Regression Class 3 - Covid Update

Model Validation and Polynomials

Load the following libraries and functions:

```
library(tidyverse)
library(lubridate) # needed for homework
library(stringr)
library(ISLR)


rmse <- function(error)
{
  sqrt(mean(error^2))
}

quadratic <- function(a,b,c){
  if(b^2-4*a*c > 0){ # first case D>0
    x_1 = (-b+sqrt(b^2-4*a*c))/(2*a)
    x_2 = (-b-sqrt(b^2-4*a*c))/(2*a)
    result = c(x_1,x_2)
  }
  else if(delta(a,b,c) == 0){ # second case D=0
    x = -b/(2*a)
  }
  else {"There are no real roots."} # third case D<0
}
```

Model Validation

Last week you built a model to predict the price of a car. You then built a testset to validate your model. The testset was just one observation, simulating a site user entering data in a process on Edmunds.com. We relied on lm to estimate model error.

Normally, we want more assurance before we deploy a model to a business process, and there are many resampling and testing methods. We'll cover the validation set approach here *(saving more advanced methods for next semester)*.

The validation set approach breaks the data into two sets, making sure that observations occur in either one or the other, but not both *(models do WAY better if they've seen the data before - which will gives an unrealistic estimate of model performance in the real world)*.

When dealing with **categorical variables**, we have to be careful that data from each and every category are included in both training and testing datasets. For example, if we train a model on different makes of cars, but we didn't include *bmw* in the training data, when *bmw* appears in the test data, the model won't know what to do with it.

Sometimes data is sparse, and we only have one observation in a category. In the simplest case *(like the one here)*, we have two choices: Go out and get more data, or drop the category. More advanced solutions

(covered next semester) include synthesizing new observations, pooling and clustering group data.

Polynomials

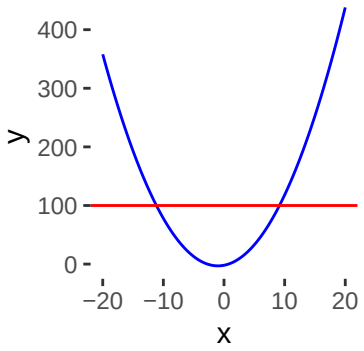
We often find data relationships are not linear. There are many ways to model this *(covered next semester)*. A simple approach for now is to just extend the linear model to accomodate non-linear relationships using polynomials. This approach modifies the value of the independent variable using an exponent, so that the model can *curve* predictions to fit the data. The model is still linear because the the coefficients are not exponentiated. *(linearity is not defined by the data or the shape of the line, it's defined by the coefficient)*. Note that we're added a function to solve quadratic equations.

Recall from High School algebra that a quadratic equation has the form:

$ax^2 + bx + c$

which for $x^2 + 2x - 2$ looks like:

```
data.frame(x = seq(from = -20, to = 20, by = .01)) %>%
  mutate(y = x^2 + 2*x - 2) %>% ggplot(aes(x, y)) +
  geom_line(color = "blue") +
  geom_hline(yintercept = 100, color = "red") +
  theme(panel.background = element_rect(fill = "white") )
```



..so, two solutions for x when y = 100, which can be solved with the quadratic formula:

$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

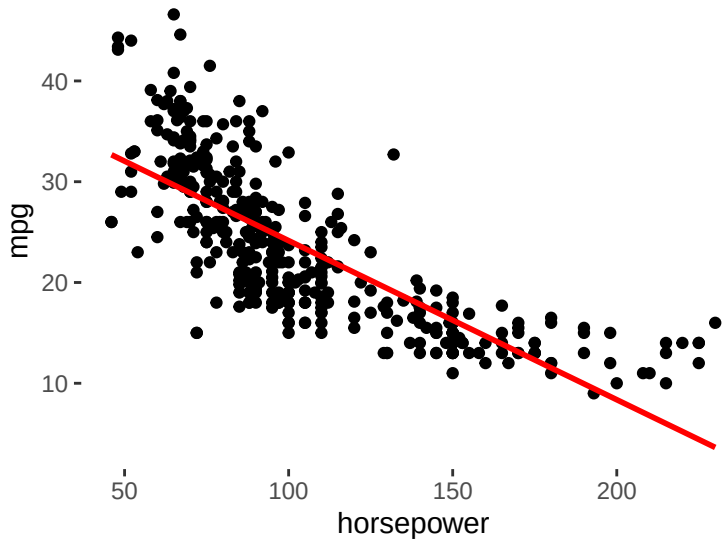
That's what the function does *(prove for yourself)*.

