1.

If we have a very correlated group of trees with respect to their predictions, then averaging all of their outcomes won't be beneficial, since variance won't decrease by much. By instead using the random forest, we get more randomness, so our trees will be less correlated. This will result in more independent trees, which will help us increase prediction accuracy. In addition, using this method allows you to more easily and quickly build trees, since we're not looking at all of the predictor variables (only a few that we pick), and there aren't parameters to tune, making it simpler.

Its simplest disadvantage is a part of its advantage – it's simple and doesn't require tuning – this can make it less useful for people who are skilled and could use other methods. Also, it makes it harder to find complicated effects, since it's not likely that two random trees will have the same paths for those effects.

We can add tree variation by instead of choosing random subsets of variables, choosing random subsets of observations. This makes it so that we're getting results from different trees, which would also lower our bias.

2. Regularization methods are required when the number of predictors exceeds the sample size to avoid over-fitting of data onto the model. This is accomplished by introducing a penalty function to the loss function, which is a function of the learned parameters in the regression. This can also be used to perform biased learning by introducing constraints and resolving sparsity. Tends to be a disadvantage if predictors are highly correlated and group selection becomes difficult. Sparsity is generally reasonable to assume as the predictors tend to outnumber the sample size in the context of boosting. However, this is not always the case, as this assumption is only valid if only a relative few predictors are statistically significant but not the case if the underlying system being modeled is more complicated.

3. The derivative of the convex power family members of penalties (except Lasso) at $\vec{a} = 0$ is 0. Thus no penalty/barrier to become non-zero (any increase in penalty causes coefficient change of equal magnitude). However, for Lasso and elastic net penalties, this is not the case and thus the loss function must improve to rectify non-zero derivative at $\vec{a} = 0$, meaning the power family tends to have non-zero values for all coefficients along the path and the elastic net family tends to have zero valued coefficients at multiple path points.

4. We have

$$j^* = \arg \min_{1 \le j \le J} \min_{\rho} E[y - \rho x_j]^2$$

$$= \min_{j,\rho} \left(E[y^2] - 2\rho E[y \cdot x_j] + \rho^2 E[x_j^2] \right)$$

As $E[x_j^2] = 1$, we have $f(\rho) = E[y^2] - 2\rho E[y \cdot x_j] + \rho^2$

Differentiating and setting the LHS to 0,

$$0 = \frac{df}{d\rho} = -2E[y \cdot x_j] + 2\rho \Rightarrow \rho = E[y \cdot x_j]$$

$$\frac{d^2 f}{d\rho^2} = 2 > 0 \quad \therefore \text{minimum by } 2^{\text{nd}} \text{ order condition}$$

$$\therefore j^* = \arg \min_{1 \le j \le J} \left(E[y^2] - (E[y \cdot x_j])^2 \right)$$

$$= \arg \max_{1 \le j \le J} (E[y \cdot x_j])^2$$

$$= \arg \max_{1 \le j \le J} |E[y \cdot x_j]| = j^*$$

5.

From lecture:

$$\bar{F}_i(z_i) = E_{z_{\setminus i}}[F(x)] = \int F(x)\, p_{\setminus i}(z_{\setminus i})\, dz_{\setminus i}$$

Given $F(x) = F_i(z_i) + F_{\setminus i}(z_{\setminus i})$ where $z_i \cup z_{\setminus i}$

$$\bar{F}_i(x) = E_{z_{\setminus i}}[F(x)] = E_{z_{\setminus i}}\left(F_i(z_i) + F_{\setminus i}(z_{\setminus i})\right)$$

$$= \int \left(F_i(z_i) + F_{\setminus i}(z_{\setminus i})\right) P(z_{\setminus i})\, dz_{\setminus i}$$

$$= \int F_i(z_i)\, P(z_{\setminus i})\, dz_{\setminus i} + \int F_{\setminus i}(z_{\setminus i}) P(z_{\setminus i})\, dz_{\setminus i}$$

$$F(x) = F_i(z_i) + \int F_{\setminus i}(z_{\setminus i})\, P(z_{\setminus i})\, dz_{\setminus i}$$

