AMS 380 Final

Prashun Dey

Part 1

Question 1

```r
data1 <- read.csv('Enigma.csv')
cat('There are', nrow(data1) - nrow(na.omit(data1)), 'missing values in the dataset.')
```

```
## There are 0 missing values in the dataset.
```

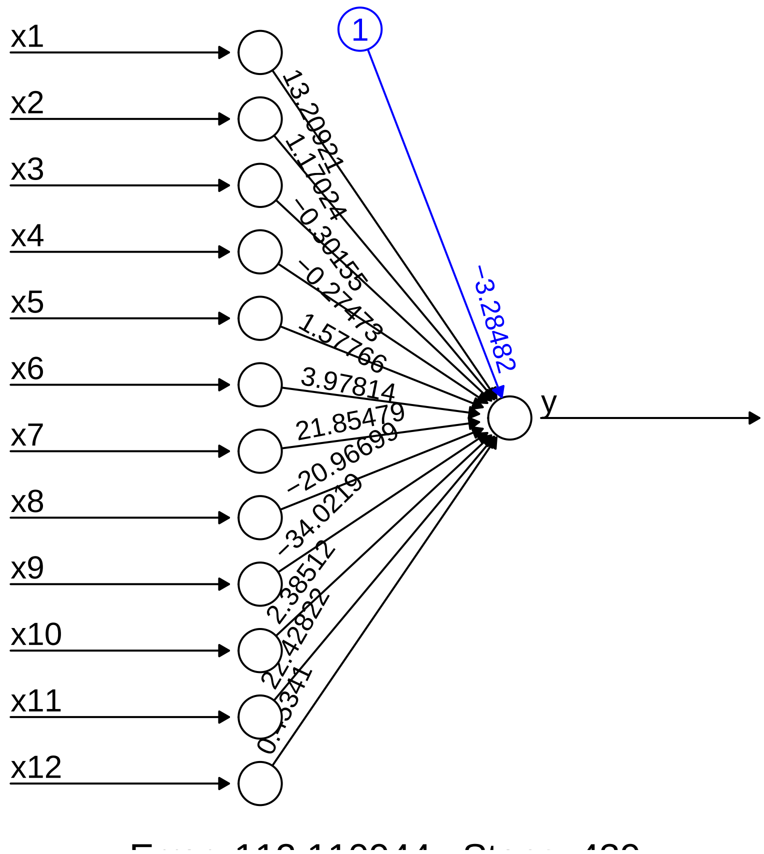
```r
# clean data without missing values
data1 <- na.omit(data1)
```

```r
set.seed(123)
training.samples <- data1$y %>% createDataPartition(p = 0.75, list = FALSE)
train.data  <- data1[training.samples, ]
test.data <- data1[-training.samples, ]
```

Question 2

2.a

```r
set.seed(123)
model <- neuralnet(y~., data = train.data, hidden = 0, err.fct = "sse", linear.output = F)
plot(model, rep = "best")
```



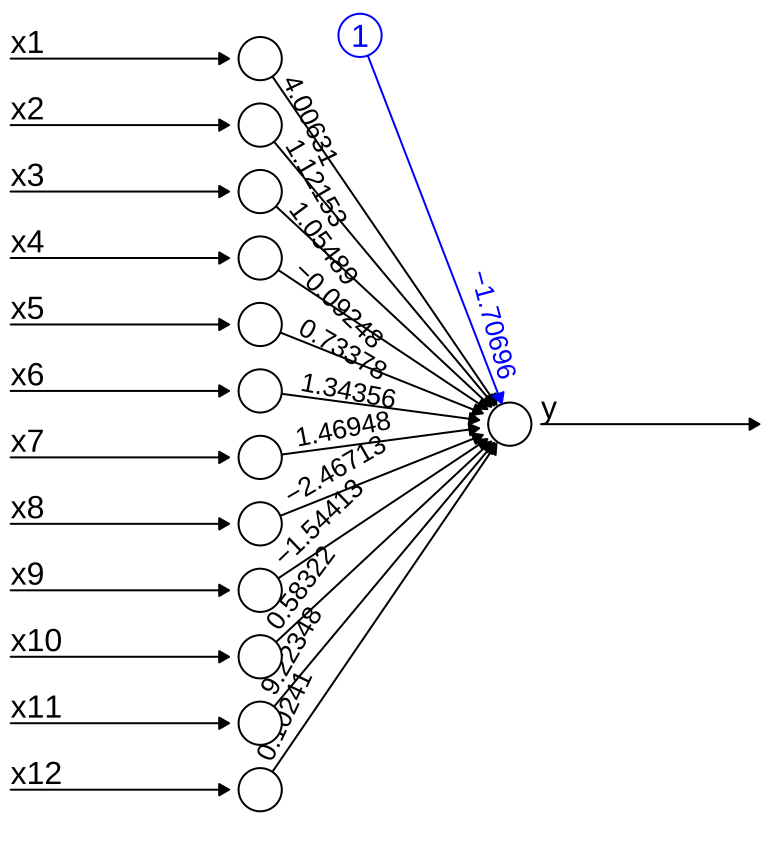

```r
probabilities <- model %>% predict(test.data) %>% as.vector()
predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
confusionMatrix(factor(predicted.classes), factor(test.data$y), positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 653  55
##          1  41 401
##
##                Accuracy : 0.9165
##                  95% CI : (0.899, 0.9319)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8246
##
##  Mcnemar's Test P-Value : 0.1846
##
##             Sensitivity : 0.8794
##             Specificity : 0.9409
##          Pos Pred Value : 0.9072
##          Neg Pred Value : 0.9223
##              Prevalence : 0.3965
```

```
##          Detection Rate : 0.3487
##    Detection Prevalence : 0.3843
##       Balanced Accuracy : 0.9102
##
##        'Positive' Class : 1
##
```

2.b

```r
set.seed(123)
model <- neuralnet(y~., data = train.data, hidden = 0, err.fct = "ce", linear.output = F)
plot(model, rep = "best")
```




```r
probabilities <- model %>% predict(test.data)
predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
confusionMatrix(factor(predicted.classes), factor(test.data$y), positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 650  97
##          1  44 359
```

```
##
##                Accuracy : 0.8774
##                  95% CI : (0.857, 0.8958)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7386
##
##  Mcnemar's Test P-Value : 1.191e-05
##
##             Sensitivity : 0.7873
##             Specificity : 0.9366
##          Pos Pred Value : 0.8908
##          Neg Pred Value : 0.8701
##              Prevalence : 0.3965
##          Detection Rate : 0.3122
##    Detection Prevalence : 0.3504
##       Balanced Accuracy : 0.8619
##
##        'Positive' Class : 1
##
```

