Unit-8c

Max

7/30/2019

Agenda

- details of categorical predictors
- Variable Selection and Model Building
- Exercises

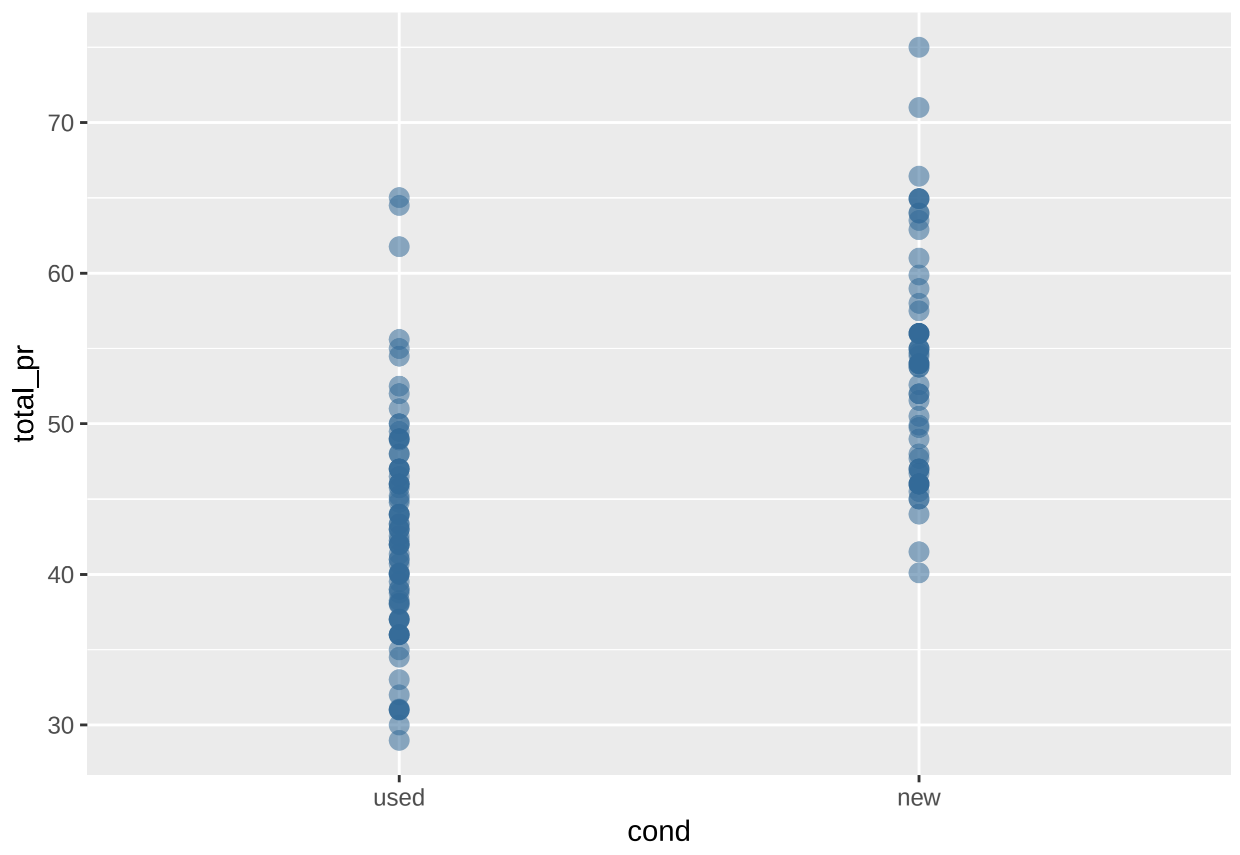
Two Level Categoricals

data: 143 auction final sale prices of *Markokart* on ebay

```
data_url <- "https://www.openintro.org/stat/data/rda/mariokart.rda"
load(url(data_url))
```

cond	total_pr
new	51.55
used	37.04
new	45.50
new	44.00
new	71.00
new	45.00

```
p=qplot(cond,total_pr,data=mariokart,size=I(3),alpha=0.5,colour=4)
p+theme(legend.position = "none")
```



- To do regression equation, we must convert the categories into a numerical form.
- We will do so using an indicator variable called `cond_new`, which takes value 1 when the game is new and 0 when the game is used.

Then linear model may be written as d $\widehat{\text{price}} = \beta_0 + \beta_1 \times \text{cond\_new}$

The R code to achieve this is quite instructive

```
mariokart <- mutate(mariokart,cond_new=as.integer(cond=="new"))
```

