STAT 6340 Mini Project 5

Matthew Lynn

April 10, 2019

Section 1

Section 2 is coded in Section 1

1(a)

We look at the Caravan dataset to determine how different the units each variable is. No suprise, we have vastly different scales for our dataset. We want to standardize the data such that each variable is fairly compared with all other variables. We use Purchase as our response variable due to being qualitative.

```
library(ISLR)
caravan = Caravan
s.caravan = scale(caravan[, -86]) #86 column is purchase
c(var(caravan[,1]),var(caravan[,2]),var(s.caravan[,1]),var(s.caravan[,2]))
```

```
## [1] 165.0378474   0.1647078   1.0000000   1.0000000
```

```
x = model.matrix(Purchase~., caravan)[,-1]
y = caravan$Purchase
```

1(b)

Here, we simply split the data into test and training. Our first 1000 obs will be our test set while the rest are training.

```
test = 1:1000
train = (-test)
```

1(c)

We fit a logistic regression model with no special features except instead of a 0.5 cutoff for the probability a person will make the purchase, we use a 0.2. Additionally, we compute the confusion matrix, sensitivity, specificity, and
overall misclassification rate based on the test data.

```
library(caret)
set.seed(1)
tc <- trainControl()
model <- train(x[train,], y[train],
               method="glm", family="binomial", trControl = tc)
model
```

```
## Generalized Linear Model
##
## 4822 samples
##   85 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4822, 4822, 4822, 4822, 4822, 4822, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9359506  0.01654115
```

```
model = glm(Purchase~., data = caravan, family = binomial, subset = train)
prob = predict(model, caravan[test,], type = "response")
pred <- ifelse(prob >= 0.2, "Yes", "No")
confusionMatrix(table(pred, y[test]))
```

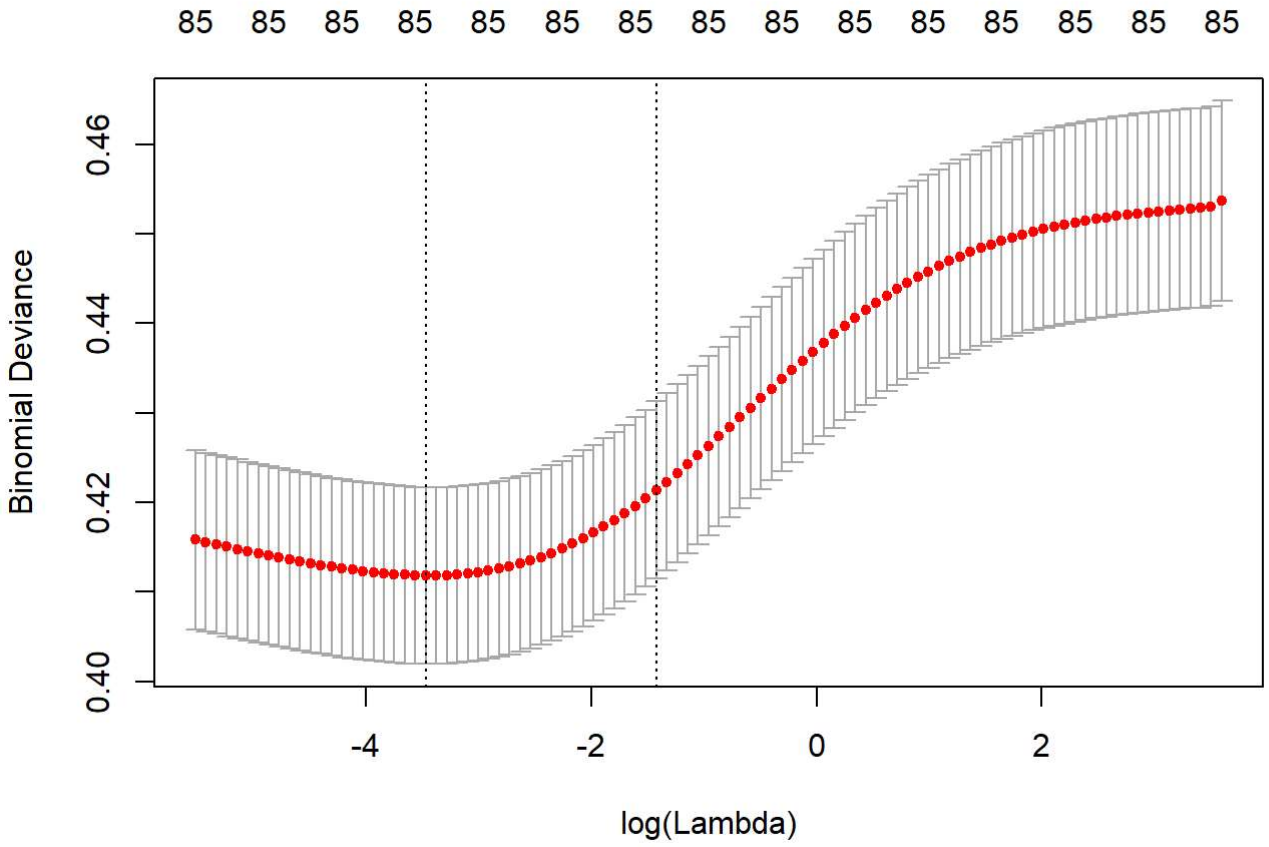
```
## Confusion Matrix and Statistics
##
##
## pred   No Yes
##   No  897  44
##   Yes  44  15
##
##                Accuracy : 0.912
##                  95% CI : (0.8927, 0.9288)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.9999
##
##                   Kappa : 0.2075
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.9532
##             Specificity : 0.2542
##          Pos Pred Value : 0.9532
##          Neg Pred Value : 0.2542
##              Prevalence : 0.9410
##          Detection Rate : 0.8970
##    Detection Prevalence : 0.9410
##       Balanced Accuracy : 0.6037
##
##        'Positive' Class : No
##
```

```
LR.ERR = mean(pred != y[test]) # Overall misclassification rate
```

1(d)

We repeat the procedure from 1(c) but use a penalty function instead, ridge-regression. Our best lambda is rather small, perhaps a normal logistic regression function is not so bad in comparison.
Note: cv.glmnet defaults to 10-fold cross validation.
The graph below shows level of variation per lambda as well as how many coefficients denoted along the top axis. The dotted lines show our best lambdas.

```
set.seed(1)
library(glmnet)
lambdas <- 10^seq(10, -2, length = 100)
ridge.model <- glmnet(x[train,], y[train], alpha = 0, lambda = lambdas,
                      family = "binomial")
ridge.cv <- cv.glmnet(x[train,], y[train], alpha = 0, family = "binomial")
plot(ridge.cv)
```




```
# Best cross-validated lambda
lambda.cv = ridge.cv$lambda.min
```

Lambda is 0.0310951, which is quite small. We may as well use least squares.

```
prob = predict(ridge.model, s = lambda.cv, newx = x[test,], type = "response")
pred <- ifelse(prob >= 0.2, "Yes", "No")
confusionMatrix(table(pred, y[test]))
```

```
## Confusion Matrix and Statistics
##
##
## pred   No Yes
##   No  918  50
##   Yes  23   9
##
##                Accuracy : 0.927
##                  95% CI : (0.9091, 0.9423)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.971093
##
##                   Kappa : 0.1631
##  Mcnemar's Test P-Value : 0.002342
##
##             Sensitivity : 0.9756
##             Specificity : 0.1525
##          Pos Pred Value : 0.9483
##          Neg Pred Value : 0.2813
##              Prevalence : 0.9410
##          Detection Rate : 0.9180
##    Detection Prevalence : 0.9680
##       Balanced Accuracy : 0.5641
##
##        'Positive' Class : No
##
```

```
# the predict function below specifies the best model by using optimal lambda
coef = predict(ridge.model, x[test,], type = "coefficients", s = lambda.cv)
length(coef[coef != 0]) # ridge uses all variables
```

