Lecture 4

Visualizing and Statistics

Andrew Jaffe
Instructor

Today

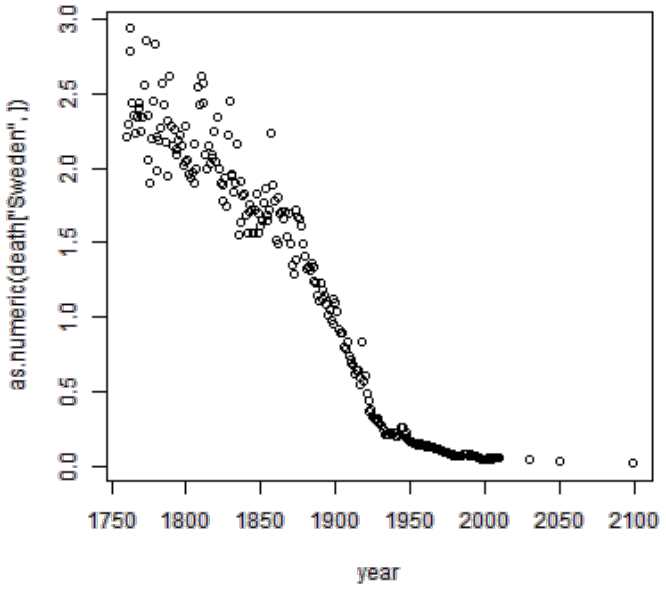
Today we are going to cover more advanced visualization and some useful statistics functions

Basic Plots

We covered some basic plots on Tuesday, but we are going to expand the ability to customize these basic graphics first.

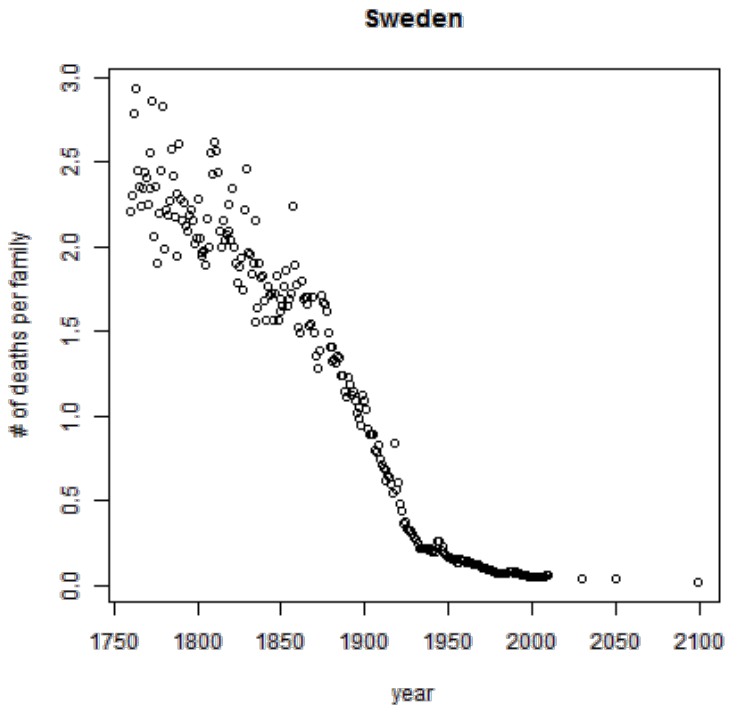
Basic Plots

```
> death = read.csv("http://biostat.jhsph.edu/~ajaffe/files/indicatordeadkids35.csv",
+     as.is = T, header = TRUE, row.names = 1)
> year = as.integer(gsub("X", "", names(death)))
> plot(as.numeric(death["Sweden", ]) ~ year)
```



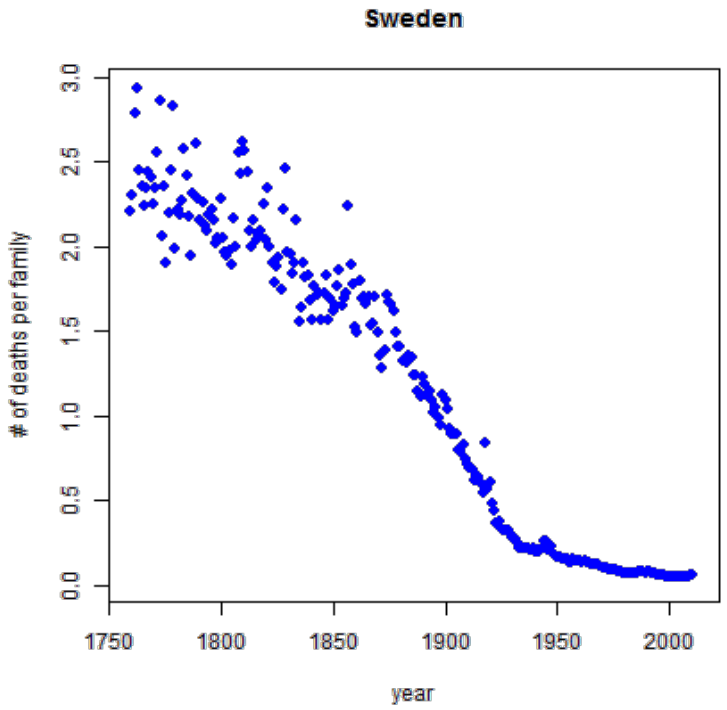
Basic Plots

```
> plot(as.numeric(death["Sweden", ]) ~ year, ylab = "# of deaths per family",
+     main = "Sweden")
```



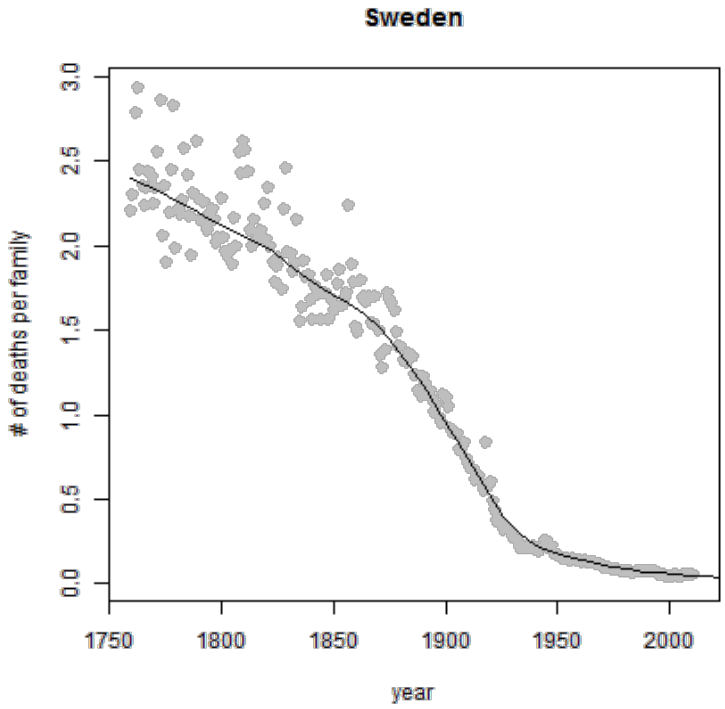
Basic Plots

```
> plot(as.numeric(death["Sweden", ]) ~ year, ylab = "# of deaths per family",
+     main = "Sweden", xlim = c(1760, 2012), pch = 19, cex = 1.2, col = "blue")
```



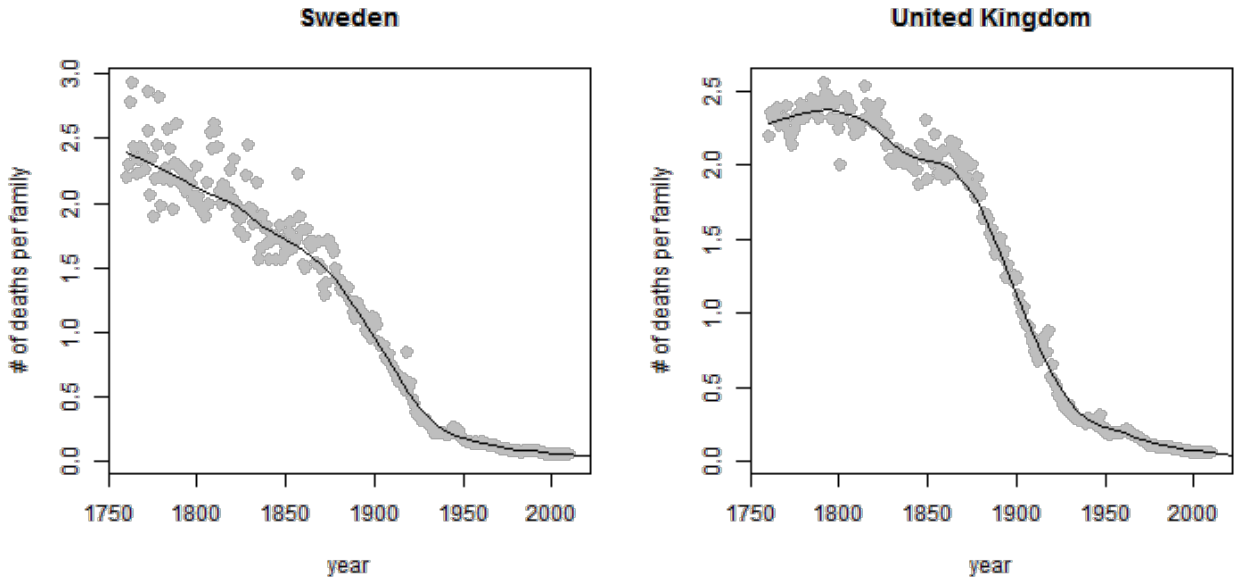
Basic Plots

```
> scatter.smooth(as.numeric(death["Sweden", ]) ~ year, span = 0.2, ylab = "# of deaths per family",
+     main = "Sweden", lwd = 3, xlim = c(1760, 2012), pch = 19, cex = 0.9, col = "grey")
```



