Homework 8

STAT 430, Spring 2017

Due: Friday, April 7 by 11:59 PM

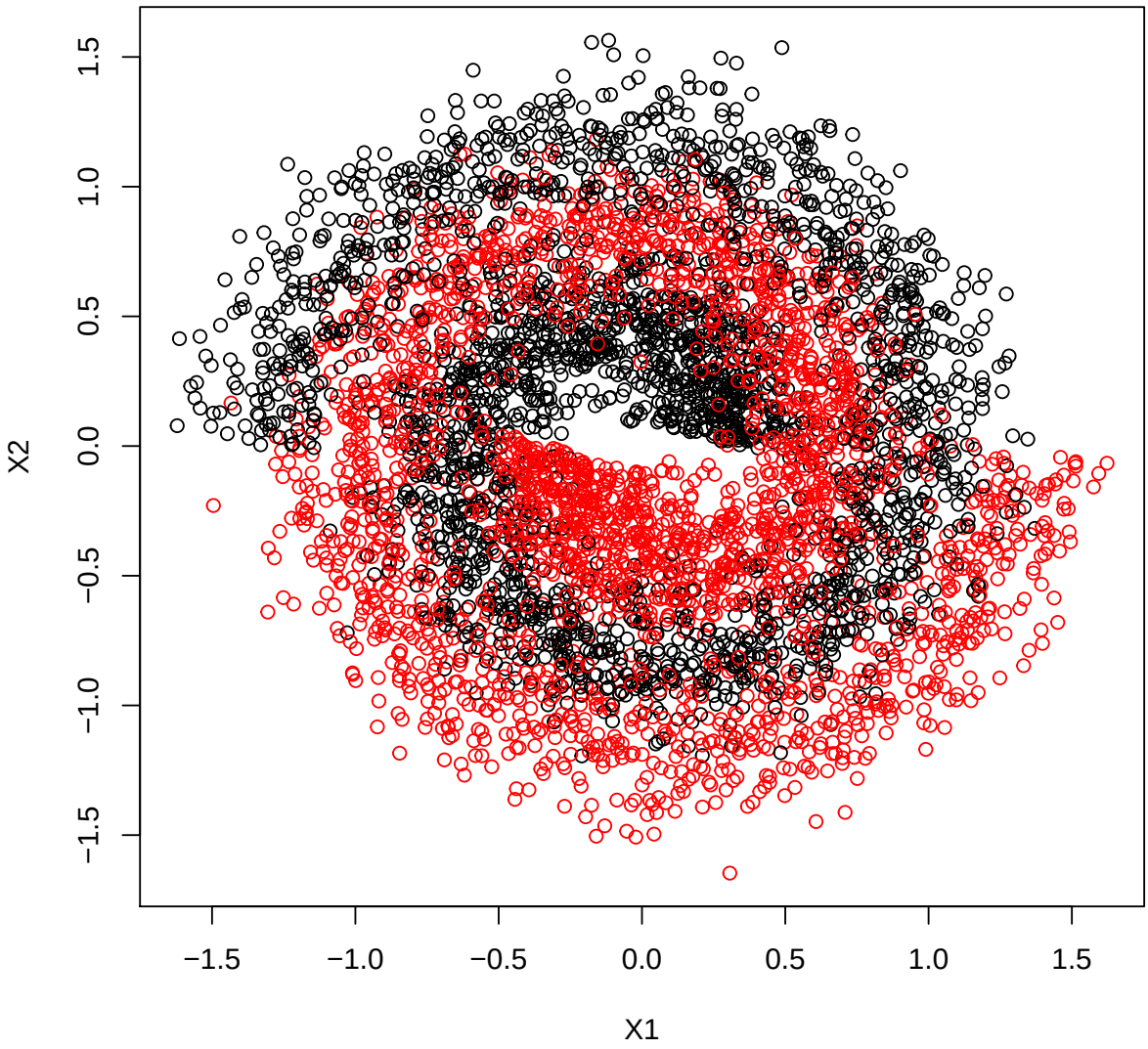
Exercise 1

[12 points] For this exercise we will create data via simulation, then asses how well certain methods perform. Use the code below to create a train and test dataset.

```
library(mlbench)
set.seed(42)
sim_trn = mlbench.spirals(n = 5000, cycles = 1.5, sd = 0.15)
sim_trn = data.frame(sim_trn$x, class = as.factor(sim_trn$classes))
sim_tst = mlbench.spirals(n = 10000, cycles = 1.5, sd = 0.15)
sim_tst = data.frame(sim_tst$x, class = as.factor(sim_tst$classes))
```

