Data Visualization & Statistics in R

Rodney J. Dyer

This activity provides an overview of basic statistical routines in R including:

- Estimation of means, variances, and other summary statistics.
- Estimation and testing of correlations.
- Analysis of Variance for 1-way models with post-hoc testing.
- Linear regression modeling with model selection.

In the course of this exercise, you will also learn how to graphically display data using built-in methods as well as how to use the `ggplot2` library to make kickin' graphics.

1 Overview

In our courses and individual research projects, dealing with data in a statistical framework is critical; that is what we are getting paid (or paying in the case of students) to do. I understand there are limitations to what students can be exposed to in the course of taking graduate statistics courses, such as STAT 543 here at VCU, and this exercise focuses on providing a basic foundation in statistics that we will commonly use in population genetics. It is my hope that everyone is comfortable with the basic statistics used here (though your mileage may vary) and that this will be more of a "how do I do this in R" exercise than a "what is this statistic" one...

As you go through this exercise, you should be actually typing in the values you see in the document so you can work with the data as you go. It is one thing to read about programming and analyses, but if you want to really learn it (and perhaps get a good grade...), you should do it as well.

2 The Data

We will use some data included in R by default to go through these routines. The data is derived from the 1974 *Motor Trend US* magazine comparing 32 different automobiles. We will first work at changing the data around a bit for clarity and in the course of this you will see how you can manipulate both raw data and data contained in `data.frames`. Here is what is in that data:

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp
##  Min.   :10.4   Min.   :4.00   Min.   : 71.1   Min.   : 52.0
##  1st Qu.:15.4   1st Qu.:4.00   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.2   Median :6.00   Median :196.3   Median :123.0
##  Mean   :20.1   Mean   :6.19   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.8   3rd Qu.:8.00   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.9   Max.   :8.00   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.76   Min.   :1.51   Min.   :14.5   Min.   :0.000
```

```
##  1st Qu.:3.08   1st Qu.:2.58   1st Qu.:16.9   1st Qu.:0.000
##  Median :3.69   Median :3.33   Median :17.7   Median :0.000
##  Mean   :3.60   Mean   :3.22   Mean   :17.8   Mean   :0.438
##  3rd Qu.:3.92   3rd Qu.:3.61   3rd Qu.:18.9   3rd Qu.:1.000
##  Max.   :4.93   Max.   :5.42   Max.   :22.9   Max.   :1.000
##        am             gear           carb
##  Min.   :0.000   Min.   :3.00   Min.   :1.00
##  1st Qu.:0.000   1st Qu.:3.00   1st Qu.:2.00
##  Median :0.000   Median :4.00   Median :2.00
##  Mean   :0.406   Mean   :3.69   Mean   :2.81
##  3rd Qu.:1.000   3rd Qu.:4.00   3rd Qu.:4.00
##  Max.   :1.000   Max.   :5.00   Max.   :8.00
```

The car models are recorded as `rownames()` of the `data.frame` and can be listed as:

```
rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

Some of these columns are self explanatory, some are not. Here is what they represent:

Table 1: Description of data columns in the `mtcars` data set. In this data set, all data is encoded numerically for historical reasons, mostly due to the tremendously small amount of computer memory available in 1974.

column	Label	Description	Column	Label	Description
1	mpg	Miles/(US) gallon	7	qsec	1/4 mile time
2	cyl	Number of cylinders	8	vs	V/S[1]
3	disp	Displacement (cu.in.)	9	am	Transmission (0 = automatic, 1 = manual)
4	hp	Gross horsepower	10	gear	Number of forward gears
5	drat	Rear axle ratio	11	carb	Number of carburetors
6	wt	Weight (lb/1000)			

In your own data, as we will do below, we should use descriptive representations rather than obscure encodings. We are no longer limited by minuscule computational resources and as such we do not need to make such shortcuts (n.b., see Y2K scare for why this is a bad idea). So in this data set, the variables `cyl`, `vs`, `am`, `gear`, & `carb` are really integer or categorical variables. Before we get into working with this data, we'll copy it to another variable (aptly named `data`) and make this more readable by replacing the numeric codes with factors.

```
data <- mtcars
data$cyl <- factor(data$cyl)
data$vs <- factor(data$vs)
data$gear <- factor(data$gear)
data$carb <- factor(data$carb)
```

With the `am` variable, I am not going to change it to a numeric integer, rather I am going to make it `Automatic` or `Stick` (e.g., as character factors) for readability. To do this, I first make a new vector consisting of only `Automatic` and then fill it in, where `data$am==1` with `Stick`. The I put it back into `data` as a factor.

```
am <- rep("Automatic", length(data$am))
am[data$am == 1] <- "Stick"
data$am <- factor(am)
```

Next, I am going to change the abbreviations of the names to full names so that we do not have to refer back to the table above.

```
names(data) <- c("MPG", "Cylinders", "Displacement", "HP", "AxelRatio", "Weight",
    "QuarterMile", "VS", "Transmission", "Gears", "Carburetors")
```

As is true for many `R` functions, `func(x)` returns the values and `func(x) <- y` sets them. You will see this over and over. Now, when we summarize the data, we can see that different kinds of data types with better names.

```
summary(data)
```

```
##       MPG        Cylinders  Displacement         HP           AxelRatio
##  Min.   :10.4   4:11       Min.   : 71.1   Min.   : 52.0   Min.   :2.76
##  1st Qu.:15.4   6: 7       1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.08
##  Median :19.2   8:14       Median :196.3   Median :123.0   Median :3.69
##  Mean   :20.1              Mean   :230.7   Mean   :146.7   Mean   :3.60
##  3rd Qu.:22.8              3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.92
##  Max.   :33.9              Max.   :472.0   Max.   :335.0   Max.   :4.93
##      Weight      QuarterMile   VS        Transmission Gears  Carburetors
##  Min.   :1.51   Min.   :14.5   0:18   Automatic:19    3:15   1: 7
##  1st Qu.:2.58   1st Qu.:16.9   1:14   Stick    :13    4:12   2:10
##  Median :3.33   Median :17.7                          5: 5   3: 3
##  Mean   :3.22   Mean   :17.8                                 4:10
##  3rd Qu.:3.61   3rd Qu.:18.9                                 6: 1
##  Max.   :5.42   Max.   :22.9                                 8: 1
```

Much nicer and in the long run, easier to work with.

3 Basic Graphical Display

As a first pass through a data set, you should plot it and take a look at how it is distributed. This is an under emphasized activity, but one that gives us a lot of insights. It is always best to know what you are working with before jumping into it, often there may be bad data points, or other issues that you need to address prior to analysis and these are easily found if you take the time to do some visual data inspection.

In all these plot mechanisms, you can look to the optional plotting arguments in each using the help functions (`?func`) to add additional plotting options.

3.1 Plot them all!

There are several routines for displaying the relationship between several variables, the easiest one is the `pairs()` routine. Below in Figure 1, I plot the continuous variables (e.g., the non-categorical ones) using this routine. Notice, that it is possible to specify a subset of rows or columns when passing an as an argument to a function. Since I did not include anything in the column location (e.g., nothing before the comma int he square brackets), it uses all the rows.[2]

```
pairs(data[, c(1, 3, 4, 5, 6, 7)])
```

[2] Using `data[c(1,2,3),]` would use the first three rows and all columns.