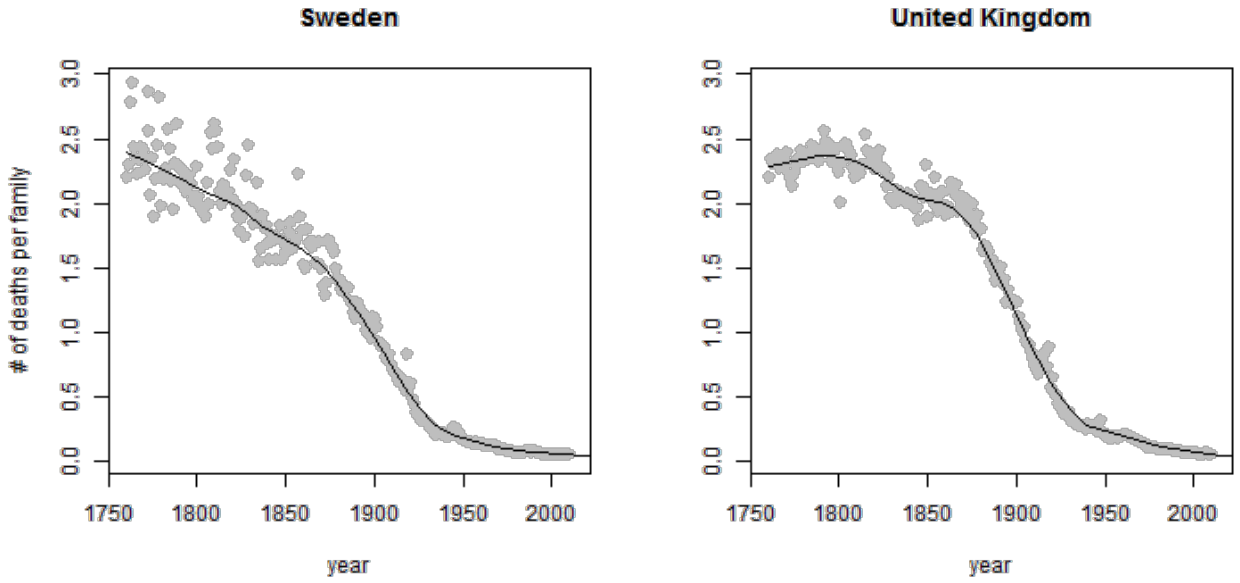
Basic Plots

```
> par(mfrow = c(1, 2))
> scatter.smooth(as.numeric(death["Sweden", ]) ~ year, span = 0.2, ylab = "# of deaths per family",
+     main = "Sweden", lwd = 3, xlim = c(1760, 2012), pch = 19, cex = 0.9, col = "grey")
> scatter.smooth(as.numeric(death["United Kingdom", ]) ~ year, span = 0.2, ylab = "# of deaths per famil
+     main = "United Kingdom", lwd = 3, xlim = c(1760, 2012), pch = 19, cex = 0.9,
+     col = "grey")
```



Basic Plots

```
> par(mfrow = c(1, 2))
> yl = range(death[c("Sweden", "United Kingdom"), ])
> scatter.smooth(as.numeric(death["Sweden", ]) ~ year, span = 0.2, ylim = yl,
+     ylab = "# of deaths per family", main = "Sweden", lwd = 3, xlim = c(1760,
+         2012), pch = 19, cex = 0.9, col = "grey")
> scatter.smooth(as.numeric(death["United Kingdom", ]) ~ year, span = 0.2, ylab = "",
+     main = "United Kingdom", lwd = 3, ylim = yl, xlim = c(1760, 2012), pch = 19,
+     cex = 0.9, col = "grey")
```



Graphical parameters

par() can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to par in tag = value form, or by passing them as a list of tagged values.

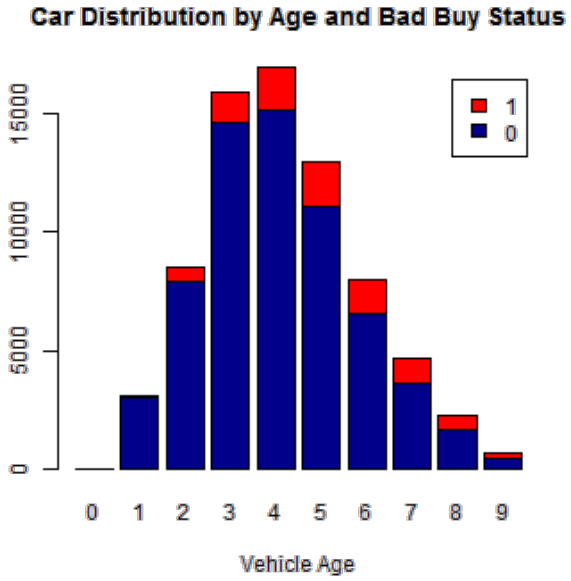
Note that some parameters are passed from plot(,...) calls whereas others need to be explicitly set using par() - like above with par(mfrow = c(nrow,ncol))

Note that some parameters are both very flexible but also very finicky, especially margins.

Bar Plots

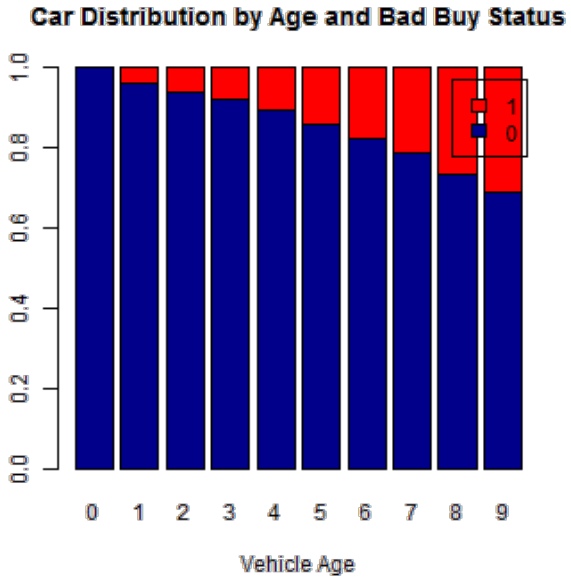
- Stacked Bar Charts are sometimes wanted to show how

```
> ## Stacked Bar Charts
> cars = read.csv("http://biostat.jhsph.edu/~ajaffe/files/kaggleCarAuction.csv",
+     as.is = T)
> counts <- table(cars$IsBadBuy, cars$VehicleAge)
> barplot(counts, main = "Car Distribution by Age and Bad Buy Status", xlab = "Vehicle Age",
+     col = c("darkblue", "red"), legend = rownames(counts))
```



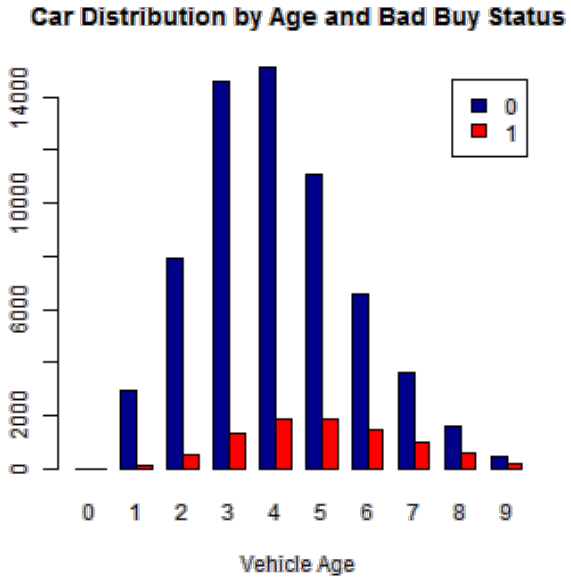
Bar Plots

```
> ## Use percentages (column percentages)
> barplot(prop.table(counts, 2), main = "Car Distribution by Age and Bad Buy Status",
+     xlab = "Vehicle Age", col = c("darkblue", "red"), legend = rownames(counts))
```



