Intermediate R Programming

Today we are going to go step-by-step through a typical Booth workflow for a regression problem. The steps involved will be:

1. **Loading the Data**
2. **Understanding the Data**
3. **Cleaning the Data**
4. **Performing Analysis**
5. **Visualizing the Results**

The dataset we will be working with today is the "mtcars" dataset, which comes preloaded with RStudio. You can load it anytime to practice by simply referencing it in R:

```
print(mtcars)
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

I asked you to download a slightly modified version of the file to allow us to practice some fundamentals. Let's load that version into R and save it as "data":

```
#The read.csv function allows us to load a comma separated values file into our R workspace
#Remember you can use the ? symbol to load the native R help files at any time!
```

```r
data <- read.csv('D:\\Users\\Jeffrey\\Downloads\\mtcars_missing_data.csv')
#Note that R for Windows requires 2 "\" when calling a filepath
```

Now that we've loaded the data, the first thing we should do is make sure we understand what the data contains. Let's try a couple functions that will be helpful for doing that!

```r
#The str() function tells us the name of each variable in a dataset, its type, and previews some of the
str(data)
```

```
## 'data.frame':    33 obs. of  12 variables:
##  $ X   : Factor w/ 33 levels "AMC Javelin",..: 18 19 5 13 14 32 7 21 20 22 ...
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : int  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : int  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : int  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : int  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: int  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: int  4 4 1 1 2 1 4 2 2 4 ...
```

```r
#The summary() function gives us descriptive statistics for each variable, and crucially the number of 
summary(data)
```

```
##                   X           mpg             cyl             disp      
##  AMC Javelin       : 1   Min.   :10.40   Min.   :4.000   Min.   : 71.1  
##  Cadillac Fleetwood: 1   1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8  
##  Camaro Z28        : 1   Median :19.20   Median :6.000   Median :196.3  
##  Chrysler Imperial : 1   Mean   :20.09   Mean   :6.188   Mean   :230.7  
##  Datsun 710        : 1   3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0  
##  Dodge Challenger  : 1   Max.   :33.90   Max.   :8.000   Max.   :472.0  
##  (Other)           :27   NA's   :1       NA's   :1       NA's   :1      
##        hp             drat             wt             qsec      
##  Min.   : 52.0   Min.   :2.760   Min.   :1.513   Min.   :14.50  
##  1st Qu.: 96.5   1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89  
##  Median :123.0   Median :3.695   Median :3.325   Median :17.71  
##  Mean   :146.7   Mean   :3.597   Mean   :3.217   Mean   :17.85  
##  3rd Qu.:180.0   3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90  
##  Max.   :335.0   Max.   :4.930   Max.   :5.424   Max.   :22.90  
##  NA's   :1       NA's   :1       NA's   :1       NA's   :1      
##        vs               am              gear            carb      
##  Min.   :0.0000   Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4375   Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :1.0000   Max.   :5.000   Max.   :8.000  
##  NA's   :1        NA's   :1        NA's   :1       NA's   :1      
```

```r
#If we just wanted to understand one column in the data, we could do that as well using the $ operator
summary(data$mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   10.40   15.43   19.20   20.09   22.80   33.90       1 
```

Oh no! Jeff is a jerk who has added some missing values to the data. This will ruin our analysis so we have no choice but to learn how to clean data with missings!

First, let's identify where the missings are in our data using the is.na() function and subsetting syntax we learned last week. Let's find all missings for the mpg variable.

```
#Remember, [] is used to subset. The number before the comma is the row, while the number after the com
#If no number is provided, R assumes you want all rows/columns
data[is.na(data$mpg),]
```

```
##                 X mpg cyl disp hp drat wt qsec vs am gear carb
## 33 Tesla Model S  NA  NA   NA NA   NA NA   NA NA NA   NA   NA
```

It appears that all data for the Tesla Model S in row 33 is missing. In this case, it makes sense to remove this row from our dataset before proceeding. Let's do that now.

```
#The complete.case() funciton is a base R function that identifies rows with no missing data. It will m
data_non_missing <- data[complete.cases(data),]
#The head() and tail() functions show you the first/last n observations in the data.frame
tail(data_non_missing,5)
```

```
##                  X  mpg cyl  disp  hp drat    wt qsec vs am gear carb
## 28   Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## 29 Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## 30   Ferrari Dino 19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## 31  Maserati Bora 15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## 32     Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

Note that row 33 is now gone from our data_non_missing data.frame.

