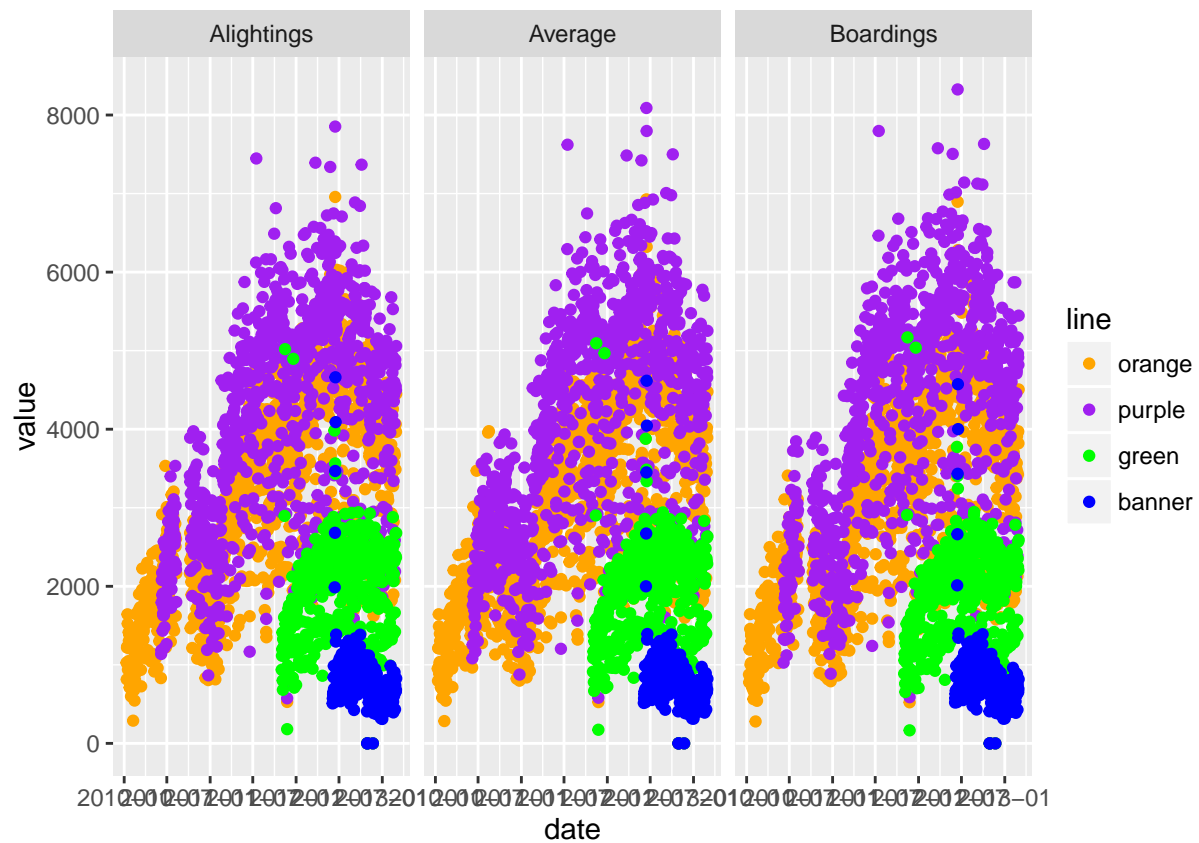
```
## Warning: Removed 5043 rows containing missing values (geom_path).
```



```
### let's make a new plot of poitns
gpoint <- g + geom_point()
### let's plot the value by the type of value - boardings/average, etc
gpoint + facet_wrap(~ type)
```