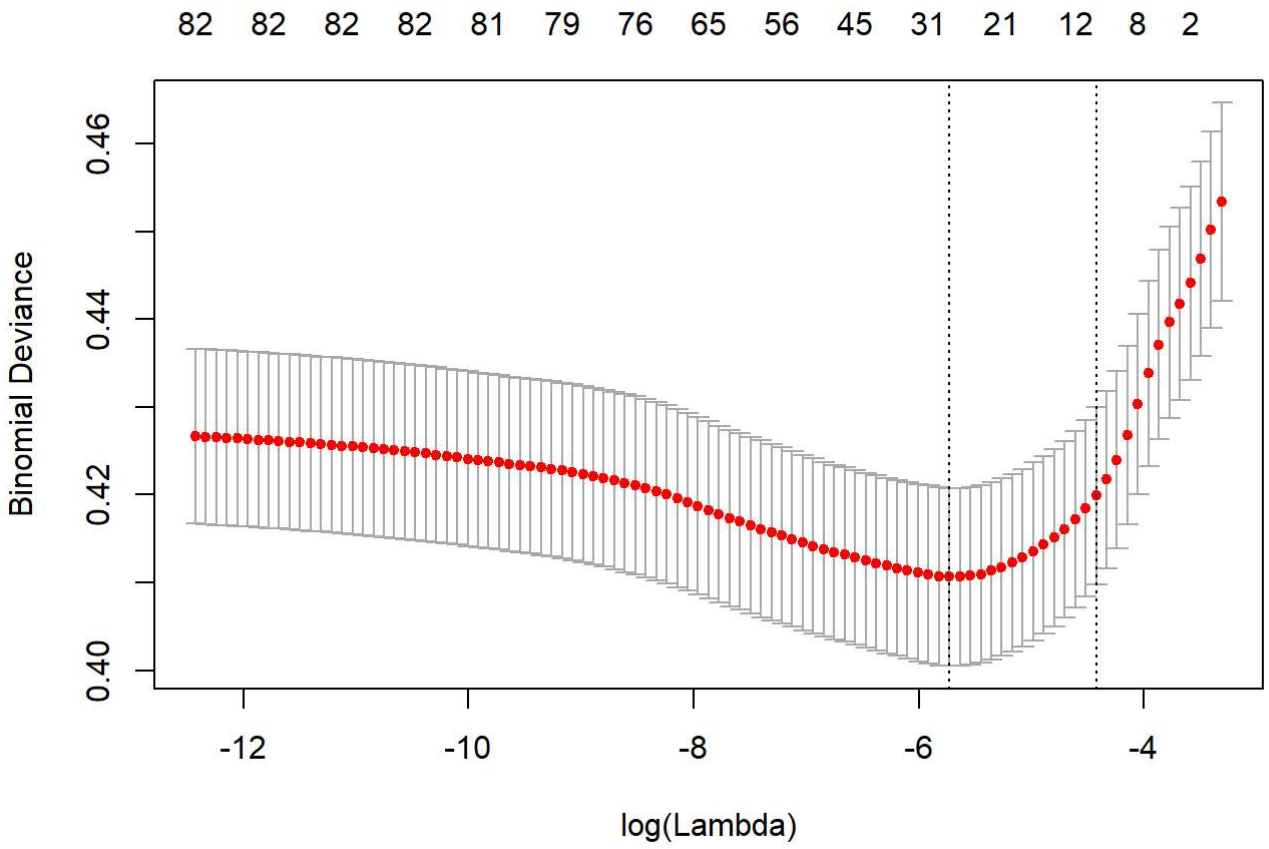
```
## [1] 86
```

```
RIDGE.ERR = mean(pred != y[test]) # Overall misclassification rate
```

1(e)

We repeat the procedure from 1(c) but use a penalty function instead,
lasso-regression. Our graph looks rather different, and has decreasing coefficients.

```
set.seed(1)
lasso.model <- glmnet(x[train,], y[train], alpha = 1, lambda = lambdas,
                      family = "binomial")
lasso.cv <- cv.glmnet(x[train,], y[train], alpha = 1, family = "binomial")
plot(lasso.cv)
```




```
# Best cross-validated lambda
lambda.cv = lasso.cv$lambda.min
```

Again, lambda is small, 0.0032576, we may as well use least squares.
But, we will see that Lasso greatly reduces our complexity.

```
prob = predict(lasso.model, s = lambda.cv, newx = x[test,], type = "response")
pred <- ifelse(prob >= 0.2, "Yes", "No")
confusionMatrix(table(pred, y[test]))
```

```
## Confusion Matrix and Statistics
##
##
## pred   No Yes
##   No  937  57
##   Yes   4   2
##
##                Accuracy : 0.939
##                  95% CI : (0.9223, 0.953)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.6379
##
##                   Kappa : 0.0512
##  Mcnemar's Test P-Value : 2.777e-11
##
##             Sensitivity : 0.9957
##             Specificity : 0.0339
##          Pos Pred Value : 0.9427
##          Neg Pred Value : 0.3333
##              Prevalence : 0.9410
##          Detection Rate : 0.9370
##    Detection Prevalence : 0.9940
##       Balanced Accuracy : 0.5148
##
##        'Positive' Class : No
##
```

```
coef = predict(lasso.model, x[test,], type = "coefficients", s = lambda.cv)
length(coef[coef != 0]) # lasso only uses 12 variables
```

```
## [1] 12
```

```
LASSO.ERR = mean(pred != y[test]) # Overall misclassification rate
```

1(f)

Lasso and normal Logistic Regression share the same overall misclassification.
Ridge uses all variables and did not do any better than LR.
Lasso managed to only use 12 variables and achieve the same results except less specificity. We would proceed with Lasso due to using a significantly smaller model while maintaining high model accuracy (lowest error rate in fact).

```
round(cbind(LR.ERR,RIDGE.ERR, LASSO.ERR), 3) # Test error rates
```

```
##      LR.ERR RIDGE.ERR LASSO.ERR
## [1,]  0.088     0.073     0.061
```

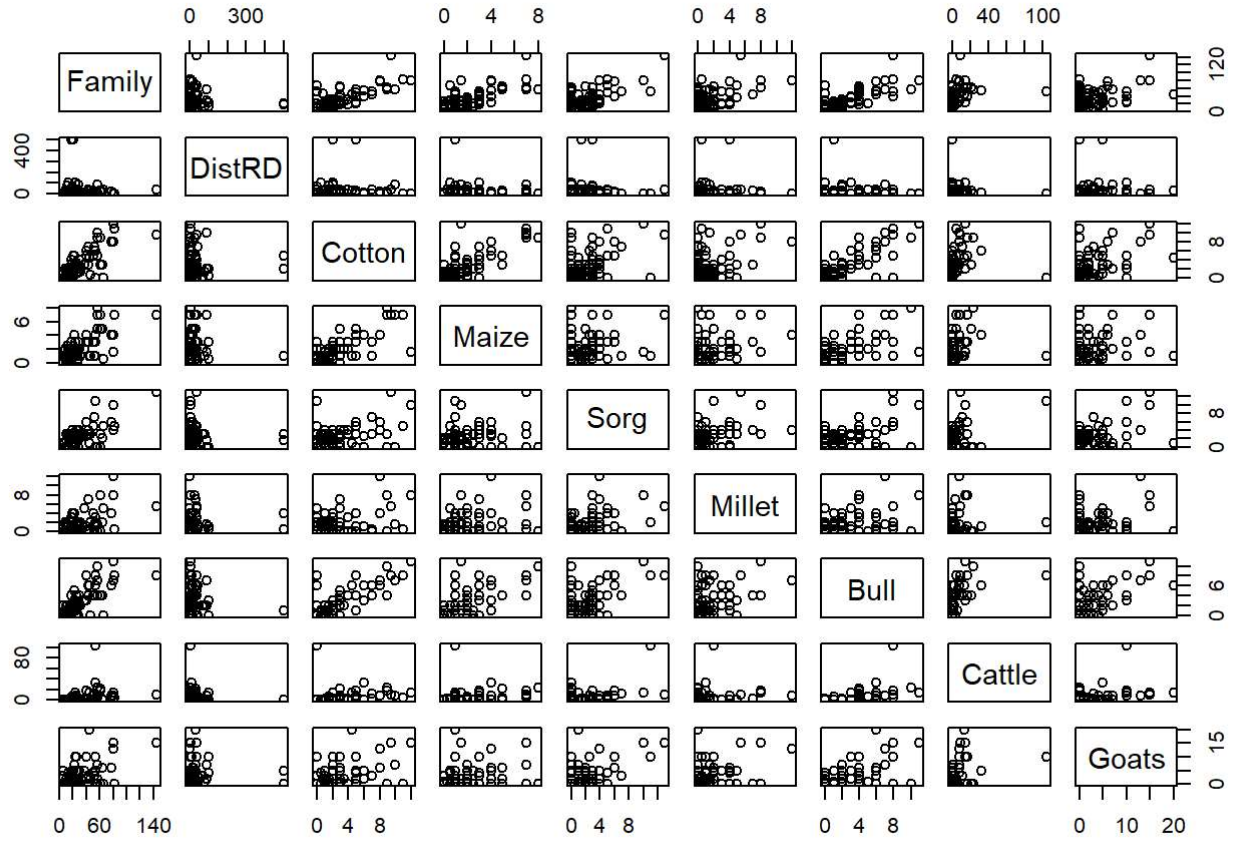
2(a)

We perform simple exploratory data analysis on the dataset mali.
Family vs DistRD is practically 0
DistRD vs Cattle is practically 0
DistRD is practically 0 with any other predictor, can probably drop
All other variables may be of use for predicting.