The training data is plotted below, with colors indicating the `class` variable, which is the response.

```
plot(sim_trn$X1, sim_trn$X2, col = sim_trn$class,
     xlab = "X1", ylab = "X2")
```



Before proceeding further, set a seed equal to your UIN.

```
uin = 123456789
set.seed(uin)
```

We'll use the following to define 5-fold cross-validation for use with `train()` from `caret`.

```
library(caret)
cv_5 = trainControl(method = "cv", number = 5)
```

We now tune two models with `train()`. First, a logistic regression using `glm`. (This actually isn't "tuned" as there are not parameters to be tuned, but we use `train()` to perform cross-validation.) Second we tune a single decision tree using `rpart`.

We store the results in `sim_glm_cv` and `sim_tree_cv` respectively, but we also wrap both function calls with `system.time()` in order to record how long the tuning process takes for each method.

```
glm_cv_time = system.time({
  sim_glm_cv  = train(
    class ~ .,
    data = sim_trn,
    trControl = cv_5,
    method = "glm")
})

tree_cv_time = system.time({
  sim_tree_cv = train(
    class ~ .,
    data = sim_trn,
    trControl = cv_5,
    method = "rpart")
})
```

We see that both methods are tuned via cross-validation in a similar amount of time.

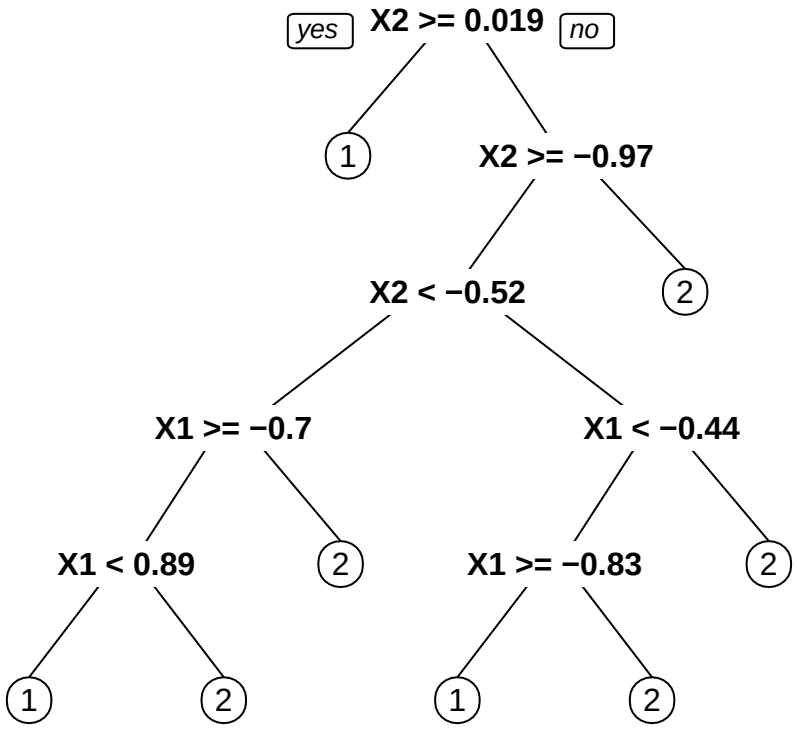
```
glm_cv_time["elapsed"]
```

```
## elapsed
##    0.95
```

```
tree_cv_time["elapsed"]
```

```
## elapsed
##    0.56
```

```
library(rpart.plot)
prp(sim_tree_cv$finalModel)
```