↑ constant

Thus, $F(x)$ is dependent on $F_i(z_i)$ + constant

Now using $E[F(x \mid z_i)]$

From lecture: $E_x[F(x) \mid z_i] = \int F(x)\, p(x \mid z_i)\, dx$

so: $E[F(x \mid z_i)] = \int \left(F(z_i) + F(z_{\setminus i})\right) p(z_{\setminus i} \mid z_i)\, dz_{\setminus i}$

$$= \int F(z_i)\, p(z_{\setminus i} \mid z_i)\, dz_{\setminus i} + \int F(z_{\setminus i})\, p(z_{\setminus i} \mid z_i)\, dz_{\setminus i}$$

if z_i and $z_{\setminus i}$ are dependent, then $p(z_{\setminus i} \mid z_i)$ is not constant always

So $F(x)$ is not dependent on $F_i(z_i)$ + constant

if they're completely independent, then $p(z_{\setminus i} \mid z_i) = p(z_{\setminus i})$, and we get

$$= \int F(z_i)\, p(z_{\setminus i})\, dz_{\setminus i} + \int F(z_{\setminus i})\, p(z_{\setminus i} \mid z_i)\, dz_{\setminus i}$$

↑ this is exactly the same as before, so only when z_i and $z_{\setminus i}$ are independent, then the two are the same.

6. Binary classification; Spam Email.

a. I fit a gbm model using the tutorial. The only main difference is that I did not need to randomize the rows, since the test and training set were already randomly made for us. In addition, I found the optimal amount of iterations using cross validation.

```
spam_all <- read.csv("Spam_Data.txt", header = FALSE)
spam_train <- read.csv("Spam_Train.txt", header = FALSE)
spam_test <- read.csv("Spam.Test.txt", header = FALSE)

rflabs<-c("make", "address", "all", "3d", "our", "over", "remove",
          "internet","order", "mail", "receive", "will",
          "people", "report", "addresses","free", "business",
          "email", "you", "credit", "your", "font","000","money",
          "hp", "hpl", "george", "650", "lab", "labs",
          "telnet", "857", "data", "415", "85", "technology", "1999",
          "parts","pm", "direct", "cs", "meeting", "original",
"project",
          "re","edu", "table", "conference", ";", "(", "[", "!", "$",
"#",
          "CAPAVE", "CAPMAX", "CAPTOT","type")

colnames(spam_all) <- rflabs
colnames(spam_train) <- rflabs
colnames(spam_test) <- rflabs

set.seed(131)

# Not necessary because train and test samples are already randomly
selected.
spam_all_r<-spam_all[sample(nrow(spam_all)),]
spam_train_r<-spam_train[sample(nrow(spam_train)),]
spam_test_r<-spam_test[sample(nrow(spam_test)),]

set.seed(444)

# Make model and look for optimal number of iterations
gbm0 <- gbm(type~., data=spam_train, train.fraction=1.0,
interaction.depth=4, shrinkage=.05, n.trees=2500, bag.fraction=0.5,
cv.folds=5, distribution="bernoulli", verbose=F)
gbm0_iterations = gbm.perf(gbm0, method="cv")
```

I then used it to predict on the test set and generated a table to see where we matched the predictions and test data.

```
gbm0_predict <- predict(gbm0, spam_test, type="response",
n.trees=gbm0_iterations)
# Make a set of 1534 0's.
gbm0_predicted = rep(0, nrow(spam_test))
# Set to 1 if predict has a value over 0.5.
gbm0_predicted[gbm0_predict>0.5] = 1

pred_actual = table(gbm0_predicted, spam_test$type)
```

```
pred_actual
spam_wrong = pred_actual[1,2]/(sum(pred_actual[,2]))
not_spam_wrong = pred_actual[2,1]/(sum(pred_actual[,1]))
spam_wrong
not_spam_wrong
```