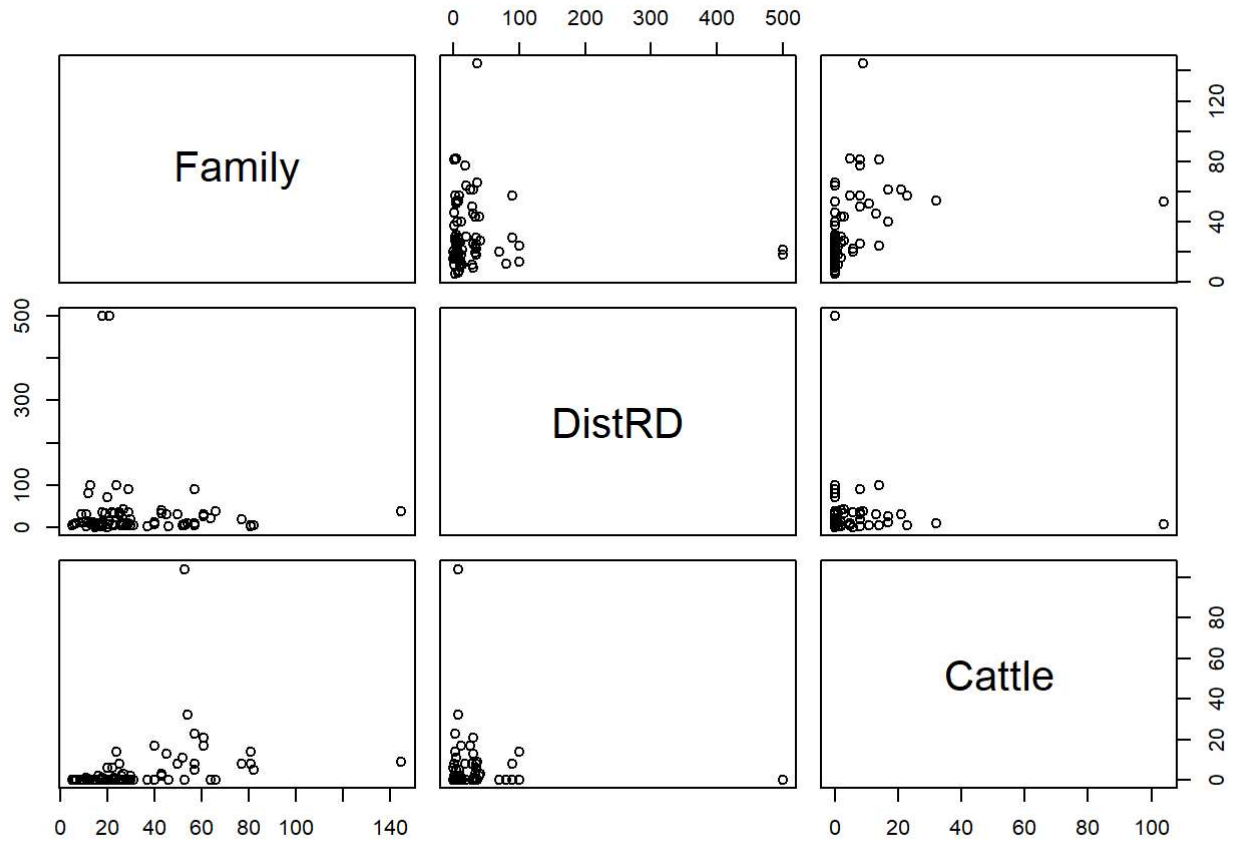
```
mali = read.csv("mali.csv", header = T)
# eda
str(mali)
```

```
## 'data.frame':    76 obs. of  9 variables:
##  $ Family: int  12 54 11 21 61 20 29 29 57 23 ...
##  $ DistRD: num  80 8 13 13 30 70 35 35 9 33 ...
##  $ Cotton: num  1.5 6 0.5 2 3 0 1.5 2 5 2 ...
##  $ Maize : num  1 4 1 2.5 5 2 2 3 5 2 ...
##  $ Sorg  : num  3 0 0 1 0 3 0 2 0 1 ...
##  $ Millet: num  0.25 1 0 0 0 0 0 0 0 0 ...
##  $ Bull  : int  2 6 0 1 4 2 0 0 4 2 ...
##  $ Cattle: int  0 32 0 0 21 0 0 0 5 1 ...
##  $ Goats : int  1 5 0 5 0 3 0 0 2 7 ...
```

```
pairs(subset(mali))
```



```
pairs(subset(mali, select = c(Family, DistRD, Cattle)))
```



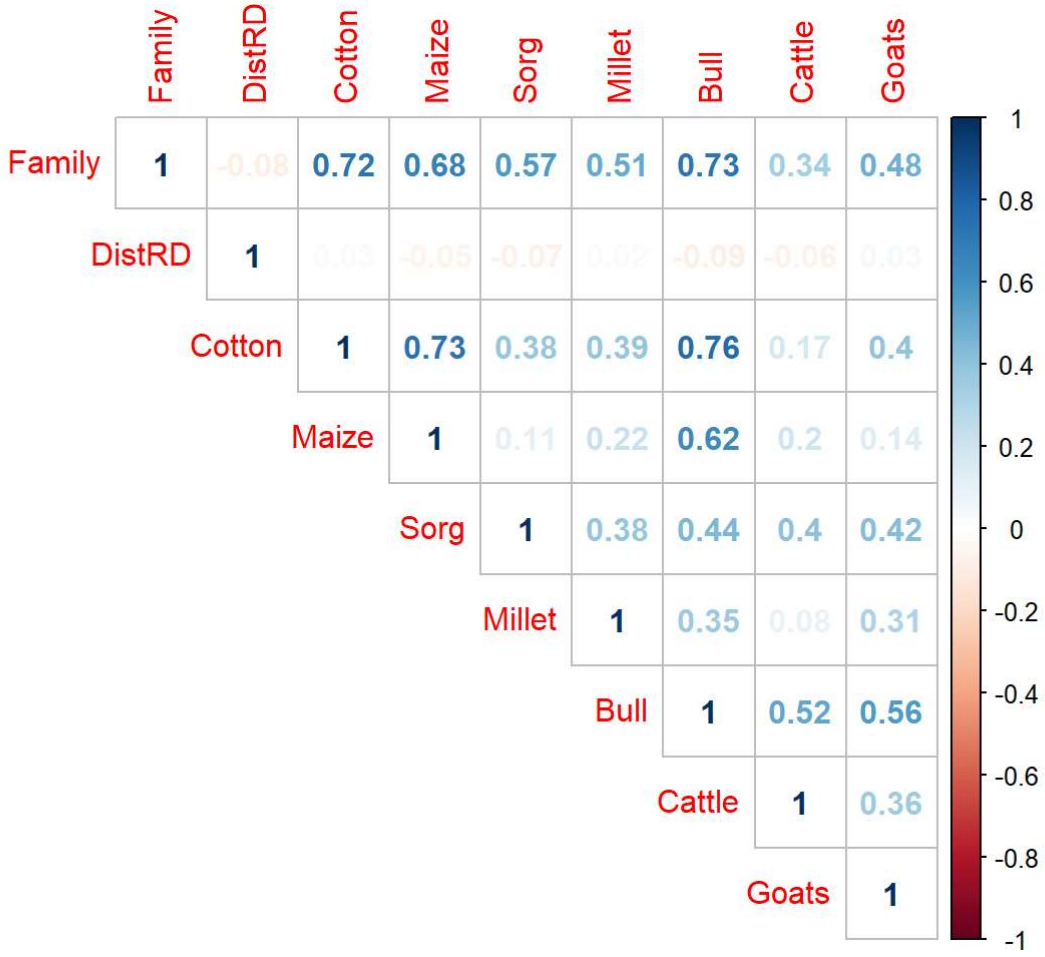

```
round(cor(mali), 3)
```

```
##         Family DistRD Cotton  Maize   Sorg Millet   Bull Cattle Goats
## Family   1.000 -0.084  0.724  0.679  0.568  0.506  0.727  0.336 0.484
## DistRD -0.084  1.000  0.028 -0.054 -0.071  0.022 -0.088 -0.063 0.031
## Cotton   0.724  0.028  1.000  0.730  0.383  0.389  0.765  0.175 0.399
## Maize    0.679 -0.054  0.730  1.000  0.109  0.217  0.623  0.197 0.136
## Sorg     0.568 -0.071  0.383  0.109  1.000  0.382  0.443  0.404 0.424
## Millet   0.506  0.022  0.389  0.217  0.382  1.000  0.353  0.081 0.305
## Bull     0.727 -0.088  0.765  0.623  0.443  0.353  1.000  0.520 0.560
## Cattle   0.336 -0.063  0.175  0.197  0.404  0.081  0.520  1.000 0.357
## Goats    0.484  0.031  0.399  0.136  0.424  0.305  0.560  0.357 1.000
```

```
round(cor(subset(mali, select = c(Family, DistRD, Cattle))), 3)
```

```
##         Family DistRD Cattle
## Family   1.000 -0.084  0.336
## DistRD -0.084  1.000 -0.063
## Cattle   0.336 -0.063  1.000
```

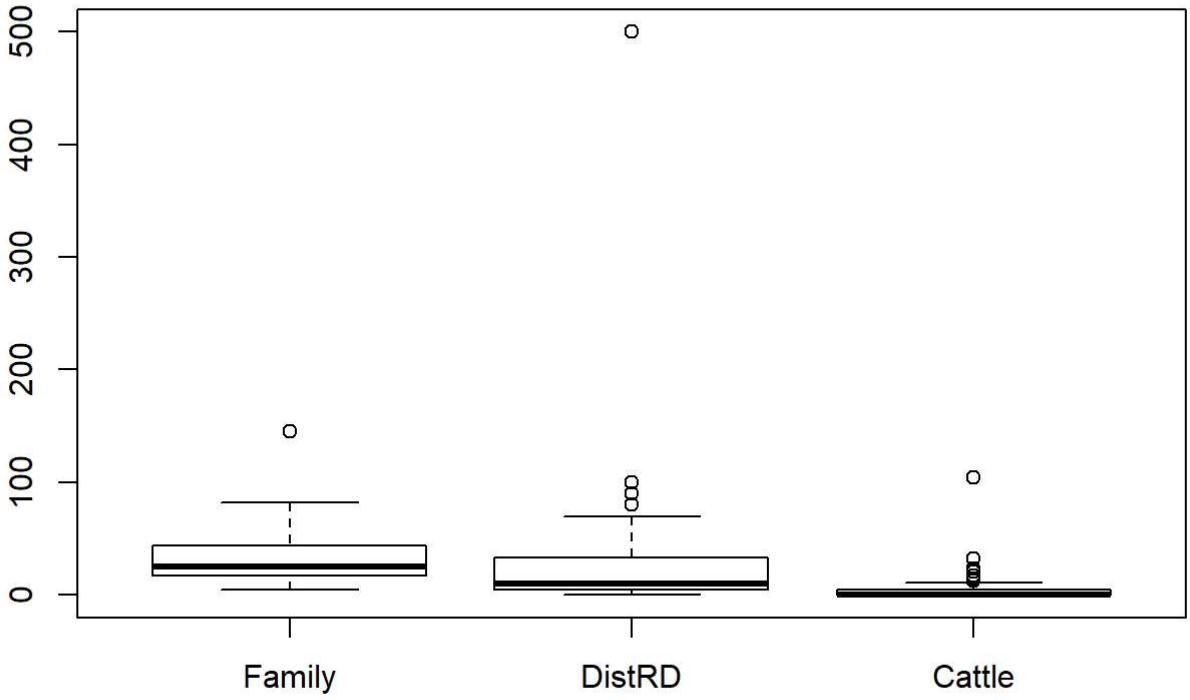
```
corrplot::corrplot(cor(mali), method = "number", type = "upper")
```




```
rbind(apply(mali, 2, mean),apply(mali, 2, var)) # => scale issues
```

```
##         Family     DistRD   Cotton    Maize     Sorg   Millet     Bull
## [1,]  32.36842   32.86184 2.991447 2.082237 2.651316 1.671053 2.631579
## [2,] 550.87579 6533.75066 8.012226 3.433980 5.700132 4.942018 7.089123
##          Cattle     Goats
## [1,]   4.657895  2.973684
## [2,] 173.081404 17.012632
```

```
# it is apparent how different they are from the mean/variance
# DistRD doesn't seem to play well with others
boxplot(subset(mali, select = c(Family, DistRD, Cattle)))$out # values of outliers
```




```
##  [1] 145  80 500 500 100 100  90  90  32  21  13 104  17  14  17  23  14
```

```
# we delete the most outlying row from each category, respectively
mali.new = mali[-c(25, 72, 34), ] # these obs are far out
```

2(b)

Most of the data is in hectares unit, however not all the data is using
the same unit measures. Therefore, without a doubt we should standardize!

2(c)

We want to use Principal Components Analysis (PCA) to reduce our model
to its most important characteristics. We look to the Proportion of
Variation Explained (PVE) and its cumulitive sum and see that
five to six variables should explain most of our variance (85%-95%).
The 1st graph below shows in what direction our components traverse,
we see that Maize & cattle got together and Millet & Sorg go together and etc.
Notice that this biplot corresponds to the first component vs the second.
The 2nd graph shows the component scores against each other, there is some
positive correlation with z1&z2 but not so much with z1&z3.
The 3rd graph shows the PVE and Cumulative PVE, seems like we only need a few!

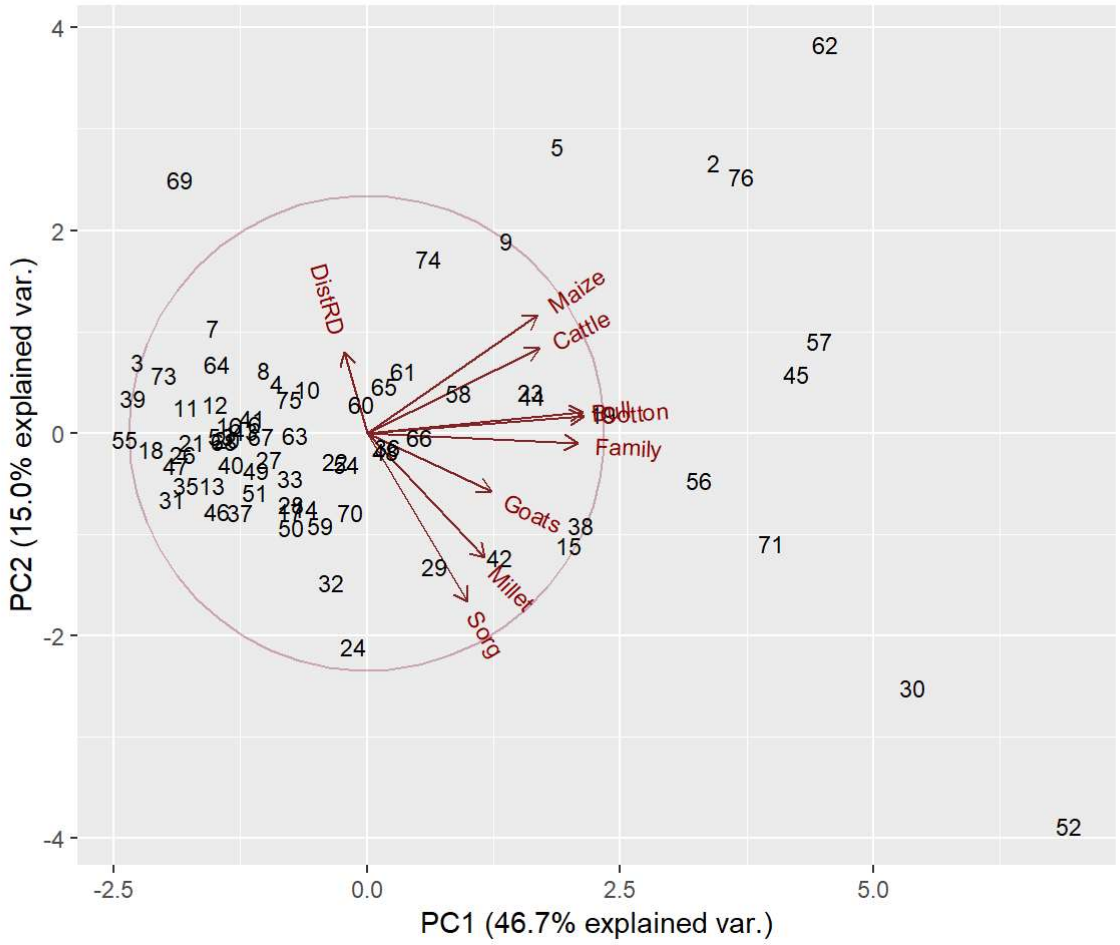
```r
library(pls)
library(ggbiplot)
require(ggplot2)
set.seed(1)
pr.out = prcomp(mali.new, scale. = T)
rbind(pr.out$center, pr.out$scale)
```

```
##        Family   DistRD   Cotton    Maize     Sorg   Millet     Bull
## [1,] 30.69863 26.77397 2.915753 2.044521 2.390411 1.582192 2.506849
## [2,] 19.67492 61.05883 2.751652 1.790863 1.847007 2.204421 2.555622
##        Cattle    Goats
## [1,] 3.301370 2.684932
## [2,] 6.387238 3.858169
```

```r
summary(pr.out)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4    PC5     PC6     PC7
## Standard deviation     2.0506 1.1612 0.9986 0.94022 0.8067 0.61693 0.48913
## Proportion of Variance 0.4672 0.1498 0.1108 0.09822 0.0723 0.04229 0.02658
## Cumulative Proportion  0.4672 0.6170 0.7278 0.82607 0.8984 0.94066 0.96724
##                           PC8     PC9
## Standard deviation     0.4157 0.34934
## Proportion of Variance 0.0192 0.01356
## Cumulative Proportion  0.9864 1.00000
```