Now we are ready to run our linear regression

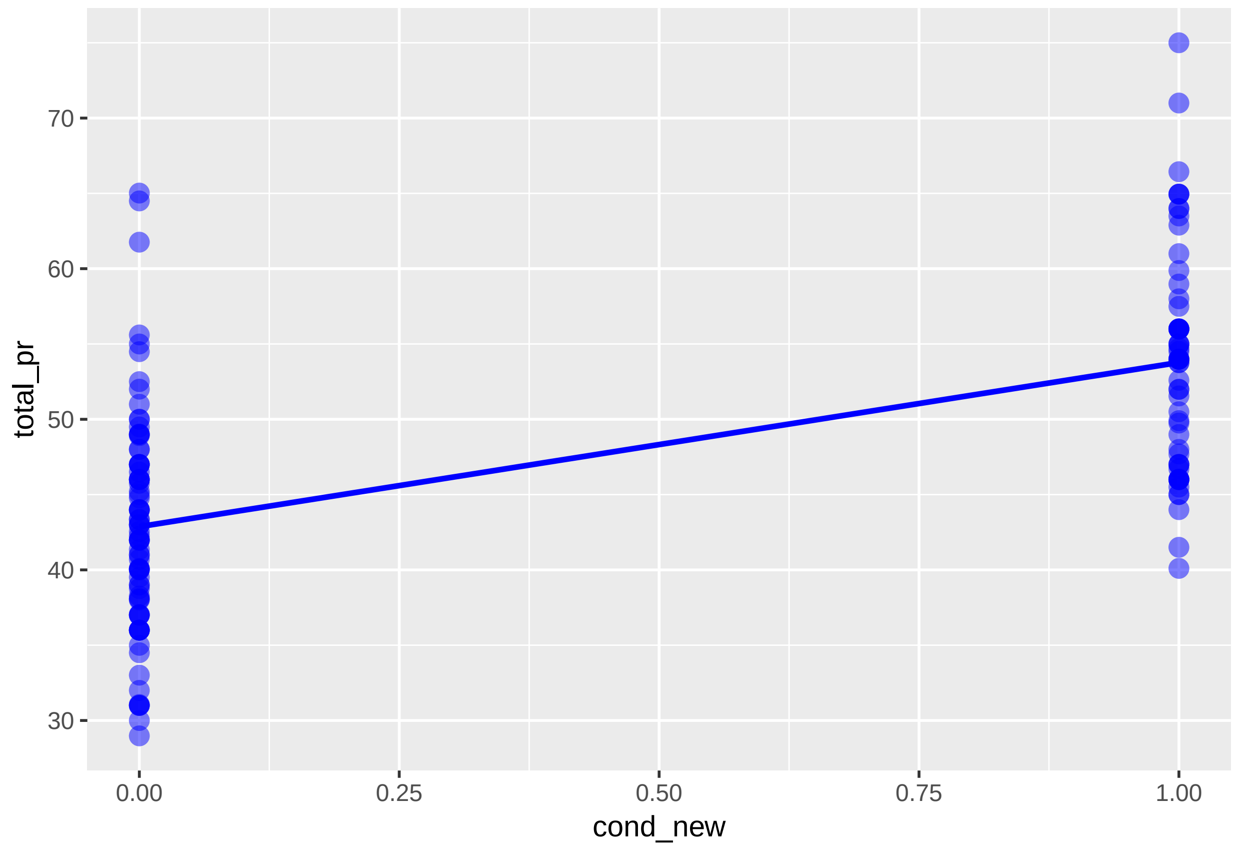
```
lm(total_pr ~ cond_new,data=mariokart)
```

```
##             Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 42.87110  0.8139806 52.668454 1.019368e-93
## cond_new    10.89958  1.2583388  8.661881 1.055698e-14
```

$$\widehat{\text{price}} = 42.87 + 10.9 \times \text{cond\_new}$$

- The estimated intercept is the value of the response variable for the first category (i.e. the category corresponding to an indicator value of 0).
- The estimated slope is the average change in the response variable between the two categories.
- Essentially, all we did was **connect the means** of the two groups with a line, (same as an anova)

```r
p <- ggplot(mariokart, aes(x = cond_new, y = total_pr))
p + geom_point(alpha = 5/10,size=3,colour=4) +
  geom_smooth(method = "lm", se = FALSE, color = "blue")
```



Numerical + categorical

data: We will look at the `mtcars` dataset

predictors: `mpg`, `hp` and `am`

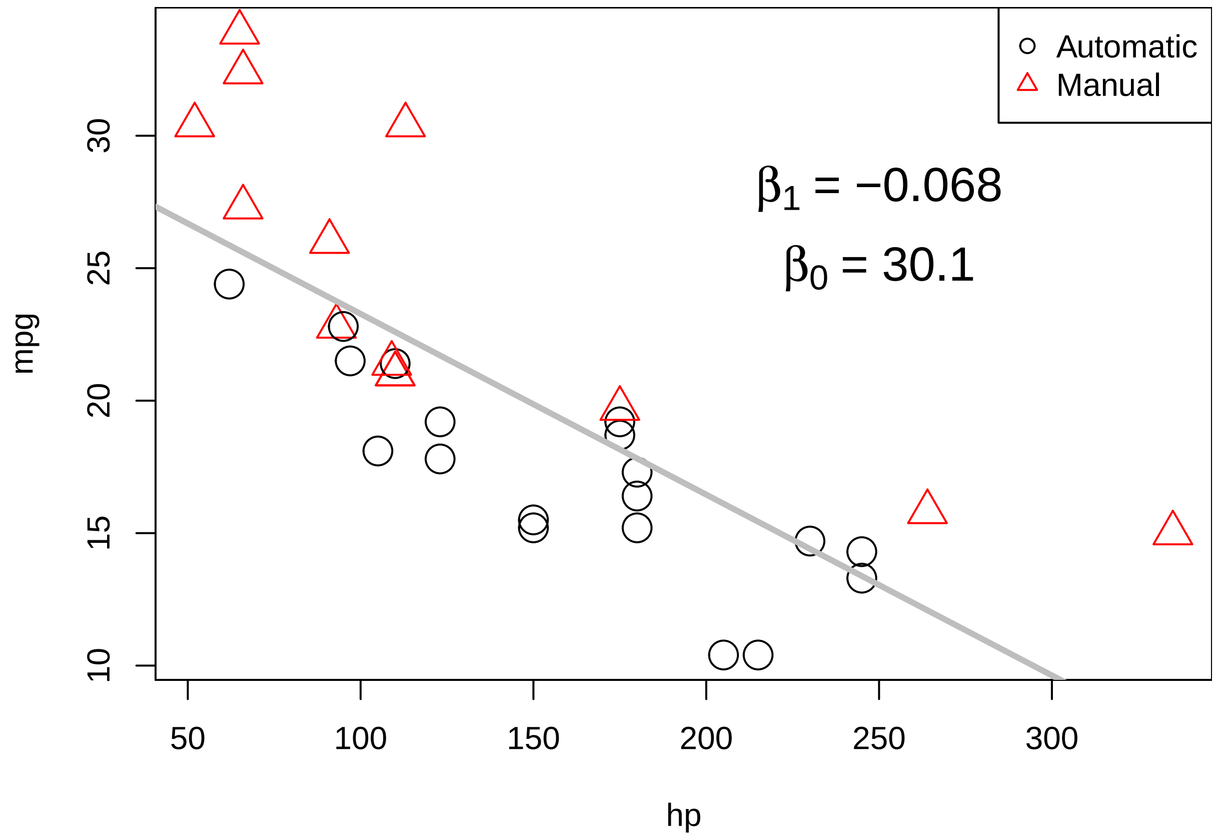
- `mpg`: fuel efficiency, in miles per gallon
- `hp`: horsepower, in foot-pounds per second
- `am`: transmission. Automatic or manual.