Case Walkthrough

Let's get some new Auto data from the ISL package, and take a look. We're adding a linear equation to the plot using geom_smooth *(method = lm)* to get a feel for how a linear model might fit the data:

```
dfMPG <- Auto

p <- ggplot(data = dfMPG) +
  geom_point(mapping = aes(x = horsepower, y = mpg)) +
  geom_smooth(method = lm, mapping = aes(x = horsepower, y = mpg), col = "red", se=F) +
  theme(
    panel.background = element_rect(fill = "white")
  )
p
```



Let's also take a look at the correlations between the independent and dependent variables *(good practice)*:

```
CorMPG <- round(cor(data.matrix(dfMPG)),2)
# notice the correlation between  cylinders, diplacement and horsepower - duh
# also notice what a mess name is, but the make is always in front
# we're going to make this simple and just take the big ones: name and horsepower

knitr::kable(CorMPG) %>%
  kable_styling(full_width = F, bootstrap_options = "striped", font_size = 9)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
mpg	1.00	-0.78	-0.81	-0.78	-0.83	0.42	0.58	0.57	0.27
cylinders	-0.78	1.00	0.95	0.84	0.90	-0.50	-0.35	-0.57	-0.28
displacement	-0.81	0.95	1.00	0.90	0.93	-0.54	-0.37	-0.61	-0.29
horsepower	-0.78	0.84	0.90	1.00	0.86	-0.69	-0.42	-0.46	-0.23
weight	-0.83	0.90	0.93	0.86	1.00	-0.42	-0.31	-0.59	-0.25
acceleration	0.42	-0.50	-0.54	-0.69	-0.42	1.00	0.29	0.21	0.13
year	0.58	-0.35	-0.37	-0.42	-0.31	0.29	1.00	0.18	0.08
origin	0.57	-0.57	-0.61	-0.46	-0.59	0.21	0.18	1.00	0.36
name	0.27	-0.28	-0.29	-0.23	-0.25	0.13	0.08	0.36	1.00

As you can see, there's a lot of varibles with ~ 70%-80% correlation to mpg. These are surrogates, so we'll just use horsepower and we also want the make *(pulling make out of name and dropping model)* We'll use str_split to pull out the make, and convert that to a factor:

```
dfMPG <- select(dfMPG, mpg, name, horsepower)
# pick up the make only
dfMPG$make <- factor(str_split_fixed(dfMPG$name, " ", 4)[,1])

dfMPG$name <- NULL
```

Now that we have the data tidy, it's time to start thinking about how we're going to build our model. There are several issues to be considered:

1. We need to break the data into training and test sets. We'll use 60% for training *(just personal preference)*.

To do that, we'll create sampleID's using the rowid_to_column function, and use SampleID as a primary key to manage unique observations.

2. We need to group the data into makes, and count the number in each group *(note, we're not summarizing here - we want to keep the individual observations so we use mutate. We just need to know how many are in the group)*. Then we just get rid of makes that don't have more than 1 observation.

3. Then we create training and test sets: we sample 60% for the grouped set *(a grouped dataframe will make sample_frac consider all groups)*. Then we create a Testset using an anti_join, which joins the SampleID's and then takes everything that's **not** in Trainset.

```
dfMPG <- rowid_to_column(dfMPG, var="SampleID") # this creates a primary key for sampling

by_MakeStyle <- dfMPG %>% group_by(make) %>% mutate(cnt = n()) %>% filter(cnt > 1)

TrainSet = sample_frac(by_MakeStyle, .6)
TestSet = anti_join(by_MakeStyle, TrainSet, by = "SampleID")
nrow(TrainSet)+nrow(TestSet)
```

```
## [1] 384
```

```
#ck
anti_join(TrainSet, TestSet, by = "make")
```

```
## # A tibble: 0 x 5
## # Groups:   make [1]
## # ... with 5 variables: SampleID <int>, mpg <dbl>, horsepower <dbl>,
## #   make <fct>, cnt <int>
```

Now we're ready to build a model, and we use the TrainSet to do that. We get the residual error *(using the TEST data)*, and save that for later comparison *(sb around 4.92)*:

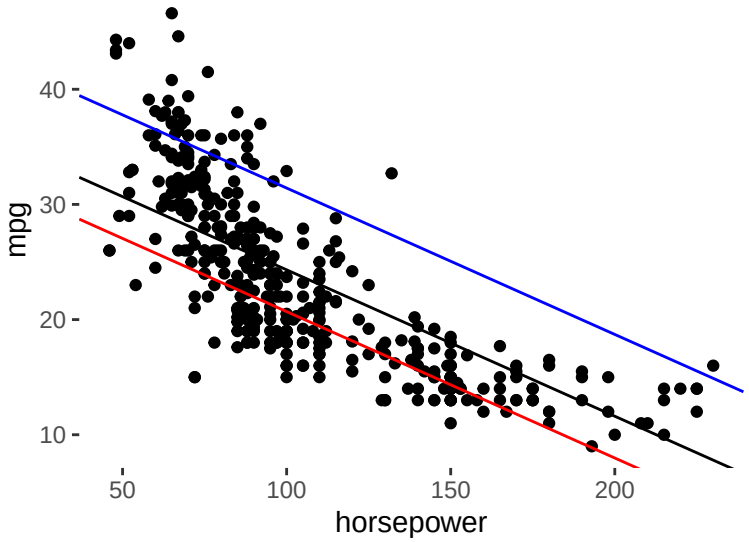
```
mod1 <- lm(mpg ~ make + horsepower, data = TrainSet)
mod1rmse = rmse(predict(mod1, TestSet) - TestSet$mpg)
```

First, we're going to narrow down to just a few makes: audi, honda, and ford. Get the coefficients and plot out the regression lines:

```
# notice that you can use the variable name instead of numeric position

Intercept <- coef(mod1)["(Intercept)"]
Audi <- coef(mod1)["makeaudi"]
Honda <- coef(mod1)["makehonda"]
Ford <- coef(mod1)["makeford"]
Horsepower <- coef(mod1)["horsepower"]

p <- ggplot(data = dfMPG) +
  geom_point(mapping = aes(x = horsepower, y = mpg)) +
  geom_abline(intercept = Intercept+Audi, slope = Horsepower) +
  geom_abline(intercept = Intercept+Honda, slope = Horsepower, color = "blue") +
  geom_abline(intercept = Intercept+Ford, slope = Horsepower, color = "red") +
  theme(
    panel.background = element_rect(fill = "white")
      )
p
```



Now, let's see if we can improve our model. If you look at the fit above, you should be able to see that the relationship between horsepower and mpg is nonlinear. We'll add a polynomial variable to the model by using **I(horsepower^2)** in lm, which tells it to square the **values** in horsepower, and then create 2 coefficients: 1 for horsepower, and 1 for $horsepower^2$:

```
mod2 = lm(mpg ~ make + horsepower + I(horsepower^2), data = TrainSet)
mod2remse = rmse(predict(mod2, TestSet) - TestSet$mpg)
```

And the rmse is reduced by .4 *(10%)*, which is a substantial improvement. Now, we'll narrow down further to just Ford *(doing this for clarity)* and take a look at the lines again *(this time, we'll just use geom_line - abline is linear only)*:

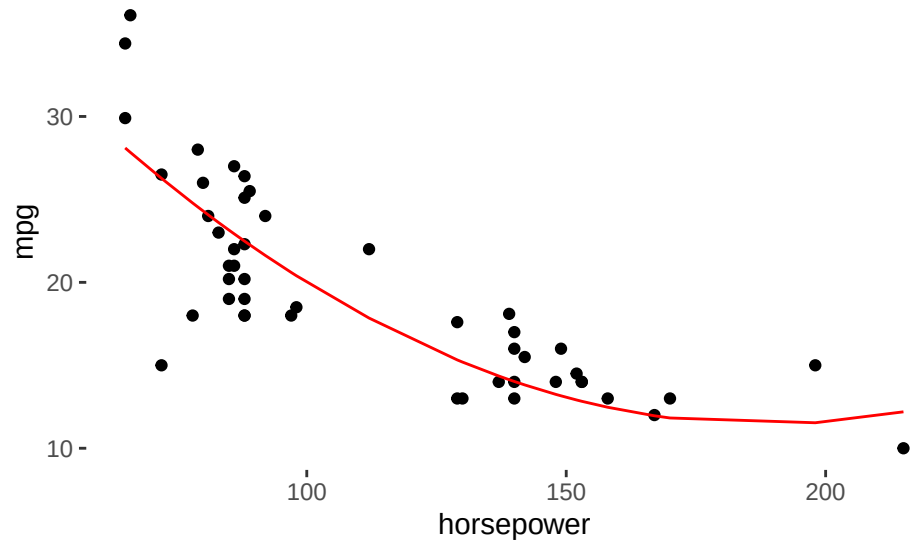
```
# pull the coefficients for Ford

Intercept <- coef(mod2)["(Intercept)"]
Ford <- coef(mod2)["makeford"]
Horsepower <- coef(mod2)["horsepower"]
Phorsepower <- coef(mod2)["I(horsepower^2)"]

dfFord <- filter(dfMPG, make == "ford")

dfFord$pPredict <- (Intercept + Ford) +
  (Horsepower*dfFord$horsepower) +
  (Phorsepower * (dfFord$horsepower^2))

pFord <- ggplot(data = dfFord, aes(x = horsepower, y= mpg)) + geom_point() +
  geom_line(data = dfFord, aes(x = horsepower, y = pPredict), color = "red") +
  theme(
    panel.background = element_rect(fill = "white") # I do this so it's easier to see in  class
  )
pFord
```



Obviously a better fit. That's good.

Now that we have good coefficients, we can answer questions like:

What's the mpg for a ford if it has 150 horsepower? If I wanted a Ford with 25mpg, how much horsepower would I get *(this uses the quadratic equation - which gives 2 solutions - you use common sense to figure out which one works)*?

```
x = 150
mpg = (Intercept + Ford) + (Horsepower*x) + (Phorsepower * (x^2))

y = 25
# using quadratic formula to solve here
hp = quadratic(Phorsepower, Horsepower, ((Intercept + Ford)-y))
```

Homework

Use the ProductSalesvs.csv file. Product is the categorical variable, and you'll need to use pivot_longer to tidy that up. Also, you'll need to convert the WkBeg to date.

Dividing up the training and test data is a little eaiser here. We want to use all the data <2015-01-01 for training, and >= 2015-01-01 for testing.

Then build your model using the Training data, and Test it using the Test data. Your rmse should be ~ 13.8. Then, predict new sales in the Test data and plot as shown *(you don't have to plot the vertical line - just for clarity)*: