521 HW 3

William Tirone

Q1

1.1

> **TRUE**. It's easier to shrink the small ones, this is also seen on p.37 of lecture 9.

1.2

> **FALSE.** We don't necessarily know what will happen to test error. (?)

1.3

> **FALSE.** We can specify a lack of knowledge with a non-informative prior.

1.4

> **FALSE**. Bayesian intervals can be used to make probabilistic statements and confidence intervals cannot, only under repeated experiments.

1.5

> **FALSE**. Bias will increase as lambda increases to reduce variance.

1.6

> **FALSE**. At some values of lambda, can have negative coefficients. Also see Lecture 10 page 11.

1.7

TRUE. If there are two collinear variables, for example, LASSO may return different solutions.

1.8

FALSE. We should scale if predictors are not on the same scale and center if we don't have an intercept term.

Q2

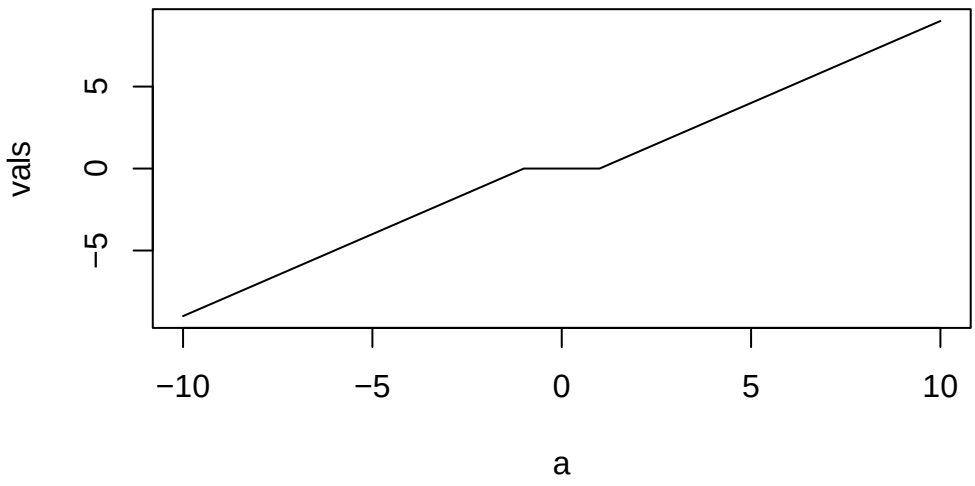
2.2

$$soft(a, 1) = sign(a)(|a| - 1)_+$$

The soft-thresholding function is non decreasing on the whole domain.

```
a = -10:10
vals = sign(a) * (abs(a) - 1)

plot(a,vals,type='l')
```



Q3

```
Registered S3 method overwritten by 'GGally':
```

```
method from
+.gg   ggplot2
```

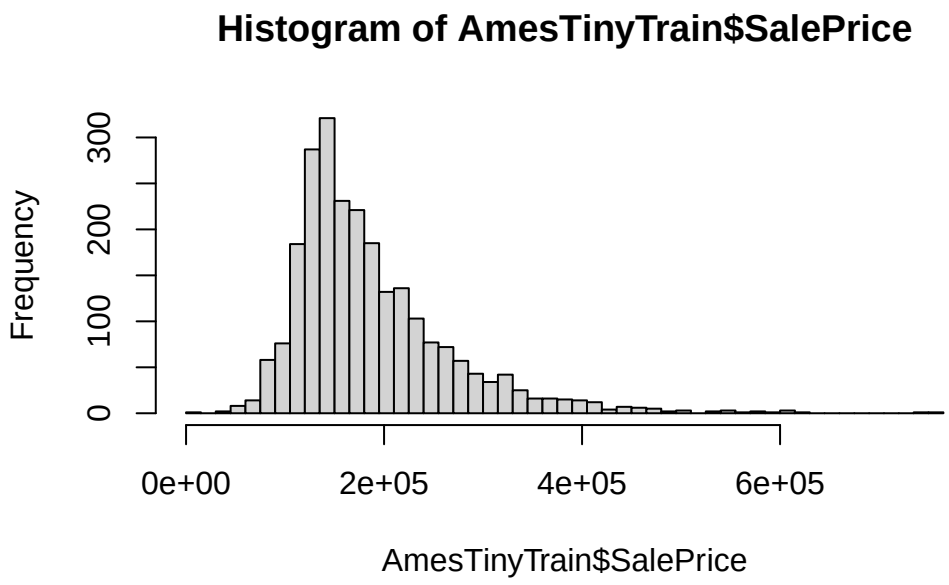
Stock project code to get started and create AmesTiny

3.1

3.1.1

Since the data is skewed, we might want a log transform to make it more unimodal.

```
hist(AmesTinyTrain$SalePrice, breaks= seq(0, 770000, 15000))
```



3.1.2

Checking for NAs. Both checks return FALSE, so no NAs present.

```
any(is.na(AmesTinyTrain))
```

```
[1] FALSE
```