First let's just look at the model

$$Y = \beta_0 + \beta_1 x_1 + \epsilon$$

where x_1 is horsepower and Y is fuel efficiency

```r
mpg_hp_slr = lm(mpg ~ hp, data = mtcars)
```



Notice there is a systematic issue with circles and triangles

---

let's try

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

where $x_2 = 0$ if automatic and $x_2 = 1$ if manual transmission

notice this effectively creates two models. When $x_2 = 0$,

$$Y = \beta_0 + \beta_1 x_1 \epsilon$$

and when $x_2 = 1$,

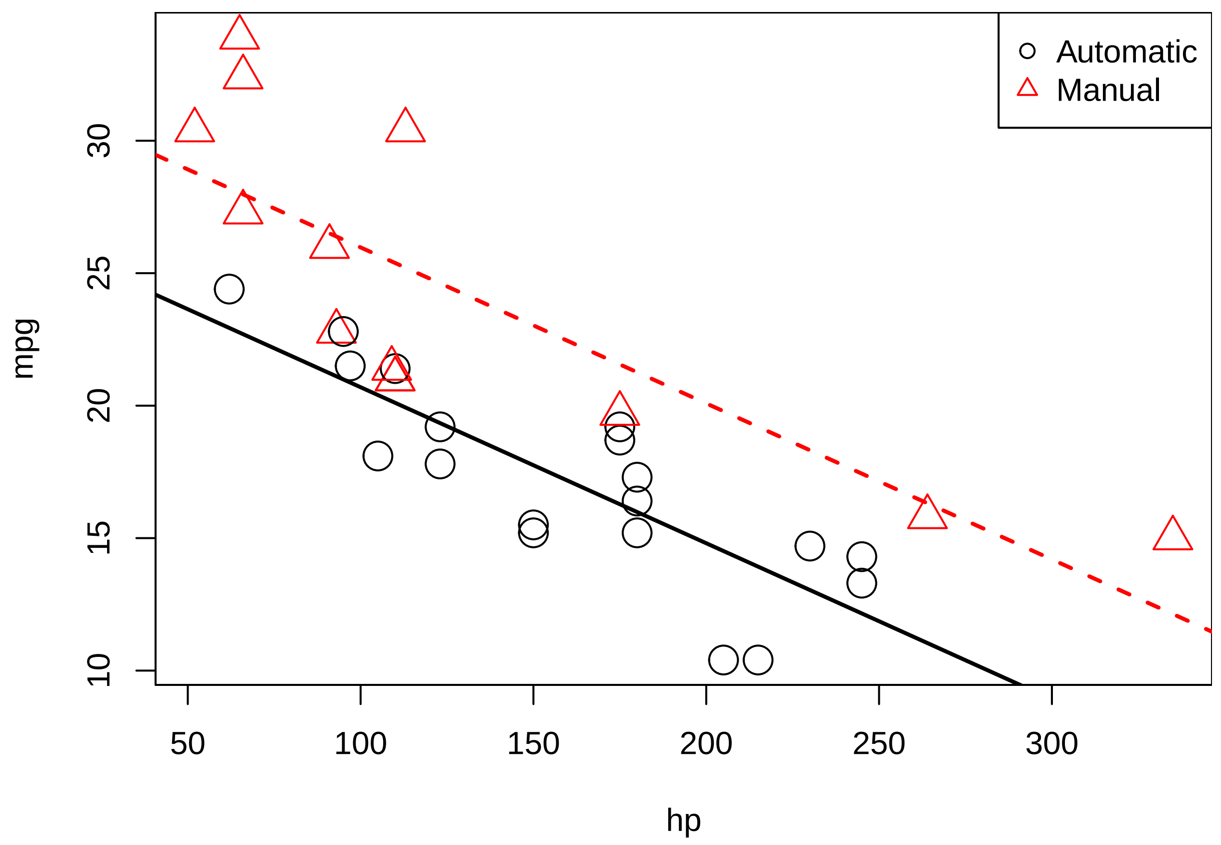
$$Y = (\beta_0 + \beta_2) + \beta_1 x_1 + \epsilon$$

The models will have same slope (parallel), different intercepts

```
mpg_hp_add = lm(mpg ~ hp + am, data = mtcars)
```

```
mpg_hp_add$coefficients
```

```
## (Intercept)          hp          am
##  26.5849137  -0.0588878   5.2770853
```