Another common data cleaning step is creating categorical variables from numeric ones. For example, let's imagine we do not care about the difference between a 6 cylinder car and an 8 cylinder car. We just want to know if a car has a low amount of cylinders (4) or a high amount of cylinders (>4). Let's create a cyl_status variable to capture this information.

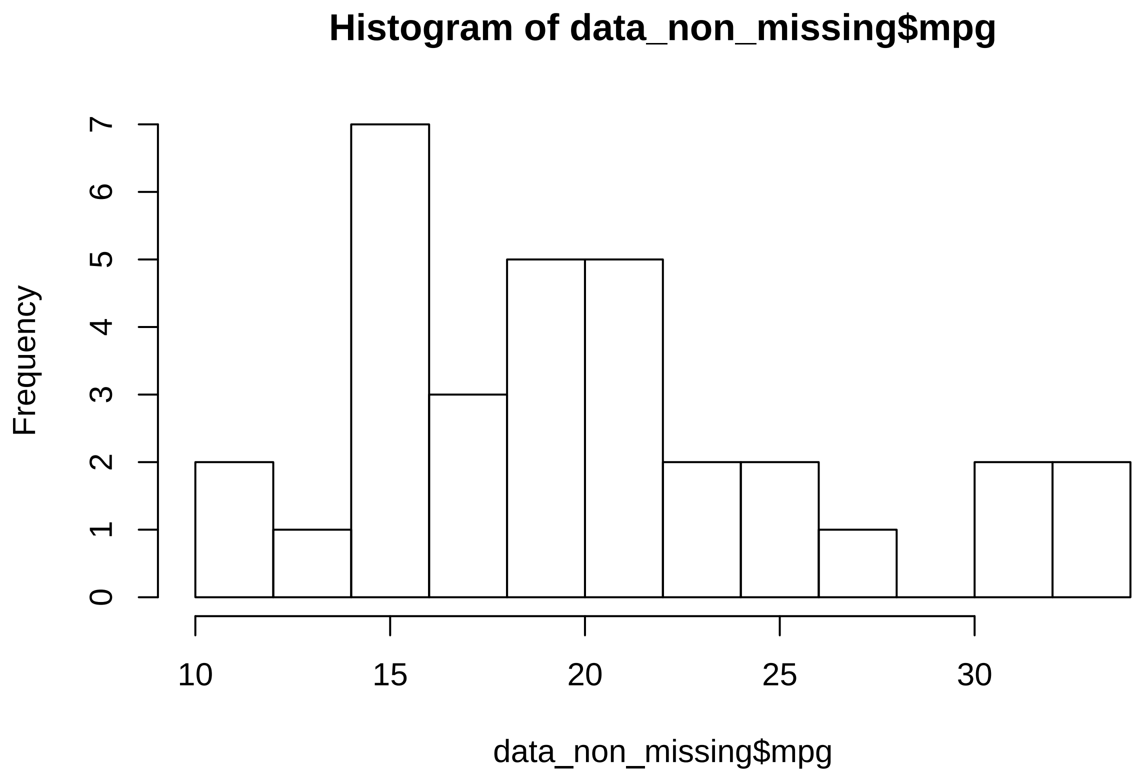
```
#You can assign data to a type by using the as.'type'() function. Here we set our  variable to type fac
#The ifelse function allows you to to specify a logical condition, a value to return if true, and one t
data_non_missing$cyl_status <- as.factor(ifelse(data_non_missing$cyl > 4, 'high', 'low'))
summary(data_non_missing$cyl_status)
```

```
## high  low
##   21   11
```

11 of ours cars have 4 cylinders, while the remaining 21 are V6 or V8s.

