N10_stats_n_equations

February 25, 2021

1 N10 formula and basic modeling

When doing statistics, data is analyzed, and values created. The real power of statistics is building models, and fitting data to those models. The models are later used to understand complex systems, and make predictions.

This notebook will cover the simpliest model, the linear model. The linear model assumes an equation in the form:

Y = A + B0*V0 + B1*V1 + B2*V2 …. + e

In this model mechanism, Y is the dependent variable, and V0, V1, V2 are independent variables. A is the intercept, and e represents an error term from the modeled Y to the actual Y. The goal of the statistics software is to minize the total error in the result. As an example, consider the mtcars data frame:

[2]:
```
names(mtcars)
```

1. 'mpg' 2. 'cyl' 3. 'disp' 4. 'hp' 5. 'drat' 6. 'wt' 7. 'qsec' 8. 'vs' 9. 'am' 10. 'gear' 11. 'carb'

Create a hypothesis, or model assumption that mpg depends on disp, hp, and wt (displacement, horsepower, and weight). This is done with a new syntax. It creates an equation or formula. This is used by the optimization software to find the dependent and independent variables:

mpg ~ disp + hp + wt

Note that the '+' does not mean add, it simply implies the variable that will be used in the model fitting optimization.

[3]:
```
model1 <- mpg ~ disp + hp + wt
```

The equation was assigned to a variable in this instance. This equation can then be used to find a model fit. For the first notebook the model will be a linear model 'lm' using the equation above.

[4]:
```
bob<-lm(model1,mtcars)
"Now, print the result"
bob
"Just the first element"
bob[1]
'You can get the coefficients by name using $'
bob$coefficients[1]
'You can get the items by name using the vector indexed with the string'
```

```
bob$coefficients["disp"]
```

'Now, print the result'

```
Call:
lm(formula = model1, data = mtcars)

Coefficients:
(Intercept)         disp           hp           wt
  37.105505    -0.000937    -0.031157    -3.800891
```

'Just the first element'

$coefficients = **(Intercept)** 37.1055052690318 **disp** -0.000937009081489654 **hp** -0.0311565508299456 **wt** -3.80089058263761

'You can get the coefficients by name using $'

(Intercept): 37.1055052690318

'You can get the items by name using the vector indexed with the string'

disp: -0.000937009081489654

The results are an equation that 'best' fits the data.

mpg = 37.1055+(-0.00937)*disp+(-0.031157)*hp+(-3.800891)*wt

The values can be accessed by getting the list elements from the $coefficients or first element of 'bob'

The second element in the results are the residuals, or error term e for each data entry:

[5]:
```
bob[2]
class(bob[2])
```

$residuals = **Mazda RX4** -2.57002989818897 **Mazda RX4 Wag** -1.60080279961632 **Datsun 710** -2.48868290932673 **Hornet 4 Drive** 0.183326888466451 **Hornet Sportabout** 0.459277999818319 **Valiant** -2.37215897262622 **Duster 360** -1.2656476663426 **Merc 240D** 1.48850107329332 **Merc 230** 0.759103273795234 **Merc 280** -0.841143190617462 **Merc 280C** -2.24114319061746 **Merc 450SE** 0.630725656368311 **Merc 450SL** 0.238422858271525 **Merc 450SLC** -1.6715326125966 **Cadillac Fleetwood** 0.0785314964176018 **Lincoln Continental** 1.04020785711813 **Chrysler Imperial** 5.48854558190914 **Fiat 128** 5.78652898226057 **Honda Civic** 1.12400525255386 **Toyota Corolla** 5.86092609974858 **Toyota Corona** -3.10158976163847 **Dodge Challenger** -3.25491890574188 **AMC Javelin** -3.89111273240693 **Camaro Z28** -1.24877729984534 **Pontiac Firebird** 2.53611904904614 **Fiat X1-9** -0.320425919413943 **Porsche 914-2** -0.0236311041590723 **Lotus Europa** 2.65504198993241 **Ford Pantera L** -0.702462515362088 **Ferrari Dino** -1.28877564306916 **Maserati Bora** 2.18315837254461 **Volvo 142E** -1.62958730997494

'list'

You can see that the data fits reasonably well in many cases, but not all. It is often good to get a 'summary' of the error terms to see how well things worked. Convert the list to a vector with the handy 'unlist' function.