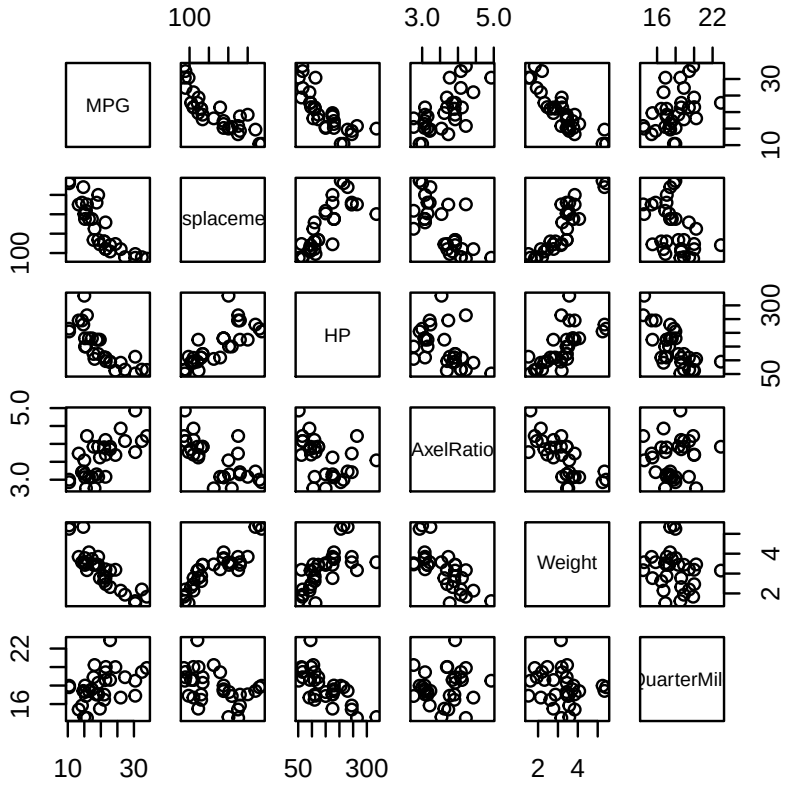


Figure 1: Plot of the continuous data columns in the data set.

3.2 Scatter Plots

The basic scatter plot consists of x and y variables, plot on a single graph. In this and other basic plotting routines in R , there are two ways to specify how the plot looks. The first uses just the names of the variables, separated by a comma in the function arguments.

```
plot(x, y)
```

This plots the x variable on the x-axis and the y variable on the y-axis. The second way to plot this is the 'functional' way of plotting.

```
plot(y ~ x)
```

Where here it is "plot y as a function of x". This is also how you specify both regression and anova model functions (see below) and it a nice shorthand way of doing it. Either way you do it, the same plot is produced. I prefer the functional approach as it is the same as we use in the statistical model estimation and I'll use it as we go forward.

For the example of this plot, I'm going to throw in a bit of extra plotting options to show how you can customize the display a bit. I add:

1. x- and y- axis labels using `xlab="X-Axis Label"` and `ylab="Y-axis label"` options to the plot.
2. symbol shape differences using `pch=` (for 'plot character').
3. changes in the box around the plot using 'box type' `bty="n"`. I prefer to have no box around my plots,

so you will see me using this option often.

And then add a legend using the `legend()` function with somewhat obvious options.

```
plot(data$AxelRatio ~ data$Weight, ylab = "Axel Ratio", xlab = "Vehicle Weight (lbs/1000)",
    pch = as.numeric(data$Gears), bty = "n")
legend(4, 5, legend = c("3 Speed", "4 Speed", "5 Speed"), pch = c(1, 2, 3),
    bty = "n")
```

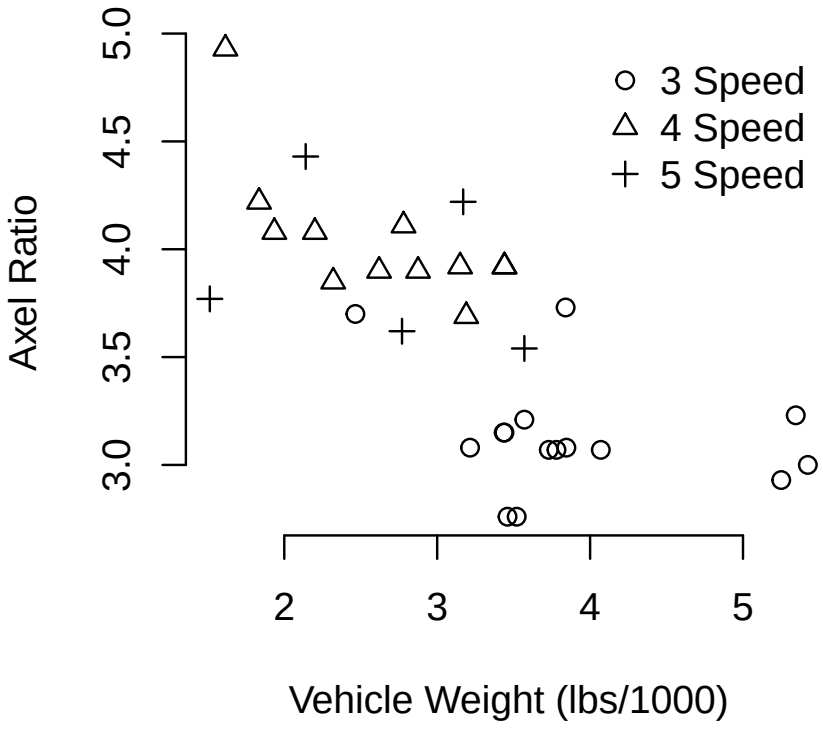


Figure 2: Plot of Axel Ratio as a function of vehicle weight (lbs/1000) for 3, 4, and 5-speed transmissions.

Notice how this is a pretty well designed plot (if I do say so myself). The axes are labelled correctly, the symbols are displayed and there is a nice caption removing any ambiguity. It stands on its own and does not need any additional explanation.

3.3 Line Plots

A line plot is identical to that in a scatter plot, you only have to tell the `plot()` function to tell it to take the data and plot it in order as lines. This is accomplished by passing `type="l"` to the function. In fact, there are several types of plots that can be produced by changing the options to `type`, see the help function for `plot` for a list. I'll skip plotting a line plot here for brevity.

3.4 Box Plots

When you have continuous data that is grouped by a categorical (e.g., `factor`), you can use the `boxplot()` function to display your data comparing factors. Here is an example showing how different size engines (in terms of the number of cylinders) are in different sized vehicles (in terms of overall weight).

```
boxplot(Weight ~ Cylinders, data = data, xlab = "Engine Cylinders", ylab = "Vehicle Weight (lbs/1000)")
```