Bar Plots

```
> # Stacked Bar Plot with Colors and Legend
> barplot(counts, main = "Car Distribution by Age and Bad Buy Status", xlab = "Vehicle Age",
+     col = c("darkblue", "red"), legend = rownames(counts), beside = TRUE)
```



Graphics parameters

Set within most plots in the base 'graphics' package:

- pch = point shape, http://voteview.com/symbols_pch.htm
- cex = size/scale
- xlab, ylab = labels for x and y axes
- main = plot title
- lwd = line density
- col = color
- cex.axis, cex.lab, cex.main = scaling/sizing for axes marks, axes labels, and title

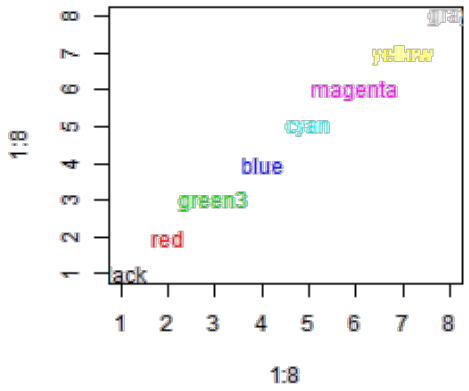
Colors

R relies on color 'palettes'.

```
> palette()
```

```
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

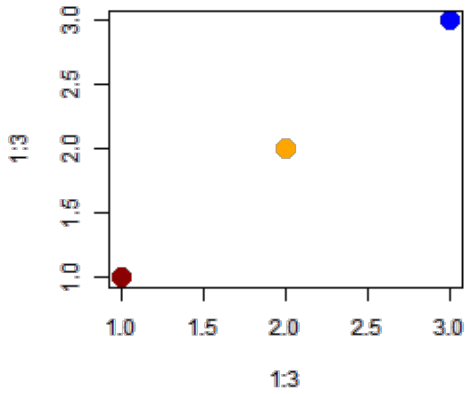
```
> plot(1:8, 1:8, type = "n")
> text(1:8, 1:8, lab = palette(), col = 1:8)
```



Colors

The default color palette is pretty bad, so you can try to make your own.

```
> palette(c("darkred", "orange", "blue"))
> plot(1:3, 1:3, col = 1:3, pch = 19, cex = 2)
```



Colors

It's actually pretty hard to make a good color palette. Luckily, smart and artistic people have spent a lot more time thinking about this. The result is the 'RColorBrewer' package

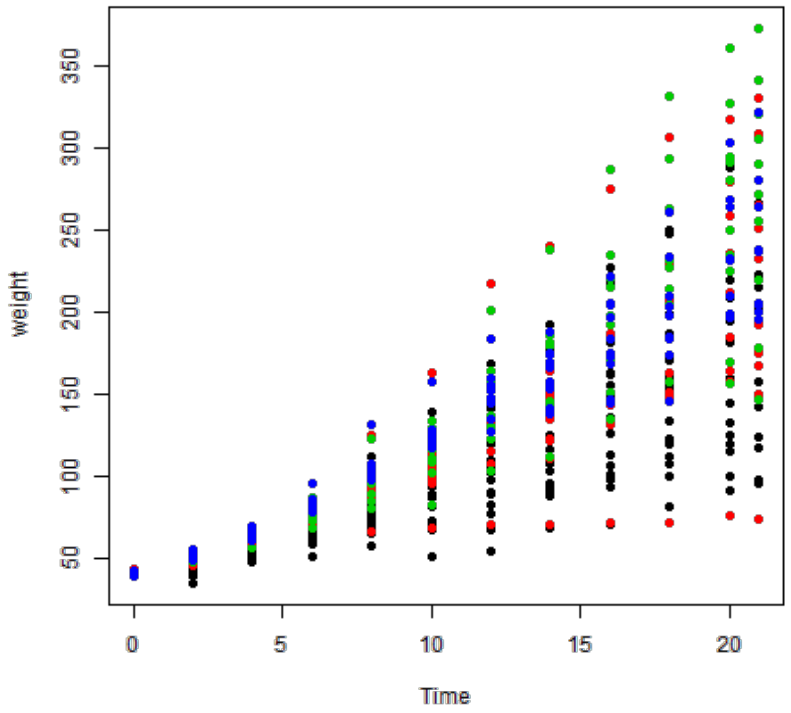
RColorBrewer::display.brewer.all() will show you all of the palettes available. You can even print it out and keep it next to your monitor for reference.

The help file for brewer.pal() gives you an idea how to use the package.

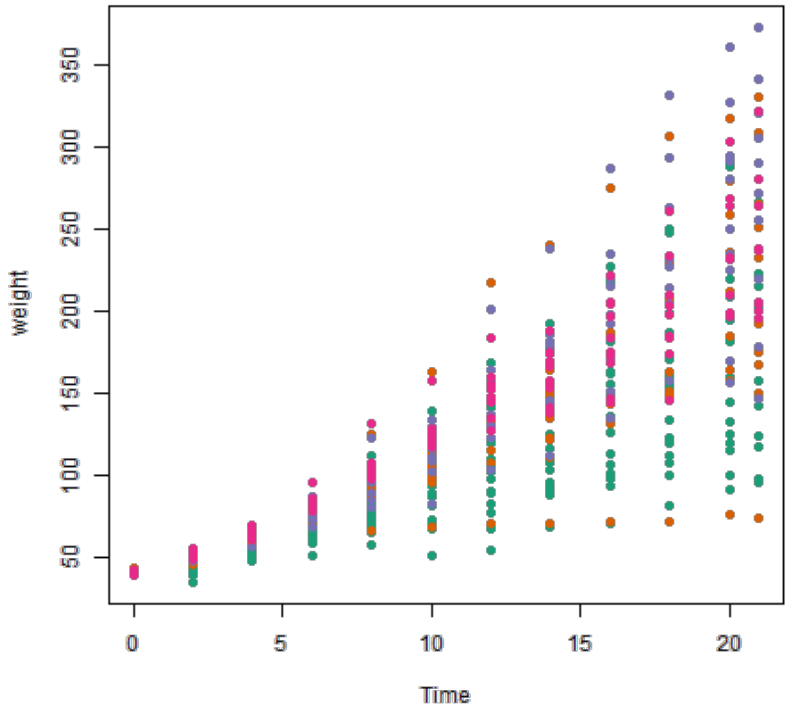
You can also get a "sneak peek" of these palettes at: www.colorbrewer2.com . You would provide the number of levels or classes of your data, and then the type of data: sequential, diverging, or qualitative. The names of the RColorBrewer palettes are the string after 'pick a color scheme:'

Colors

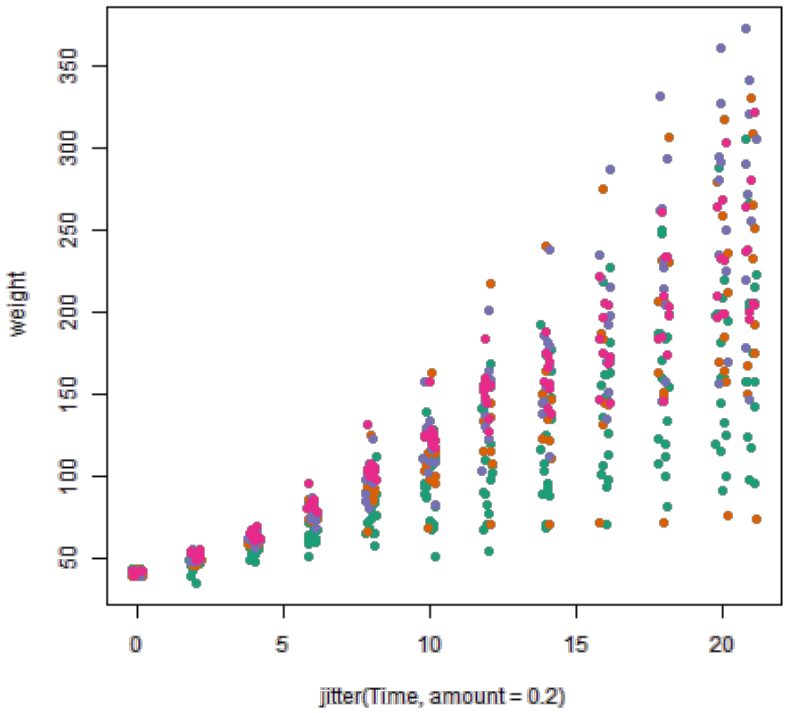
```
> palette("default")
> with(ChickWeight, plot(weight ~ Time, pch = 19, col = Diet))
```