Repeat the above analysis using a random forest, twice. The first time use 5-fold cross-validation. The second time, tune the model using OOB samples. We only have two predictors here, so, for both, use the following tuning grid.

```
rf_grid = expand.grid(mtry = c(1, 2))
```

```
oob  = trainControl(method = "oob")
rf_oob_time = system.time({
  sim_rf_oob = train(
    class ~ .,
    data = sim_trn,
    trControl = oob,
    tuneGrid = rf_grid)
})

rf_cv_time = system.time({
  sim_rf_cv = train(
    class ~ .,
    data = sim_trn,
    trControl = cv_5,
    tuneGrid = rf_grid)
})
```

(a) Compare the time taken to tune each model. Is the difference between the OOB and CV result for the random forest similar to what you would have expected?

Solution:

```
## Logistic, CV     Tree, CV      RF, OOB       RF, CV
##         0.95         0.56         4.55        12.01
```

```
rf_cv_time["elapsed"] / rf_oob_time["elapsed"]
```

```
## elapsed
## 2.63956
```

The speed-up for OOB is only about three times that of 5-fold CV, instead of the five times that would have been expected. There appears to be some additional overhead in using OOB.

(b) Compare the tuned value of `mtry` for each of the random forests tuned. Do they choose the same model?

Solution:

```
sim_rf_oob$bestTune
```

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

```
sim_rf_cv$bestTune
```

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

They choose the same model, although, there were only two to choose from, and they are not very different. In practice, the two methods may differ more.

(c) Report the CV and OOB accuracy for the random forests.

Solution:

```
## RF, OOB  RF, CV
##  0.7720  0.7746
```

Note that, in this case they are extremely similar. However, in practice they may differ more.

(d) Compare the test accuracy of each of the four procedures considered. Briefly explain these results.

Solution:

```
accuracy = function(actual, predicted) {
  mean(actual == predicted)
}
```

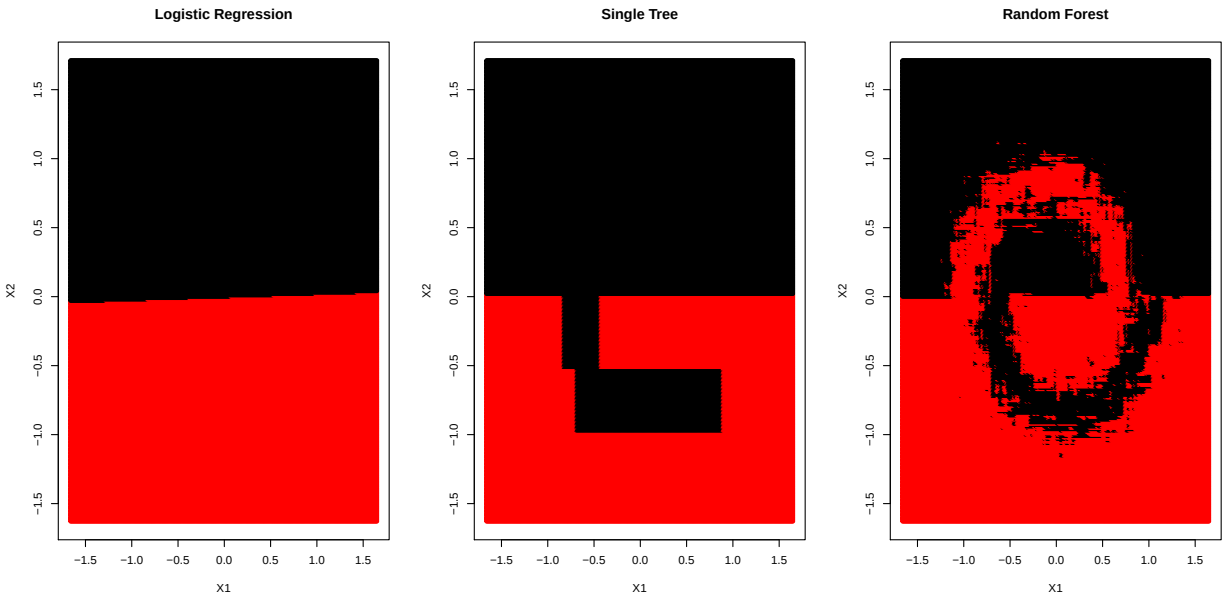
```
glm_cv_tst_acc = accuracy(
  predicted = predict(sim_glm_cv, sim_tst),
  actual    = sim_tst$class
)

tree_cv_tst_acc = accuracy(
  predicted = predict(sim_tree_cv, sim_tst),
  actual    = sim_tst$class
)

rf_cv_tst_acc = accuracy(
  predicted = predict(sim_rf_cv, sim_tst),
  actual    = sim_tst$class
)

rf_oob_tst_acc = accuracy(
  predicted = predict(sim_rf_oob, sim_tst),
```

```
  actual    = sim_tst$class
)
```

```
## Logistic, CV     Tree, CV      RF, OOB       RF, CV
##       0.6614       0.7021       0.7791       0.7780
```

- Logistic: Performs the worst. This is expected as clearly a non-linear decision boundary is needed.
- Single Tree: Better than logistic, but not the best seen here. We see above that this is not a very deep tree. It will have non-linear boundaries, but since it uses binary splits, they will be rectangular regions.
- Random Forest: First note that both essentially fit the same model. (The exact forests will be different due to randomization.) By using many trees (500) the boundaries will become less rectangular than the single tree, and will better match the spiral data in the data.
- See below for plots of decision boundaries created by making predictions from the different models.

```
plot_grid = expand.grid(
  X1 = seq(min(sim_tst$X1), max(sim_tst$X1), by = 0.01),
  X2 = seq(min(sim_tst$X2), max(sim_tst$X2), by = 0.01)
)

glm_pred  = predict(sim_glm_cv, plot_grid)
tree_pred = predict(sim_tree_cv, plot_grid)
rf_pred   = predict(sim_rf_oob, plot_grid)

par(mfrow = c(1, 3))
plot(plot_grid$X1, plot_grid$X2, col = glm_pred,
     xlab = "X1", ylab = "X2", pch = 20, main = "Logistic Regression")
plot(plot_grid$X1, plot_grid$X2, col = tree_pred,
     xlab = "X1", ylab = "X2", pch = 20, main = "Single Tree")
plot(plot_grid$X1, plot_grid$X2, col = rf_pred,
     xlab = "X1", ylab = "X2", pch = 20, main = "Random Forest")
```



Exercise 2

[**12 points**] For this question we will predict the `Salary` of `Hitters`. (`Hitters` is also the name of the dataset.) We first remove the missing data:

```
library(ISLR)
Hitters = na.omit(Hitters)
```

After changing `uin` to your UIN, use the following code to test-train split the data.

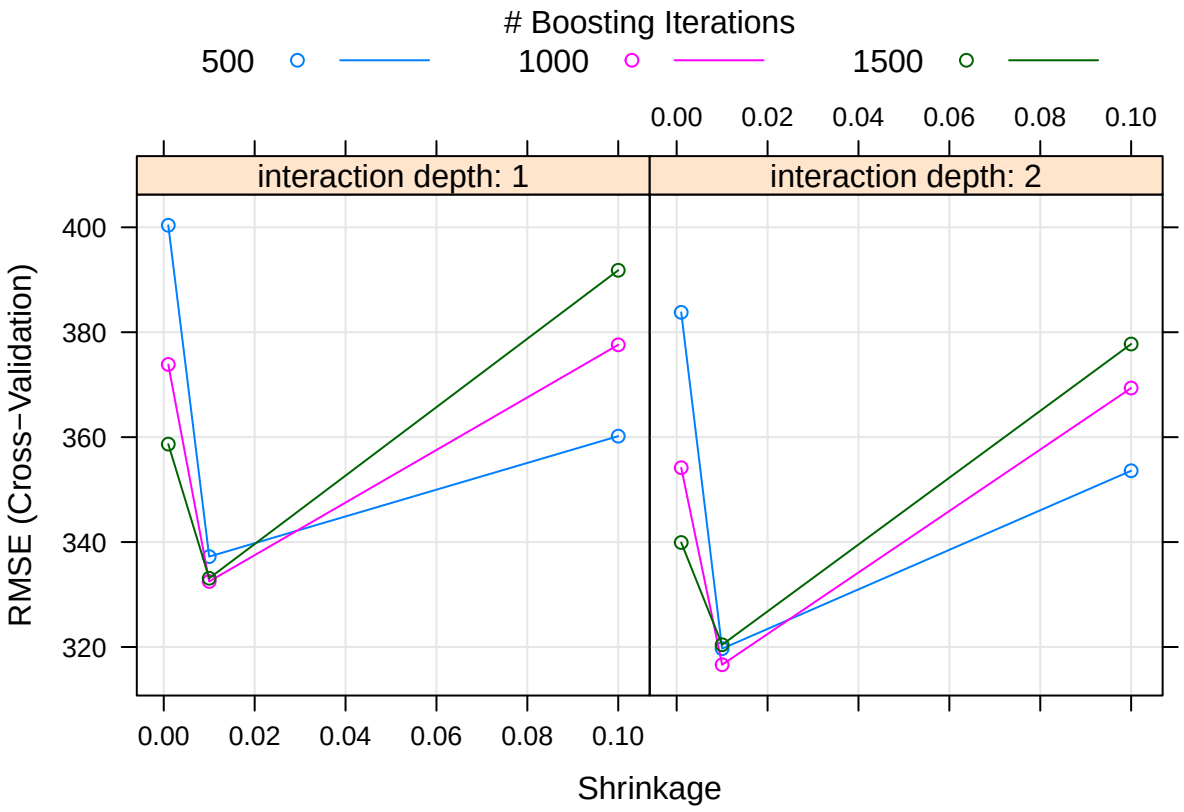
```
uin = 123456789
set.seed(uin)
hit_idx = createDataPartition(Hitters$Salary, p = 0.6, list = FALSE)
hit_trn = Hitters[hit_idx,]
hit_tst = Hitters[-hit_idx,]
```

(a) Tune a boosted tree model using the following tuning grid and 5-fold cross-validation. Create a plot that shows the tuning results.

```
gbm_grid = expand.grid(interaction.depth = c(1, 2),
                       n.trees = c(500, 1000, 1500),
                       shrinkage = c(0.001, 0.01, 0.1),
                       n.minobsinnode = 10)
```

```
hit_gbm = train(Salary ~ ., data = hit_trn,
                method = "gbm",
                trControl = cv_5,
                verbose = FALSE,
                tuneGrid = gbm_grid)
```

```
plot(hit_gbm)
```



(b) What is the CV-RMSE and tuning parameters of the tuned boosted tree model?

```
hit_gbm$bestTune
```

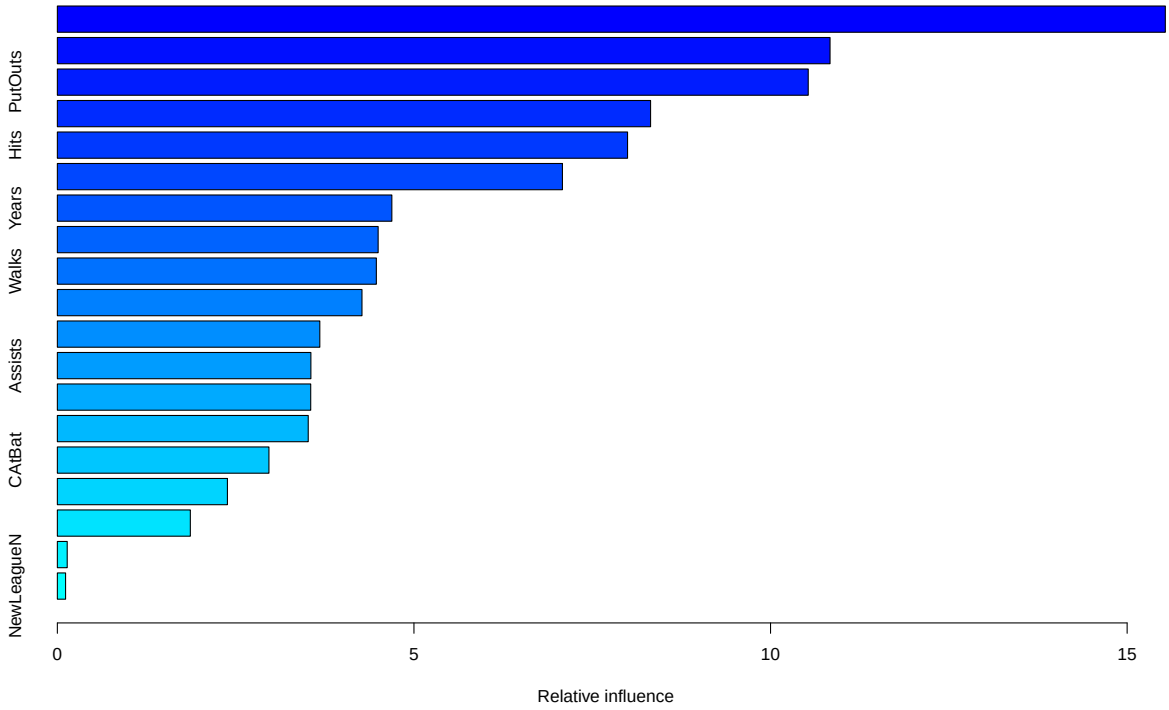
```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 11    1000                 2      0.01             10
```

```
min(hit_gbm$results$RMSE)
```

```
## [1] 316.6123
```

(c) According to the boosted model, what are the two most important predictors?

```
hit_var_imp = summary(hit_gbm$final)
```




```
head(hit_var_imp)
```

```
##             var   rel.inf
## CHmRun   CHmRun 15.535481
## CRuns     CRuns 10.834235
## PutOuts PutOuts 10.528464
## RBI         RBI  8.317985
## Hits       Hits  7.995127
## CRBI       CRBI  7.082149
```

Here we see the two most important variables are CHmRun and CRuns.

(d) Tune a random forest using OOB resampling and **all** possible values of `mtry`. Report the best value of `mtry` as well as the OOB RMSE for both the best model as well as the bagged model. (It is possible, but unlikely that they are the same. If they are the same, simply report that the best model is the bagged model and report only that model.)

```
rf_grid = rf_grid = expand.grid(mtry = 1:(ncol(hit_trn) - 1))
hit_rf  = train(Salary ~ ., data = hit_trn,
                method = "rf",
                trControl = oob,
                tuneGrid = rf_grid)
```

```
hit_rf$bestTune
```

```
##   mtry
## 5    5
```

```
# best mtry
hit_rf$results[hit_rf$bestTune$mtry, ]$RMSE
```

```
## [1] 296.6724
```

```
# bagged
hit_rf$results[(ncol(hit_trn) - 1), ]$RMSE
```

```
## [1] 302.1073
```

(e) Report the test RMSE for the tuned boosted tree model, the tuned random forest, and a bagged tree model.

```
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
```

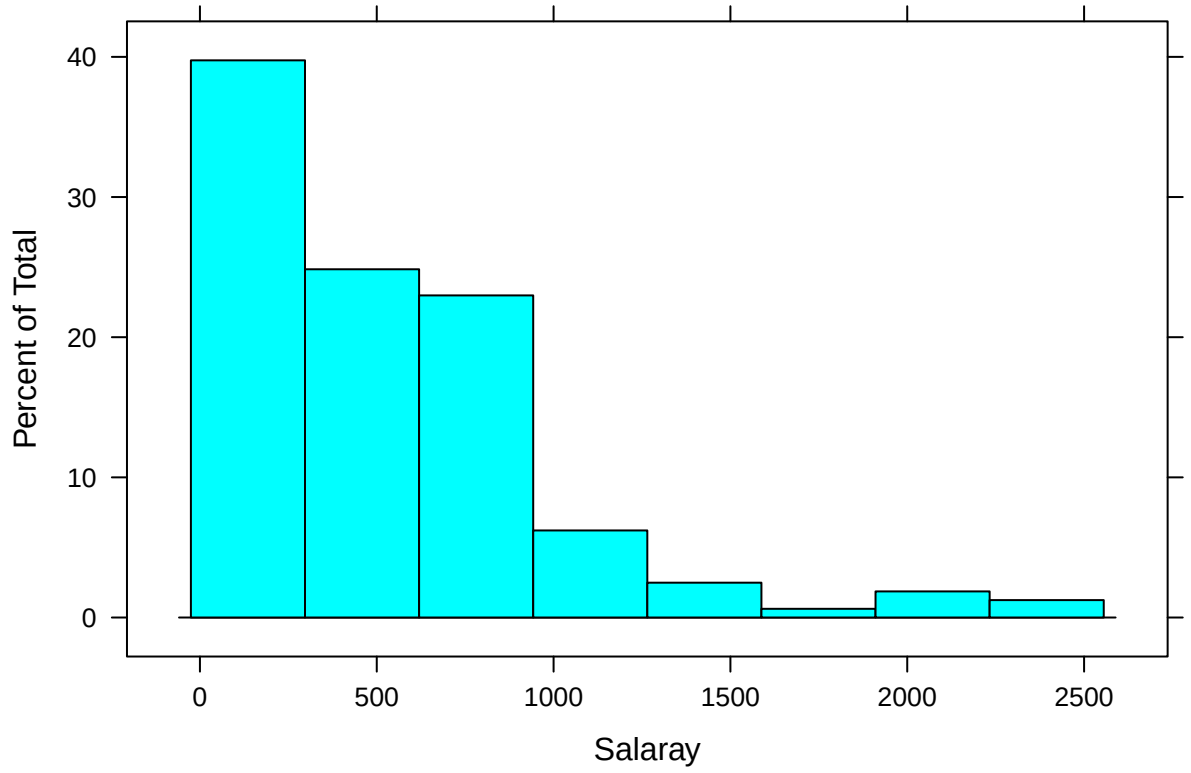
```
gbm_tst_rmse = rmse(
  predicted = predict(hit_gbm, hit_tst),
  actual    = hit_tst$Salary
)

rf_tst_rmse = rmse(
  predicted = predict(hit_rf, hit_tst),
  actual    = hit_tst$Salary
)

# storing the bagged model for making predictions
hit_bag = train(Salary ~ ., data = hit_trn,
                method = "rf",
                trControl = oob,
                tuneGrid = data.frame(mtry = (ncol(hit_trn) - 1)))

bag_tst_rmse = rmse(
  predicted = predict(hit_bag, hit_tst),
  actual    = hit_tst$Salary
)
```

```
##      gbm       rf      bag
## 258.5069 258.4821 280.0406
```

Exercise 3

[6 points] Continue with the data from Exercise 2. The book, ISL, suggests log transforming the response, `Salary`, before fitting a random forest. Is this necessary? Re-tune a random forest as you did in Exercise 2, except with a log transformed response. Report test RMSE for both the untransformed and transformed model. Based on these results, do you think the transformation was necessary?

```
histogram(hit_trn$Salary, xlab = "Salaray")
```




```
hit_rf_log  = train(log(Salary) ~ ., data = hit_trn,
                    method = "rf",
                    trControl = oob,
                    tuneGrid = rf_grid)
```

```
# without transformation
rmse(
  predicted = predict(hit_rf, hit_tst),
  actual = hit_tst$Salary
)
```

```
## [1] 258.4821
```

```
# with log transformation
rmse(
  predicted = exp(predict(hit_rf_log, hit_tst)),
  actual = hit_tst$Salary
)
```

```
## [1] 275.5441
```

Here we see that the untransformed model actually performs better. However, they are relatively close, so either could be acceptable. Note that a random forest can model a non-linear relationship, which is why the transformation is not necessary.