Numerical-Categorical interaction

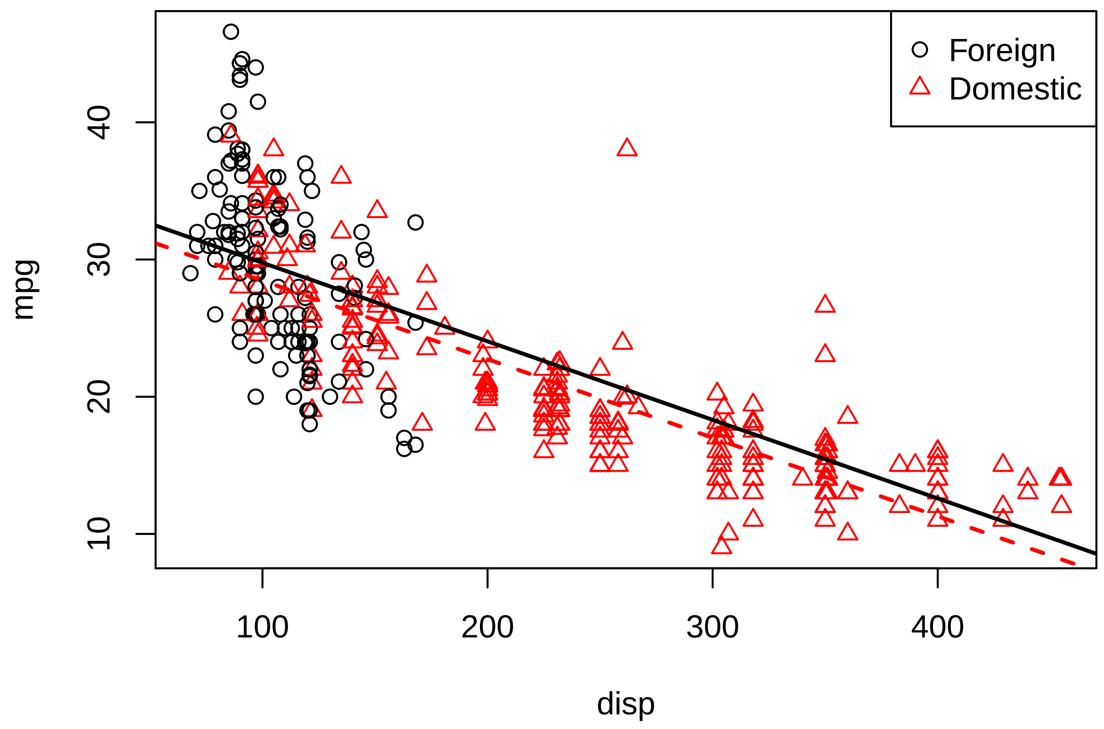
To remove the "same slope" restriction, we discuss interaction

predictors: `mpg`, `hp` and `am`

- Y is `mpg`: fuel efficiency, in miles per gallon
- x_1 is `disp`: the displacement in cubic inches,
- x_2 is `domestic`: an indicator variable domestic or foreign make

The non-interacting model is:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$



the issue is that the fit is ok for the red, not great for black

---

we would like a model that allows for two different slopes

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \epsilon$$

this creates two sub-models with **different slopes**.

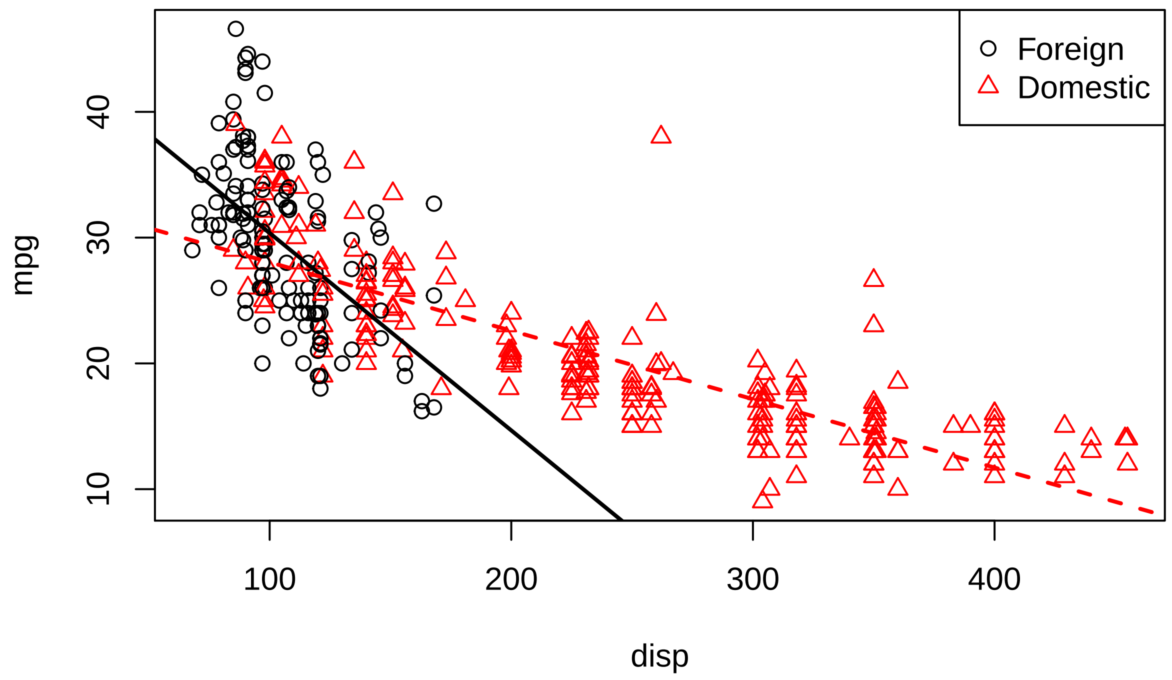
When $x_2 = 0$, (foreign cars)

$$Y = \beta_0 + \beta_1 x_1 + \epsilon$$

and when $x_2 = 1$ (domestic cars),

$$Y = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) x_1 + \epsilon$$

```
mpg_disp_int = lm(mpg ~ disp + domestic+disp:domestic, data = autompg)
```