```
> library(RColorBrewer)
> palette(brewer.pal(5, "Dark2"))
> with(ChickWeight, plot(weight ~ Time, pch = 19, col = Diet))
```



```
> library(RColorBrewer)
> palette(brewer.pal(5, "Dark2"))
> with(ChickWeight, plot(weight ~ jitter(Time, amount = 0.2), pch = 19, col = Diet),
+     xlab = "Time")
```



Adding legends

The legend() command adds a legend to your plot. There are tons of arguments to pass it.

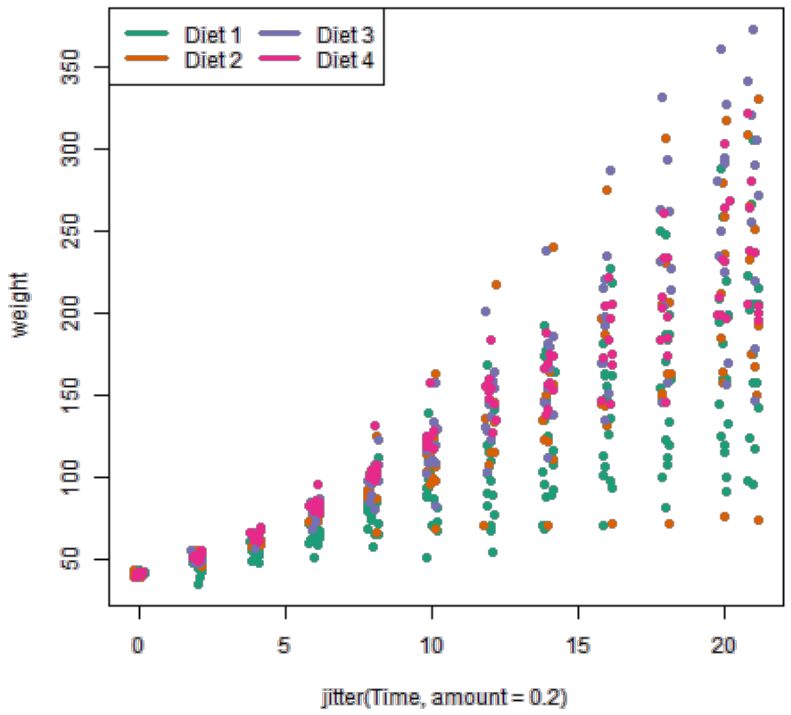
x, y=NULL: this just means you can give (x,y) coordinates, or more commonly just give x, as a character string: "top","bottom","topleft","bottomleft","topright","bottomright".

legend: unique character vector, the levels of a factor

pch, lwd: if you want points in the legend, give a pch value. if you want lines, give a lwd value.

col: give the color for each legend level

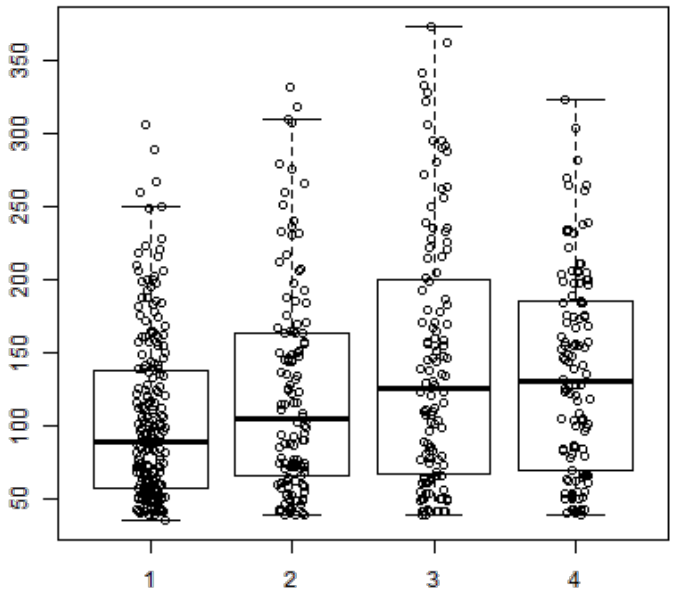
```
> palette(brewer.pal(5, "Dark2"))
> with(ChickWeight, plot(weight ~ jitter(Time, amount = 0.2), pch = 19, col = Diet),
+     xlab = "Time")
> legend("topleft", paste("Diet", levels(ChickWeight$Diet)), col = 1:length(levels(ChickWeight$Diet)),
+     lwd = 3, ncol = 2)
```



Boxplots, revisited

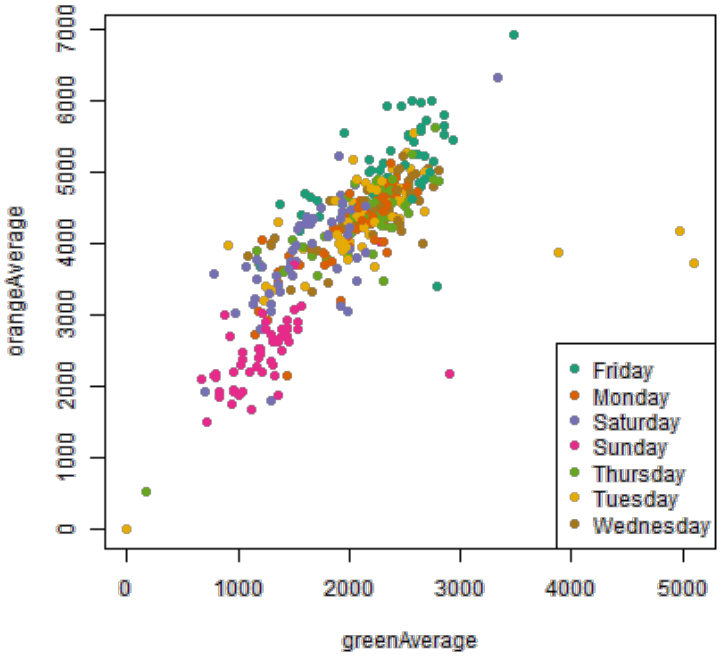
These are one of my favorite plots. They are way more informative than the barchart + antenna...

```
> with(ChickWeight, boxplot(weight ~ Diet, outline = FALSE))
> points(ChickWeight$weight ~ jitter(as.numeric(ChickWeight$Diet), 0.5))
```



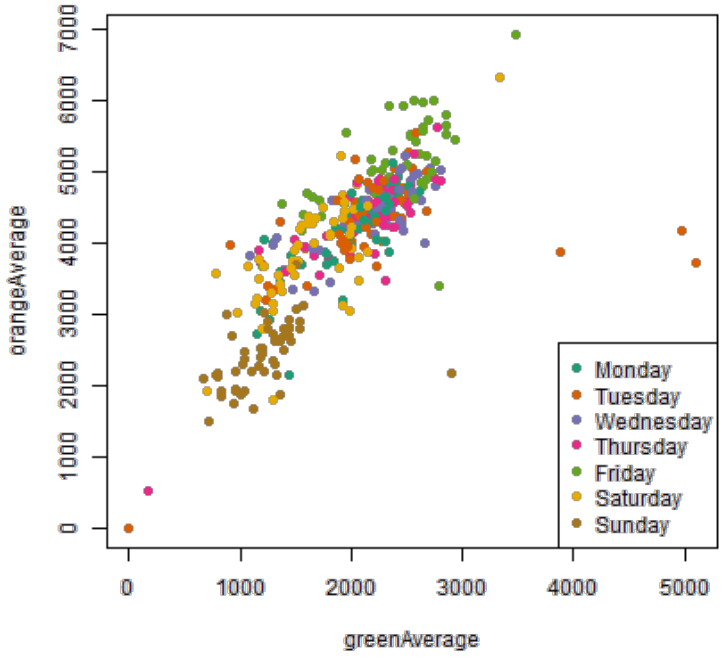
Coloring by variable

```
> load("../lecture2/charmcirc.rda")
> palette(brewer.pal(7, "Dark2"))
> dd = factor(dat$day)
> with(dat, plot(orangeAverage ~ greenAverage, pch = 19, col = as.numeric(dd)))
> legend("bottomright", levels(dd), col = 1:length(dd), pch = 19)
```



Coloring by variable