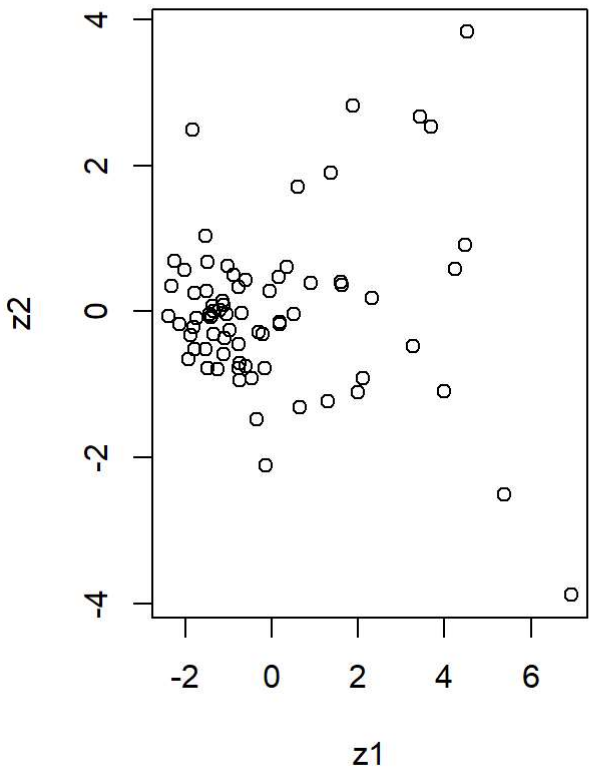
```r
# biplot(pr.out, scale=0)
ggbiplot(pr.out, obs.scale = 1, var.scale = 1, labels=row.names(mali.new),
         ellipse = TRUE, circle = TRUE) +
  scale_color_discrete(name = '') +
  theme(legend.direction = 'horizontal', legend.position = 'top')
```

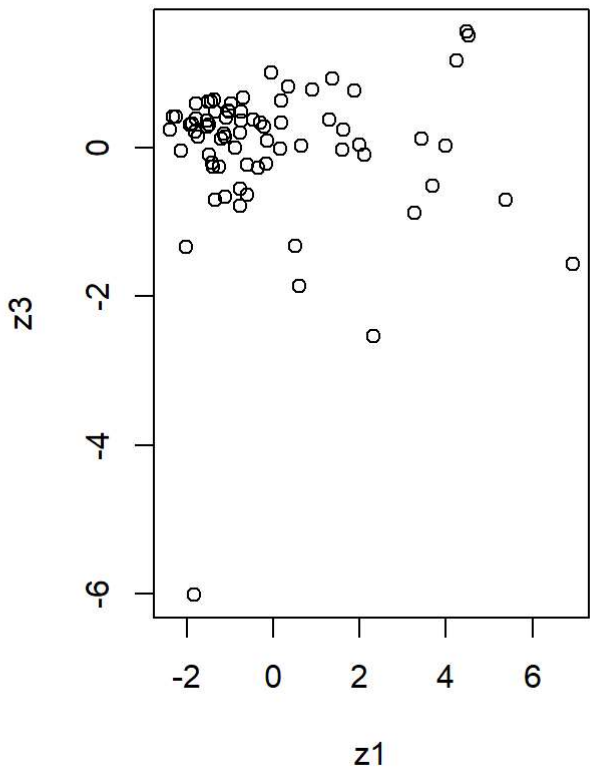



```
pr.var = pr.out$sdev^2
pve = pr.var/sum(pr.var)
pve
```

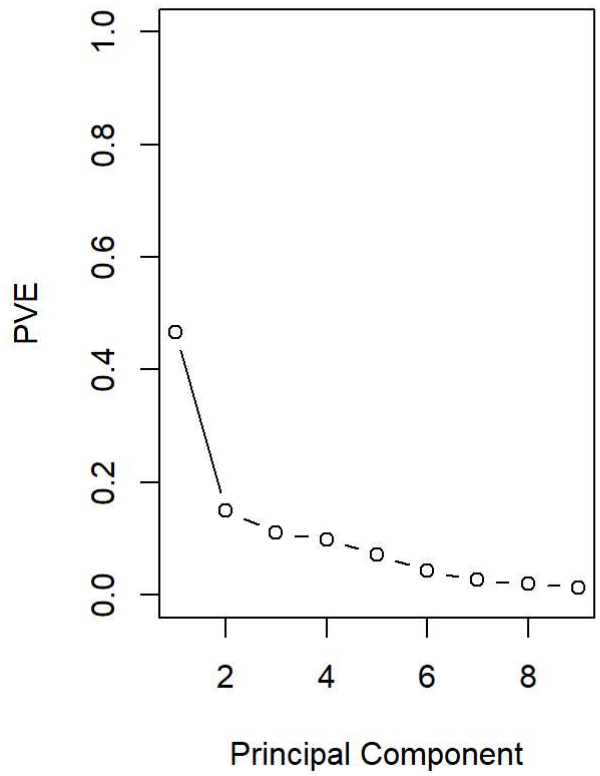
```
## [1] 0.46722674 0.14982206 0.11079577 0.09822438 0.07230117 0.04228870
## [7] 0.02658357 0.01919814 0.01355946
```

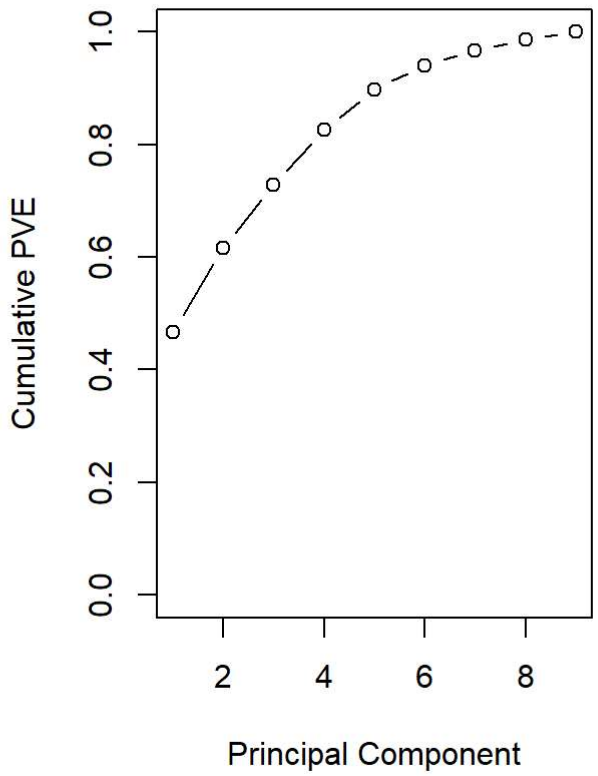
```
par(mfrow = c(1,2))
plot(pr.out$x[,1:2], xlab = "z1", ylab = "z2")
plot(pr.out$x[,c(1,3)], xlab = "z1", ylab = "z3")
```






```
plot(pve, xlab = "Principal Component", ylab = "PVE", ylim = c(0,1), type = 'b')
plot(cumsum(pve), xlab = "Principal Component",
     ylab = "Cumulative PVE", ylim = c(0,1), type = 'b')
```





2(d)