- In general, you can have factors (categorical predictors) with multiple levels, as well as numerical-numerical interactions
- Multiple levels can be treated with multiple indicator variables
- To learn more about this check out

https://daviddalpiaz.github.io/appliedstats/categorical-predictors-and-interactions.html#factor-variables

Exact Collinearity

```
gen_exact_collin_data = function(num_samples = 100) {
  x1 = rnorm(n = num_samples, mean = 80, sd = 10)
  x2 = rnorm(n = num_samples, mean = 70, sd = 5)
  x3 = 2 * x1 + 4 * x2 + 3
  y = 3 + x1 + x2 + rnorm(n = num_samples, mean = 0, sd = 1)
  data.frame(y, x1, x2, x3)
}
exact_collin_data = gen_exact_collin_data()
```

```
##          y        x1       x2       x3
## 1 154.5899  87.07094 63.18365 429.8765
## 2 149.9183  83.78278 61.96626 418.4306
## 3 147.8021  78.99524 67.12420 429.4873
## 4 180.3074 100.68561 75.63863 506.9257
## 5 159.6184  88.02572 69.69488 457.8310
## 6 142.7751  65.78119 74.47113 432.4469
```

---

x_1, x_2, x_3 are not linearly ind, so Cov matrix is singular

```
X = cbind(1, as.matrix(exact_collin_data[,-1]))
solve(t(X) %*% X)
```

```
## Error in solve.default(t(X) %*% X): system is computationally singular: reciprocal condition number
```

```
lm(y ~ . , data=exact_collin_data)
```

```
##
## Call:
## lm(formula = y ~ ., data = exact_collin_data)
##
## Coefficients:
## (Intercept)           x1           x2           x3
##       1.562        1.007        1.013           NA
```

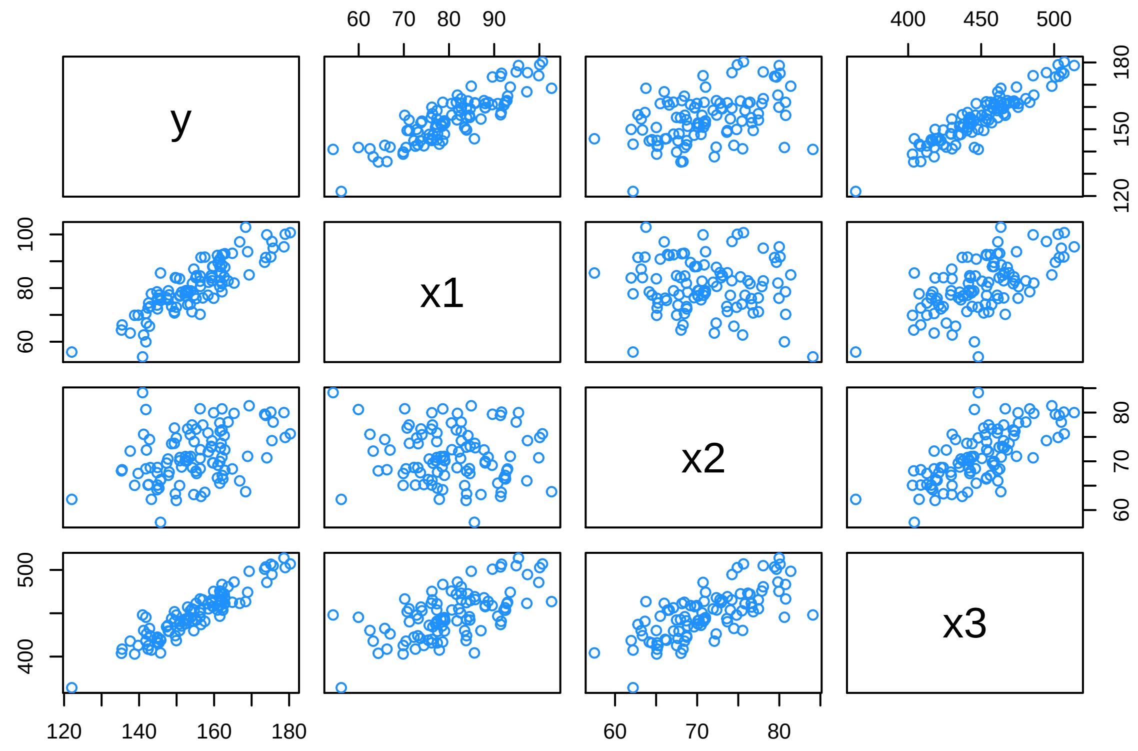
- Note that linear independence is not exactly the same thing as correlation
- However, to get an idea of collinearity you can examine the correlations, and this is usually an indicator of collinearity
- You can use the `cov()` and `cor()` functions in R to see the (symmetric) matrices

```
round(cor(X),3)
```

```
##       x1    x2    x3
## x1 1.000 0.828 0.947
## x2 0.828 1.000 0.964
## x3 0.947 0.964 1.000
```

You can check for correlations between variables with `pairs`

```
pairs(exact_collin_data, col = "dodgerblue")
```



The estimator for the β vector is

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- Typically some variables will be approximately collinear but not exactly
- Collinearity results in higher variance in the estimator $\hat{\beta}$ but the same point estimates
- Collinearity does not affect prediction errors, but does affect explanatory error
- That is collinearity increases $\text{Var}(\hat{\beta})$ but does not increase the MSE
- Collinearity will affect your model's ability to **explain things**

Quality Criterion

Impossible to add a predictor and make R^2 or $RMSE$ worse

But too many parameters is also a bad thing

Reminder of Notations:

$\text{RMSE} = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$

$R^2 = 1 - \frac{RSS}{SST}$

$SST = \sum(y_i - \overline{y})^2$ *total sum of squares* (AKA SS_{tot})

$SSG = \sum(\hat{y}_i - \overline{y})^2$ *explained sum of squares* (AKA SS_{reg})

$RSS = \sum(y_i - \hat{y}_i)^2$ *residual sum of squares* (AKA SS_{res})

The Akaike information criteria (AIC) is

$$\text{AIC} = n \log\left(\frac{RSS}{n}\right) + 2p$$

- Reflects the tradeoff between reduced error (RSS) and increasing the number of parameters, p
- If AIC is smaller, then you have a **better** model

The adjusted R^2_a similarly adds a penalty on number of parameters

$R^2_a = 1 - \frac{n-1}{n-p}(1 - R^2)$

- like R^2, for adjusted, larger is still better

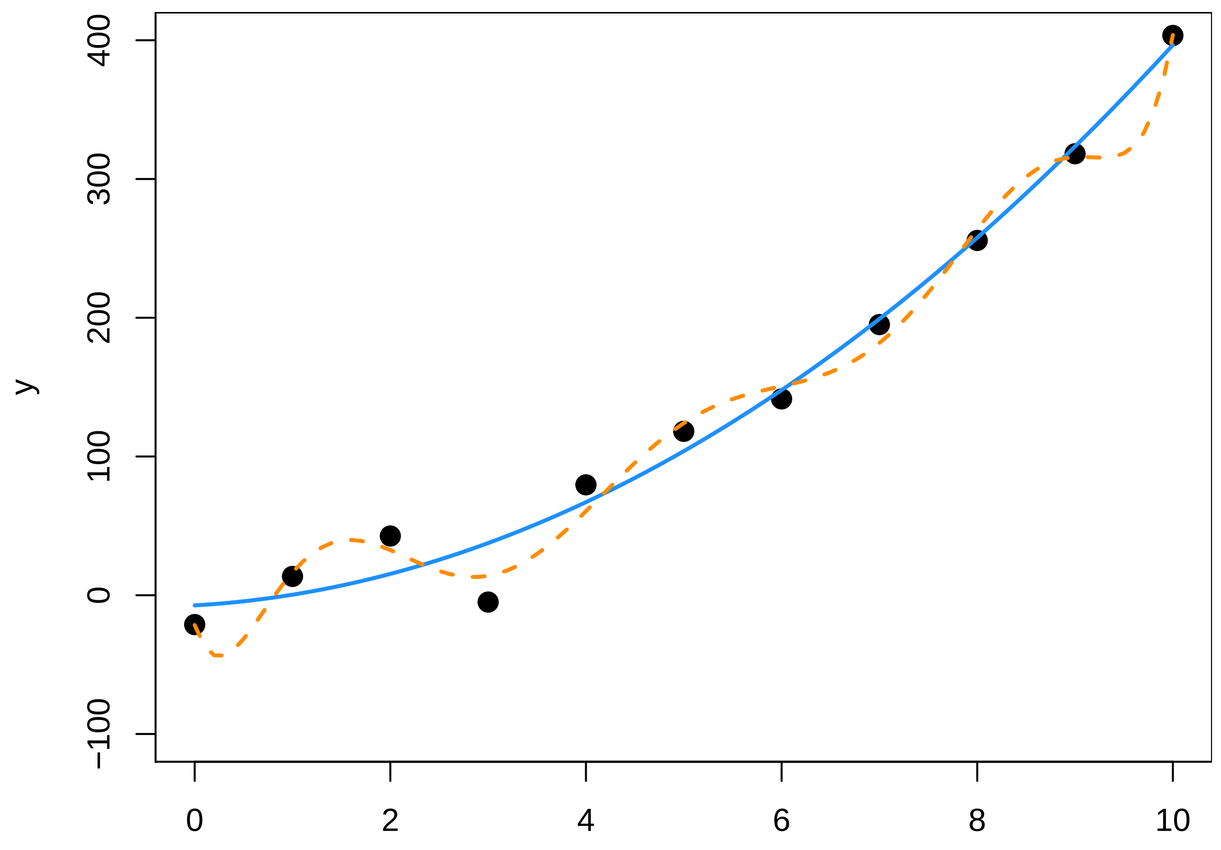
Cross-Validated RMSE

```r
make_poly_data = function(sample_size = 11) {
  x = seq(0, 10)
  y = 3 + x + 4 * x ^ 2 + rnorm(n = sample_size, mean = 0, sd = 20)
  data.frame(x, y)
}
set.seed(1234)
poly_data = make_poly_data()
```

Here we have generated data where the mean of Y is a quadratic function of a single predictor x, specifically,

$$Y = 3 + x + 4x^2 + \epsilon$$

```r
fit_quad = lm(y ~ poly(x, degree = 2), data = poly_data)
fit_big  = lm(y ~ poly(x, degree = 8), data = poly_data)
```



The dashed orange curve fits the points better, making smaller errors, however it is fitting the random noise. This is **overfitting**.

```r
sqrt(mean(resid(fit_quad) ^ 2))
```

```
## [1] 17.61812
```

```r
sqrt(mean(resid(fit_big) ^ 2))
```

```
## [1] 10.4197
```

overfitting is usually controlled by looking at cross-validation error, which leaves out some data while estimating β.