[6]:
```
sally <- unlist(bob[2])
class(sally)
summary(sally)
```

'numeric'

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
-3.8910 -1.6400 -0.1720  0.0000  1.0610  5.8610 
```

This shows that the model is off by most ~-4 to 6 mpg. The first and third quartiles are off by about 1.5 mpg. Not a great fit, but not too bad.

The values of disp, hp, and wt don't have the same units. Display a line of the mtcars to see the typical units.

[7]:
```
mtcars[2,]
```

A data.frame: 1 × 11

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0

The displacement is in cubic inches, the hp in units of one, but the weight appears to be in thousands of pounds.

Make a model function. I used the sum function to get the sum of the vector multiplication. Since the first element is a constant, 1.0 was placed there. This will be multiplied with the intercept. The second variable disp will be multiplied by the B0 term, and so on to the vector length.

[8]:
```
mpgmodel <- function(disp,hp,wt) { sum(unlist(bob[1])*c(1.0,disp,hp,wt))}
```

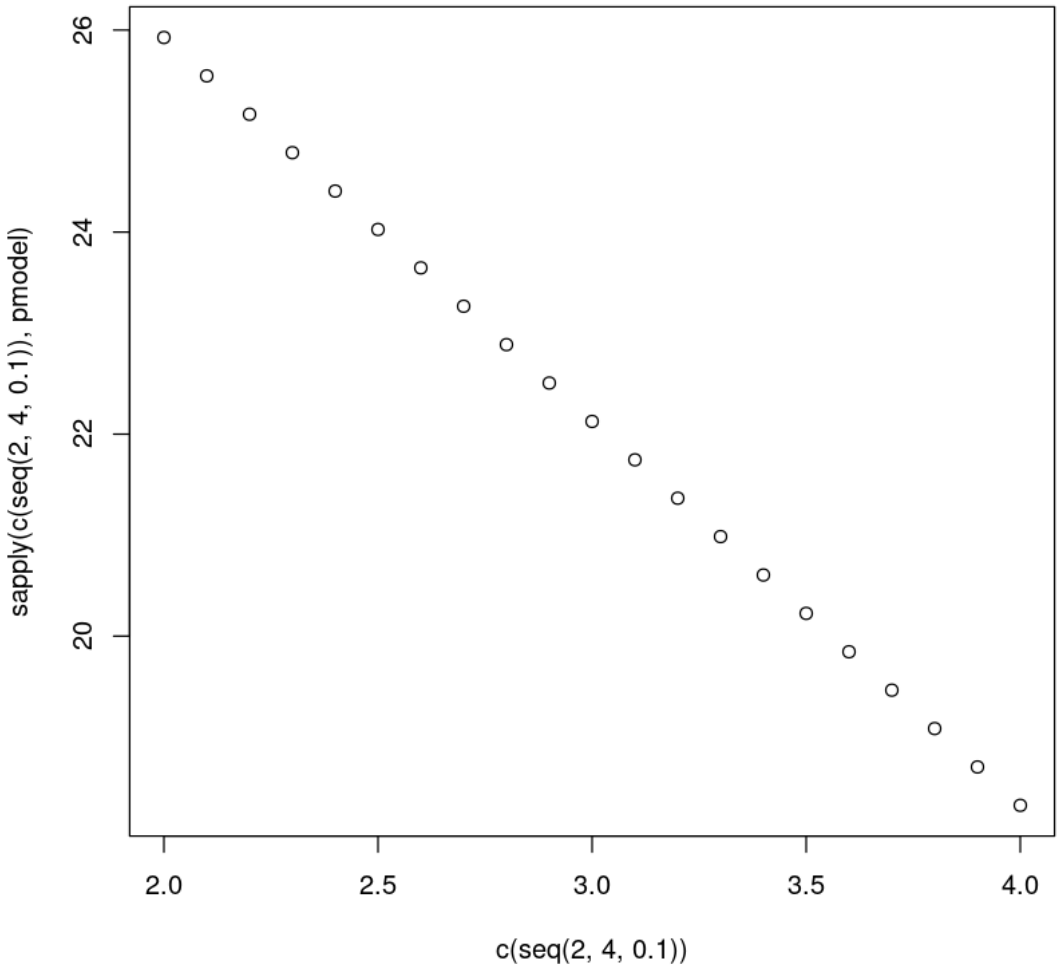
[9]:
```
mpgmodel(160,110,2.9)
```

22.5057805350504

A plot is nice. Let's run it from 2000 to 4000 pounds. It needs a floating point number, so use the 'seq' function to get a sequence with an increment. (Here 0.1)

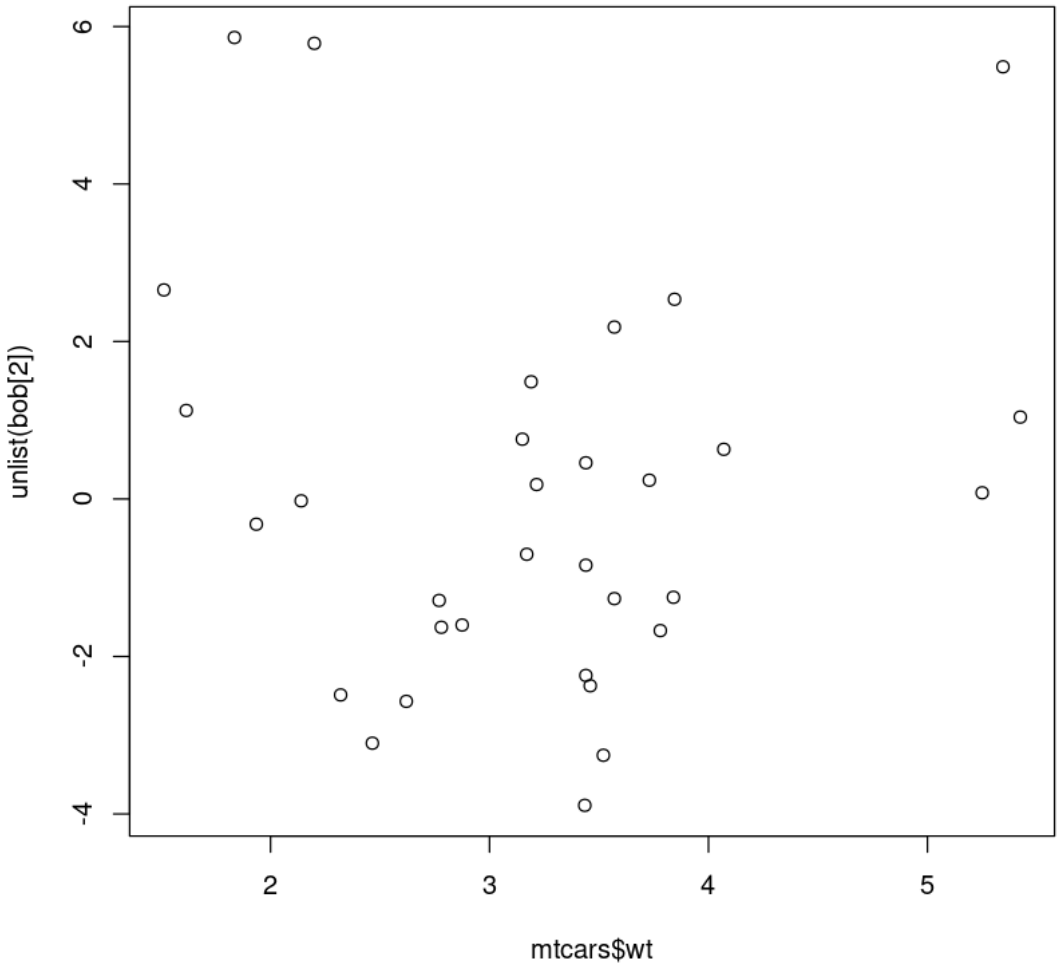
[12]:
```
seq(0,1,0.2)
pmodel <- function(x) { mpgmodel(160,110,x)}
plot(c(seq(2.0,4.0,0.1)),sapply(c(seq(2.0,4.0,0.1)),pmodel))