```
> dd = factor(dat$day, levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
+     "Friday", "Saturday", "Sunday"))
> with(dat, plot(orangeAverage ~ greenAverage, pch = 19, col = as.numeric(dd)))
> legend("bottomright", levels(dd), col = 1:length(dd), pch = 19)
```



Devices

By default, R displays plots in a separate panel. From there, you can export the plot to a variety of image file types, or copy it to the clipboard.

However, sometimes its very nice to save many plots made at one time to one pdf file, say, for flipping through. Or being more precise with the plot size in the saved file.

R has 5 additional graphics devices: bmp(), jpeg(), png(), tiff(), and pdf()

The syntax is very similar for all of them:

```
pdf("filename.pdf", width=8, height=8) # inches
plot() # plot 1
plot() # plot 2
# etc
dev.off()
```

Basically, you are creating a pdf file, and telling R to write any subsequent plots to that file. Once you are done, you turn the device off. Note that failing to turn the device off will create a pdf file that is corrupt, that you cannot open.

More powerful graphics

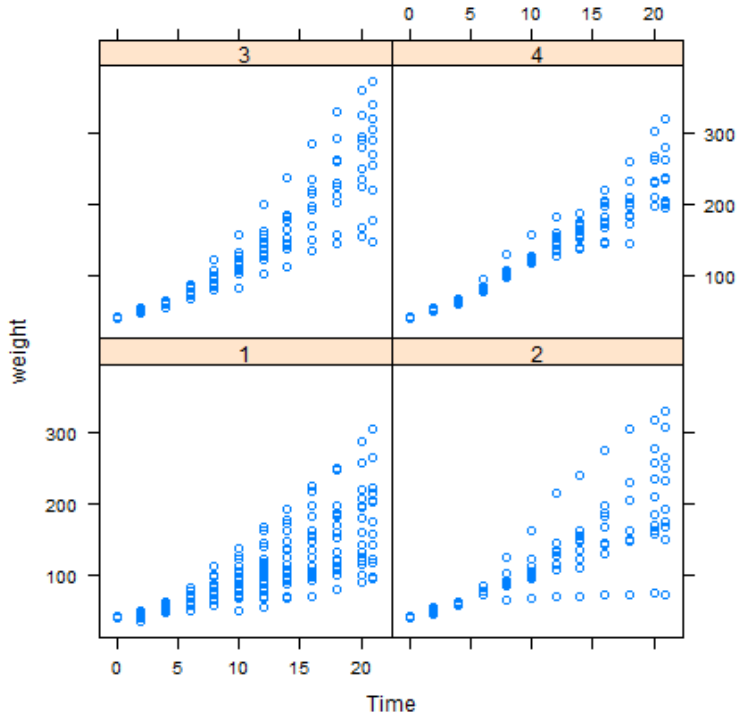
There are two very common packages for making very nice looking graphics.

lattice: http://lmdvr.r-forge.r-project.org/figures/figures.html

ggplot2: http://docs.ggplot2.org/current/index.html

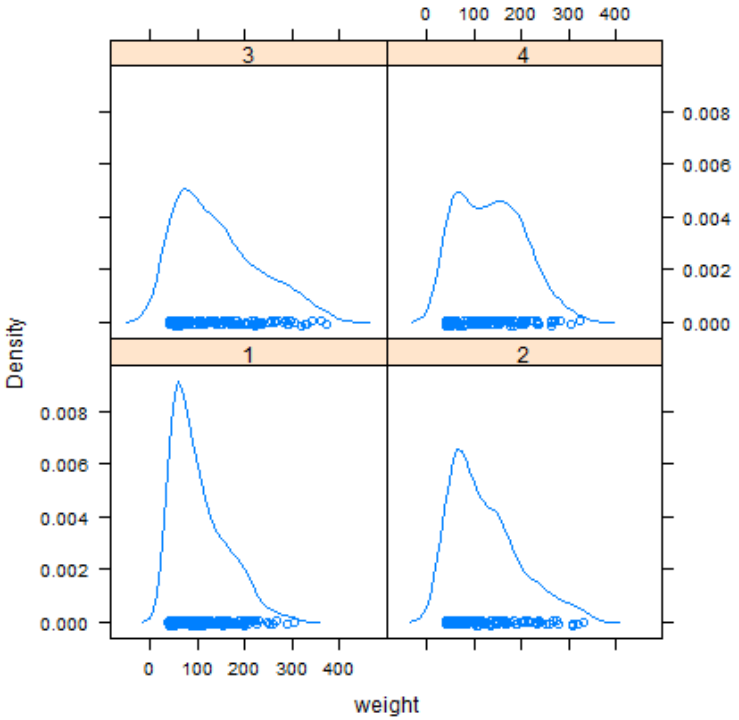
Lattice

```
> library(lattice)
> xyplot(weight ~ Time | Diet, data = ChickWeight)
```



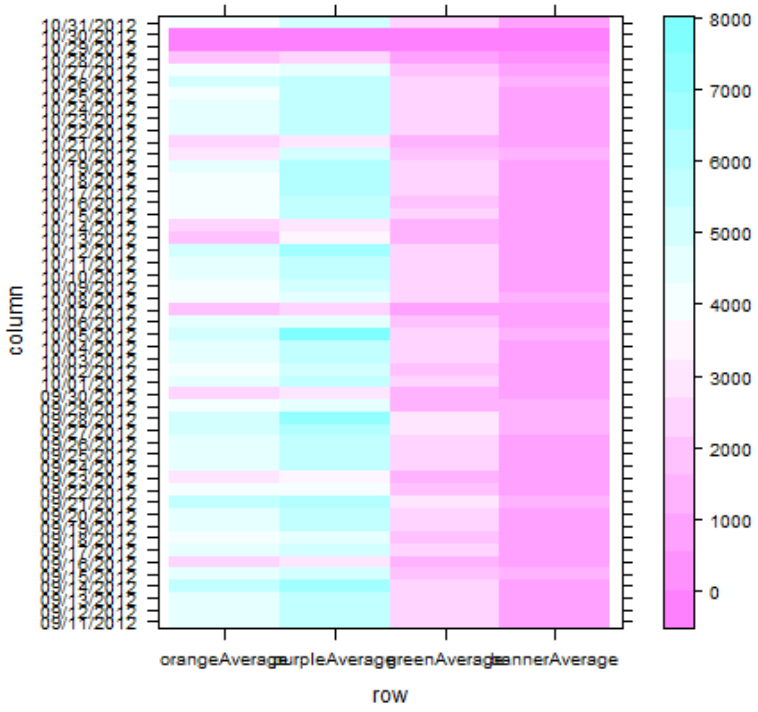
Lattice

```
> densityplot(~weight | Diet, data = ChickWeight)
```



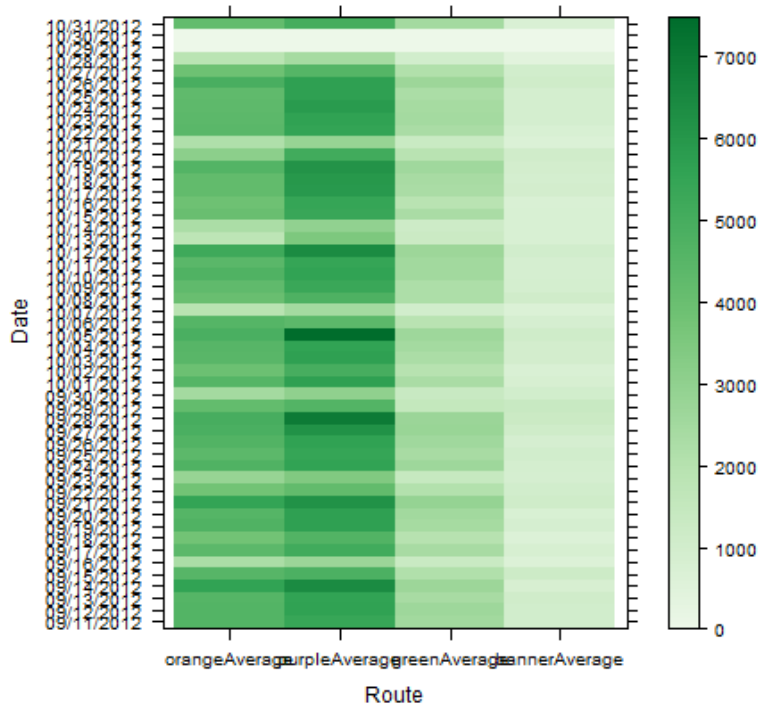
Lattice

```
> rownames(dat2) = dat2$date
> mat = as.matrix(dat2[975:nrow(dat2), 3:6])
> levelplot(t(mat), aspect = "fill")
```



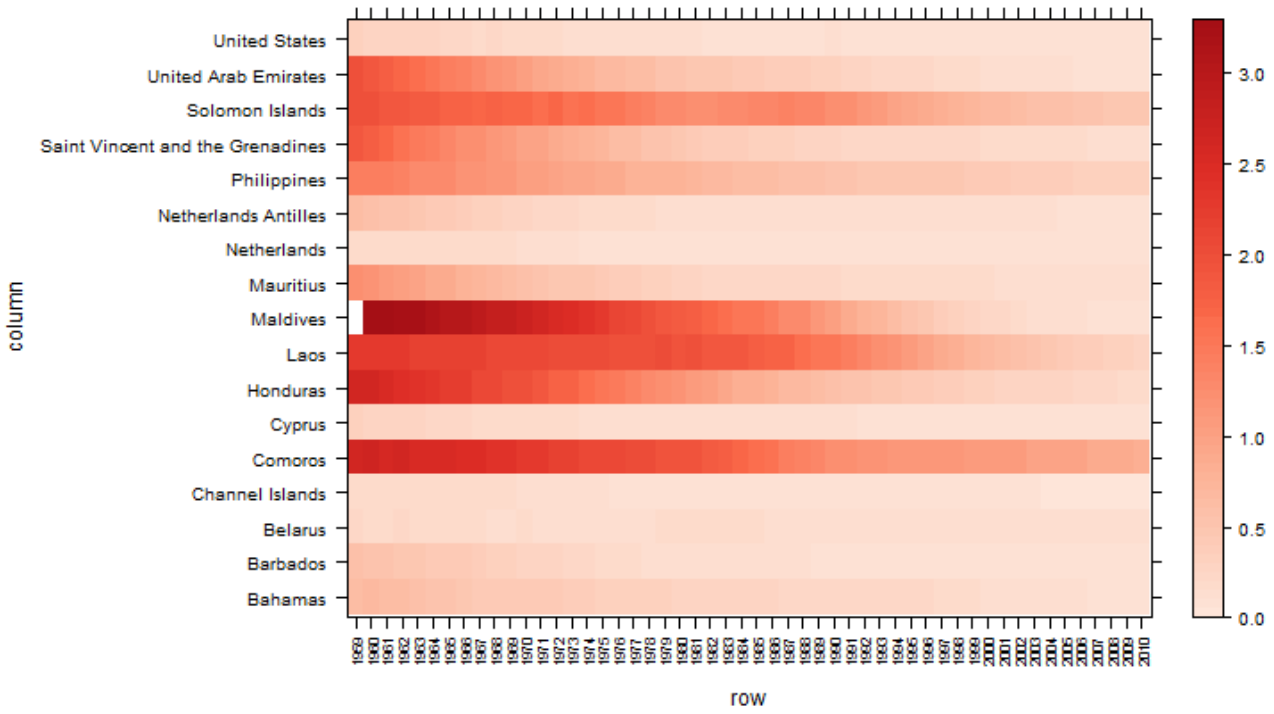
Lattice

```
> theSeq = seq(0, max(mat), by = 50)
> my.col <- colorRampPalette(brewer.pal(5, "Greens"))(length(theSeq))
> levelplot(t(mat), aspect = "fill", at = theSeq, col.regions = my.col, xlab = "Route",
+     ylab = "Date")
```



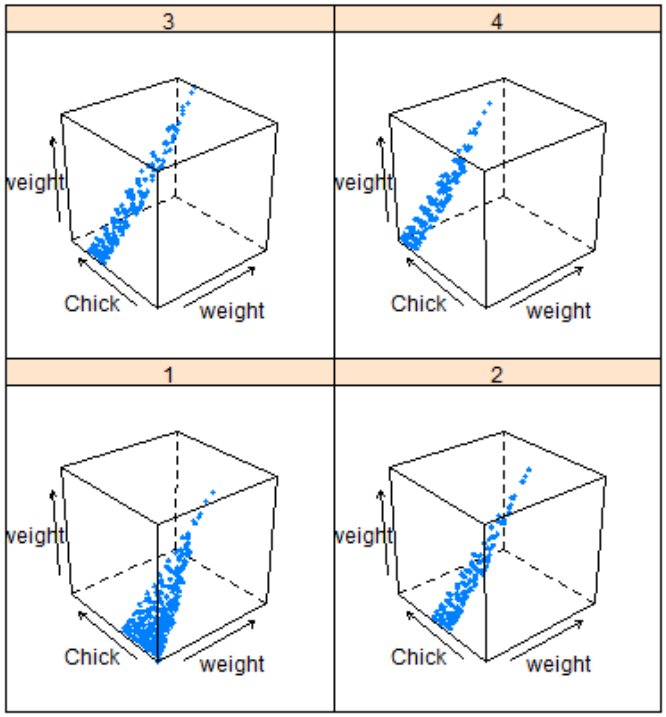
Lattice

```
> tmp = death[grep("s$", rownames(death)), 200:251]
> yr = gsub("X", "", names(tmp))
> theSeq = seq(0, max(tmp, na.rm = TRUE), by = 0.05)
> my.col <- colorRampPalette(brewer.pal(5, "Reds"))(length(theSeq))
> levelplot(t(tmp), aspect = "fill", at = theSeq, col.regions = my.col, scales = list(x = list(label = y
+     rot = 90, cex = 0.7)))
```



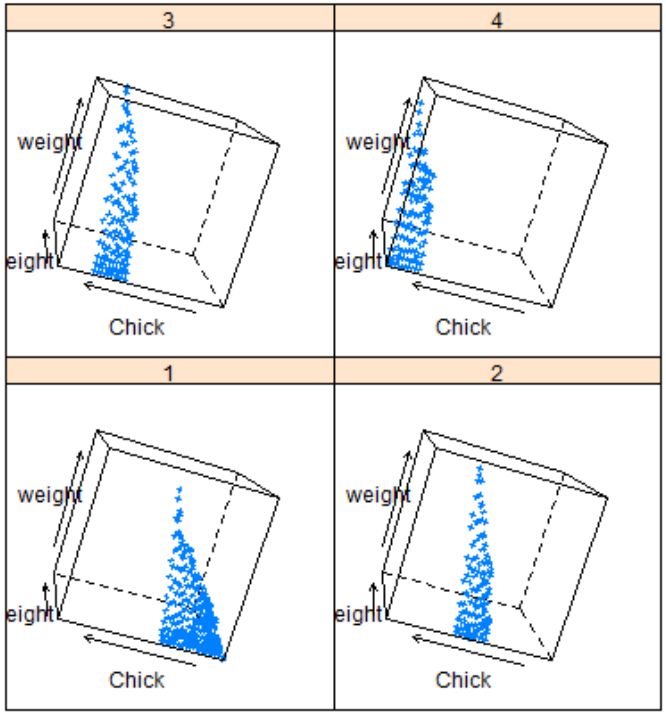
Lattice

```
> cloud(weight ~ weight * Chick | Diet, data = ChickWeight)
```



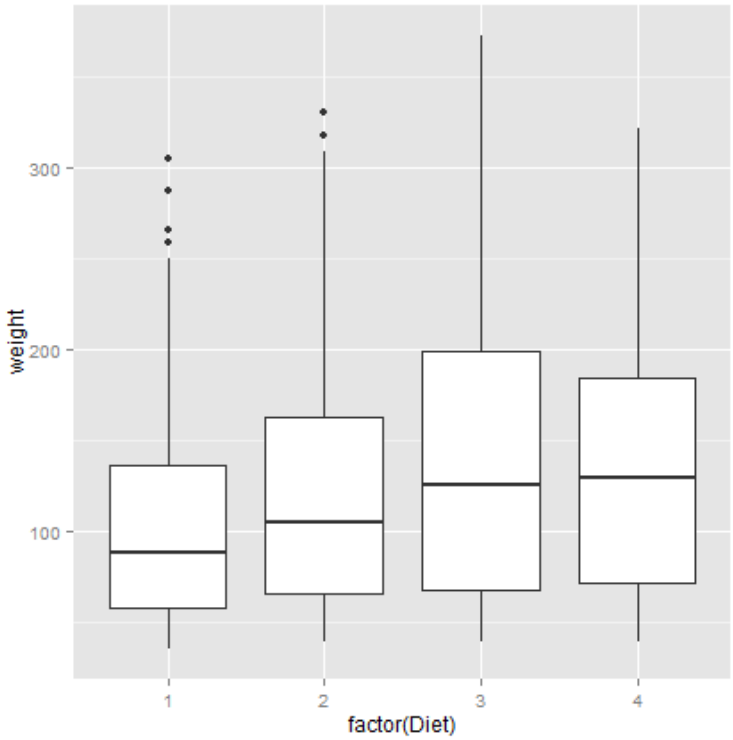
Lattice

```
> cloud(weight ~ weight * Chick | Diet, data = ChickWeight, screen = list(z = 40,
+     x = -70, y = 60))
```



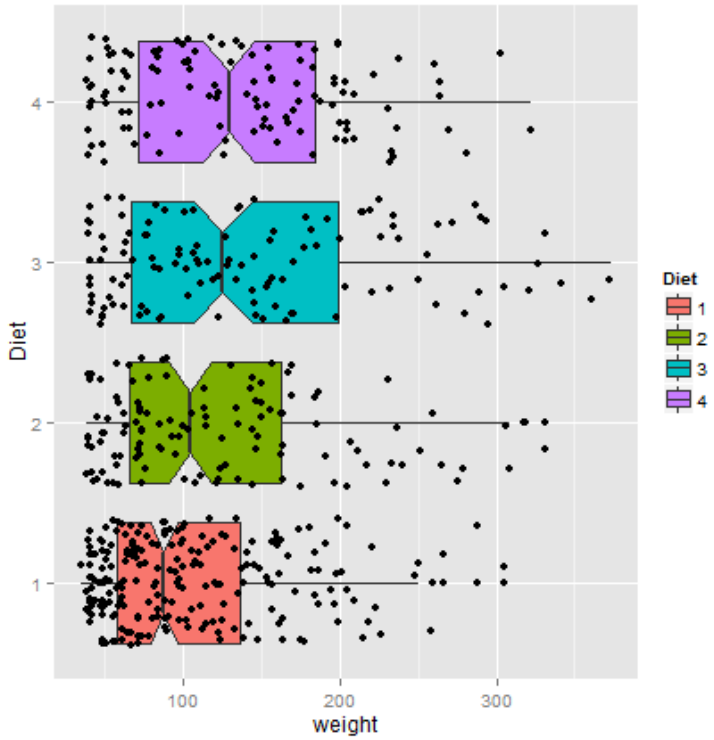
ggplot2

```
> library(ggplot2)
> qplot(factor(Diet), weight, data = ChickWeight, geom = "boxplot")
```



ggplot2

```
> p = ggplot(ChickWeight, aes(Diet, weight))
> p + geom_boxplot(notch = TRUE, aes(fill = Diet)) + geom_jitter() + coord_flip()
```



Statistics

Now we are going to cover how to perform a variety of basic statistical tests in R.

- Correlation
- T-tests
- Proportion tests
- Chi-squared
- Fisher's Exact Test
- Linear Regression

Note: We will be glossing over the statistical theory and "formulas" for these tests. There are plenty of resources online for learning more about these tests, as well as dedicated Biostatistics series at the School of Public Health

Correlation

cor() performs correlation in R

```
cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
```

```
> cor(dat2$orangeAverage, dat2$purpleAverage)
```

```
[1] NA
```

```
> cor(dat2$orangeAverage, dat2$purpleAverage, use = "complete.obs")
```

```
[1] 0.9208
```

Correlation

You can also get the correlation between matrix columns

```
> signif(cor(dat2[, grep("Average", names(dat2))], use = "complete.obs"), 3)
```

	orangeAverage	purpleAverage	greenAverage	bannerAverage
orangeAverage	1.000	0.889	0.837	0.441
purpleAverage	0.889	1.000	0.843	0.441
greenAverage	0.837	0.843	1.000	0.411
bannerAverage	0.441	0.441	0.411	1.000

Or between columns of two matrices, column by column.

```
> signif(cor(dat2[, 3:4], dat2[, 5:6], use = "complete.obs"), 3)
```

	greenAverage	bannerAverage
orangeAverage	0.837	0.441
purpleAverage	0.843	0.441

Correlation

You can also use cor.test() to test for whether correlation is significant (ie non-zero). Note that linear regression is probably your better bet.

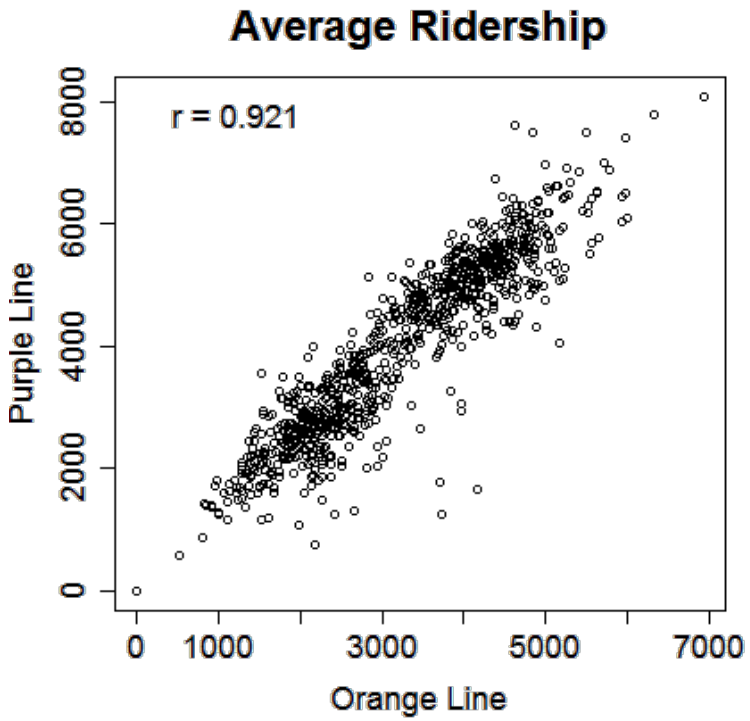
```
> ct = cor.test(dat2$orangeAverage, dat2$purpleAverage, use = "complete.obs")
> ct
```

```

    Pearson's product-moment correlation

data:  dat2$orangeAverage and dat2$purpleAverage
t = 69.65, df = 871, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9100 0.9303
sample estimates:
   cor
0.9208
```

Correlation

Note that you can add the correlation to a plot, via the legend() functinon.

```
> plot(dat2$orangeAverage, dat2$purpleAverage, xlab = "Orange Line", ylab = "Purple Line",
+     main = "Average Ridership", cex.axis = 1.5, cex.lab = 1.5, cex.main = 2)
> legend("topleft", paste("r =", signif(ct$estimate, 3)), bty = "n", cex = 1.5)
```



Correlation

For many of these testing result objects, you can extract specific slots/results as numbers, as the 'ct' object is just a list.

```
> # str(ct)
> names(ct)
```

```
[1] "statistic"   "parameter"   "p.value"     "estimate"    "null.value"
[6] "alternative" "method"      "data.name"   "conf.int"
```

```
> ct$statistic
```

```
    t
69.65
```

```
> ct$p.value
```

```
[1] 0
```

T-tests

The T-test is performed using the t.test() function, which essentially tests for the difference in means of a variable between two groups.

```
> tt = t.test(dat2$orangeAverage, dat2$purpleAverage)
> tt
```

```
    Welch Two Sample t-test

data:  dat2$orangeAverage and dat2$purpleAverage
t = -16.22, df = 1745, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1141.5  -895.2
sample estimates:
mean of x mean of y
     2994      4013
```

```
> names(tt)
```

```
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"
```

T-tests

You can also use the 'formula' notation.

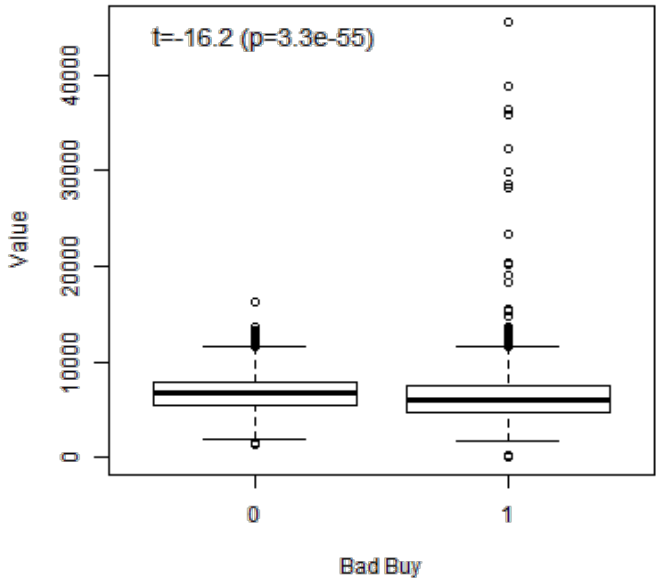
```
> tt2 = t.test(VehBCost ~ IsBadBuy, data = cars)
> tt2$estimate
```

```
mean in group 0 mean in group 1
           6797            6259
```

T-tests

You can add the t-statistic and p-value to a boxplot.

```
> boxplot(VehBCost ~ IsBadBuy, data = cars, xlab = "Bad Buy", ylab = "Value")
> leg = paste("t=", signif(tt$statistic, 3), " (p=", signif(tt$p.value, 3), ")",
+     sep = "")
> legend("topleft", leg, cex = 1.2, bty = "n")
```



Proportion tests

prop.test() can be used for testing the null that the proportions (probabilities of success) in several groups are the same, or that they equal certain given values.

```
prop.test(x, n, p = NULL,
          alternative = c("two.sided", "less", "greater"),
          conf.level = 0.95, correct = TRUE)
```

```
> prop.test(x = 15, n = 32)
```

```
    1-sample proportions test with continuity correction

data:  15 out of 32, null probability 0.5
X-squared = 0.0312, df = 1, p-value = 0.8597
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.2951 0.6497
sample estimates:
     p
0.4688
```

Chi-squared tests

chisq.test() performs chi-squared contingency table tests and goodness-of-fit tests.

```
chisq.test(x, y = NULL, correct = TRUE,
           p = rep(1/length(x), length(x)), rescale.p = FALSE,
           simulate.p.value = FALSE, B = 2000)
```

```
> tab = table(cars$IsBadBuy, cars$IsOnlineSale)
> tab
```

```
       0     1
  0 62375  1632
  1  8763   213
```

Chi-squared tests

```
> cq = chisq.test(tab)
> cq
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  tab
X-squared = 0.9274, df = 1, p-value = 0.3356
```

```
> names(cq)
```

```
[1] "statistic" "parameter" "p.value"   "method"    "data.name" "observed"
[7] "expected"  "residuals" "stdres"
```

```
> cq$p.value
```

```
[1] 0.3356
```

Chi-squared tests

Note that does the same test as prop.test, for a 2x2 table.

```
> chisq.test(tab)
```

```
    Pearson's Chi-squared test with Yates' continuity correction

data:  tab
X-squared = 0.9274, df = 1, p-value = 0.3356
```

```
> prop.test(tab)
```

```
    2-sample test for equality of proportions with continuity
    correction

data:  tab
X-squared = 0.9274, df = 1, p-value = 0.3356
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.005208  0.001674
sample estimates:
prop 1 prop 2
0.9745 0.9763
```

Chi-squared tests

Note that does the same test as prop.test, for a 2x2 table.

chisq.test(tab) prop.test(tab)

Linear Regression

Now we will briefly cover linear regression. I will use a little notation here so some of the commands are easier to put in the proper context.

y_i = \alpha + \beta * x_i + \epsilon_i

where:

- y_i is the outcome for person i
- \alpha is the intercept
- \beta is the slope
- x_i is the predictor for person i
- \epsilon_i is the residual variation for person i

Linear Regression

The 'R' version of the regression model is:

y ~ x

where:

- y is your outcome
- x is/are your predictor(s)

Linear Regression

```
> fit = lm(VehOdo ~ VehicleAge, data = cars)
> fit
```

```

Call:
lm(formula = VehOdo ~ VehicleAge, data = cars)

Coefficients:
(Intercept)   VehicleAge
      60127         2723
```

'(Intercept)' is \alpha

'VehicleAge' is \beta

Linear Regression

```
> summary(fit)
```

```
Call:
lm(formula = VehOdo ~ VehicleAge, data = cars)

Residuals:
   Min     1Q Median     3Q    Max
-71097  -9500   1383  10323  41037

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  60127.2      134.8   446.0   <2e-16 ***
VehicleAge    2722.9       29.9    91.2   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13800 on 72981 degrees of freedom
Multiple R-squared: 0.102,  Adjusted R-squared: 0.102
F-statistic: 8.31e+03 on 1 and 72981 DF,  p-value: <2e-16
```

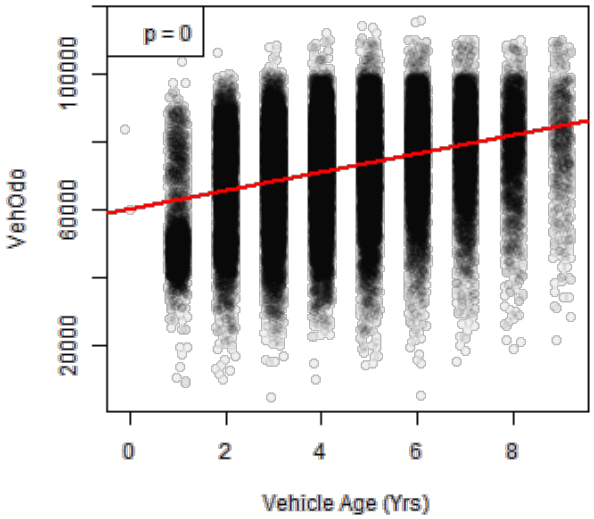
Linear Regression

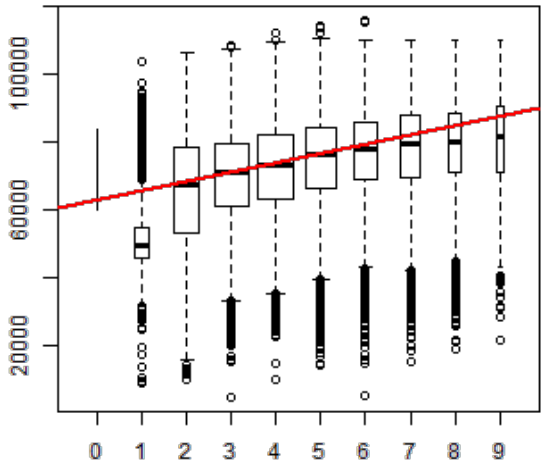
```
> summary(fit)$coef
```

```
             Estimate Std. Error t value Pr(>|t|)
(Intercept)     60127     134.80  446.04        0
VehicleAge       2723      29.86   91.18        0
```

Linear Regression

```
> library(scales)
> par(mfrow = c(1, 2))
> plot(VehOdo ~ jitter(VehicleAge, amount = 0.2), data = cars, pch = 19, col = alpha("black",
+     0.05), xlab = "Vehicle Age (Yrs)")
> abline(fit, col = "red", lwd = 2)
> legend("topleft", paste("p =", summary(fit)$coef[2, 4]))
> boxplot(VehOdo ~ VehicleAge, data = cars, varwidth = TRUE)
> abline(fit, col = "red", lwd = 2)
```





Linear Regression

Note that you can have more than 1 predictor in regression models.

The interpretation for each slope is change in the predictor corresponding to a one-unit change in the outcome, holding all other predictors constant.

```
> fit2 = lm(VehOdo ~ VehicleAge + WarrantyCost, data = cars)
> summary(fit2)
```

```
Call:
lm(formula = VehOdo ~ VehicleAge + WarrantyCost, data = cars)

Residuals:
   Min     1Q Median     3Q    Max
-67895  -8673    940   9305  45765

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.24e+04   1.46e+02   359.1   <2e-16 ***
VehicleAge   1.94e+03   2.89e+01    67.4   <2e-16 ***
WarrantyCost 8.58e+00   8.25e-02   104.0   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12900 on 72980 degrees of freedom
Multiple R-squared: 0.218,  Adjusted R-squared: 0.218
F-statistic: 1.02e+04 on 2 and 72980 DF,  p-value: <2e-16
```

Linear Regression

Factors get special treatment in regression models - lowest level of the factor is the comparison group, and all other factors are relative to its values.

```
> fit3 = lm(VehOdo ~ factor(TopThreeAmericanName), data = cars)
> summary(fit3)
```

```
Call:
lm(formula = VehOdo ~ factor(TopThreeAmericanName), data = cars)

Residuals:
   Min     1Q Median     3Q    Max
-71947  -9634   1532  10472  45936

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                            68249         93  733.98  < 2e-16 ***
factor(TopThreeAmericanName)FORD        8524        158   53.83  < 2e-16 ***
factor(TopThreeAmericanName)GM          4952        129   38.39  < 2e-16 ***
factor(TopThreeAmericanName)NULL       -2005       6362   -0.32  0.75267
factor(TopThreeAmericanName)OTHER        585        160    3.66  0.00026 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14200 on 72978 degrees of freedom
Multiple R-squared: 0.0482, Adjusted R-squared: 0.0482
F-statistic:  924 on 4 and 72978 DF,  p-value: <2e-16
```

Lab

Load the property taxes dataset from the website: http://biostat.jhsph.edu/~ajaffe/files/propertyTaxes.rda

1. How many houses were assessed?
2. Make boxplots using a) default and b) ggplot2 graphics showing cityTax by whether the property is a principal residence or not.
3. Subset the data to only retain those houses that are principal residences. How many such houses are there?
4. Describe the distribution of property taxes on these residences.
5. Convert the 'lotSize' variable to a numeric square feet variable. Assume hyphens represent decimal places within measurements. Also, 1 acre = 43560 square feet
6. Plot this numeric lotSize versus cityTax on principal residences. What is the correlation between these two variables? Fit a linear regression model with cityTax as the outcome - what is the coefficient for lotSize, as well as its t-statistic and p-value?