For example, we will consider the "leave-one-out" $RMSE - CV$ error, which calculates residuals in each case leaving that data point out of consideration of the fit

This and other types of CV can be done with `caret` library

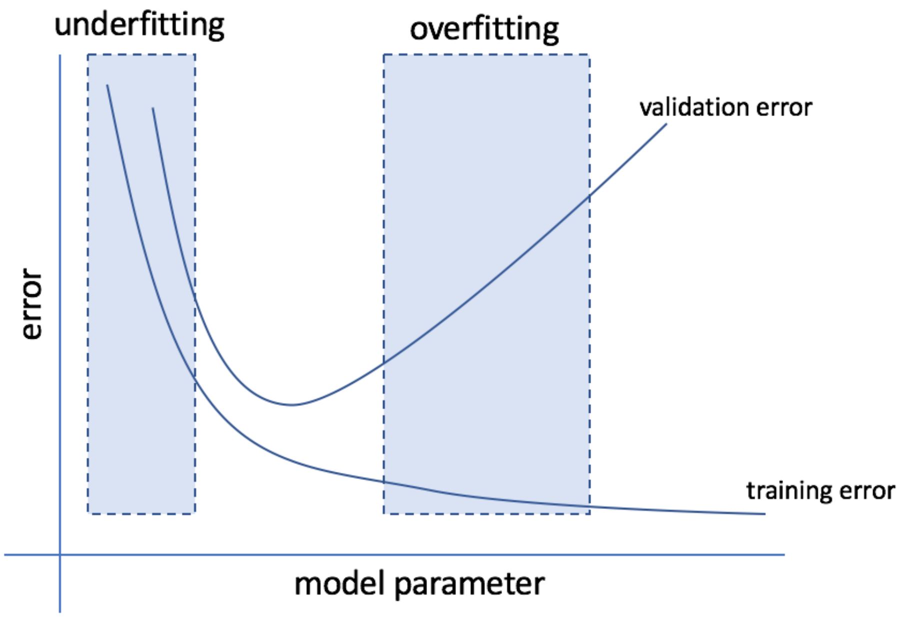
```r
library('caret')
train.control <- trainControl(method = "LOOCV")
train(y ~ poly(x, degree = 2), data = poly_data,
      method = "lm",trControl = train.control)
```

```
## Linear Regression
##
## 11 samples
##   1 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 10, 10, 10, 10, 10, 10, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   23.57189  0.9686724  18.90261
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
train(y ~ poly(x, degree = 8), data = poly_data,
      method = "lm",trControl = train.control)
```

```
## Linear Regression
##
## 11 samples
##  1 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 10, 10, 10, 10, 10, 10, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1334.357  0.4475182  667.524
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

The general picture is



Selection Procedures

data: `seatpos`: physical measurements of people sitting in a car

```r
library(faraway) # contains dataset
model <- lm(hipcenter ~ .,data=seatpos) #hipcenter measured in mm
round(model$coefficients,2)
```

```
## (Intercept)         Age      Weight     HtShoes          Ht      Seated
##      436.43        0.78        0.03       -2.69        0.60        0.53
##         Arm       Thigh         Leg
##       -1.33       -1.14       -6.44
```

The data contains $p = 8 + 1 = 9$ predictors.

Remember, the intercept β_0 counts as a predictor.

---

so the number of possible first order models is

$$\sum_{k=0}^{p-1} \binom{p-1}{k} = 2^{p-1} = 256$$

In a **backwards search** we start with the model `hipcenter ~ .`, which is otherwise known as `hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh + Leg`

then R repeatedly tries to delete until `hipcenter ~ 1`

At each "step", R reports the current model, its AIC, and the possible deletions steps with their RSS

```r
hipcenter_mod = lm(hipcenter ~ ., data = seatpos)
hipcenter_mod_back_aic = step(hipcenter_mod, direction = "backward")
```

```
## Start:  AIC=283.62
## hipcenter ~ Age + Weight + HtShoes + Ht + Seated + Arm + Thigh +
##     Leg
##
##           Df Sum of Sq   RSS    AIC
## - Ht       1      5.01 41267 281.63
## - Weight   1      8.99 41271 281.63
## - Seated   1     28.64 41290 281.65
## - HtShoes  1    108.43 41370 281.72
## - Arm      1    164.97 41427 281.78
## - Thigh    1    262.76 41525 281.87
## <none>                 41262 283.62
## - Age      1   2632.12 43894 283.97
## - Leg      1   2654.85 43917 283.99
##
## Step:  AIC=281.63
## hipcenter ~ Age + Weight + HtShoes + Seated + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Weight   1     11.10 41278 279.64
## - Seated   1     30.52 41297 279.66
## - Arm      1    160.50 41427 279.78
## - Thigh    1    269.08 41536 279.88
## - HtShoes  1    971.84 42239 280.51
## <none>                 41267 281.63
## - Leg      1   2664.65 43931 282.01
## - Age      1   2808.52 44075 282.13
##
## Step:  AIC=279.64
## hipcenter ~ Age + HtShoes + Seated + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Seated   1     35.10 41313 277.67
## - Arm      1    156.47 41434 277.78
## - Thigh    1    285.16 41563 277.90
## - HtShoes  1    975.48 42253 278.53
## <none>                 41278 279.64
## - Leg      1   2661.39 43939 280.01
## - Age      1   3011.86 44290 280.31
##
## Step:  AIC=277.67
## hipcenter ~ Age + HtShoes + Arm + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Arm      1    172.02 41485 275.83
## - Thigh    1    344.61 41658 275.99
## - HtShoes  1   1853.43 43166 277.34
## <none>                 41313 277.67
## - Leg      1   2871.07 44184 278.22
## - Age      1   2976.77 44290 278.31
##
## Step:  AIC=275.83
## hipcenter ~ Age + HtShoes + Thigh + Leg
##
##           Df Sum of Sq   RSS    AIC
## - Thigh    1     472.8 41958 274.26
## <none>                 41485 275.83
## - HtShoes  1    2340.7 43826 275.92
## - Age      1    3501.0 44986 276.91
## - Leg      1    3591.7 45077 276.98
##
## Step:  AIC=274.26
## hipcenter ~ Age + HtShoes + Leg
##
##           Df Sum of Sq   RSS    AIC
## <none>                 41958 274.26
## - Age      1    3108.8 45067 274.98
## - Leg      1    3476.3 45434 275.28
## - HtShoes  1    4218.6 46176 275.90
```

