Lab 6: Hands Off my Stack

Paul Nguyen

10/30/2019

```
library(tidyr)
library(infer)
library(data.table)
library(dplyr)
library(ggplot2)
```

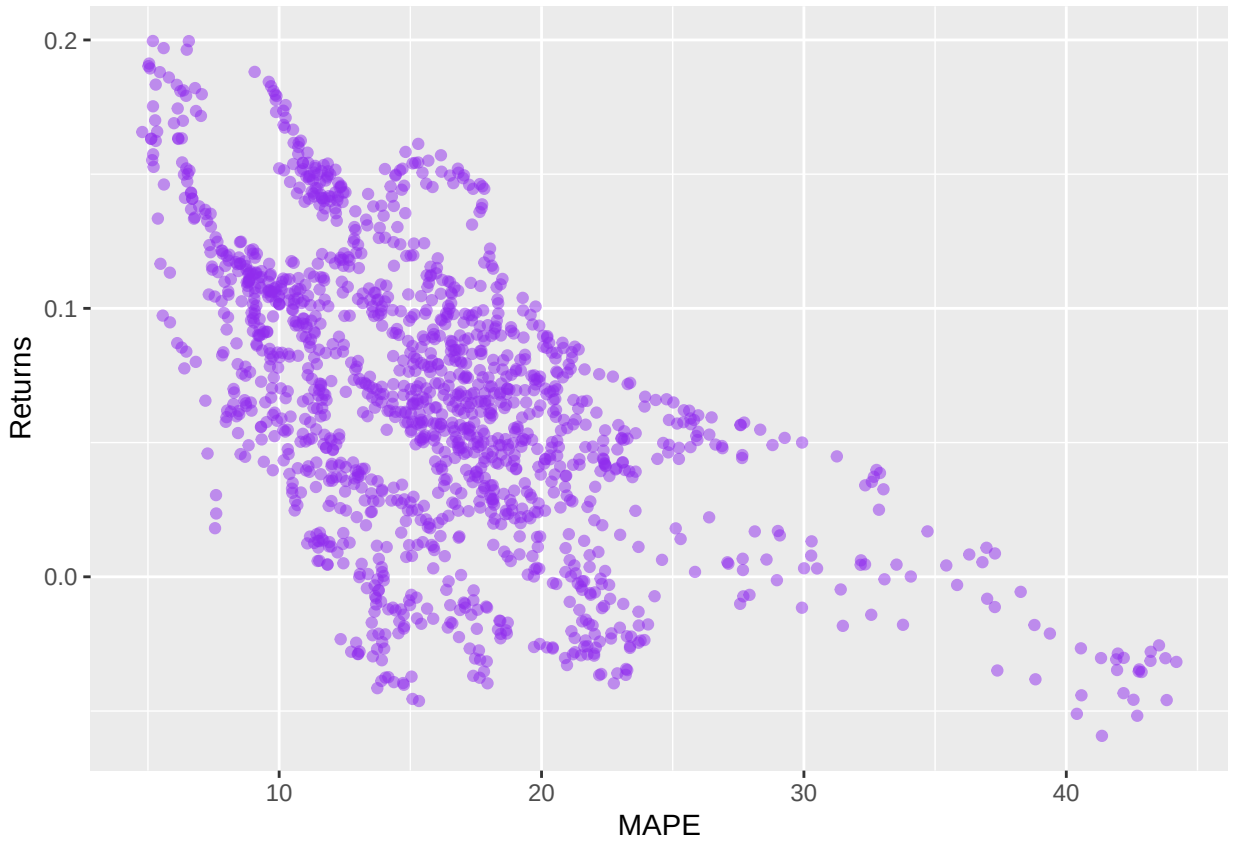
```
d <- read.csv("https://bit.ly/36kibHZ")

d1 <- d %>%
  mutate(MAPE = Price / Earnings_10MA_back)
summary(d1)
```

```
##       Date          Price            Earnings      Earnings_10MA_back
##  Min.   :1871   Min.   :  64.76   Min.   :  4.01   Min.   : 8.51
##  1st Qu.:1907   1st Qu.: 161.84   1st Qu.: 11.94   1st Qu.:13.89
##  Median :1943   Median : 237.53   Median : 18.89   Median :17.48
##  Mean   :1943   Mean   : 442.82   Mean   : 27.02   Mean   :25.70
##  3rd Qu.:1979   3rd Qu.: 553.62   3rd Qu.: 36.59   3rd Qu.:37.19
##  Max.   :2015   Max.   :2056.51   Max.   :103.17   Max.   :77.00
##                                                    NA's   :120
##   Return_cumul      Return_10_fwd           MAPE
##  Min.   :    0.99   Min.   :-0.05925   Min.   : 4.785
##  1st Qu.:   15.13   1st Qu.: 0.03400   1st Qu.:11.708
##  Median :   95.89   Median : 0.06827   Median :15.947
##  Mean   : 1456.57   Mean   : 0.06788   Mean   :16.554
##  3rd Qu.: 1060.67   3rd Qu.: 0.10481   3rd Qu.:19.959
##  Max.   :12950.92   Max.   : 0.19960   Max.   :44.196
##                     NA's   :120        NA's   :120
```

```
d1 <- d1 %>%
  na.omit()

ggplot(data = d1, mapping = aes(x = MAPE, y = Return_10_fwd)) +
  geom_point(color = "purple2", alpha = .5) +
  ylab("Returns")
```




```
linearmodel <- lm(Return_10_fwd ~ MAPE, data = d1)
summary(linearmodel)
```

```
##
## Call:
## lm(formula = Return_10_fwd ~ MAPE, data = d1)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.116777 -0.029650  0.004347  0.028478  0.093157
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1383475  0.0029889   46.29   <2e-16 ***
## MAPE        -0.0045885  0.0001727  -26.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04321 on 1482 degrees of freedom
## Multiple R-squared:  0.3226, Adjusted R-squared:  0.3221
## F-statistic: 705.8 on 1 and 1482 DF,  p-value: < 2.2e-16
```

```
pred1 <- predict(linearmodel, data = d1) %>%
  tibble() %>%
  mutate(MAPE = d1$MAPE)
```

There 120 na's because Earnings_10MA_back refers to the 10 year average of earnings looking backwards from the current date. The first 120 rows that the data tracks were from the 10 initial years where they could

not have gone 10 years back. (12 months for those 10 years)

My Linear Model's intercept is .138 and it's coefficient for MAPE is -.0046; both are significant. The coefficient's std. error is .0001727.

The MSE for my model is .00188

```
set.seed(408)
k <- 5

partitionindex <- rep(1:k, each = nrow(d1)/k) %>%
  sample()

partitionindex <- partitionindex %>%
  append(c(1, 2, 4, 3))

d1 <- d1 %>%
  mutate(fold = partitionindex)
navector <- rep(NA, k)

for(i in 1:k) {
  test <- d1 %>%
    filter(fold == i)
  train <- d1 %>%
    filter(fold != i)
  lm <- lm(Return_10_fwd ~ MAPE, data = train)
  prediction <- predict(lm, newdata = test)
  mse <- sum((test$Return_10_fwd - prediction)^2) / nrow(test)
  navector[i] <- mse
}

mse <- sum(navector) / k
mse
```

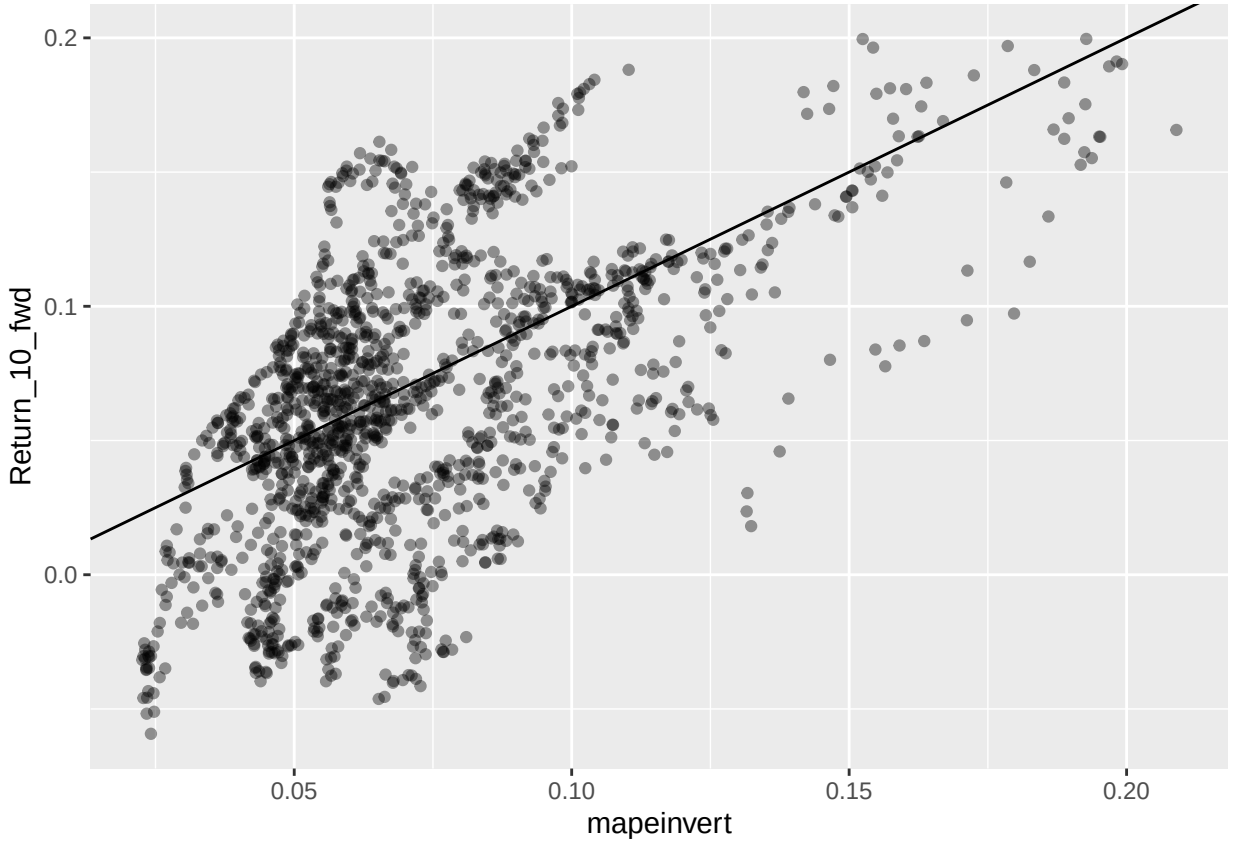
```
## [1] 0.00188007
```

```
#inverting the variable
d1 <- d1 %>%
  mutate(mapeinvert = (1/MAPE))
invertlm <- lm(Return_10_fwd ~ mapeinvert, data = d1)
summary(invertlm)
```

```
##
## Call:
## lm(formula = Return_10_fwd ~ mapeinvert, data = d1)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.106298 -0.030839  0.002955  0.028179  0.103866
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.007659   0.002878  -2.661  0.00788 **
## mapeinvert   0.995904   0.036513  27.275  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.04284 on 1482 degrees of freedom
## Multiple R-squared:  0.3342, Adjusted R-squared:  0.3338
## F-statistic: 743.9 on 1 and 1482 DF,  p-value: < 2.2e-16
```

```
ggplot(data = d1, mapping = aes(x = mapeinvert, y = Return_10_fwd)) +
  geom_point(alpha = .4) +
  geom_abline(intercept = 0, slope = 1)
```




```
invertnavector <- rep(NA, k)
set.seed(408)
for(i in 1:k) {
  test <- d1 %>%
    filter(fold == i)
  train <- d1 %>%
    filter(fold != i)
  invertlm <- lm(Return_10_fwd ~ mapeinvert, data = train)
  prediction <- predict(invertlm, newdata = test)
  invertmse <- sum((test$Return_10_fwd - prediction)^2) / nrow(test)
  invertnavector[i] <- invertmse
}

invertmse <- sum(invertnavector) / k
invertmse
```

```
## [1] 0.001843889
```

```
pred2 <- predict(invertlm, newdata = d1) %>%
  tibble() %>%
  mutate(MAPE = d1$MAPE)
```

The coefficent for my inverted MAPE model is .9959, with an intercept of -.0077. They are both significant, with standard errors of .0365 and .00288.