2.c

```
set.seed(123)
model <- glm(y~., family = binomial, data = train.data)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
model
```

```
##
## Call:  glm(formula = y ~ ., family = binomial, data = train.data)
##
## Coefficients:
## (Intercept)           x1           x2           x3           x4           x5
##    -1.70687      4.00598      1.12141      1.05502     -0.09251      0.73377
##          x6           x7           x8           x9          x10          x11
##     1.34362      1.46950     -2.46714     -1.54422      0.58319      9.22351
##         x12
##     0.10240
##
## Degrees of Freedom: 3450 Total (i.e. Null);  3438 Residual
## Null Deviance:       4625
## Residual Deviance: 2147  AIC: 2173
```

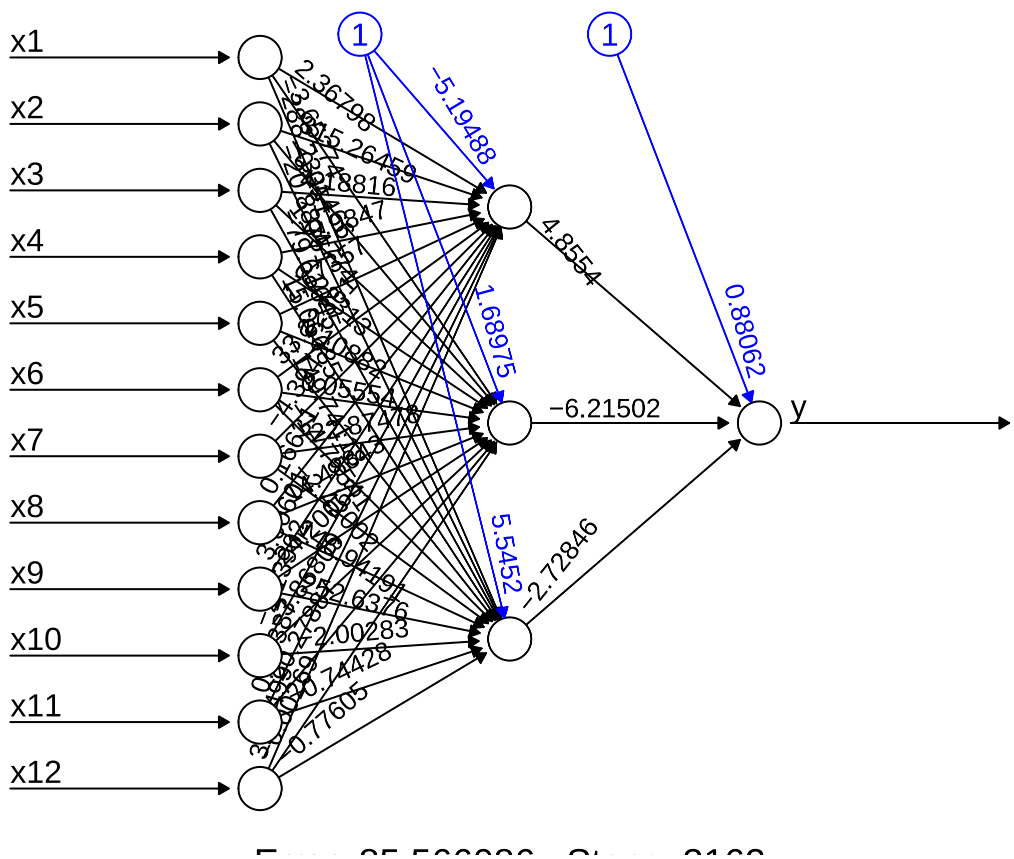
The CE Loss function model better resembles logistic regression model.

```
probabilities <- model %>% predict(test.data, type = 'response')
predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
confusionMatrix(factor(predicted.classes), factor(test.data$y), positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 650  97
##          1  44 359
##
##                Accuracy : 0.8774
##                  95% CI : (0.857, 0.8958)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7386
##
##  Mcnemar's Test P-Value : 1.191e-05
##
##             Sensitivity : 0.7873
##             Specificity : 0.9366
##          Pos Pred Value : 0.8908
##          Neg Pred Value : 0.8701
##              Prevalence : 0.3965
##          Detection Rate : 0.3122
##    Detection Prevalence : 0.3504
##       Balanced Accuracy : 0.8619
##
##        'Positive' Class : 1
##
```

2.d

```r
set.seed(123)
model <- neuralnet(y~., data = train.data, hidden = 3, err.fct = "sse", linear.output = F)
plot(model, rep = "best")
```




```r
probabilities <- model %>% predict(test.data)
predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
confusionMatrix(factor(predicted.classes), factor(test.data$y), positive = '1')
```

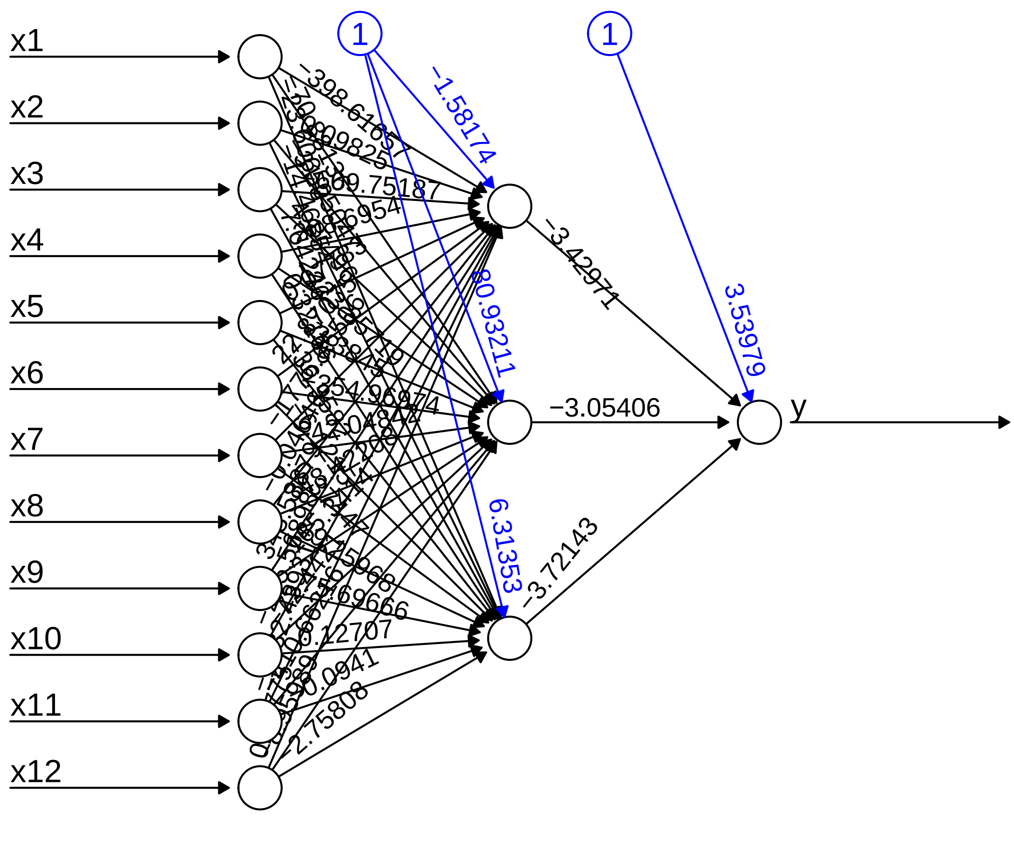
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 662  42
##          1  32 414
##
##                Accuracy : 0.9357
##                  95% CI : (0.9199, 0.9491)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.865
##
##  Mcnemar's Test P-Value : 0.2955
##
##             Sensitivity : 0.9079
##             Specificity : 0.9539
##          Pos Pred Value : 0.9283
##          Neg Pred Value : 0.9403
##              Prevalence : 0.3965
```

```
##          Detection Rate : 0.3600
##    Detection Prevalence : 0.3878
##       Balanced Accuracy : 0.9309
##
##        'Positive' Class : 1
##
```

The prediction with hidden layers is better than no hidden layer.‘

2.e

```r
set.seed(123)
model <- neuralnet(y~., data = train.data, hidden = 3, err.fct = "ce", linear.output = F)
plot(model, rep = "best")
```




```r
probabilities <- model %>% predict(test.data)
predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
confusionMatrix(factor(predicted.classes), factor(test.data$y), positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 647  38
##          1  47 418
##
##                Accuracy : 0.9261
##                  95% CI : (0.9094, 0.9405)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8461
##
##  Mcnemar's Test P-Value : 0.3855
##
##             Sensitivity : 0.9167
##             Specificity : 0.9323
##          Pos Pred Value : 0.8989
##          Neg Pred Value : 0.9445
##              Prevalence : 0.3965
##          Detection Rate : 0.3635
##    Detection Prevalence : 0.4043
##       Balanced Accuracy : 0.9245
##
##        'Positive' Class : 1
##
```

The prediction with hidden layers is better than no hidden layer.

Question 3

3.a

```r
########### To conduct the random forest, need factorize the response data, or will become regression r
train.data$y <- factor(train.data$y)
test.data$y <- factor(test.data$y)
##################################

set.seed(123)
model <- train(
  y ~., data = train.data, method = "rf",
  trControl = trainControl("cv", number = 10),
  importance = TRUE
  )
# Best tuning parameter
model$bestTune
```

```
##   mtry
## 1    2
```

```r
model$finalModel
```

```
##
## Call:
```

```
##  randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.29%
## Confusion matrix:
##      0    1 class.error
## 0 2028   66  0.03151862
## 1  151 1206  0.11127487
```

Overall Accuracy

```
(2028+1206)/(2028+1206+151+66)
```

```
## [1] 0.9371197
```

Sensitivity

```
1206/(1206+151)
```

```
## [1] 0.8887251
```

Specificity

```
2028/(2028+66)
```

```
## [1] 0.9684814
```

3.b

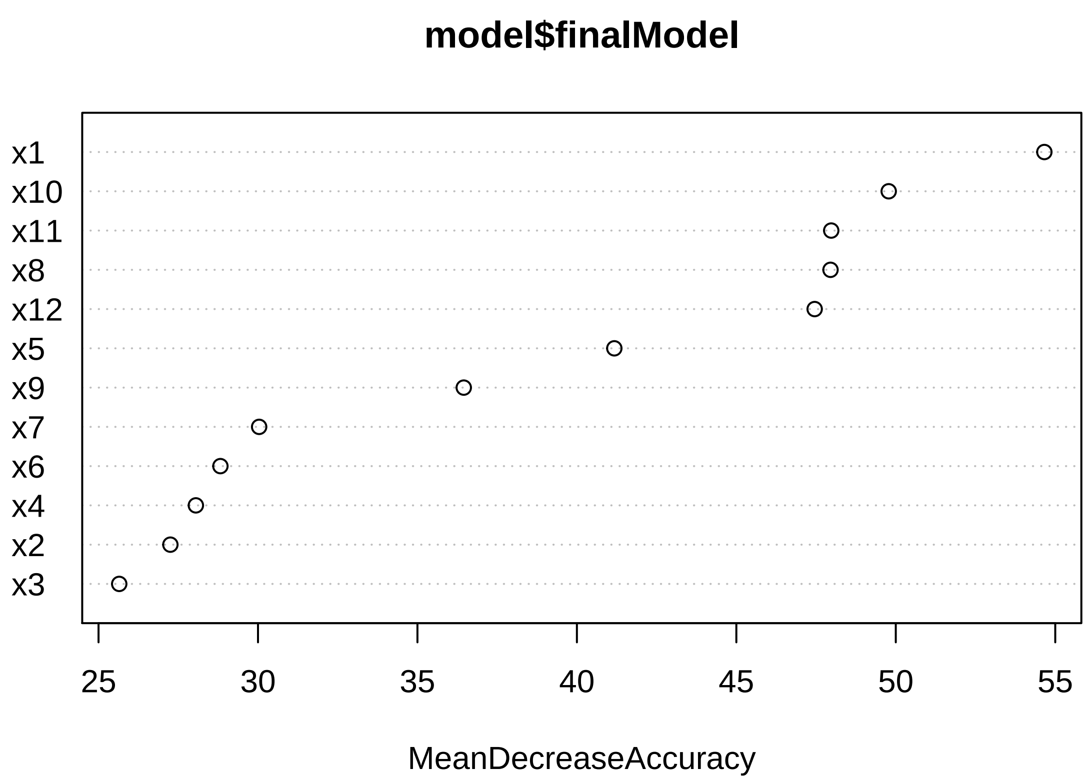
```
pred <- model %>% predict(test.data)
#predict(model, test)
confusionMatrix(pred, test.data$y, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 672  58
##          1  22 398
##
##                Accuracy : 0.9304
##                  95% CI : (0.9142, 0.9445)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8526
##
```

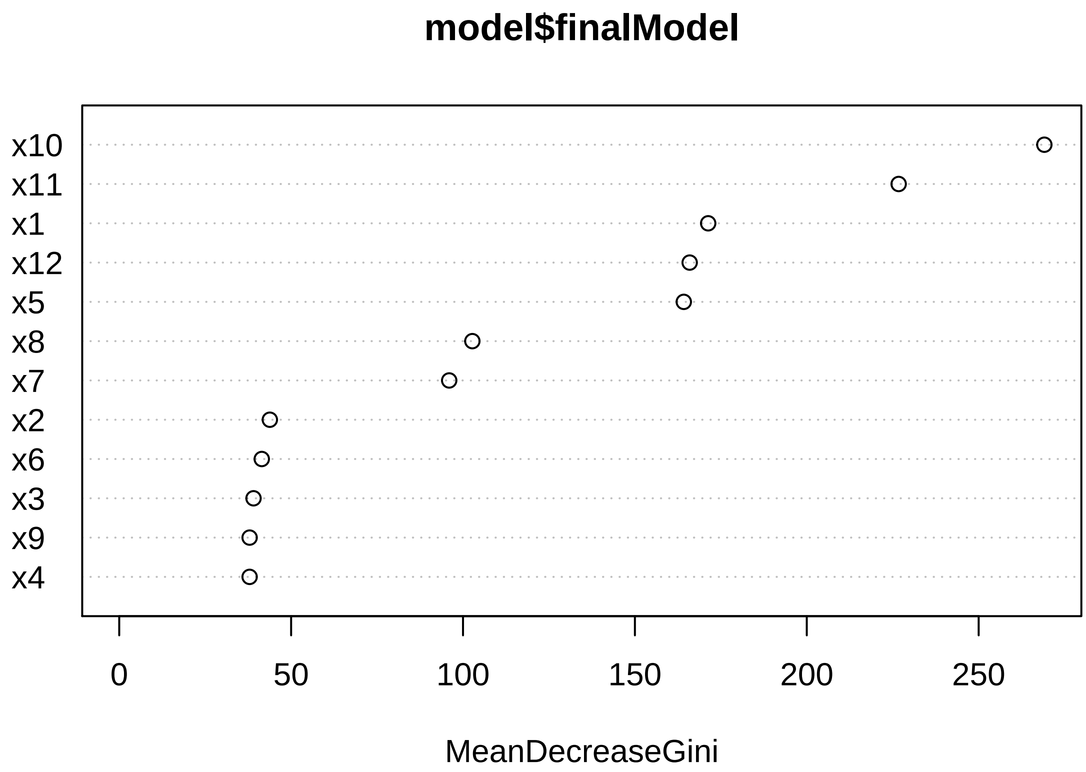
```
##  Mcnemar's Test P-Value : 9.111e-05
##
##             Sensitivity : 0.8728
##             Specificity : 0.9683
##          Pos Pred Value : 0.9476
##          Neg Pred Value : 0.9205
##              Prevalence : 0.3965
##          Detection Rate : 0.3461
##    Detection Prevalence : 0.3652
##       Balanced Accuracy : 0.9206
##
##        'Positive' Class : 1
##
```

3.c