```
##
## Call:
## lm(formula = hipcenter ~ Age + HtShoes + Leg, data = seatpos)
##
## Coefficients:
## (Intercept)          Age      HtShoes          Leg
##    456.2137       0.5998      -2.3023      -6.8297
```

The step keeps going until the AIC no longer goes down. This is the model that we finally end up with from back steps

```
hipcenter_mod_back_aic
```

```
##
## Call:
## lm(formula = hipcenter ~ Age + HtShoes + Leg, data = seatpos)
##
## Coefficients:
## (Intercept)          Age      HtShoes          Leg
##    456.2137       0.5998      -2.3023      -6.8297
```

```
extractAIC(hipcenter_mod_back_aic) # returns p and AIC
```

```
## [1]   4.0000 274.2597
```

The resulting model increases the adjusted R_a^2

```
summary(hipcenter_mod)$adj.r.squared
```

```
## [1] 0.6000855
```

```
summary(hipcenter_mod_back_aic)$adj.r.squared
```

```
## [1] 0.6531427
```

There are also **forward searches**, **exhaustive searches**, and **step searches** all of them are trying to find the **Best model**

You can also include interactions and higher order polynomial terms and then do step searches for best models for example

```
Y ~ .^2 + I(X_1^2) + I(X_2^2) + ... + I(X_p-1^2)
```

What if you have a million features?

- In situations like image recognition you really need a machine learning approach to decide what are the important features
- But This is exactly what an ANN is doing - its figuring out how to combine the features and at what order, automatically!

Exercises unit-8c

data; We will work again with the birthweight data set

FOLLOW ALL INSTRUCTIONS AND EXECUTE OR WRITE THE REQUESTED CODE IN THE RSTUDIO CONSOLE OR SCRIPT

Do the setup:

```
library(tidyverse)
library(MASS)
head(birthwt)
```

In order to understand the predictors type

```
?birthwt
```

We will model birth weight in grams, `bwt`, as a response to all other variables as predictors

let's look at the non-interacting case, type:

```
birthwt_full_model <- lm(bwt ~ ., data = birthwt)
```

call `summary(birthwt_full_model)` and inspect the results.

call `cor(birthwt)` and inspect the results.

Can you find any suggestions of collinearity?

(You could also do `pairs(birthwt)` but this plot is not so instructive)

Do an AIC stepthrough of the full model you just created

```
step(birthwt_full_model,direction = "backward")
```

look at all of the console output as it steps through

Trace through the deletions in the console output.

What predictors are retained in the backwards search model?

What are the adjusted R_a^2 for the full model and the optimal model (based on AIC criteria)?

HINT: you could use

```
summary(birthwt_full_model)$adj.r.squared
summary(step(birthwt_full_model,direction = "backward"))$adj.r.squared
```

or instead you can save the results of the step search

---

let's factor the race variable and name the values so it is more intelligible. Right now the race is encoded numerically (needs to become categorical)

```
birthwt$race =
recode_factor(birthwt$race,`1`="white",`2`="black",`3`="other")
```

Let's look at a simple interaction model

```
birthwt.lm.interact <- lm(bwt ~ race * age, data = birthwt)

summary(birthwt.lm.interact)
```

The interaction allows us to fit the data with lines of different slopes. Next, visualize with

```
 qplot(x = age, y = bwt, color = race, data = birthwt)
 + stat_smooth(method = "lm", se = FALSE, fullrange = TRUE)
```

Exercise 2 Numerical cross-validation check

We will try to fit a simulated model, and find the optimal number of parameters

$$Y = 3 + x_1 + 4x_1^2 + 5x_2 + 2x_2^4 + \epsilon$$

data: enter all codes below to construct the numerical dataset

```
sample_size = 37
x_1 = 1:sample_size
x_2 = c(72,4,1,7,6,3,6,2,2,5,6,2,34,3,34,6,2,4,3,4,63,
  2,5,6,4,56,6,3,5,46,33,4,2,3,5,55,4)
y = 3 + x_1 + 4 * x_1^2 +5*x_2+2*x_2^4 +
  rnorm(n = sample_size, mean = 0, sd = 20)
poly_dat =  data.frame(x_1,x_2, y)
```

We want to see the relation between cross-validation error and the number of parameters in the model.

Calculate the cross-validation error for polynomials from degree 1 to degree 8. Plot the cross-validation error versus the number of parameters.

for example the model below will have **five** parameters (including the intercept)

```
fit_quad = lm(y ~ poly(x_1, degree = 2) + poly(x_2, degree = 2), data = poly_dat)
```

You can use the `caret` library to get the LOOCV error, start with

```
library('caret')
train.control <- trainControl(method = "LOOCV")
train(y ~ poly(x_1, degree = 1)+poly(x_2, degree = 1), data = poly_dat,
      method = "lm",trControl = train.control)
```

---

and you should end with the following

```
train(y ~ poly(x_1, degree = 8)+poly(x_2,degree=8), data = poly_dat,
      method = "lm",trControl = train.control)
```

each time you should record the LOOCV MSE and then finally plot versus number of parameters.

fin