The CV MSE of this inverted MAPE model is .00184, which is just slightly less than the model with the normal MAPE variable.

```
#simple model
simplemodelpred <- train$mapeinvert
#simpledata <- data.frame(d1$mapeinvert, d1$Return_10_fwd) %>%
 # setnames(old=c("d1.mapeinvert","d1.Return_10_fwd"),
  #         new=c("mapeinvert", "Return_10_fwd"))


#sm <- lm(Return_10_fwd ~ simplemodel, data = train)
#smprediction <- predict(sm, newdata = d1)

trainingmse <- sum((train$Return_10_fwd - simplemodelpred)^2)/ nrow(train)
trainingmse
```

```
## [1] 0.001904254
```

```
#tmse <- sum((train$Return_10_fwd - smprediction)^2)/ nrow(train)
#tmse
```

The training mse for my simple model is .001922. The training MSE is equivalent to the test MSE in this scenario because this model does not use a specific training or test set for its predictions. It simply takes whatever observation you give it and returns the inverted MAPE variable. The training and test set are randomly scrambled, so there shouldn't be too big of a difference in 1/MAPE's and the corresponding MSE's.

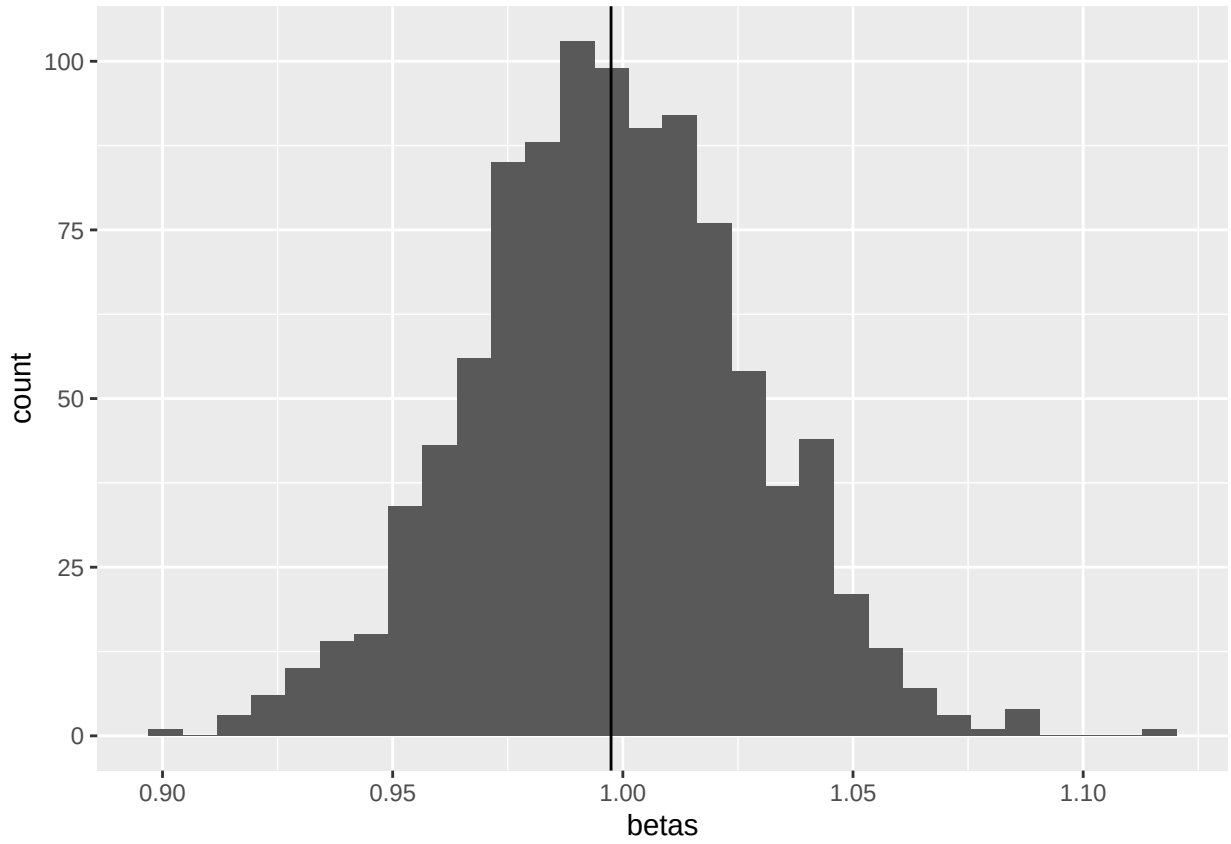
```
boot_ind <- sample(1:nrow(d1),
                   size = nrow(d1),
                   replace = TRUE)
d1_boot <- d1[boot_ind,]

betas <- rep(NA, 1000)

for(i in 1:1000){
  boot_ind <- sample(1:nrow(d1),
                   size = nrow(d1),
                   replace = TRUE)
  d1_boot <- d1[boot_ind,]
  betas[i] <- coef(lm(Return_10_fwd ~ mapeinvert, data = d1_boot))[2]
}

ggplot(mapping = aes(x = betas)) +
  geom_histogram() +
  geom_vline(xintercept = mean(betas))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
sd(betas)
```

```
## [1] 0.02988939
```

```
stat <- coef(lm(Return_10_fwd ~ mapeinvert, data = d1))[2]
lb <- stat - 1.96*sd(betas)
ub <- stat + 1.96*sd(betas)
ci <- c(lb, ub)
ci
```

```
## mapeinvert mapeinvert
##  0.9373204  1.0544868
```

```
confint(invertlm, level = .95)
```

```
##                   2.5 %       97.5 %
## (Intercept) -0.0150236 -0.002393385
## mapeinvert   0.9424821  1.102938190
```