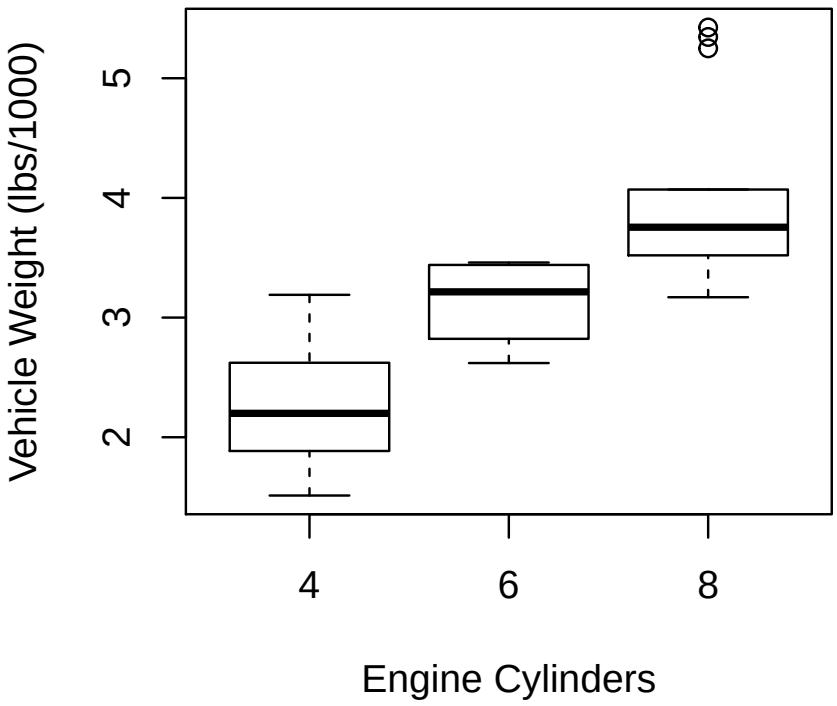


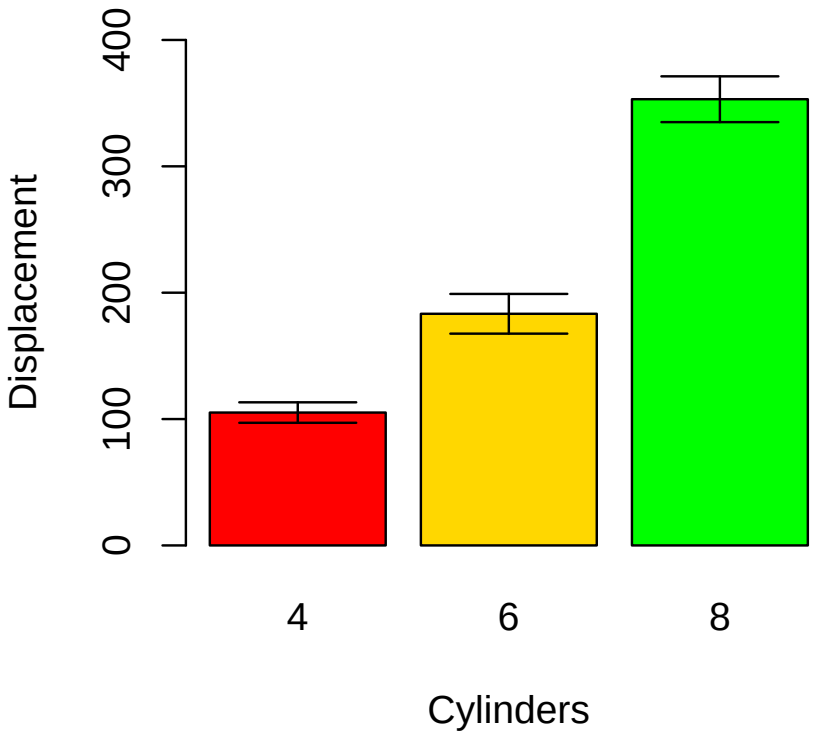
Figure 3: Vehicle weights observed in 1974 for alternate engine configurations.

3.5 Barplot

A barplot is a display of continuous data plot across categories. There are often some options added to barplots that add to the information such as error bars, etc. Here is an example using mean displacement across cylinders estimated using the `by(data,category,function)` approach. The `by()` function is *really* helpful for summaries. It returns a list keyed by the levels of the `category` of the `data` as analyzed by `function`. I also use it for the standard error of the mean as well for the error bars displayed using `arrows`.

```
mean.disp <- by(data$Displacement, data$Cylinders, mean)
sd.mean <- by(data$Displacement, data$Cylinders, sd)/sqrt(by(data$Displacement,
    data$Cylinders, length))
x <- barplot(mean.disp, xlab = "Cylinders", ylab = "Displacement", ylim = c(0,
    400), col = c("red", "gold", "green"))
arrows(x, mean.disp, x, mean.disp + sd.mean, angle = 90)
```

```
arrows(x, mean.disp, x, mean.disp - sd.mean, angle = 90)
```



There are a few things to point out here.

1. The `barplot()` function, if asked, returns the midpoints (x-axis coordinates) for the bars. This is something that is determined at the time of plotting. Here I assign that to `x` and use it in the `arrows()` function.
2. The `arrows()` function takes four coordinates, `x0`, `y0`, `x1`, and `y1` for where to plot the bars. Here I take the mean value (e.g., the bar height) and set the offset of a sd.mean for both positive and negative bars
3. The arrows, with an `angle=90` looks like an error bar...
4. The `col=` gives the color of the bar (or symbol in other plots).

4 Basic Statistics

OK, so we've gone through the data and you should be able to provide a plot of various aspects of this data set.

4.1 Summary Statistics

There are several kinds of summary statistics available for you in R . Some of the most common ones are those that deal with moments and

```
mean(data$MPG)
```

```
## [1] 20.09
```

```
var(data$MPG)
```

```
## [1] 36.32
```

```
range(data$MPG)
```

```
## [1] 10.4 33.9
```

```
summary(data$MPG)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.4    15.4    19.2    20.1    22.8    33.9
```

4.2 T-Tests

A t-test examines the mean value of one or two variables. To start, I will examine a one-variable situation, seeing if the vehicles with 4-cylinders have a specified level of efficiency.

H_O : In 1974, 4-cylinder cars did not get at least 20 miles per gallon.

Actually, with a one sample t-test, R tests that the mean is equal to zero. So in this example, I take the mpg -20 as the value.

```
cars.4.cylinder <- data$MPG[data$Cylinders == 4] - 20
t.test(cars.4.cylinder, alternative = "greater")
```

```
##
##  One Sample t-test
##
## data:  cars.4.cylinder
## t = 4.901, df = 10, p-value = 0.0003113
## alternative hypothesis: true mean is greater than 0
## 95 percent confidence interval:
##  4.199   Inf
## sample estimates:
## mean of x
##     6.664
```

Conclusion: Reject H_O at a pretty good level, cars back in 1974 got more than 20 miles per gallon.

A two-sample t-test can be performed the same way. You need to pass two sets of data to the function. Here I test the hypothesis:

H_O : Mean horsepower is the same for automatic and stick shift vehicles in 1974.

```
hp.auto <- data$HP[data$Transmission == "Automatic"]
hp.stick <- data$HP[data$Transmission == "Stick"]
t.test(hp.auto, hp.stick)
```

```
##
##  Welch Two Sample t-test
##
## data:  hp.auto and hp.stick
## t = 1.266, df = 18.71, p-value = 0.221
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
##  -21.88  88.71
## sample estimates:
## mean of x mean of y
##     160.3     126.8
```

Conclusion: Do not reject H_O, the mean horsepower seems to be the same.

4.3 Correlation

Correlations are a measure of how much one variable changes in relation to changes in another variable. Here, we look at the relationship between vehicle weight and miles per gallon.

```
r <- cor(data$Weight, data$MPG)
r
```

```
## [1] -0.8677
```

It looks like r = -0.8677 is pretty far away from 0 suggesting that there may be a correlation between these two variables (and it is a negative one as we would expect). But is it *significantly* different? That is the question we need to address. To to that, we use `cor.test()`:

```
wm_corr <- cor.test(data$Weight, data$MPG)
wm_corr
```

```
##
##  Pearson's product-moment correlation
##
## data:  data$Weight and data$MPG
## t = -9.559, df = 30, p-value = 1.294e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9338 -0.7441
## sample estimates:
##     cor
## -0.8677
```

Again, I assign the results of the test to a variable. I do this because there may be reasons why I need to get access to the various parts of the test results, like say that r = -0.8677 as I write this text without copying from R and pasting it into the document.