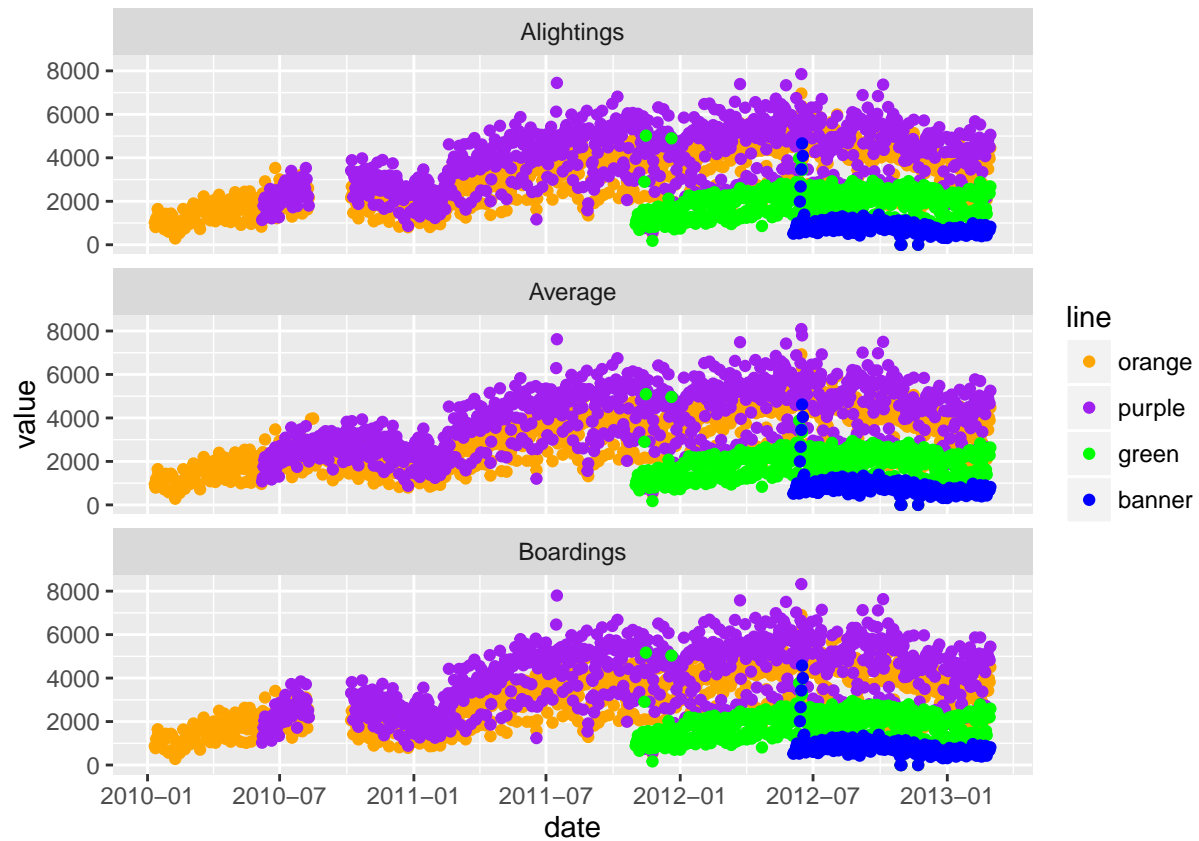
```
## Warning: Removed 5328 rows containing missing values (geom_point).
```





OK let's turn off some warnings - making `warning=FALSE` (in knitr) as an option.