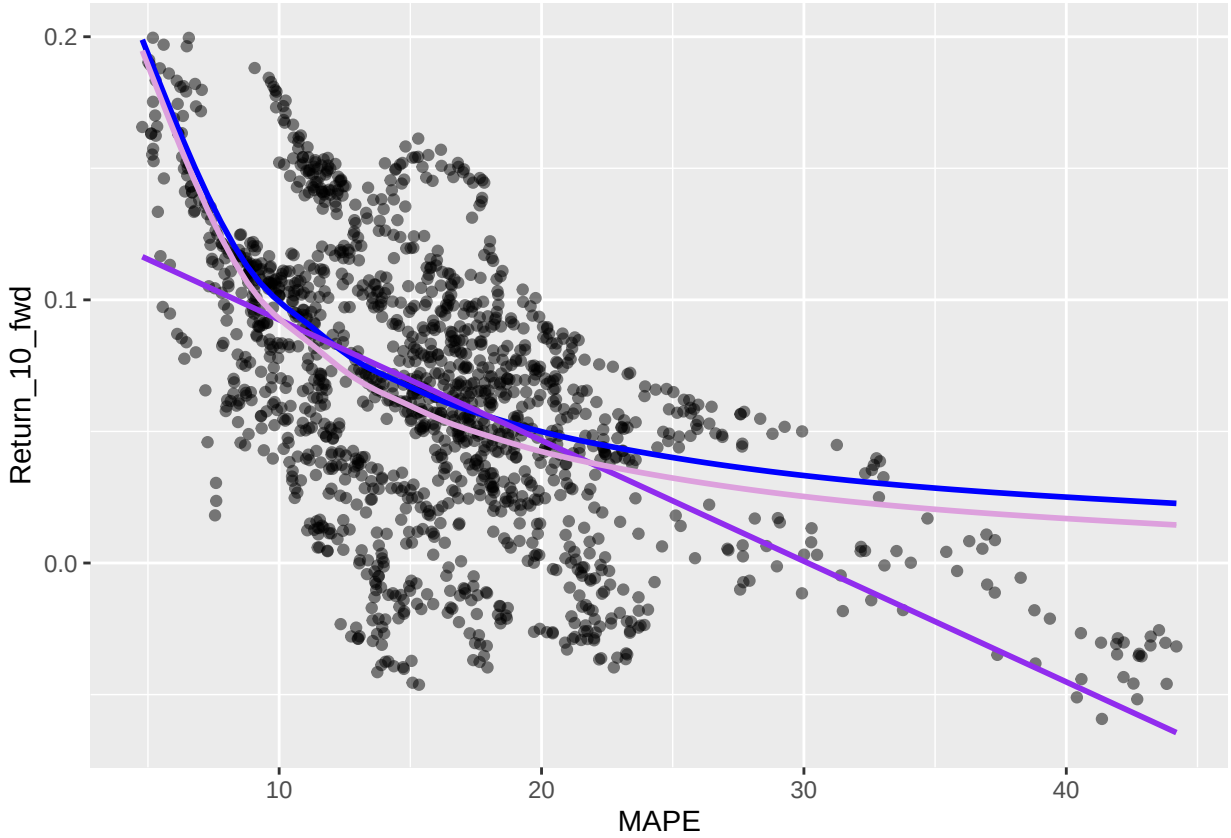
my approximate 95% bootstrap confidence interval for the slope is (.937, 1.055). My interval returned by running the confint command on my model was (.942, 1.10). The approximate ci is pretty similar to the ci given by the confint function. The ci's are the same length, but have a difference in ends of .005, which is pretty small and can be explained by sampling variation of our data.

```
#ggplot(data = d1, mapping = aes(x = MAPE,  y = Return_10_fwd)) +
 # geom_point(alpha = .5, color = "black") +
  #geom_abline(intercept = (coef(lm))[1], slope = (coef(lm))[2], color = "tomato") +
  #geom_abline(intercept = (coef(invertlm)[1]), slope = (coef(invertlm))[2],
   #           color = "purple2") +
  #ylim(0,5)
```

```
  #how to plot simple and model 2 on this graph?

ggplot(data = d1, mapping = aes(x = MAPE,  y = Return_10_fwd)) +
  geom_point(alpha = .5, color = "black") +
  geom_smooth(data = d1, aes(x = MAPE, y = I(1/MAPE)), color = "blue") +
  geom_smooth(data = pred1, mapping = aes(x = MAPE, y = .), color = "purple2") +
  geom_smooth(data = pred2, mapping = aes(x = MAPE, y = .), color = "plum")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```