The result of most analyses will have a bunch of data that is summarized when you print it out as we did above. The variable `wm_corr` is actually a kind of list object. If you look into it, you can see all the data and other stuff inside it. Often it is necessary to work with parts of an analysis and it is a good idea to know what is actually inside the results. In a broad context, it takes a lot of the magic out of R , when you realize that almost everything you see is derived from those basic data types we discussed in the last exercise.

```
names(wm_corr)
```

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

```
wm_corr$p.value
```

```
## [1] 1.294e-10
```

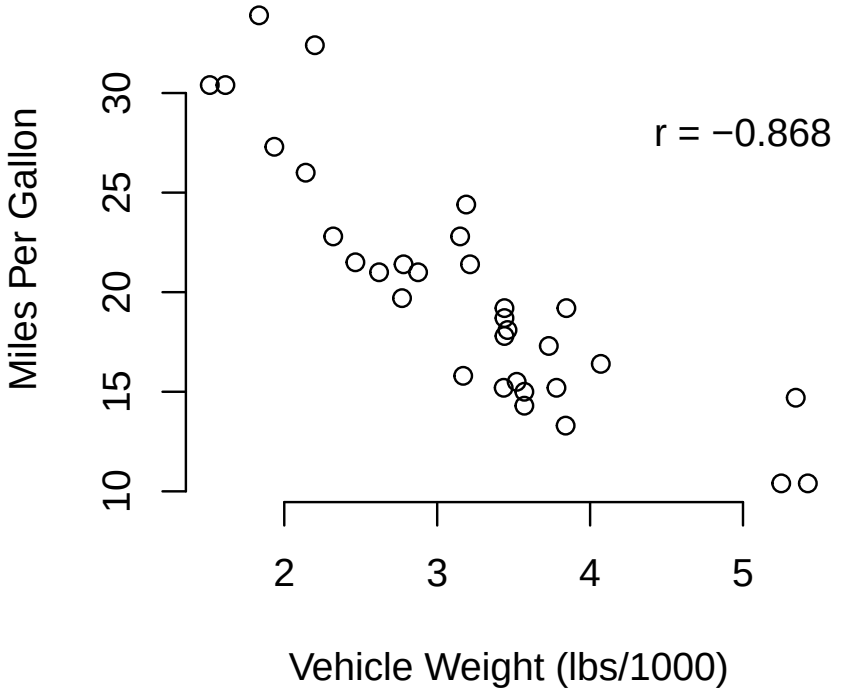
```
wm_corr$statistic
```

```
##      t
## -9.559
```

Here is an example of how I could use the results of the analysis in a subsequent plot (n.b., the `round` is

necessary to not plot out 20 digits).

```
plot(MPG ~ Weight, data = data, xlab = "Vehicle Weight (lbs/1000)", ylab = "Miles Per Gallon",
    bty = "n")
text(5, 28, paste("r =", round(wm_corr$estimate, digits = 3)))
```



4.4 Analysis of Variance & *Post Hoc* Testing

The analysis of variance, and associated statistics, looks at differences among several treatments (usually more than two). I like to think of it as a linear model such as:

$$Y_{ij} = \mu + \tau_i + \epsilon_{ij} \tag{1}$$

Where

- Y_{ij} is an observation,
- μ is the mean of all observations across the data set
- τ_i is the deviance of the i^{th} treatment level from the overall mean μ (the treatment mean)
- ϵ_{ij} is deviance of the j^{th} observation from the mean of the i^{th} treatment (the error term)

So if there is no significant differences among treatments, then the $\tau_i = 0$. Actually, this is the precise null hypothesis

$H_O : \tau = 0$

that is tested. In R , it is a bit confusing at first because we first do the analysis of variance using the function `aov()` and then if we want to look at the ANOVA table (the thing usually reported), we use the function `anova()` and pass the model from the `aov()` function. At first, it seems that perhaps the `anova()` model should have been named something else... It is one of those things you'll have to get used to. Here is the output of testing if having differently geared transmissions produce significantly different running inefficiencies.

```
fit.mpg_trans <- aov(MPG ~ Gears, data = data)
fit.mpg_trans
```

```
## Call:
##    aov(formula = MPG ~ Gears, data = data)
##
## Terms:
##                   Gears Residuals
## Sum of Squares  483.2     642.8
## Deg. of Freedom     2        29
##
## Residual standard error: 4.708
## Estimated effects may be unbalanced
```

OK, that does not tell us much, lets construct the ANOVA table and look at that.

```
anova(fit.mpg_trans)
```

```
## Analysis of Variance Table
##
## Response: MPG
##           Df Sum Sq Mean Sq F value  Pr(>F)
## Gears      2    483   241.6    10.9 0.00029 ***
## Residuals 29    643    22.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So from the ANOVA table for the model `MPG` $\sim$ `Gears` we see:

- We have three treatments (`Gears`) as there are 2 degrees of freedom.
- The estimated F Value is large given the sample sizes, resulting in a very small probability of seeing treatment differences (the τ_i's) observed in the data.

Conclusion: Reject H_O, the different gearing in 1974 cars have different mileage inefficiencies.

That is all find and dandy but there is one last thing to consider. With several different treatments, there are actually many ways to get a rejections of the Null hypothesis. Consider:

1. All treatments are significantly different from each other.
2. On treatment is different from the rest which are the same.
3. Two treatments are different from the rest but not from each other.
4. Two treatments are different from the rest and each other.
5. ...

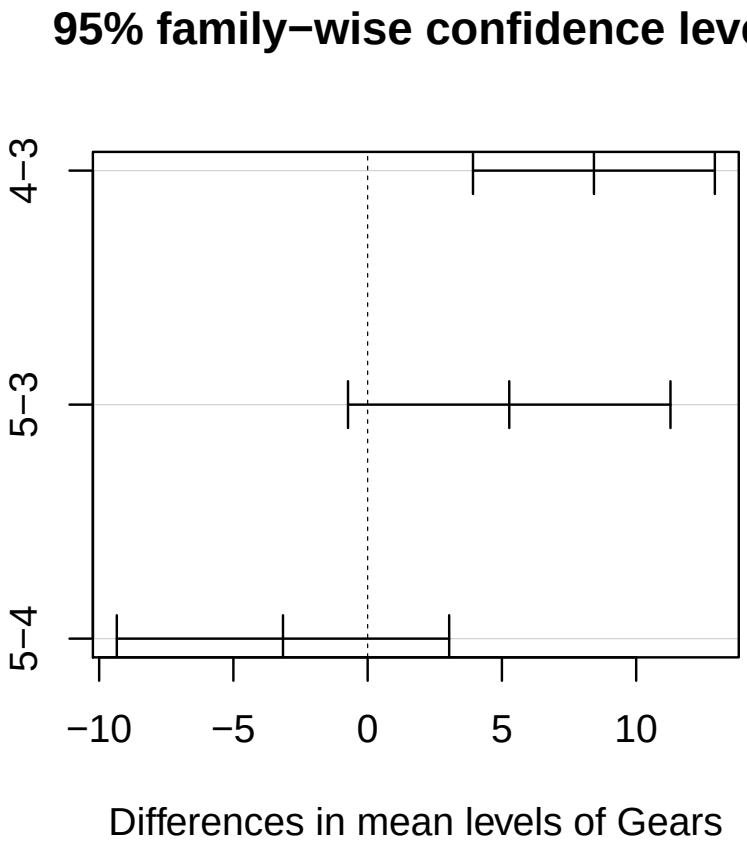
As you can see, simply rejecting the Null hypothesis only tells us that "At least one of the treatments is different from the rest." What it does not tell us is which one! This is where we need *post hoc* tests. The most common one used is the Tukey test, this tests for the differences in treatment means (e.g., $\tau_i - \tau_j = 0$). To perform it in R , you pass the model to it and it returns an object that can be textually or graphically displayed.