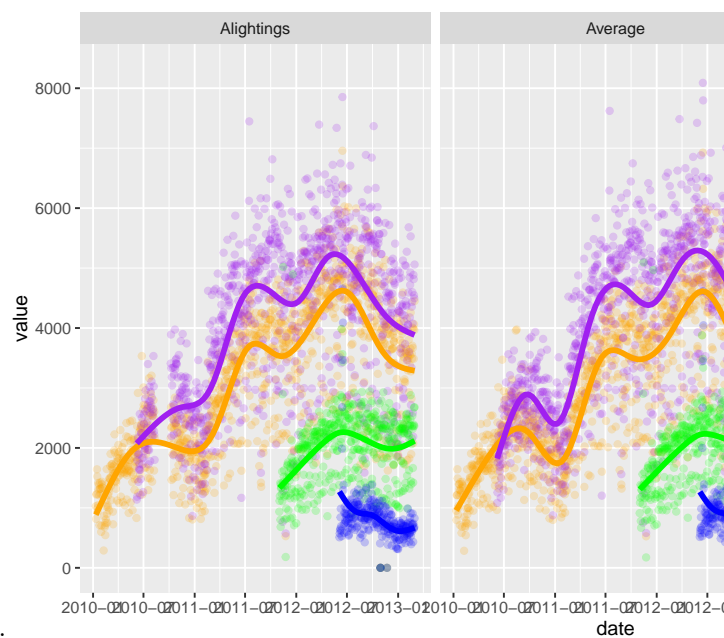
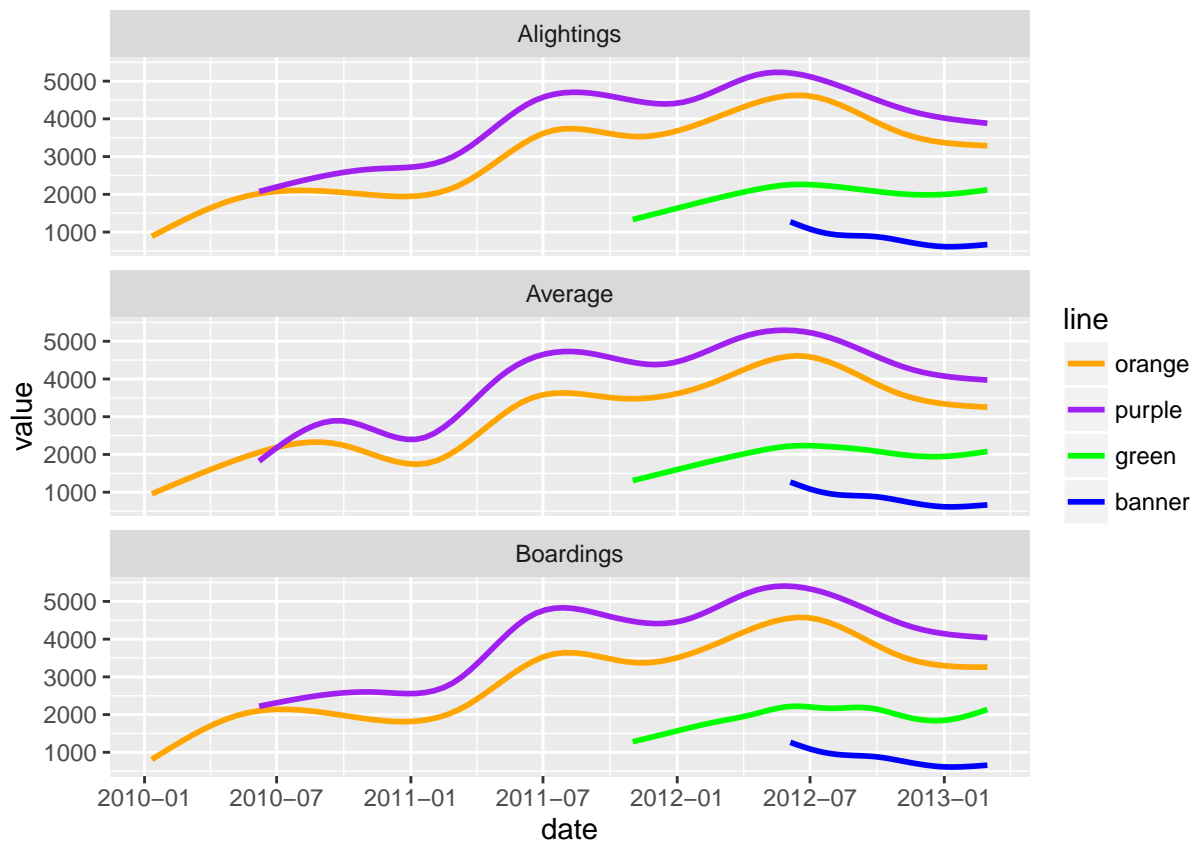
```
## let's compare vertically
gpoint + facet_wrap(~ type, ncol=1)
```



```
gfacet = g + facet_wrap(~ type, ncol=1)
```

We can also smooth the data to give us a overall idea of how the average changes over time. I don't want to do a standard error (se).

```
## let's smooth this - get a rough estimate of what's going on
gfacet + geom_smooth(se=FALSE)
```



OK, I've seen enough code, let's turn that off, using `echo=FALSE`.

There are still messages, but we can turn these off with `message = FALSE`