```
#ggplot(data = d1, mapping = aes(x = mapeinvert, y = Return_10_fwd)) +
 # geom_point(alpha = .5) +
  #geom_abline(intercept = (coef(invertlm)[1]), slope = (coef(invertlm))[2],
   #           color = "purple2") +
#  geom_smooth(data = d1)
```

blue line = simple model plum line = inverted MAPE model purple line = MAPE model

Using CV MSE, the model I would select to make predictions of returns would be the inverted MAPE model, as it has the lowest CV MSE of .00184 (MAPE = .00188, simple = .019). Looking at the plot, I would say this is a decent model for prediction. Its strength is including the curve that the data takes on.. the true model definitely does not seem to be exactly linear. For its weakness, I would say that it tends to over predict towards the right side of the data, where MAPE was higher.

The Confidence Interval that I came up with through my bootstrapping procedure for the slope of the linear model using 1/MAPE as a predictor was from .94 to 1.10. The simple model predicts that the returns is exactly equal to the inverted MAPE value, so the form $y = b + m(x)$ turns to returns = 1*invertmape. The

slope for this model is 1, which is in between my 95% confidence interval. Thus, the simple minded model is a plausible model given our data.

Problem Set

4) how might we estimate the standard deviation of our prediction? Our first step would be to select n observations from the data set *with replacement* to form one bootstrap data set. We then predict our value for y. Doing this many times will give us many bootstrap datasets and multiple predictions. We can then compute the standard error of our bootstrap estimates with the formula: (using α as our predicttion)

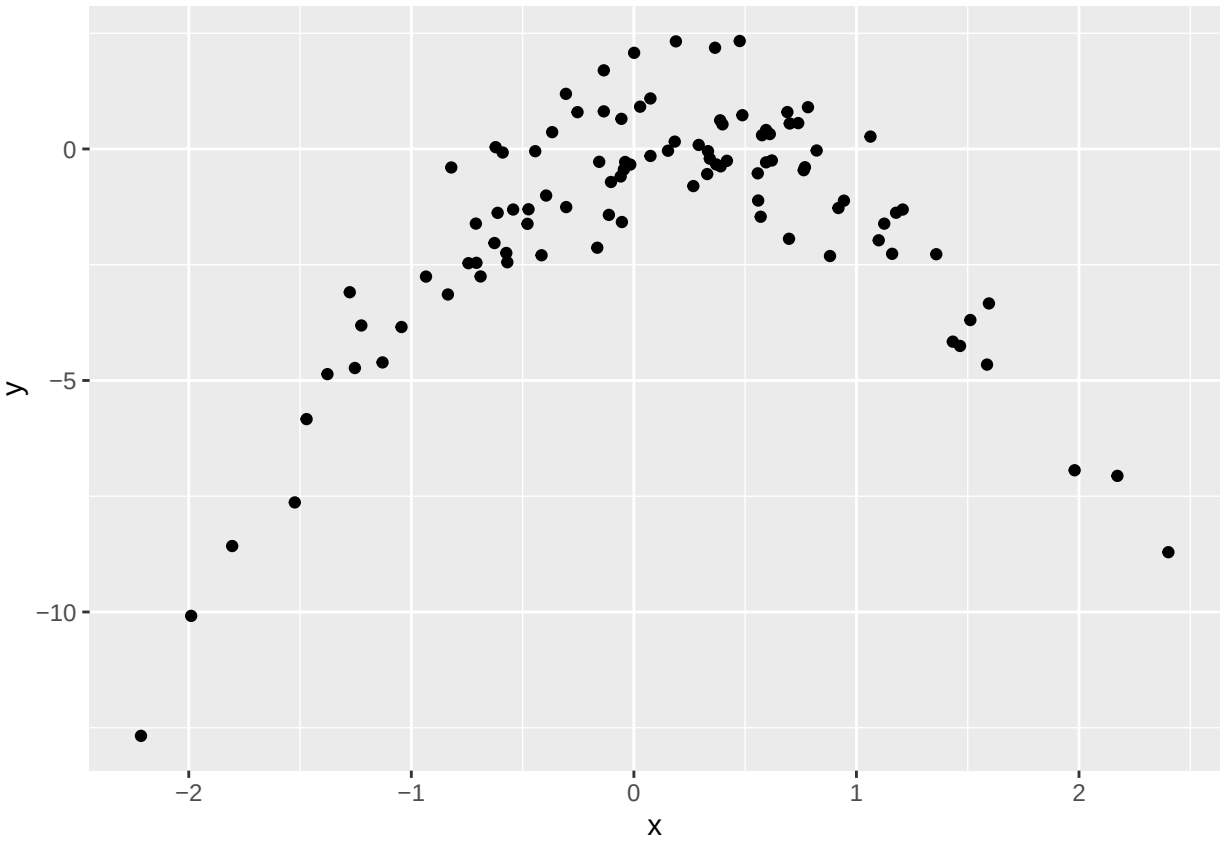
$$SE_B(\hat{\alpha} = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'})^2}$$

This should equal the sum of each bootstrap estimate minus the average bootstrap estimate squared, divided by the number of boostrap samples minus 1, and then taking the square root. This is an estimate of the standard error of $\hat{\alpha}$ estimated from orignal dataset.

8)