```r
# Plot MeanDecreaseAccuracy
varImpPlot(model$finalModel, type = 1)
```




```r
# Plot MeanDecreaseGini
varImpPlot(model$finalModel, type = 2)
```



3.d

```r
varImp(model, type = 1)
```

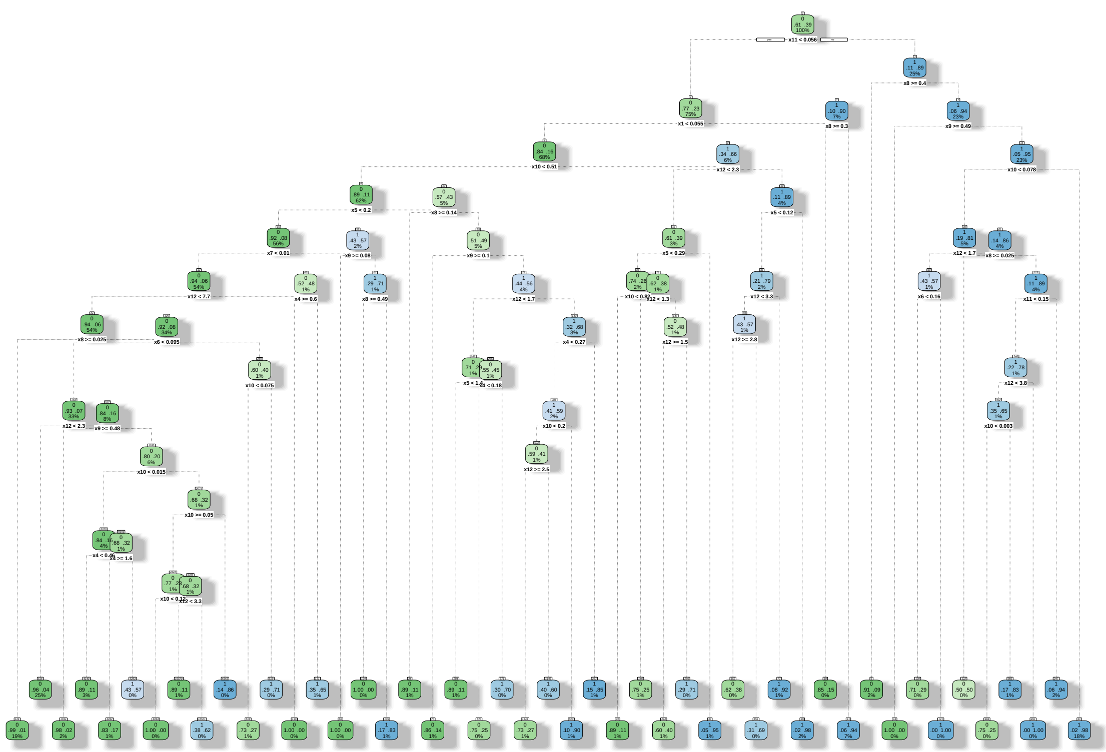
```
## rf variable importance
##
##      Overall
## x1   100.000
## x10   83.176
## x11   76.963
## x8    76.885
## x12   75.171
## x5    53.517
## x9    37.240
## x7    15.124
## x6    10.935
## x4     8.283
## x2     5.527
## x3     0.000
```

Question 4

4.a

```r
model <- rpart(y ~., data = train.data, control = rpart.control(cp=0))
par(xpd = NA)
fancyRpartPlot(model)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2021−May−11 09:21:14 prashundey

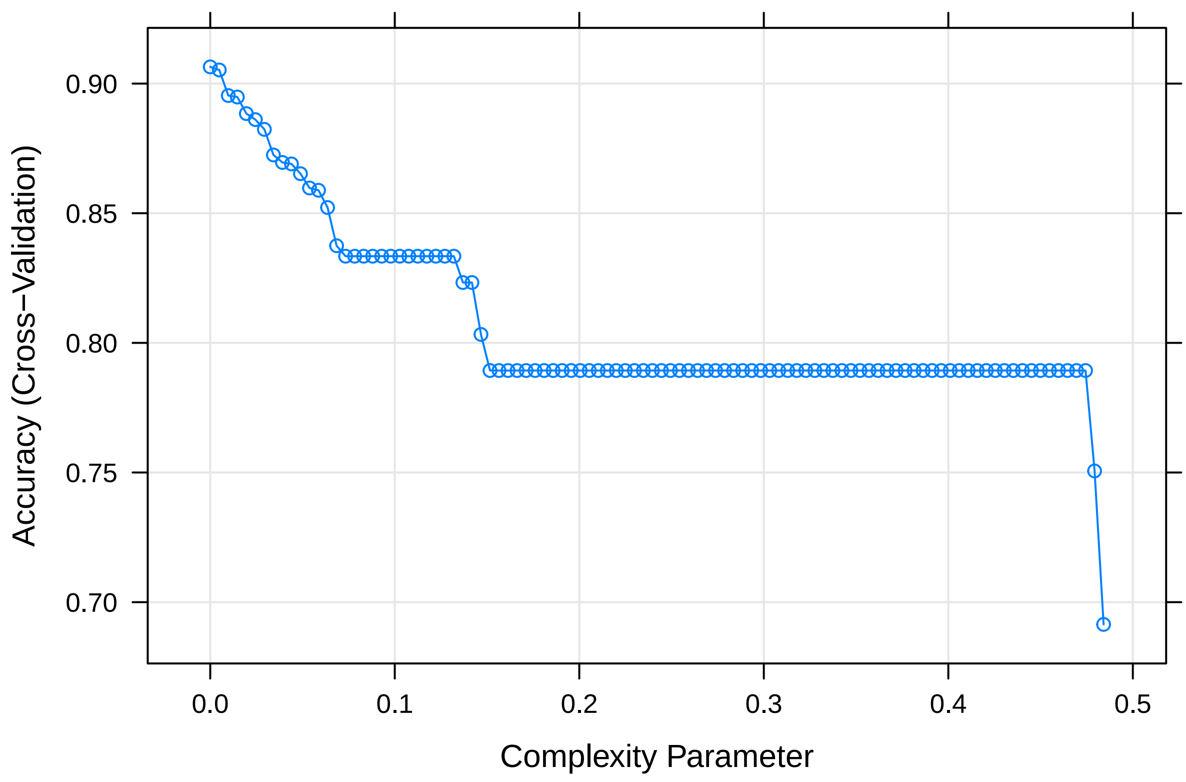
```r
pred_full <- predict(model, newdata = test.data, type ='class')
confusionMatrix(pred_full, test.data$y, positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 657  54
##          1  37 402
##
##                Accuracy : 0.9209
##                  95% CI : (0.9037, 0.9358)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : < 2e-16
##
```

```
##                  Kappa : 0.8336
##
##  Mcnemar's Test P-Value : 0.09349
##
##             Sensitivity : 0.8816
##             Specificity : 0.9467
##          Pos Pred Value : 0.9157
##          Neg Pred Value : 0.9241
##              Prevalence : 0.3965
##          Detection Rate : 0.3496
##    Detection Prevalence : 0.3817
##       Balanced Accuracy : 0.9141
##
##        'Positive' Class : 1
##
```

4.b

```r
set.seed(123)
model2 <- train(
  y ~., data = train.data, method = "rpart",
  trControl = trainControl("cv", number = 10),
  tuneLength = 100)
plot(model2)
```



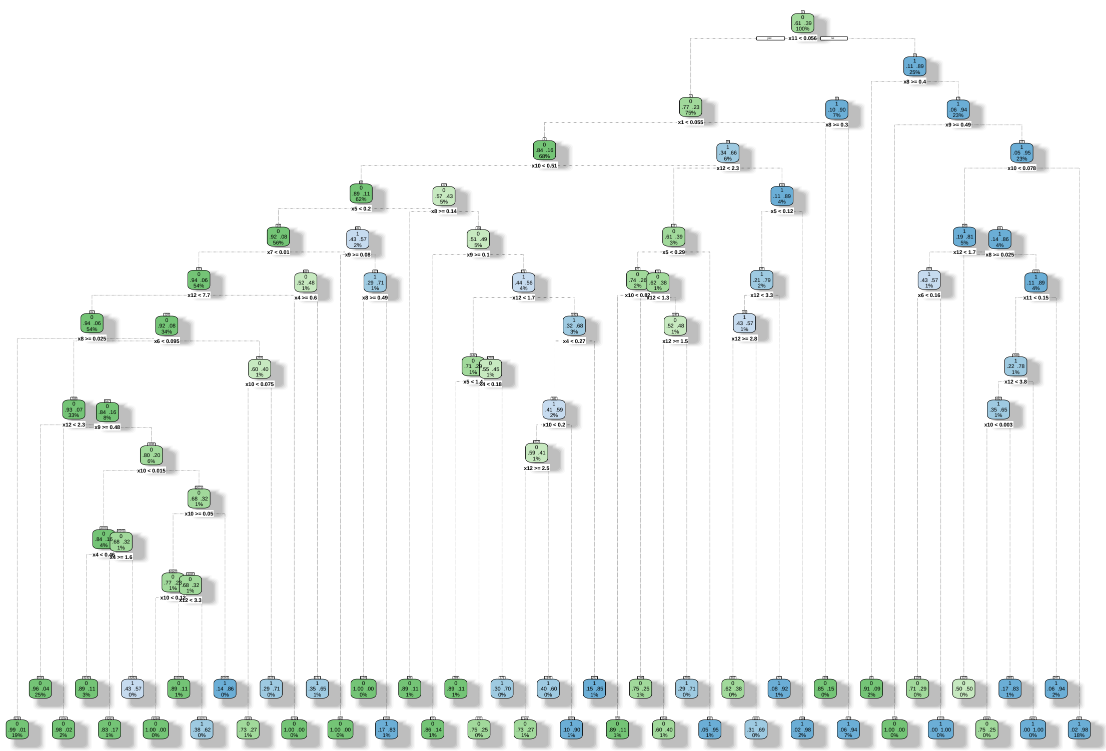
```r
model2$bestTune
```

```
##   cp
## 1  0
```

The best pruned tree is just the fully grown tree.

```r
fancyRpartPlot(model2$finalModel)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2021−May−11 09:21:21 prashundey

4.c

```r
pred_prune <- predict(model2, newdata = test.data)
confusionMatrix(pred_prune, test.data$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 657  54
```

```
##          1  37 402
##
##                Accuracy : 0.9209
##                  95% CI : (0.9037, 0.9358)
##     No Information Rate : 0.6035
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8336
##
##  Mcnemar's Test P-Value : 0.09349
##
##             Sensitivity : 0.9467
##             Specificity : 0.8816
##          Pos Pred Value : 0.9241
##          Neg Pred Value : 0.9157
##              Prevalence : 0.6035
##          Detection Rate : 0.5713
##    Detection Prevalence : 0.6183
##       Balanced Accuracy : 0.9141
##
##        'Positive' Class : 0
##
```

Question 5

5.a

```r
# if the pruned tree is different with fully grown tree, it should not be 1
mean(pred_full == pred_prune)
```

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

5.b

```r
(402 + 657) / (402 + 657 + 37 + 54)
```

```
## [1] 0.9208696
```

Both classification methods, resulted in the same percentage because they were the same tree. 92.09% accuracy.

Part 2

Question 1

```r
data2 <- read.csv('Mystery.csv')
cat('There are', nrow(data2) - nrow(na.omit(data2)), 'missing values in the dataset.')
```

```
## There are 0 missing values in the dataset.
```

```r
data2 <- na.omit(data2)
```

```r
set.seed(123)
training.samples <- data2$y %>% createDataPartition(p = 0.75, list = FALSE)
train.data  <- data2[training.samples, ]
test.data <- data2[-training.samples, ]
```

Question 2

2.a

```r
x <- model.matrix(y ~., train.data)[,-1]
y <- train.data$y
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 0)
cv$lambda.min
```

```
## [1] 64.52856
```

The best lambda for ridge regression is 64.52856

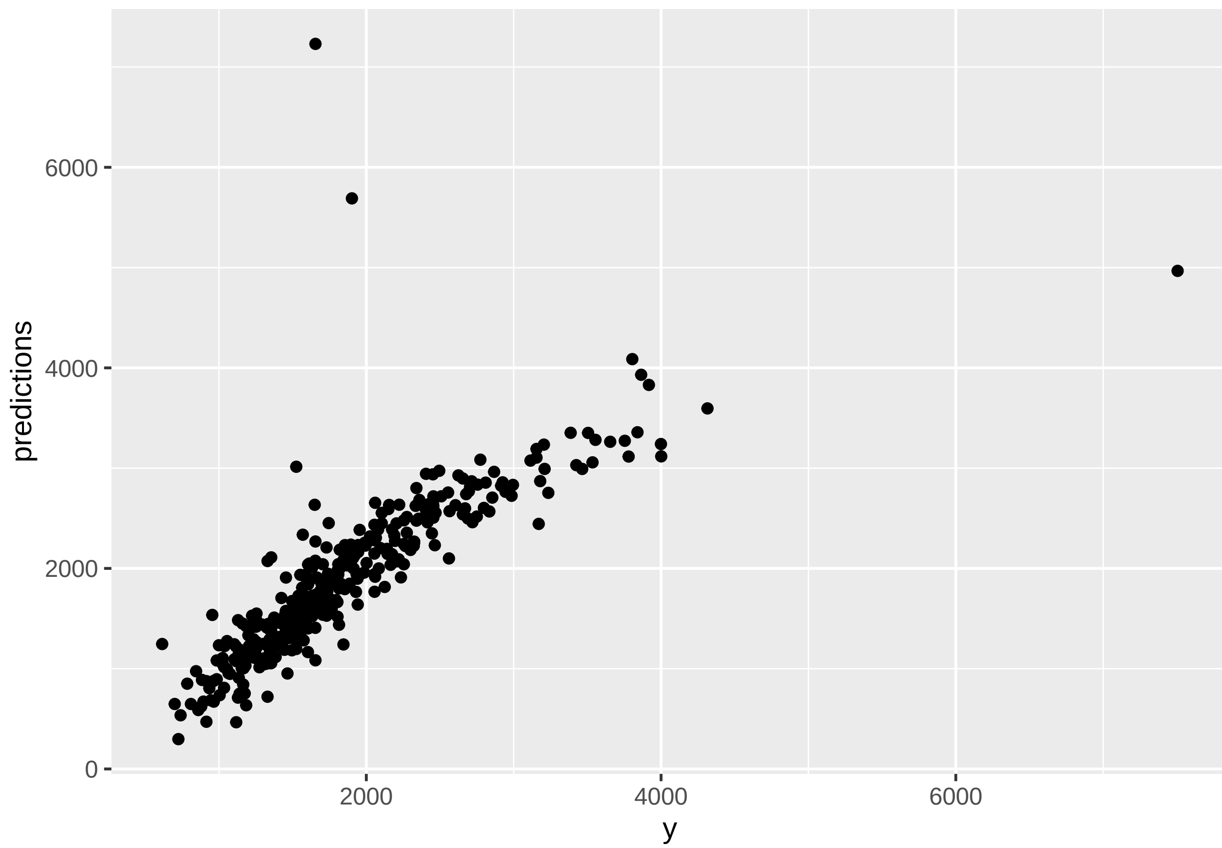
```r
model <- glmnet(x, y, alpha = 0, lambda = cv$lambda.min)
coef(model)
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -413.9886502
## x1            -4.6964088
## x2             0.6864429
## x3           160.5289196
## x4            52.3290629
## x5             0.5142872
## x6             0.1781592
## x7             0.3941634
## x8           -12.5414328
## x9          -117.6681106
## x10         -301.2609710
## x11           44.4442926
## x12           42.4704369
## x13           27.1686916
## x14            0.3509977
```

```r
x.test <- model.matrix(y ~., test.data)[,-1]
predictions <- model %>% predict(x.test) %>% as.vector()
data.frame(
  RMSE = RMSE(predictions, test.data$y),
  Rsquare = R2(predictions, test.data$y)
)
```

```
##       RMSE   Rsquare
## 1 467.3119 0.6687667
```

```r
ggplot(data = test.data, aes(x = y, y = predictions)) + geom_point()
```



2.b

```r
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 1)
cv$lambda.min
```

```
## [1] 3.211601
```

best lambda for LASSO is 3.211601

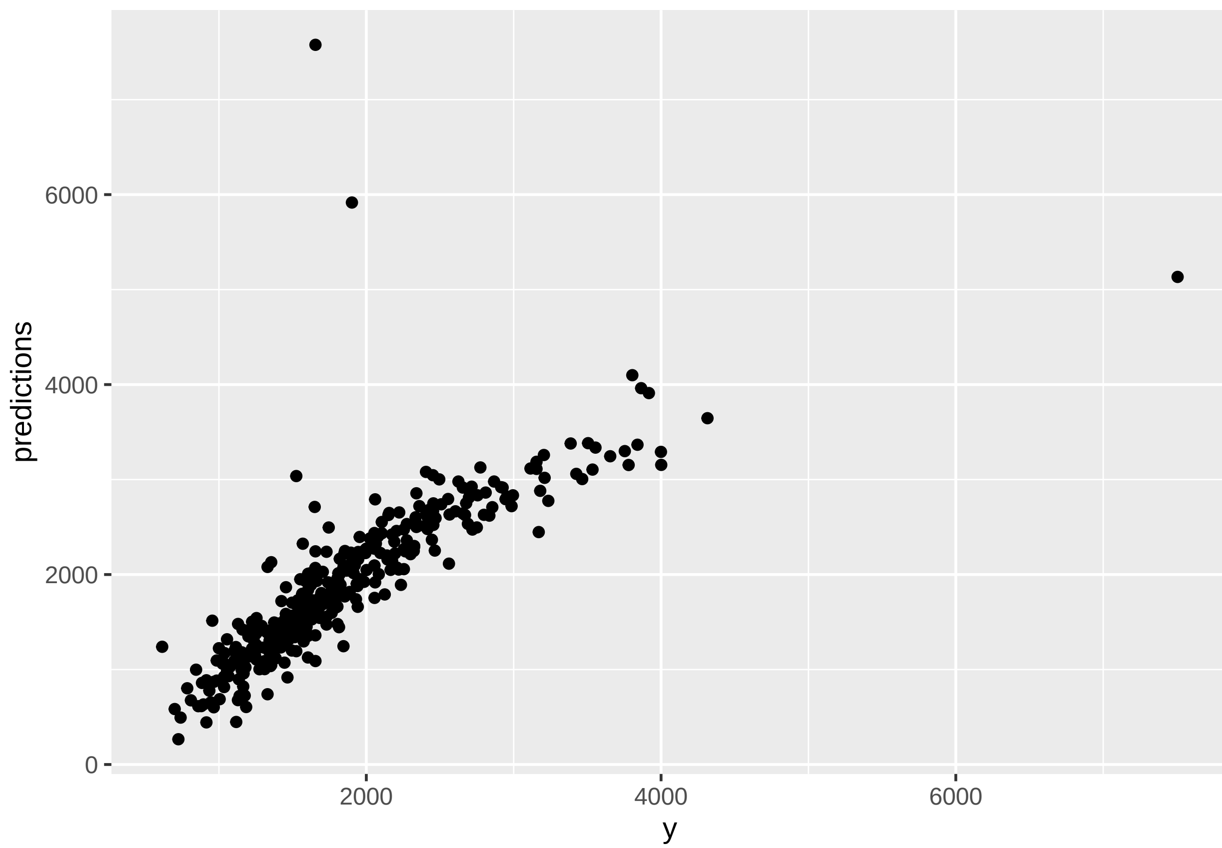
```r
model <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)
coef(model)
```

```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -448.9186214
## x1            -5.1787954
## x2             0.6924462
## x3           170.1536077
## x4            56.7784937
## x5             0.3590211
## x6             .
## x7             0.6535211
## x8           -38.3496997
## x9          -128.6100861
## x10         -279.6002558
## x11           32.8964866
## x12           22.0024680
## x13            .
## x14            0.3581317
```

```r
x.test <- model.matrix(y ~., test.data)[,-1]
predictions <- model %>% predict(x.test) %>% as.vector()
data.frame(
  RMSE = RMSE(predictions, test.data$y),
  Rsquare = R2(predictions, test.data$y)
)
```

```
##      RMSE   Rsquare
## 1 485.264 0.6601552
```

```r
ggplot(data = test.data, aes(x = y, y = predictions)) + geom_point()
```



2.c

```r
set.seed(123)
model <- train(
 y ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
model$bestTune
```

```
##    alpha   lambda
## 15   0.2 8.491095
```

```r
coef(model$finalModel, model$bestTune$lambda)
```

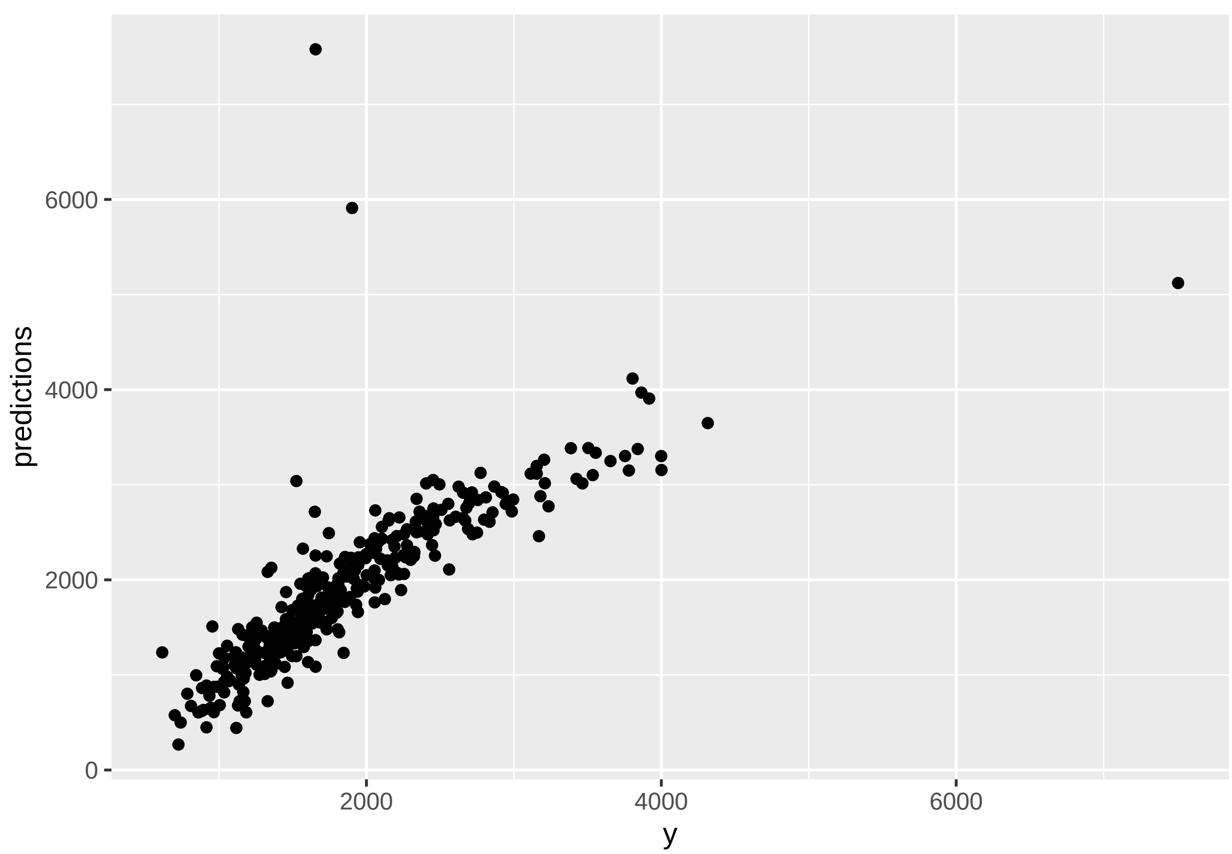
```
## 15 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) -442.0745573
## x1            -5.1803023
## x2             0.7001039
## x3           168.5663124
## x4            57.4930484
## x5             0.5108956
```