```
tukey.mpg_trans <- TukeyHSD(fit.mpg_trans)
tukey.mpg_trans
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = MPG ~ Gears, data = data)
##
## $Gears
##       diff     lwr    upr  p adj
## 4-3  8.427  3.9235 12.930 0.0002
## 5-3  5.273 -0.7309 11.278 0.0937
## 5-4 -3.153 -9.3424  3.036 0.4296
```

You can see that the mean difference between the treatments (4-3, 5-3, 5-4) is displayed, the confidence intervals are given and the adjusted probability is shown. Graphically, this is

```
plot(tukey.mpg_trans)
```



Showing the mean and confidence intervals for the difference in $\tau_i - \tau_j$ for all treatments. NOW, we can say something cool about it, the differences in efficiency are significant between 3 and 4 gears but other than that, they are not significantly different. There are other *post hoc* tests, this is just a commonly used one. Be careful with complicated models, particularly nested ones, the number of treatment differences can be exceedingly large.=

4.5 Linear Regression & Model Selection

Regression is an approach where you are assuming a functional relationship between at least two variables. This is the common linear model:

$$Y_i = \beta_0 + \beta_1 X + \epsilon_i \tag{2}$$

Where:

- Y_{ij} is the i^{th} observation.
- β_0 is the intercept term.
- β_1 is the slope of change in X (rise over run).
- And ϵ_i is again the error term for the i^{th} observation.

It is not surprising that the linear model for both regression and anova are similar. In fact, mathematically, they are identical except that in ANOVA the predictor variable is categorical whereas in regression analysis it is a continuous variable. I generally think of all linear models like a regression, it is just easier that way.

So, in this case, lets look at the $\frac{1}{4}$ mile time as a function of displacement in a linear model context. In fact, in `R` , to do a regression you use the function `lm()` (linear model) and pass it the functional form of the relationship. I'll show the results in the ANOVA table format as above.

```
fit.lm <- lm(QuarterMile ~ Displacement, data = data)
summary(fit.lm)
```

```
##
## Call:
## lm(formula = QuarterMile ~ Displacement, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.885 -1.218  0.045  0.923  4.489
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  19.29146    0.61903   31.16   <2e-16 ***
## Displacement -0.00625    0.00237   -2.64    0.013 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.64 on 30 degrees of freedom
## Multiple R-squared:  0.188,Adjusted R-squared:  0.161
## F-statistic: 6.95 on 1 and 30 DF,  p-value: 0.0131
```

The summary gives us some diagnostic data but not the whole ANOVA table, that is received from `anova()`.

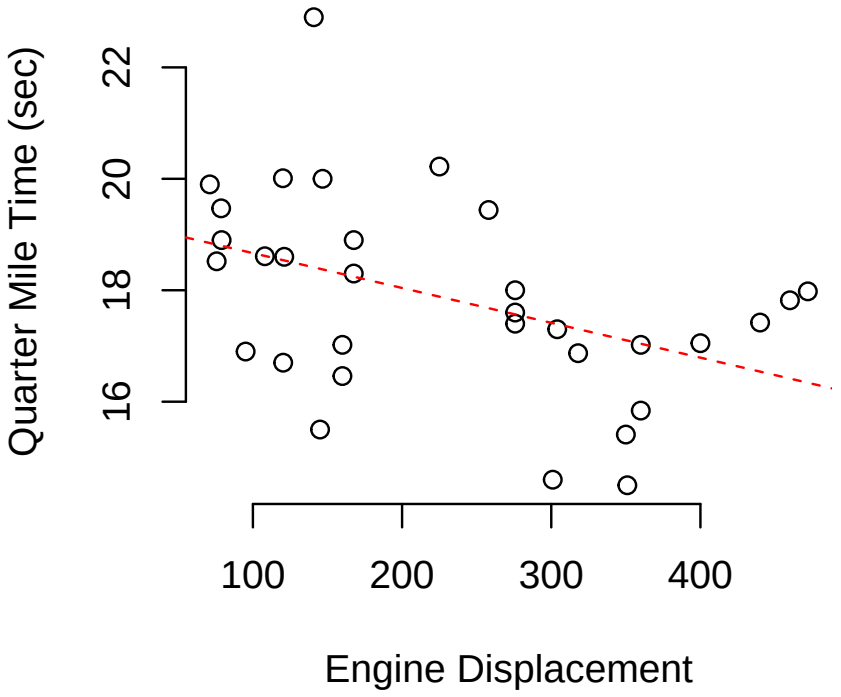
```
anova(fit.lm)
```

```
## Analysis of Variance Table
##
## Response: QuarterMile
##              Df Sum Sq Mean Sq F value Pr(>F)
## Displacement  1   18.6   18.62    6.95  0.013 *
## Residuals    30   80.4    2.68
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You can plot the `fit.lm` object using the `plot()` function and it gives a series of diagnostic plots. You can also pass analyses from `aov()` and it will show the same set of plots, I'll not do that here for brevity. You can

plot the original variables and then overlay the regression line onto the plot using `abline(intercept,slope)` (I make the line red and dashed using `col` and `lty`).

```
plot(QuarterMile ~ Displacement, data = data, bty = "n", ylab = "Quarter Mile Time (sec)",
    xlab = "Engine Displacement")
abline(fit.lm$coefficients[1], fit.lm$coefficients[2], col = "red", lty = 2)
```



So, from these results, we could conclude that engine displacement (a measure of how much volume the cylinders have) has some relationship with how fast a vehicle can travel a quarter mile. The overall model is significant (e.g., the model F =6.9501 is large, describing R^2 =0.161).

But is it the 'best' model we can get or just one that has a significant slope term? Testing the fit of different models is where we hope to get more meaningful interpretations of how the data relate to each other. Simply rejecting H_O does not tell us that this model is the best, just that it is sufficient. The easiest way to test alternative models is to use AIC. Here I am going to look at a sequence of models with additional terms `HP` and `AxelRatio`. The overall question is which subset of variables provide the best model for `QuarterMile`, one with only displacement or ones with more terms?

Here is the model with `Displacement` and `HP` together:

```
fit.lm2 <- lm(QuarterMile ~ Displacement + HP, data = data)
summary(fit.lm2)
```

```
##
## Call:
## lm(formula = QuarterMile ~ Displacement + HP, data = data)
##
## Residuals:
```

```
##    Min     1Q Median     3Q    Max
## -2.027 -0.916  0.168  0.611  4.175
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  20.45411    0.53116   38.51  < 2e-16 ***
## Displacement  0.00487    0.00295    1.65     0.11
## HP           -0.02542    0.00534   -4.76  4.9e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.25 on 29 degrees of freedom
## Multiple R-squared:  0.544,Adjusted R-squared:  0.513
## F-statistic: 17.3 on 2 and 29 DF,  p-value: 1.12e-05
```

And the one for `Displacement`, `HP`, and `AxelRatio`

```
fit.lm3 <- lm(QuarterMile ~ Displacement + HP + AxelRatio, data = data)
summary(fit.lm3)
```

```
##
## Call:
## lm(formula = QuarterMile ~ Displacement + HP + AxelRatio, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.900 -0.749 -0.023  0.618  4.235
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.69347    2.61462    9.06  8.1e-10 ***
## Displacement  0.00171    0.00385    0.44  0.66004
## HP           -0.02360    0.00548   -4.31  0.00018 ***
## AxelRatio    -0.77204    0.61042   -1.26  0.21638
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.23 on 28 degrees of freedom
## Multiple R-squared:  0.569,Adjusted R-squared:  0.523
## F-statistic: 12.3 on 3 and 28 DF,  p-value: 2.56e-05
```

Both of these models provide a larger R^2 than the original model, but are they 'significantly' better? There are several ways to look at this. We could compare the R^2 values and just grab the largest one (e.g., the one that explains the most of the total SS) but that is a bit misleading. A common approach is that of using the statistic AIC. The AIC approach estimates the fit of the model but penalizes you for adding more terms. It is easy to get a model with high R^2 by adding more and more predictor variables but are they really making a better model.

The general rule for using AIC is that a smaller AIC is better.

```
aic_lm <- AIC(fit.lm, fit.lm2, fit.lm3)
aic_lm
```

```
##         df   AIC
## fit.lm   3 126.3
## fit.lm2  4 109.8
## fit.lm3  5 110.0
```

So looking at the three models we have, we see that the lowest AIC is 109.799 for the second model. As a result, we can interpret this as suggesting that model 2 (`QuarterMile  Displacement + HP`) is the best

model. However, there is only a small distance between the second and third model in terms of AIC. This is where the second rule comes in, models with AIC values separated by less than 2.0 are considered equally good. Here I take the AIC values, subtract the minimum one and print out the results.

```
aic_lm$deltaAIC <- aic_lm$AIC - min(aic_lm$AIC)
aic_lm
```

```
##         df   AIC deltaAIC
## fit.lm   3 126.3  16.4817
## fit.lm2  4 109.8   0.0000
## fit.lm3  5 110.0   0.2221
```

So the difference between the second and third model are pretty small and as such we cannot rule out the third model as being good as well. The first model is 'right out' (so to speak). As a result, we must entertain both models 2 and 3 as potential descriptors of what is going on and further experimentation (or other model types) should be examined to get a better understanding.