```
set.seed(1)
x = rnorm(100)
y = x-2*x^2 +rnorm(100)

q8 <- tibble(x = x, y = y)
ggplot(data = q8, mapping = aes(x = x, y = y)) +
  geom_point()
```



In this dataset, we have n (100) observations and p (1) predictor. Looking at the graph, I would say that the data seems parabolic, with x centered just a bit above 0.

```
vector <- rep(NA, 100)

ff15 <- function(model){
for(i in 1:100){
  partition <- (1:100)
  q8 %>%
    mutate(fold = partition)
  test <- q8 %>%
    filter(partition == i)
  train <- q8 %>%
    filter(partition != i)
  prediction <- predict(model, newdata = test)
  MSEtest <- (test$y - prediction)^2
  vector[i] <- MSEtest
  CV <- sum(vector)/100
}
  CV
}

set.seed(408)
x = rnorm(100)
y = x-2*x^2 +rnorm(100)
q8 <- tibble(x = x, y = y)

model1 <- lm(y ~ x, data = train)
```

```
ff15(model1)
```

```
## [1] 6.195823
```

```
model2 <- lm(y ~ x + I(x^2), data = train)
ff15(model2)
```

```
## [1] 0.9450712
```

```
model3 <- lm(y ~ x + I(x^2) + I(x^3), data = train)
ff15(model3)
```

```
## [1] 0.9400236
```

```
model4 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = train)
ff15(model4)
```

```
## [1] 0.9144541
```

```
set.seed(323)
x = rnorm(100)
y = x-2*x^2 +rnorm(100)
q8 <- tibble(x = x, y = y)
ff15(model1)
```

```
## [1] 12.96238
```

```
ff15(model2)
```

```
## [1] 1.263181
```

```
ff15(model3)
```

```
## [1] 1.3556
```

```
ff15(model4)
```

```
## [1] 1.249137
```

My results are not the same as I got in (c). This is because the data has changed, so the coefficients for my models are slightly different each time. This is why the MSE's are different, due to sampling variability.

For my models in (c), the model with the lowest LOOCV error was model two, with just the linear and quadratic term. This isn't too surprising to me, as this is the model that emulated the true model the closest. The other 2 models with the cubic and fourth term were too flexible, and did not estimate the missing response very well. The first model was way too simple, and was too biased to make an accurate prediction for y.

```
set.seed(408)
x = rnorm(100)
y = x-2*x^2 +rnorm(100)
q8 <- tibble(x = x, y = y)
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.0246  -0.4894   0.5470   1.4658   4.3370
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8186     0.2520  -7.216 1.14e-10 ***
## x             1.9061     0.2552   7.469 3.38e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.514 on 98 degrees of freedom
## Multiple R-squared:  0.3628, Adjusted R-squared:  0.3563
## F-statistic: 55.79 on 1 and 98 DF,  p-value: 3.379e-11
```

```
summary(model2)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2), data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.62421 -0.53061 -0.03273  0.61092  2.62298
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.07985    0.12408  -0.644    0.521
## x            1.08483    0.10624  10.211   <2e-16 ***
## I(x^2)      -1.83867    0.07920 -23.215   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9871 on 97 degrees of freedom
## Multiple R-squared:  0.9028, Adjusted R-squared:  0.9008
## F-statistic: 450.5 on 2 and 97 DF,  p-value: < 2.2e-16
```

```
summary(model3)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3), data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.57854 -0.54447 -0.05922  0.58035  2.67991
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.09456    0.12607  -0.750    0.455
## x            0.98311    0.17724   5.547 2.57e-07 ***
## I(x^2)      -1.80910    0.08944 -20.226  < 2e-16 ***
## I(x^3)       0.04187    0.05831   0.718    0.475
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9895 on 96 degrees of freedom
## Multiple R-squared:  0.9033, Adjusted R-squared:  0.9003
```

```
## F-statistic:   299 on 3 and 96 DF,  p-value: < 2.2e-16
```

```
summary(model4)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4), data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.71170 -0.50474 -0.07794  0.56483  2.63440
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.22819    0.14949  -1.526    0.130
## x            1.04210    0.17942   5.808 8.36e-08 ***
## I(x^2)      -1.48440    0.21807  -6.807 8.93e-10 ***
## I(x^3)       0.00128    0.06295   0.020    0.984
## I(x^4)      -0.07389    0.04534  -1.630    0.106
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9811 on 95 degrees of freedom
## Multiple R-squared:  0.9059, Adjusted R-squared:  0.902
## F-statistic: 228.8 on 4 and 95 DF,  p-value: < 2.2e-16
```

In model 1, we see that the linear term is statistically significant. In model 2, we see that both the linear and quadratic term are statistically significant. In model 3, again, the linear and quadratic term are significant, but the cubic term has a p value of .758, rendering it unnecessary. In model 4, again, the linear and quadratic term are significant, but this time, the cubic term has a p value of .921 and the term to the fourth has one of .194. Both here are not statistically significant. These results do agree with my cross validation conclusions, that the second model would be the best one in predicting y. We can see that models 3 and 4 include unnecessary terms, increasing its flexibility/variance too much.