Based on the biplot above and the rotation matrix below, we see that Family, Cotton, and Bull each have similar weight.
It appears each component corresponds to a measure of farmer's resource density.

```
pr.out$rotation[,1:2]
```

```
##                PC1         PC2
## Family  0.43335948 -0.03640144
## DistRD -0.04734891  0.29462528
## Cotton  0.44443599  0.06193349
## Maize   0.35098001  0.42716213
## Sorg    0.20691334 -0.61257733
## Millet  0.24223514 -0.45284956
## Bull    0.44398103  0.07644839
## Cattle  0.35435834  0.30726657
## Goats   0.25508301 -0.21210097
```

```
summary(pr.out)$importance[2:3,1:2]
```

```
##                            PC1     PC2
## Proportion of Variance 0.46723 0.14982
## Cumulative Proportion  0.46723 0.61705
```

2(e)

Component 1: Farmers with multi-purpose farms
Component 2: Farmland specifically used for cereal/grain crops
Component 3: Goats coincide with roads, perhaps by being more free-range
Component 4: Similar to component 3 with increases to cattle and maize
Component 5: Similar to component 2

```
pr.out$rotation[,1:5]
```

```
##                PC1         PC2         PC3         PC4         PC5
## Family  0.43335948 -0.03640144  0.07425531 -0.18036258  0.06010274
## DistRD -0.04734891  0.29462528 -0.80373951 -0.49986985 -0.10371608
## Cotton  0.44443599  0.06193349  0.04494546 -0.09981657 -0.22679103
## Maize   0.35098001  0.42716213  0.27980176 -0.25276136  0.03208743
## Sorg    0.20691334 -0.61257733 -0.04008551 -0.15060355 -0.63957831
## Millet  0.24223514 -0.45284956 -0.01255267 -0.39528072  0.68796089
## Bull    0.44398103  0.07644839 -0.06409519  0.11015108 -0.07635223
## Cattle  0.35435834  0.30726657 -0.05269781  0.32564112  0.01168883
## Goats   0.25508301 -0.21210097 -0.50944103  0.58876493  0.21177812
```

3(a)

Here we fit a regular linear model

```
auto = subset(ISLR::Auto, select = -name) # exclude name because it gets nasty
set.seed(1)
x<-model.matrix(mpg~., auto)[,-1]
y<-auto$mpg
fit = lm(y~x)
summary(fit)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5903 -2.1565 -0.1169  1.8690 13.0604
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -17.218435   4.644294  -3.707  0.00024 ***
## xcylinders     -0.493376   0.323282  -1.526  0.12780
## xdisplacement   0.019896   0.007515   2.647  0.00844 **
## xhorsepower    -0.016951   0.013787  -1.230  0.21963
## xweight        -0.006474   0.000652  -9.929  < 2e-16 ***
## xacceleration   0.080576   0.098845   0.815  0.41548
## xyear           0.750773   0.050973  14.729  < 2e-16 ***
## xorigin         1.426141   0.278136   5.127 4.67e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.328 on 384 degrees of freedom
## Multiple R-squared:  0.8215, Adjusted R-squared:  0.8182
## F-statistic: 252.4 on 7 and 384 DF,  p-value: < 2.2e-16
```

```
pred = predict(fit, newx = x)
LS.ERR = mean((pred - y)^2) # test MSE (same as train MSE)
LS = round(summary(fit)$coefficients[,1], 3)
```

3(b)

Now we perform best subset and see what we get

```
set.seed(1)
totpred <- ncol(auto) - 1
fit <- leaps::regsubsets(mpg~., auto, nvmax = totpred)
fit.summary <- summary(fit)
fit.summary
```

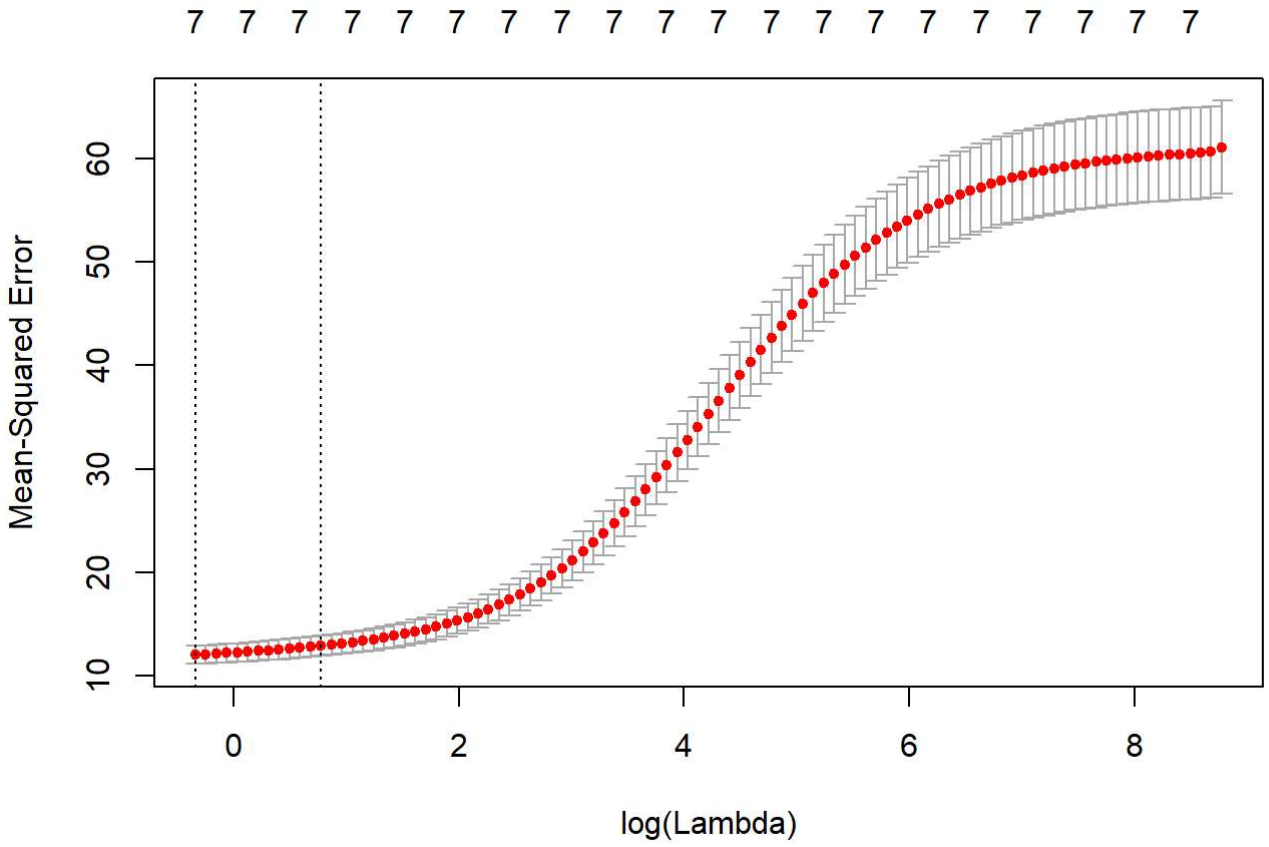
```
## Subset selection object
## Call: regsubsets.formula(mpg ~ ., auto, nvmax = totpred)
## 7 Variables  (and intercept)
##              Forced in Forced out
## cylinders        FALSE      FALSE
## displacement     FALSE      FALSE
## horsepower       FALSE      FALSE
## weight           FALSE      FALSE
## acceleration     FALSE      FALSE
## year             FALSE      FALSE
## origin           FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: exhaustive
##          cylinders displacement horsepower weight acceleration year origin
## 1  ( 1 ) " "       " "          " "        "*"    " "          " "  " "
## 2  ( 1 ) " "       " "          " "        "*"    " "          "*"  " "
## 3  ( 1 ) " "       " "          " "        "*"    " "          "*"  "*"
## 4  ( 1 ) " "       "*"          " "        "*"    " "          "*"  "*"
## 5  ( 1 ) " "       "*"          "*"        "*"    " "          "*"  "*"
## 6  ( 1 ) "*"       "*"          "*"        "*"    " "          "*"  "*"
## 7  ( 1 ) "*"       "*"          "*"        "*"    "*"          "*"  "*"
```

```
point = as.numeric(which.min(fit.summary$bic)) # graph gives best model (not shown)
# function required to perform prediction on regsubsets
predict.regsubsets = function(object, newdata, id, ...) {
    form = as.formula(object$call[[2]])
    mat = model.matrix(form, newdata)
    coefi = coef(object, id = id)
    mat[, names(coefi)] %*% coefi
}
pred = predict.regsubsets(fit, auto, point)
BS.ERR = mean((pred - y)^2)
BS = round(coef(fit, point), 3) # only uses 3 (from point)
Term = names(coef(fit, 7)) # to be used later
```

3(c)