5 Using GGPlot2 for Graphics Awesomeness

R itself comes with a wild variety of graphic routines, most of which are both quick and easy. However, they do not provide really nice ways to look at data. Leland Wilkinson's book *The Grammar of Graphics* (2005) led towards a different way of constructing quality graphics in modern statistical analyses. The `ggplot2` library is one implementation of this approach and one that I really like. And since you are my guinnea pigs in this class, you'll be exposed to it as well. Here is the basic idea behind using `ggplot2` (as I interpret it).

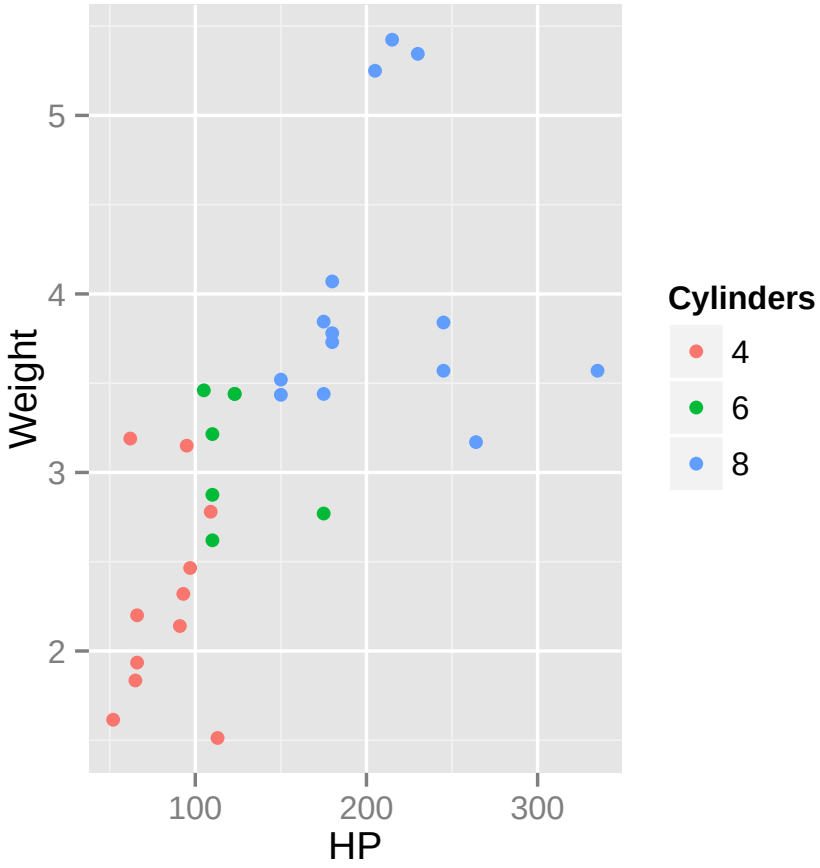
1. You data needs to be in a `data.frame`, which is how we usually work with it.
2. A graphic is made up of several layers, one added upon the next, similar to how we overlay objects on an overhead. These layers are *geometries* (consisting of points, lines, networks, bars, etc. These geometries are named using the `geom_*` convention (e.g., `geom_line()`, `geom_point()`, etc.)
3. Each geometry is plot using some *aesthetics* (using the funciton `aes()`) determining the coordinates or the items, their colors, shapes, etc. Several `geom_*` layers can use the same *aes* objects. However, when you up a property like color, shape, or fill into an aesthetic, usually this is tied to a variable in the underlying `data.frame`. For example, I may want to make a scatter plot of populations from `arapat` but have them colored by the Cluster they are in defined by the column *Cluster*.
4. The coordinate system can be modified for the entire plot switching to `coord_cartesian()`, `coord_polar`, etc.
5. Categorical groupings of the data, say different loci, can be highlighted by creating a matrix of plots (or fascets in the `ggplot` paralance) so that the pattern in each group is easily interpretable.

For all of these things, we essentially add these components together to make the plot. I like to think of it in just the same way that we add together components to create a multiple regression.

5.1 A Basic → Complex Scatter Plot

When we start, we begin with a `ggplot()` object and then add to it as we go.

```
require(ggplot2)
p <- ggplot(data, aes(x = HP, y = Weight))
p <- p + geom_point(aes(color = Cylinders))
p
```



You could have gotten the same thing by:

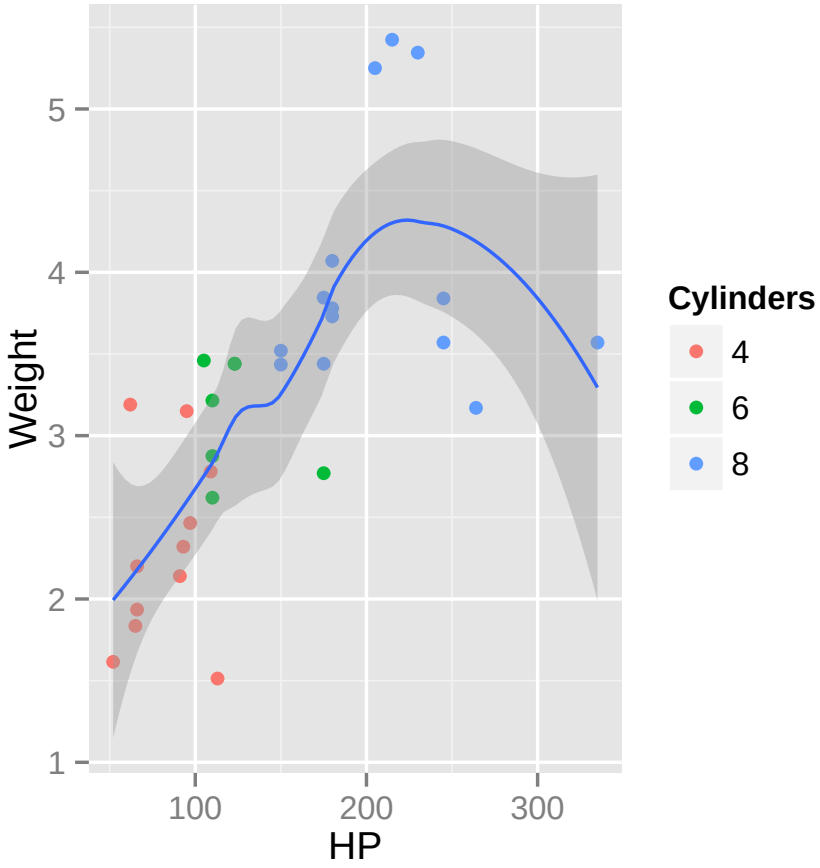
```
ggplot(data) + geom_point(aes(x = HP, y = Weight, color = Cylinders))
```

We can get more sophisticated here by adding a shape variable for `Transmission`, producing two legends on the right side. I'm also going to move the x- and y- aesthetics back into ggplot() so that additional layers can also use them and I don't have to retype the same thing for the other objects.

```
p <- ggplot(data, aes(x = HP, y = Weight)) + geom_point(aes(color = Cylinders,
    shape = Transmission))
p
```