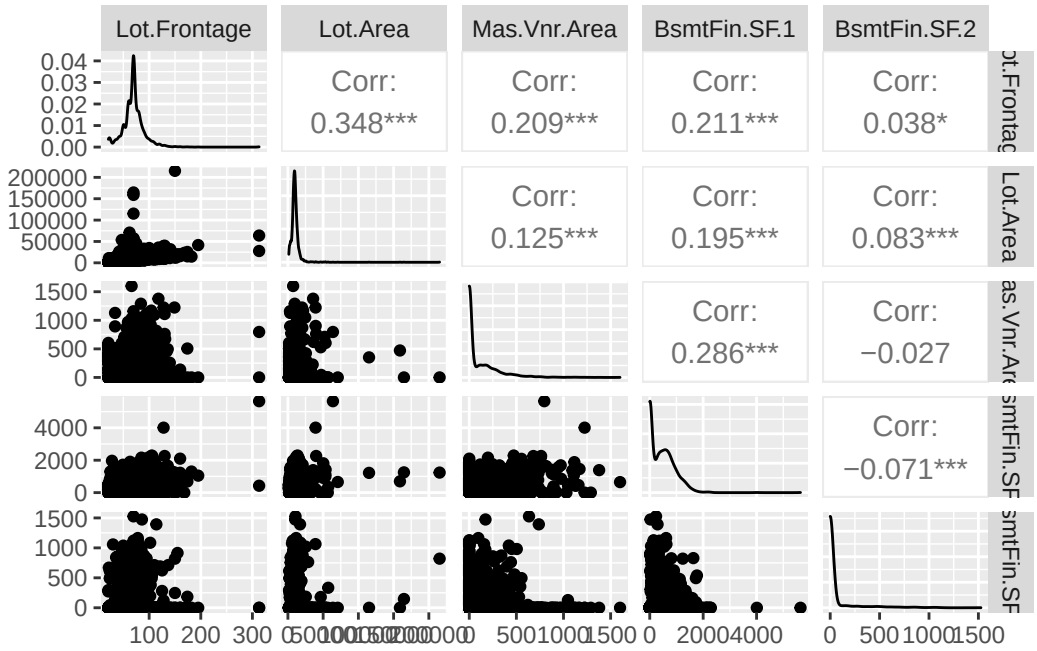
```
any(is.na(AmesTinyTest))
```

```
[1] FALSE
```

3.1.3

Does not look like there is collinearity.

```
v = continuousVar[1:5]
ggpairs(AmesTiny[,v], progress = FALSE)
```



3.2

Fitting the lm

```
models = c()

for (i in 1:11) {
    fit <- lm(reformulate(Xnames[1:Xname_stops[i]],
                response='log(SalePrice + 1)'), data = AmesTinyTrain)
    models = c(models, list(fit))
}
```

3.2.1

Function for MSE

```
MSE = function(y,X,B) {
  n = dim(X)[1]
  (1/n) * (norm(y- X %*% B,type = "2")^2)
}
```

3.2.2

Function for R2

```
R2 = function(y,X,B) {
  cor(y,X %*% B)^2
}
```

3.2.3

```
mse_train = c()
R2_train = c()
mse_RImp = c()
R2_RImp = c()

for (i in seq(1,11)){

  y.i = log(AmesTinyTrain$SalePrice + 1)
  X.i = model.matrix(models[[i]])
  B.i = matrix(models[[i]]$coefficients)

  mse_i = MSE(y = y.i,
              X = X.i,
              B = B.i)

  r2_i = R2(y = y.i,
            X = X.i,
            B = B.i)

  mse_train = c(mse_train, mse_i)
  R2_train = c(R2_train, r2_i)
```

```
  mse_RImp = c(mse_RImp, mean(models[[i]]$residuals^2))
  R2_RImp = c(R2_RImp, summary(models[[i]])$r.squared)

}

#my implementation of MSE and R Squared
modelQuality = data.frame(mse_train, R2_train)

# R implementation of MSE and R Squared
modelQualityRImp = data.frame(mse_RImp, R2_RImp)

print(modelQuality)
```

```
   mse_train  R2_train
1  0.05771607 0.6147249
2  0.04895374 0.6732166
3  0.04815538 0.6785459
4  0.02333795 0.8442110
5  0.02114386 0.8588573
6  0.01825005 0.8781745
7  0.01605491 0.8928278
8  0.01531166 0.8977893
9  0.01480165 0.9011938
10 0.01478316 0.9013172
11 0.01475757 0.9014880
```

```
print(modelQualityRImp)
```

```
    mse_RImp   R2_RImp
1  0.05771607 0.6147249
2  0.04895374 0.6732166
3  0.04815538 0.6785459
4  0.02333795 0.8442110
5  0.02114386 0.8588573
6  0.01825005 0.8781745
7  0.01605491 0.8928278
8  0.01531166 0.8977893
9  0.01480165 0.9011938
10 0.01478316 0.9013172
11 0.01475757 0.9014880
```

```
dim(modelQuality)
```

```
[1] 11  2
```