Ridge Resgression, the graph is used as in question 1 for determining best lambda.

```
set.seed(1)
require(glmnet)
lambdas <- 10^seq(10, -2, length = 100)
x<-model.matrix(mpg~., auto)[,-1]
y<-auto$mpg
fit <- glmnet(x, y, alpha = 0, lambda = lambdas, family = "gaussian")
fit.cv <- cv.glmnet(x, y, alpha = 0)
plot(fit.cv)
```



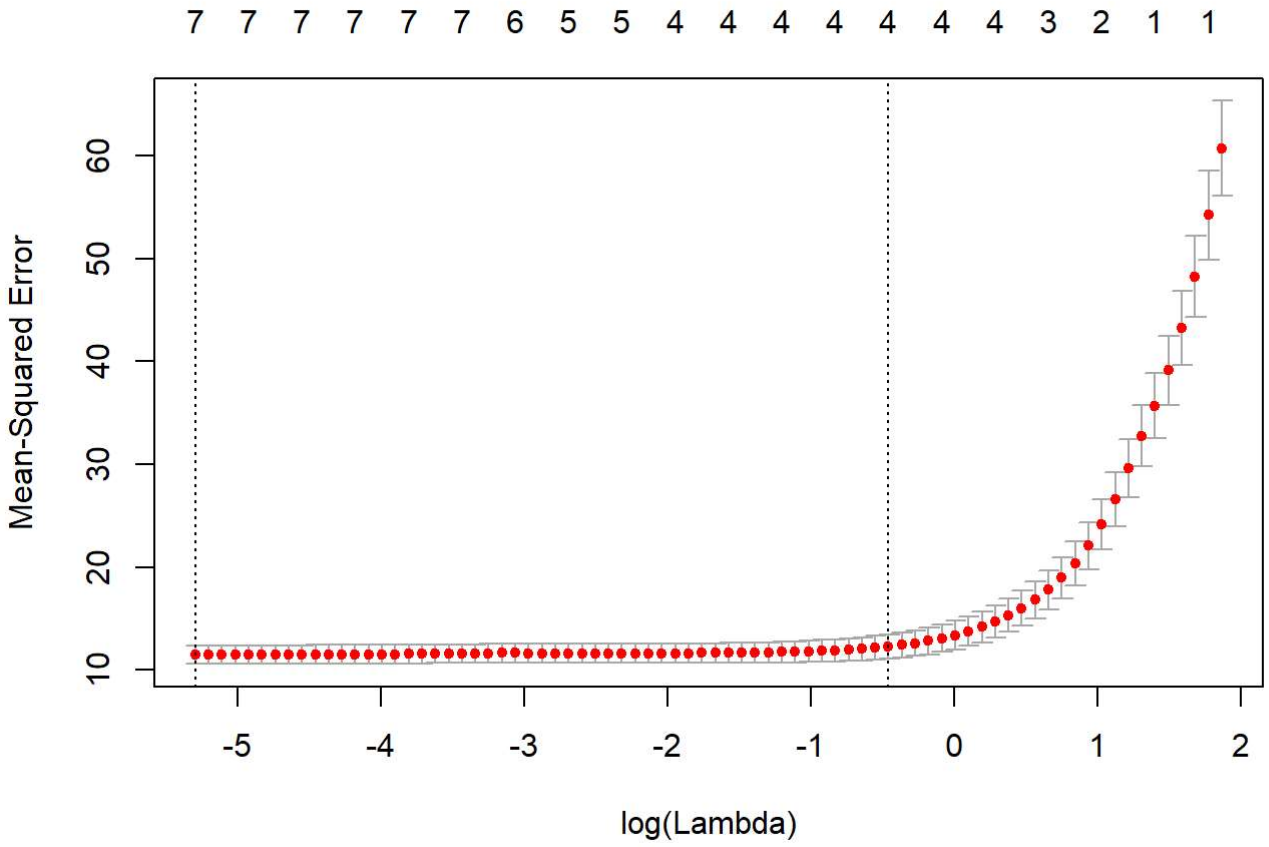

```
# Best cross-validated lambda
lambda = fit.cv$lambda.min
pred = predict(fit, s = lambda, newx = x)
RIDGE.ERR = mean((pred - y)^2)
coef = predict(fit, x, type = "coefficients", s = lambda)
Ridge = round(coef[,1], 3)
length(coef[coef != 0]) # ridge uses all variables
```

```
## [1] 8
```

3(d)

Lasso Regression, graph below appears to show 4 coefficients but the model chosen included all predictors. This may be an error in coding or the fact that the function decided the seemingly non existent decreaase in MSE was sufficient.

```
set.seed(1)
lambdas <- 10^seq(10, -2, length = 100)
fit <- glmnet(x, y, alpha = 1, lambda = lambdas, family = "gaussian")
fit.cv <- cv.glmnet(x, y, alpha = 1)
plot(fit.cv)
```




```
# Best cross-validated lambda
lambda = fit.cv$lambda.min
pred = predict(fit, s = lambda, newx = x)
LASSO.ERR = mean((pred - y)^2)
coef = predict(fit, x, type = "coefficients", s = lambda)
Lasso = round(coef[,1], 3)
length(coef[coef != 0]) # lasso uses all variables
```

```
## [1] 8
```

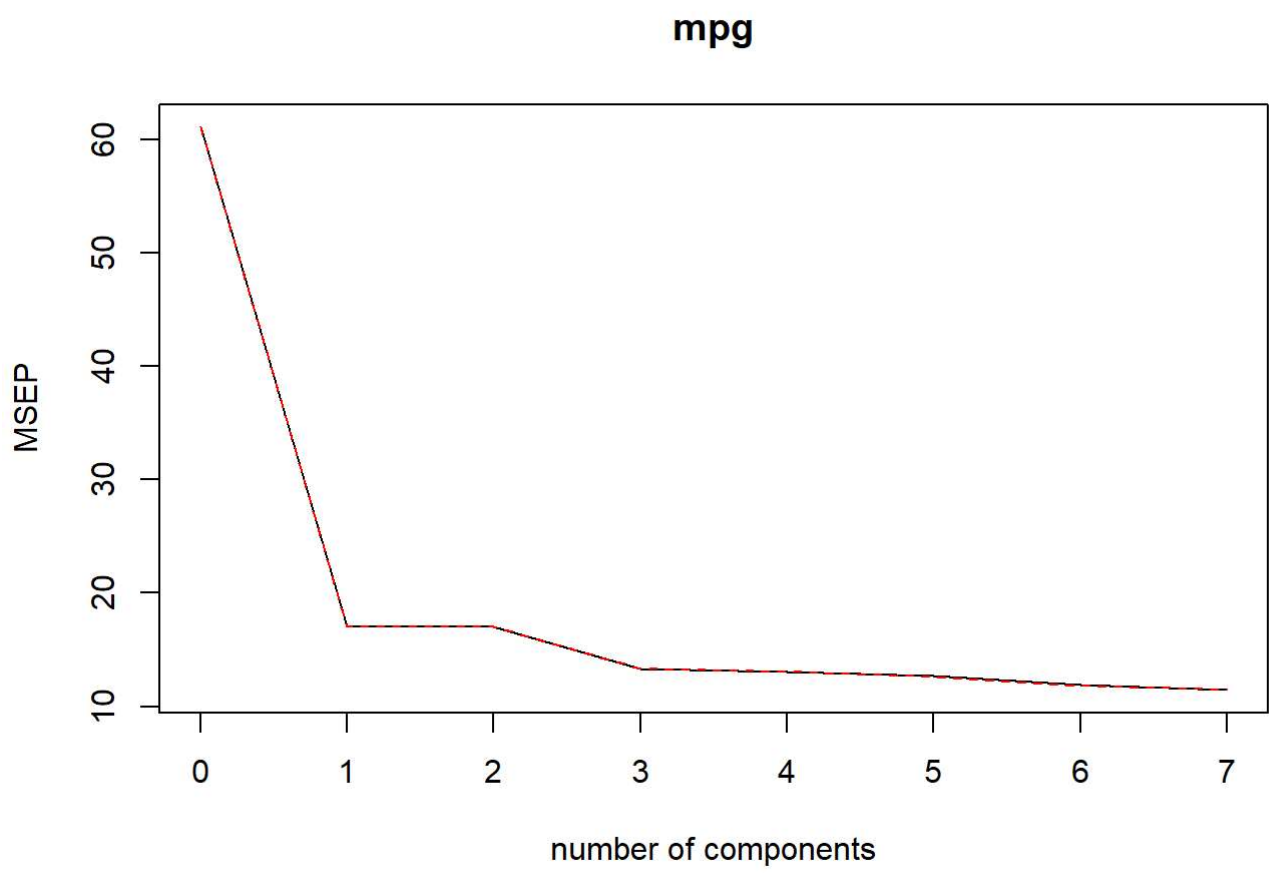
3(e)

Now we use Principal Component Regression, the graph below is similar to those used in auto selection methods. From a viewer point, 3 predictors seem to be ideal.
However, the function decided on a full model fit similar to 3(d)'s model.

```
require(pls)
set.seed(1)
fit = pcr(mpg~., data = auto, scale = T, validation = "CV")
summary(fit)
```

```
## Data:    X dimension: 392 7
##  Y dimension: 392 1
## Fit method: svdpc
## Number of components considered: 7
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           7.815    4.128    4.126    3.649    3.616    3.559    3.449
## adjCV        7.815    4.126    4.124    3.646    3.612    3.554    3.443
##        7 comps
## CV       3.389
## adjCV    3.384
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      65.89    79.32    89.95    96.82    98.67    99.52   100.00
## mpg    72.28    72.51    78.83    79.30    80.22    81.50    82.15
```

```
validationplot(fit, val.type = "MSEP")
```



```
fit = pcr(mpg~., data = auto, scale = T, ncomp = 7)
pred = predict(fit, newx = x)
PCR.ERR = mean((pred - y)^2)
summary(fit)
```

```
## Data:    X dimension: 392 7
##  Y dimension: 392 1
## Fit method: svdpc
## Number of components considered: 7
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      65.89    79.32    89.95    96.82    98.67    99.52   100.00
## mpg    72.28    72.51    78.83    79.30    80.22    81.50    82.15
```

```
PCR = round(coef(fit, intercept = T), 3)
```

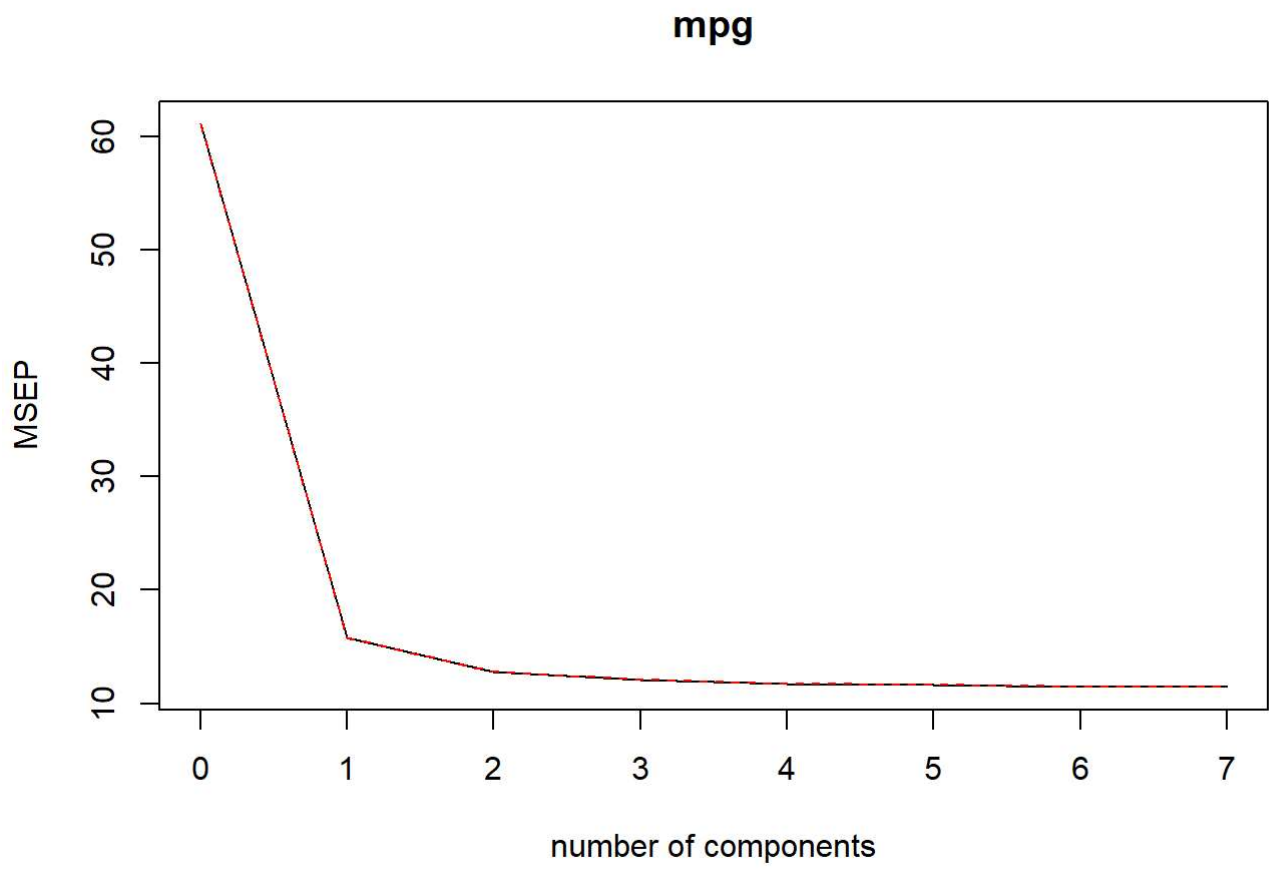
3(f)

Finally we do Partial Least Squares.
Here we use less components because PLS explains the predictor more effectively.
Graph is smoother than 3(e)'s but we conclude similar reasonings except PLS does suggest just two components in comparison to PCR which chose all components.

```
set.seed(1)
fit = plsr(mpg~., data = auto, scale = T, validation = "CV")
summary(fit)
```

```
## Data:    X dimension: 392 7
##  Y dimension: 392 1
## Fit method: kernelpls
## Number of components considered: 7
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           7.815    3.973    3.575    3.482    3.428    3.412    3.391
## adjCV        7.815    3.971    3.570    3.478    3.425    3.408    3.386
##        7 comps
## CV       3.389
## adjCV    3.384
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      65.75    76.46    82.97    91.01    97.97    99.40   100.00
## mpg    74.44    79.88    80.93    81.60    81.86    82.11    82.15
```

```
validationplot(fit, val.type = "MSEP") # 2 components look good
```




```
fit = plsr(y~x, data = auto, scale = T, ncomp = 2)
pred = predict(fit, newx = x)
PLS.ERR = mean((pred - y)^2)
summary(fit)
```

```
## Data:    X dimension: 392 7
##  Y dimension: 392 1
## Fit method: kernelpls
## Number of components considered: 2
## TRAINING: % variance explained
##    1 comps  2 comps
## X    65.75    76.46
## y    74.44    79.88
```

```
PLS = round(coef(fit, intercept = T), 3)
```

3(g)

Below is a table representing the weights of each coefficient for each of the different models alongside their test MSE. We see that Best Subset Selection results in fewer coefficients being used but Partial Least Squares resulted in the smallest test MSE. We recommend the best subset model because of the aforementioned, its test MSE is only 0.7 higher than then PLS's.

```
# round(cbind(LS, BS, Ridge, Lasso, PCR, PLS), 3)
table = rowr::cbind.fill(Term, LS, BS, Ridge, Lasso, PCR, PLS, fill = NA)
colnames(table)<-c("Term", "LS", "BS", "Ridge", "Lasso", "PCR", "PLS")
# note the NA in Term is error row
table = rbind(table, c(LS.ERR, BS.ERR, RIDGE.ERR, LASSO.ERR, PCR.ERR, PLS.ERR))
table
```

Term <fctr>	**LS** <dbl>	**BS** <dbl>	**Ridge** <dbl>	**Lasso** <dbl>	**PCR** <dbl>	**PLS** <dbl>
(Intercept)	-17.21800	-18.04600	-10.99000	-17.22700	-17.21800	-17.27000
cylinders	-0.49300	-0.00600	-0.40100	-0.35700	-0.84200	-1.08500
displacement	0.02000	0.75700	-0.00300	0.01500	2.08200	-1.12600
horsepower	-0.01700	1.15000	-0.02800	-0.01500	-0.65200	-1.13300
weight	-0.00600	*NA*	-0.00400	-0.00600	-5.49900	-1.88300
acceleration	0.08100	*NA*	-0.05200	0.07200	0.22200	-0.42200
year	0.75100	*NA*	0.65500	0.74700	2.76600	2.72300
origin	1.42600	*NA*	1.28500	1.36600	1.14900	1.11900
NA	11.09211	11.50372	10.85812	13.29987	13.87994	10.84748

9 rows