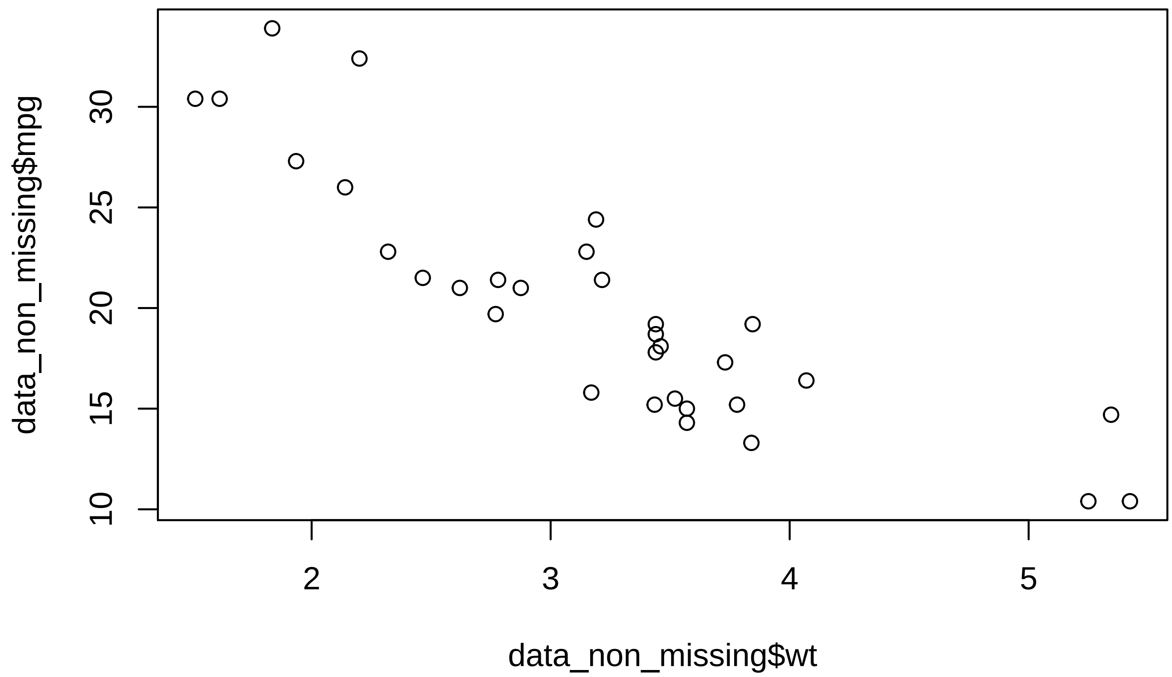
Now that we've removed all missings, let's introduce the concept of plotting. First, let's plot miles per gallon since it will eventually become the dependent variable in our regression.

```
#A histogram is a useful plot for showing a univariate distribution. You can specify the number of buck
hist(data_non_missing$mpg, 10)
```



Now let's try to understand how other variables relate to miles per gallon. To do this, let's begin by plotting the relationship between a car's weight (wt) and its mpg.

```
#The plot function is a base R package for plotting. Eventually you'll want to use ggplot2 to create th
plot(data_non_missing$wt,data_non_missing$mpg)
```



Clearly miles per gallon tends to fall as weight increases, but is this relationship statistically meaningful? Let's find out by running a simple linear regression!

The lm() function stands for 'Linear Model' and is used to run regressions in R. lm() requires a formula detailing the dependent and independent variable(s) in the format 'y ~ x'