```
## x6             0.1547817
## x7             0.4931062
## x8           -42.2043070
## x9          -132.1914470
## x10         -288.9940694
## x11           37.8783780
## x12           23.9553465
## x13            .
## x14            0.3652305
```

```
x.test <- subset(test.data, select = -y)
predictions <- model %>% predict(x.test)
# model$fianlModel %>% predict(x.test,model$bestTune$lambda)
data.frame(
  RMSE = RMSE(predictions, test.data$y),
  Rsquare = R2(predictions, test.data$y)
)
```

```
##       RMSE   Rsquare
## 1 484.8419 0.6605653
```

```
ggplot(data = test.data, aes(x = y, y = predictions)) + geom_point()
```



2.d

```r
# to make sure to get same result, we alwasy set lambda range like this
lambda <- 10^seq(-3, 3, length = 100)
```

```r
set.seed(123)
ridge <- train(
  y ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 0, lambda = lambda)
  )
```

```r
set.seed(123)
lasso <- train(
  y ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda)
  )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```r
set.seed(123)
elastic <- train(
  y ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
```

```r
models <- list(ridge = ridge, lasso = lasso, elastic = elastic)
resamples(models) %>% summary( metric = "RMSE")
```

```
##
## Call:
## summary.resamples(object = ., metric = "RMSE")
##
## Models: ridge, lasso, elastic
## Number of resamples: 10
##
## RMSE
##             Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ridge   242.3666 296.3807 309.2699 329.9067 372.5931 449.5673    0
## lasso   237.8107 297.2884 312.0110 328.7980 377.5471 442.3513    0
## elastic 239.0043 298.3003 311.3291 328.7150 376.8959 442.2240    0
```

Elastic model is the best with lowest mean and second lowest median of RMSE.

Question 3

3.a

```
models <- regsubsets(y~., data = train.data, nvmax = 5)
summary(models)
```

```
## Subset selection object
## Call: regsubsets.formula(y ~ ., data = train.data, nvmax = 5)
## 14 Variables  (and intercept)
##     Forced in Forced out
## x1      FALSE      FALSE
## x2      FALSE      FALSE
## x3      FALSE      FALSE
## x4      FALSE      FALSE
## x5      FALSE      FALSE
## x6      FALSE      FALSE
## x7      FALSE      FALSE
## x8      FALSE      FALSE
## x9      FALSE      FALSE
## x10     FALSE      FALSE
## x11     FALSE      FALSE
## x12     FALSE      FALSE
## x13     FALSE      FALSE
## x14     FALSE      FALSE
## 1 subsets of each size up to 5
## Selection Algorithm: exhaustive
##           x1  x2  x3  x4  x5  x6  x7  x8  x9  x10 x11 x12 x13 x14
## 1  ( 1 ) " " " " "*" " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 ) " " " " "*" " " " " " " "*" " " " " " " " " " " " " " "
## 3  ( 1 ) " " " " "*" " " "*" "*" " " " " " " " " " " " " " " " "
## 4  ( 1 ) "*" " " "*" " " "*" " " "*" " " " " " " " " " " " " " "
## 5  ( 1 ) "*" " " "*" " " "*" " " "*" " " "*" " " " " " " " " " "
```

```
for(i in 1:5){
  cat('The best model with', i, 'variable(s) is:\n')
  predictors <- names(which(summary(models)$which[i,-1] == TRUE))
  predictors <- paste(predictors, collapse = "+")
  cat('y ~' , predictors, '\n')
}
```

```
## The best model with 1 variable(s) is:
## y ~ x3
## The best model with 2 variable(s) is:
## y ~ x3+x7
## The best model with 3 variable(s) is:
## y ~ x3+x5+x6
## The best model with 4 variable(s) is:
## y ~ x1+x3+x5+x7
## The best model with 5 variable(s) is:
## y ~ x1+x3+x5+x7+x9
```

3.b

```r
get_model_formula <- function(id, object, outcome){
  # get models data
  models <- summary(object)$which[id,-1]
  # Get outcome variable
  #form <- as.formula(object$call[[2]])
  #outcome <- all.vars(form)[1]
  # Get model predictors
  predictors <- names(which(models == TRUE))
  predictors <- paste(predictors, collapse = "+")
  # Build model formula
  as.formula(paste0(outcome, "~", predictors))
}
```

```r
get_cv_error <- function(model.formula, data){
  set.seed(123)
  train.control <- trainControl(method = "cv", number = 5)
  cv <- train(model.formula, data = data, method = "lm",
              trControl = train.control)
  cv$results$RMSE
}
```

```r
model.ids <- 1:5
cv.errors <-  map(model.ids, get_model_formula, models, "y") %>%
  map(get_cv_error, data = train.data) %>%
  unlist()
cv.errors
```

```
## [1] 488.6770 413.4632 383.5591 367.4798 357.8093
```

The overall best model is the model with 5 variables:
y ~ x1+x3+x5+x7+x9
This model has the lowest error.

3.c

```r
res.lm <- lm(y ~., data = train.data)
step <- stepAIC(res.lm, direction = "both", trace = FALSE)
step
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3 + x4 + x5 + x6 + x8 + x9 + x10 +
##     x11 + x14, data = train.data)
##
## Coefficients:
## (Intercept)           x1           x2           x3           x4           x5
##   -446.2585      -5.1991       0.7301     170.0607      59.6451       1.0407
##          x6           x8           x9          x10          x11          x14
##      0.6782     -62.0522    -143.7630    -309.2414      43.2446       0.3561
```

The best model with step wise regression is:
y ~ x1 + x2 + x3 + x4 + x5 + x6 + x8 + x9 + x10 + x11 + x14

3.d

```r
# add step wise and best subset regression into the model list
best_sub <- lm(y ~ x1+x3+x5+x7+x9, data = train.data)
models <- list(ridge = ridge, lasso = lasso, elastic = elastic, best_sub = best_sub, step = step)
# Here asked to use testing data and resamples funciton can only be applied on train result
# so we need use another way to compare them.
lapply(models %>% predict(test.data), RMSE, test.data$y)
```

```
## $ridge
## [1] 467.2919
##
## $lasso
## [1] 485.8628
##
## $elastic
## [1] 484.8419
##
## $best_sub
## [1] 496.1158
##
## $step
## [1] 491.658
```

The best model is ridge regression with lowest RMSE.