```

1. 0 2. 0.2 3. 0.4 4. 0.6 5. 0.8 6. 1



You can see it is a straight line. The line comes from the linear model. Take a look at the error terms. (Get a list by unlist on the second element of the lm result)

[11]:
```
plot(mtcars$wt,unlist(bob[2]))
```



No clear pattern can be seen.

Try a correlation of mtcars, and look for the highest correlating items to mpg.

[63]:
```
cor(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs
mpg	1.0000000	-0.8521620	-0.8475514	-0.7761684	0.68117191	-0.8676594	0.41868403	0.6640389
cyl	-0.8521620	1.0000000	0.9020329	0.8324475	-0.69993811	0.7824958	-0.59124207	-0.8108118
disp	-0.8475514	0.9020329	1.0000000	0.7909486	-0.71021393	0.8879799	-0.43369788	-0.7104159
hp	-0.7761684	0.8324475	0.7909486	1.0000000	-0.44875912	0.6587479	-0.70822339	-0.7230967
drat	0.6811719	-0.6999381	-0.7102139	-0.4487591	1.00000000	-0.7124406	0.09120476	0.4402785
wt	-0.8676594	0.7824958	0.8879799	0.6587479	-0.71244065	1.0000000	-0.17471588	-0.5549157
qsec	0.4186840	-0.5912421	-0.4336979	-0.7082234	0.09120476	-0.1747159	1.00000000	0.7445354
vs	0.6640389	-0.8108118	-0.7104159	-0.7230967	0.44027846	-0.5549157	0.74453544	1.0000000
am	0.5998324	-0.5226070	-0.5912270	-0.2432043	0.71271113	-0.6924953	-0.22986086	0.1683451
gear	0.4802848	-0.4926866	-0.5555692	-0.1257043	0.69961013	-0.5832870	-0.21268223	0.2060233
carb	-0.5509251	0.5269883	0.3949769	0.7498125	-0.09078980	0.4276059	-0.65624923	-0.5696071