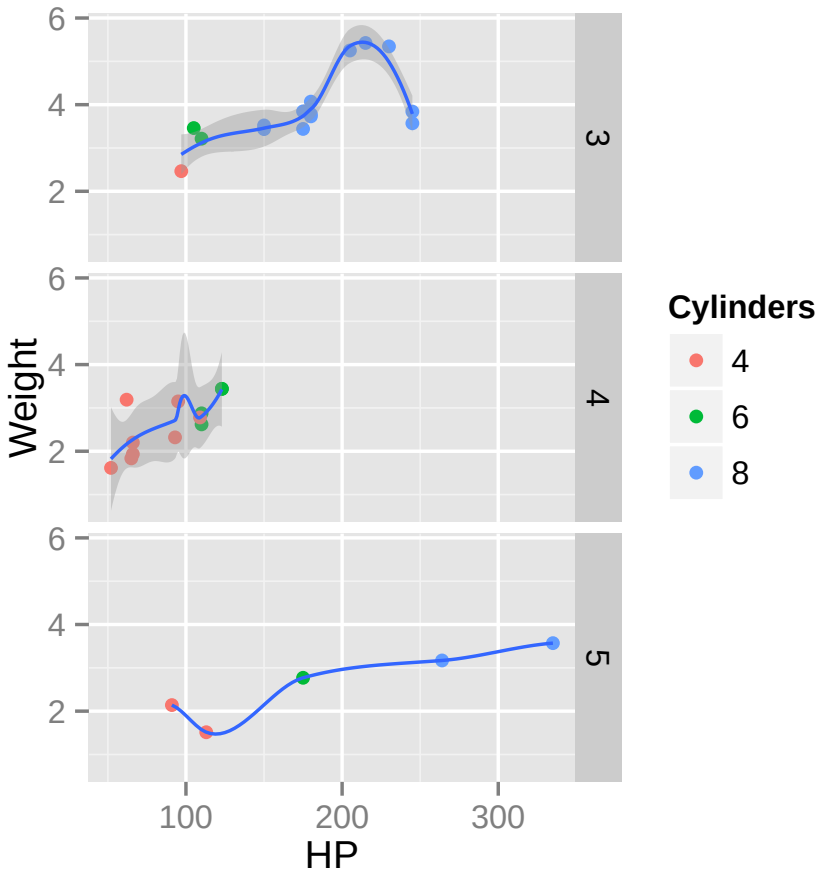
Now, I will add a smoothing line though that relationship

```
p <- p + stat_smooth(method = "loess")
p
```



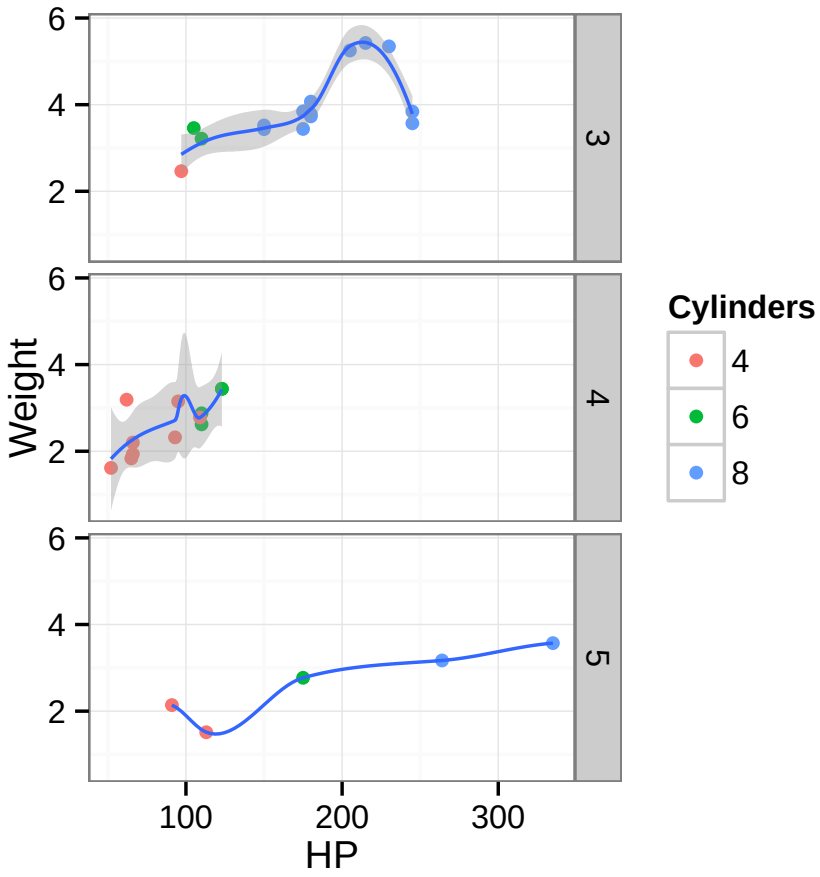
I'm now going to plot the different gears in different plots, to separate them out.

```
p <- p + facet_grid(Gears ~ .)
p
```



And to close it off, I'm going to make the overall color theme a bit more basic:

```
p + theme_bw()
```

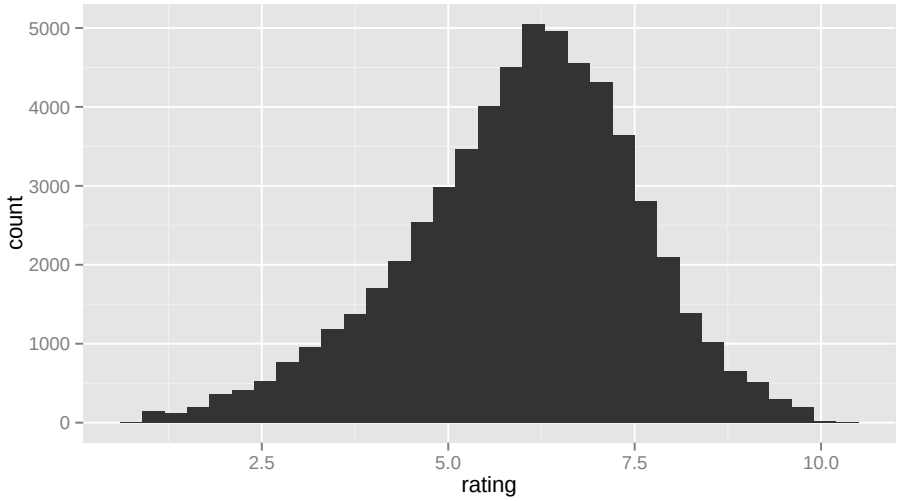


5.2 Frequency Distributions

Another common type of data are histograms, displaying the distribution of some variable throughout its range. This is not to be confused with a bar plot, which contrasts different factors. I'm going to switch over to another data set, `movies`, which consists of 28,819 data rows for 24 variables (see `?movies`) as it has a lot more data than the `mtcars` data set. For histograms, you use the function `geom_histogram()`, that can take several differnt kinds of options. Here is an example using the ratings (on a scale of 1-10)

```
p <- ggplot(movies) + geom_histogram(aes(x = rating))
p
```

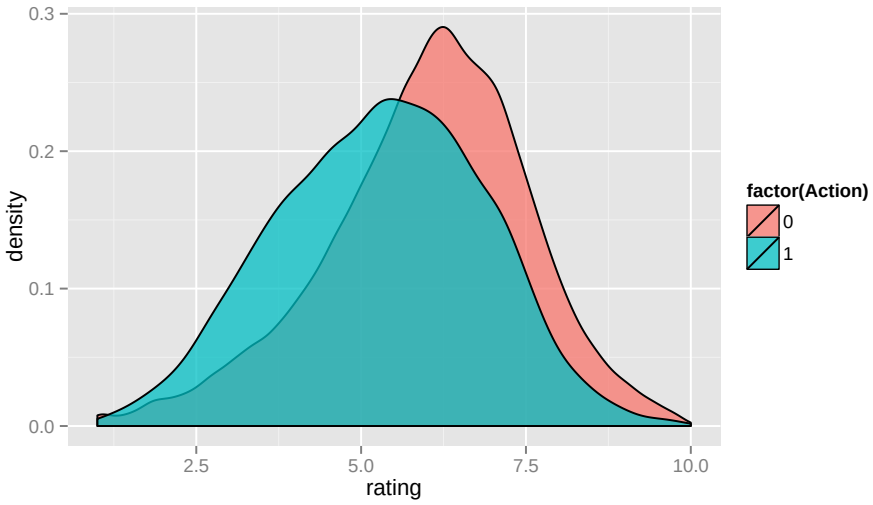
```
## stat_bin:  binwidth defaulted to range/30.  Use 'binwidth = x' to adjust this.
```



Notice by default, it assumes a bin size (e.g., how much of the x-axis is attributed to each bar) of `range(movies$ratings) / 30`. You can supply this value if you like (say `binwidth=0.1` as that corresponds to the levels in the raw data.)