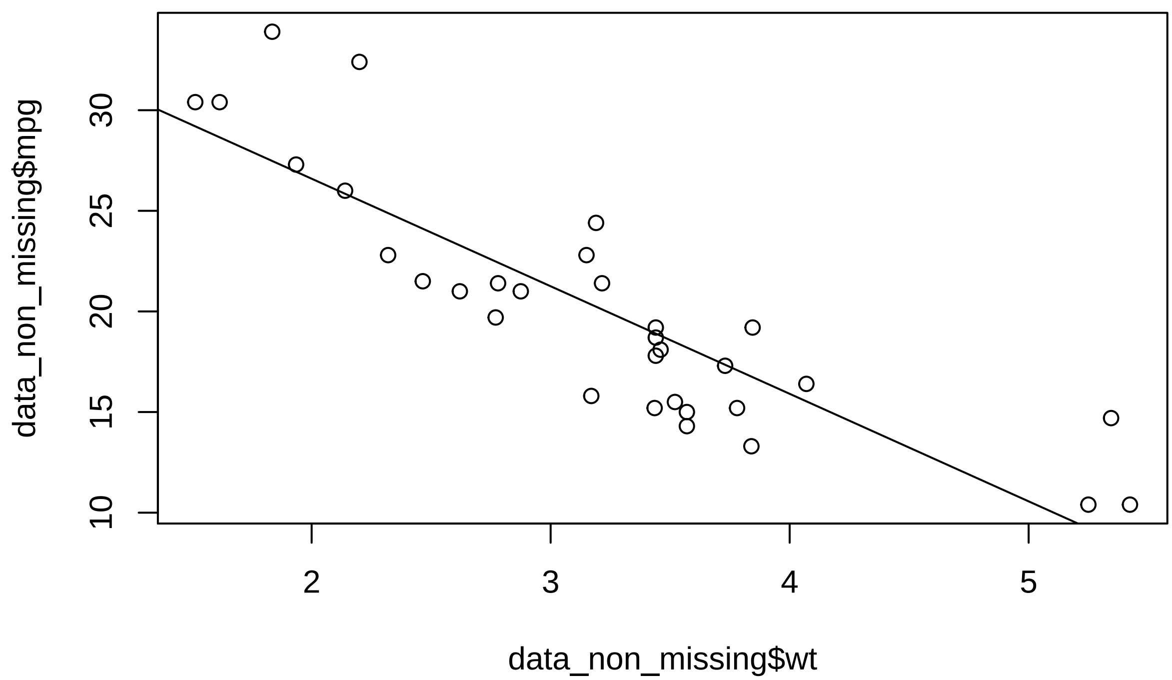
```
#You can save your regression model as an object in your R environment the same as any other variable
reg <- lm(mpg~wt,data=data_non_missing)
#Note that I have used the data= argument so I don't have to reference the dataset each variable comes
#Now let's output the results of the regression using summary()
summary(reg)
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = data_non_missing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

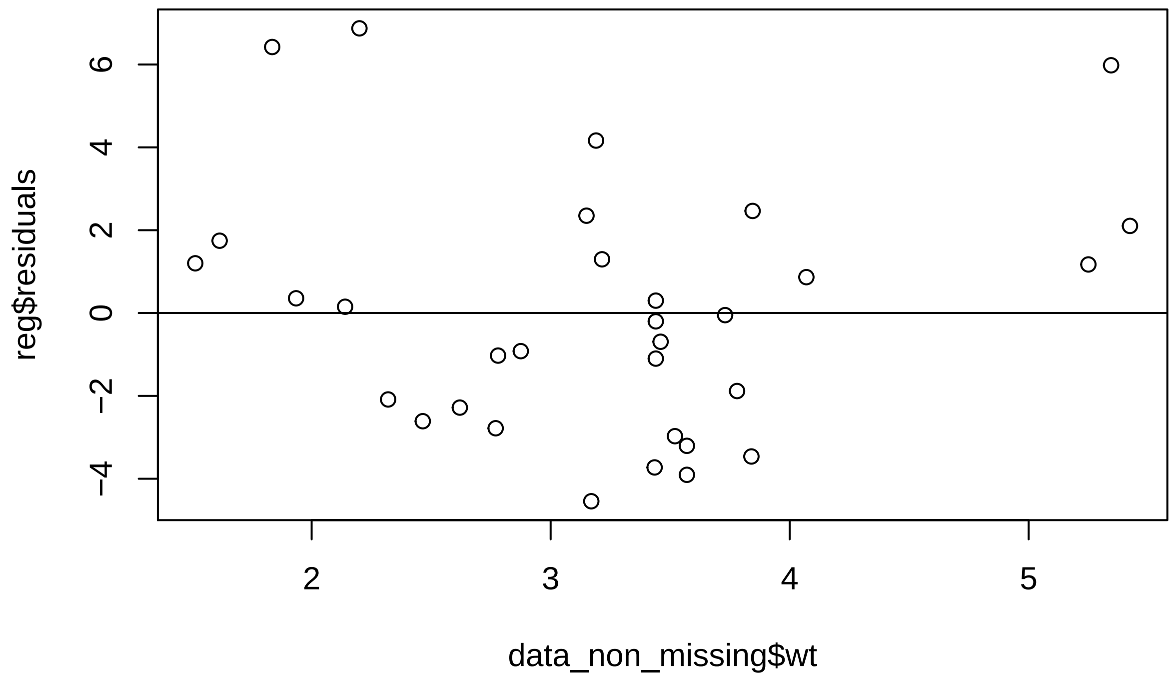
The summary function outputs the formula for the regression, the range of the residuals, the coefficient estimates and their significance, an R-squared value, and a F-statistic. We can see in our regression that an increase in weight of 1 is associated with a decrease in mpg of -5.3445, which is highly significant. Let's overlay a plot of this linear model on top of our data.

```
#You can overlay a regression line on a plot by simply calling the abline() command and adding the regr
plot(data_non_missing$wt,data_non_missing$mpg)
abline(reg)
```



Interesting! It seems like the line is too low at the ends, and too high in the middle. Let's explore this further by graphing the residuals.

```
#One of the components of the regression output are the residuals, or the difference between the predic
#The regression output is in the same order as the input data, so we can simply graph our independent v
plot(data_non_missing$wt,reg$residuals)
abline(0,0)
```



As we suspected, the residuals are not normally distributed! Perhaps we should consider adding a squared weight term to our model. A perfect excuse to explore multiple linear regression!

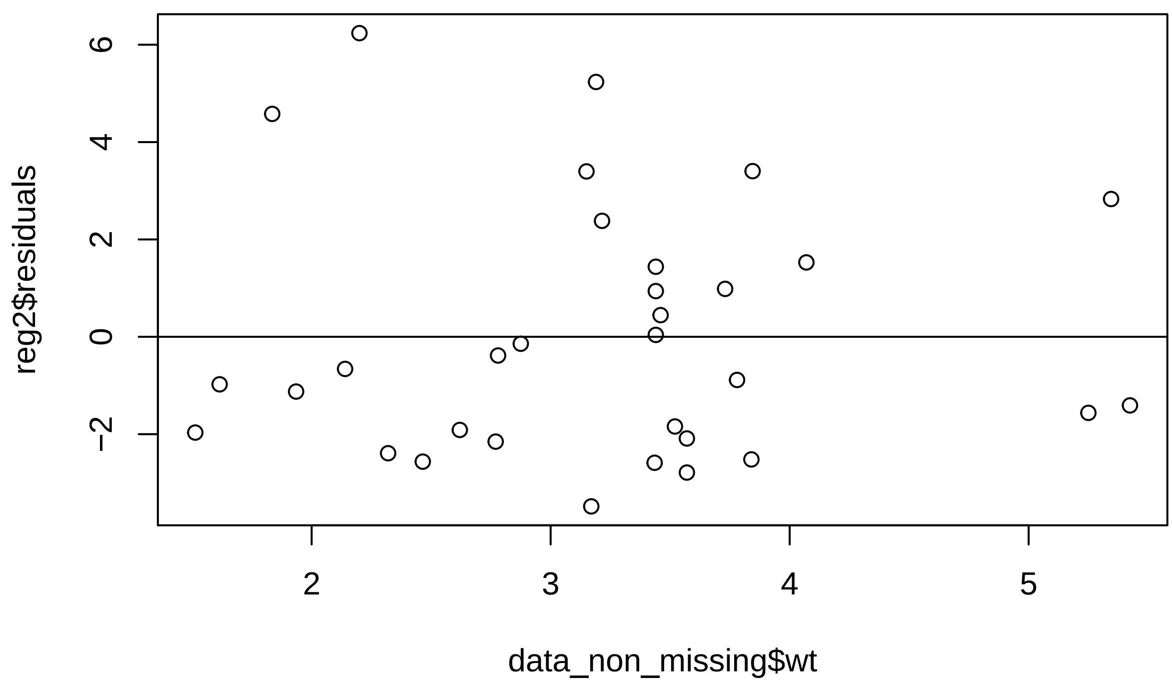
```
#First lets create a squared weight term
data_non_missing$wt2 <- data_non_missing$wt^2

#Multiple linear regression is exactly the same as our prior example, except that additional variables a
reg2 <- lm(mpg~wt+wt2,data=data_non_missing)
summary(reg2)
```

```
##
## Call:
## lm(formula = mpg ~ wt + wt2, data = data_non_missing)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.483 -1.998 -0.773  1.462  6.238
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  49.9308     4.2113  11.856 1.21e-12 ***
## wt          -13.3803     2.5140  -5.322 1.04e-05 ***
## wt2           1.1711     0.3594   3.258  0.00286 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.651 on 29 degrees of freedom
```

```
## Multiple R-squared:  0.8191, Adjusted R-squared:  0.8066
## F-statistic: 65.64 on 2 and 29 DF,  p-value: 1.715e-11
```

Adding a squared term certainly improved our adjusted R-squared. Let's check our residuals again.

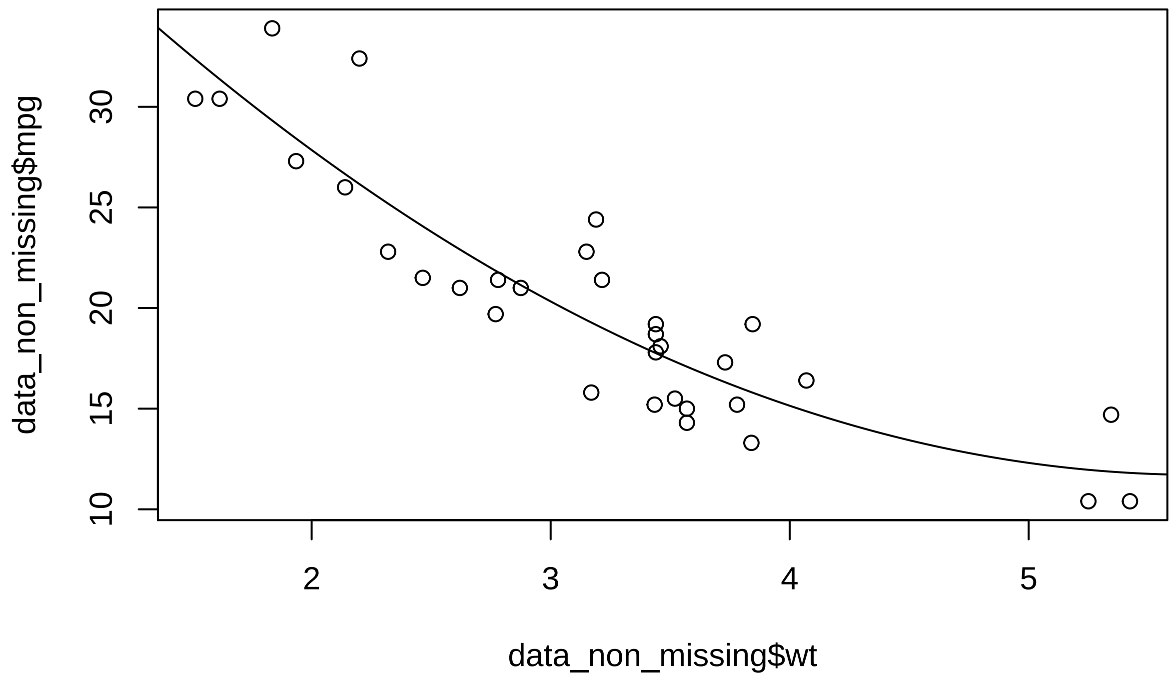
```
plot(data_non_missing$wt,reg2$residuals)
abline(0,0)
```



That certainly looks better!

Because base R functions only plot straight lines, we're going to have to get a little fancy and introduce another important part of regressions in R, the predict() function. The predict function will take a set of independent variable values you give it and output the predict the associated dependent variable value. By creating a sequence of wt measures that are spaced very close together, we will be able to approximate a curve matching our polynomial model. Let's give it a shot.

```
#We want to plot our data from a wt range of 1 to 6. In order to do that let's use the seq() function.
#We'll need a lot of points inbetween to make a smooth-looking curve and a dataframe to store them in
predict_range <- data.frame(wt = seq(1,6,by=0.001), wt2 = seq(1,6,by=0.001)^2)

#Now let's calculate the predicted mpg for each weight using the predict function
#CAUTION: It is paramount that your new_data variables have the EXACT SAME NAME as your regression mode
predict_range$fitted<-predict(reg2,newdata=predict_range)

#Now lets plot the result
plot(data_non_missing$wt,data_non_missing$mpg)
lines(predict_range$wt,predict_range$fitted)
```



Not bad! Obviously this has been a tremendously simplified version of what you will actually do in class, but now you should have the basic skills to get started. Any questions?