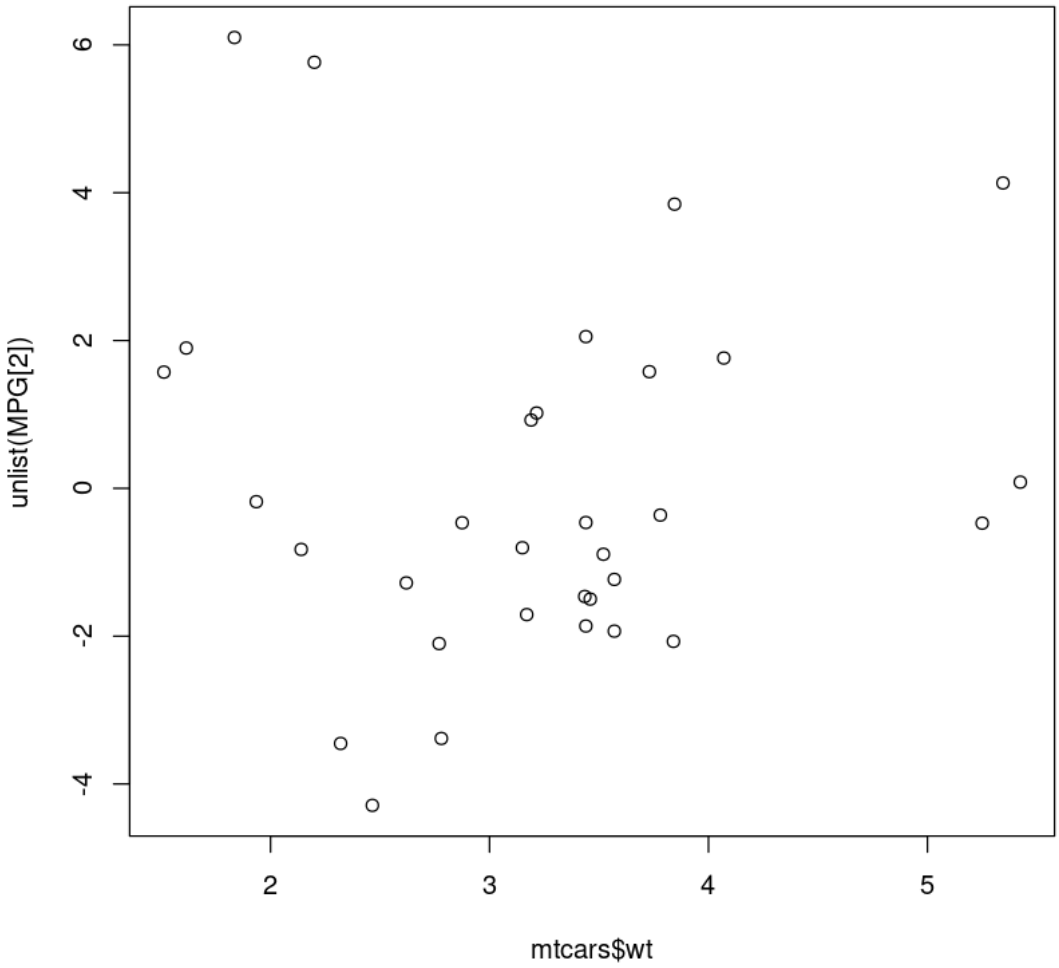
Cyl and wt are the two highest correlating items. (Correlation assumes a linear relationship, and fits the lm well). As an exercise, create an equation with mpg as the dependent variable, and get a summary of the error terms.

[20]:
```
# exercise goes here
core <- mpg ~ cyl + wt
MPG<-lm(core,mtcars)
MPG
error_sum<- unlist(MPG[2])
class(error_sum)
sum
```

```
Call:
lm(formula = core, data = mtcars)

Coefficients:
(Intercept)          cyl           wt
     39.686       -1.508       -3.191
```

[21]:
```
plot(mtcars$wt,unlist(MPG[2]))
```



To submit the results, first save the notebook, then on the JUPYTER, not the browser file area, click print preview. This will result in a nice html image. You can screen capture that, or use the browser print feature to print to a pdf file. (This will depend on how your PC is set up). Upload the results to Canvas.