For a more continuous approximation of a frequency distribution, you can use the function `geom_density()`. Here I show the density of ratings comparing Action movies to non-action movie types.

```
p <- ggplot(movies, aes(x = rating)) + geom_density(aes(fill = factor(Action)),
    alpha = 0.75)
p
```



It would probably be better to use a factor column with values of movie type rather than a bunch of 0/1 indexes. Here is how I would collapse the 7 last columns into a single factor variable and then plot ratings of all types together. Remember, a key strength of R is its ability to manipulate data and this is a good example of how to do that.

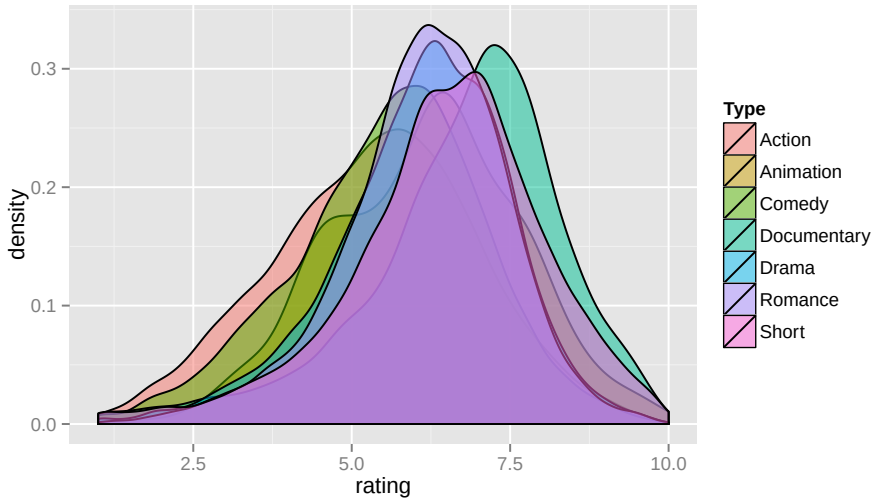
```
Type <- rep("Action", length(movies$Action))
Type[movies$Animation == 1] <- "Animation"
Type[movies$Comedy == 1] <- "Comedy"
Type[movies$Drama == 1] <- "Drama"
Type[movies$Documentary == 1] <- "Documentary"
Type[movies$Romance == 1] <- "Romance"
```

```
Type[movies$Short == 1] <- "Short"
movies$Type <- factor(Type)
table(Type)
```

```
## Type
##      Action   Animation      Comedy Documentary       Drama     Romance
##       14826         343        8897        2598       18090        4576
##       Short
##        9458
```

OK, now we can look at the rating density for all these types at once with a nice easy and clean methodology.

```
p <- ggplot(movies, aes(x = rating)) + geom_density(aes(fill = Type), alpha = 0.5)
p
```

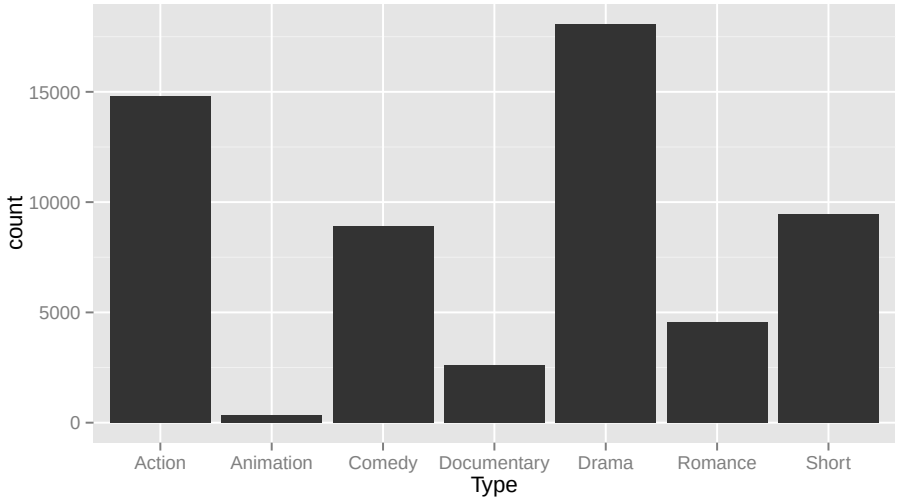


Looks OK, but a bit messy, perhaps a `facet_grid()` approach would be better.

5.3 Barplots

A barplot is one that displays the values for different categorical data. Here is an example of how many types of different movie types we have in the `data.frame`.

```
p <- ggplot(movies, aes(x = Type)) + geom_bar()
p
```

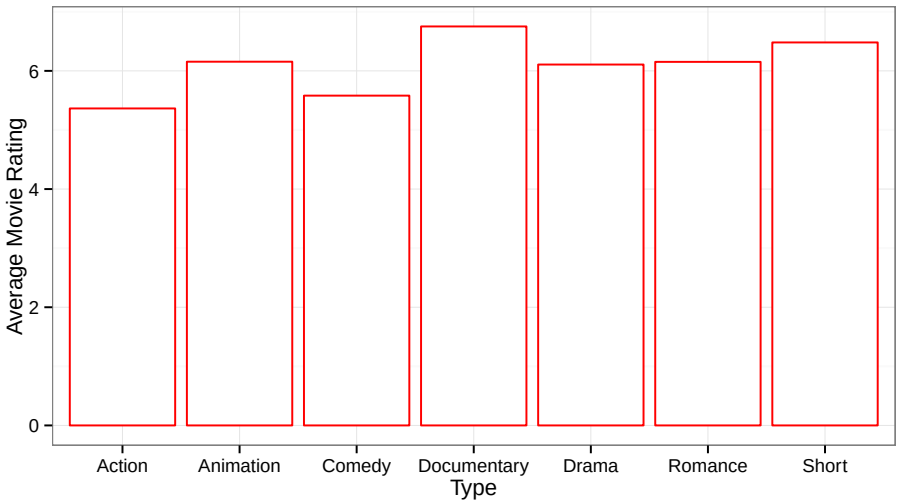


Here is an example of how we can take the mean rating for each type. I use the funciton `by()` to create a vector of `mean` values in `rating` for each `type`. I then shove it into a `data.frame` and have `ggplot` plot it out (and adjust the colors just for fun).

```
mean.rating <- by(movies$rating, movies$Type, mean)
df <- data.frame(Type = names(mean.rating), Rating = as.numeric(mean.rating))
df
```

```
##          Type Rating
## 1      Action  5.364
## 2   Animation  6.155
## 3      Comedy  5.581
## 4 Documentary  6.752
## 5       Drama  6.107
## 6     Romance  6.153
## 7       Short  6.481
```

```
ggplot(df, aes(x = Type, y = Rating)) + geom_bar(stat = "identity", fill = "white",
    color = "red") + theme_bw() + ylab("Average Movie Rating")
```



There are a lot more kinds of plots available in the `ggplot2` library. I will be using them thoughout this course and you will become more familiar with them as we go forward.