The table that we got (where the rows are predicted and the columns are test) is shown below.

```
gbm0_predicted   0   1
             0 889  33
             1  27 585
```

That means our overall misclassification score is 3.9%.
For the spam emails (column for 1), 5.3% were misclassified.
For the non-spam emails (column for 0), 2.9% were misclassified.

b. For this, we used the same process as before, but we also use the weights parameter that gbm supplies. Through testing, we found a weight of 1:25 (where 25 is spam email) makes our model have ~0.3% misclassification of non-spam emails. Another change was that I had to lower the n.trees parameter to avoid overfitting the data. The code is as follows:

```
# Repeat the above but with a weight vector

# Make a set of 3067 1's. 3067 is the size of the training set. Then
set all 0's to a different weight
spam_weights = rep(1, nrow(spam_train))
spam_weights[spam_train$type == 0] = 25

# Decrease # of trees to avoid overfitting so we can minimize error.
gbm1 <- gbm(type~., data=spam_train, train.fraction=1.0, weights =
spam_weights, interaction.depth=4, shrinkage=.05, n.trees=150,
bag.fraction=0.5, cv.folds=5, distribution="bernoulli", verbose=F)
gbm1_iterations = gbm.perf(gbm1, method="cv")
gbm1_iterations
gbm1.predict <- predict(gbm1, spam_test, type="response",
n.trees=gbm1_iterations)
# Make a set of 1534 0's. 1534 is size of predicted (or test set)
gbm1_predicted = rep(0, nrow(spam_test))
# Set to 1 if predict has a value over 0.5.
gbm1_predicted[gbm1.predict>0.5] = 1

gbm1_pred_actual = table(gbm1_predicted, spam_test$type)
gbm1_pred_actual

misclass_scores =
gbm1_pred_actual[2][1]/(gbm1_pred_actual[1][1]+gbm1_pred_actual[2][1])
misclass_scores
```

Using this, we find our misclassification scores of:

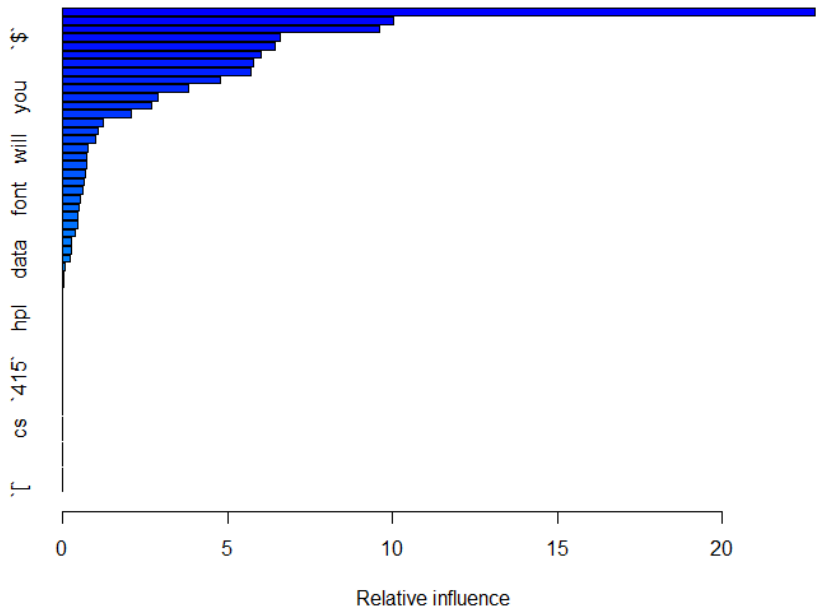
```
gbm1_predicted   0   1
             0 913 211
             1   3 407
```

This gives us an overall misclassification score of non-spam emails of 0.33%.

i. The overall misclassification score of this is 13.95%.
The misclassification of non-spam emails is .33%.
The misclassification of spam emails is 33.82%.

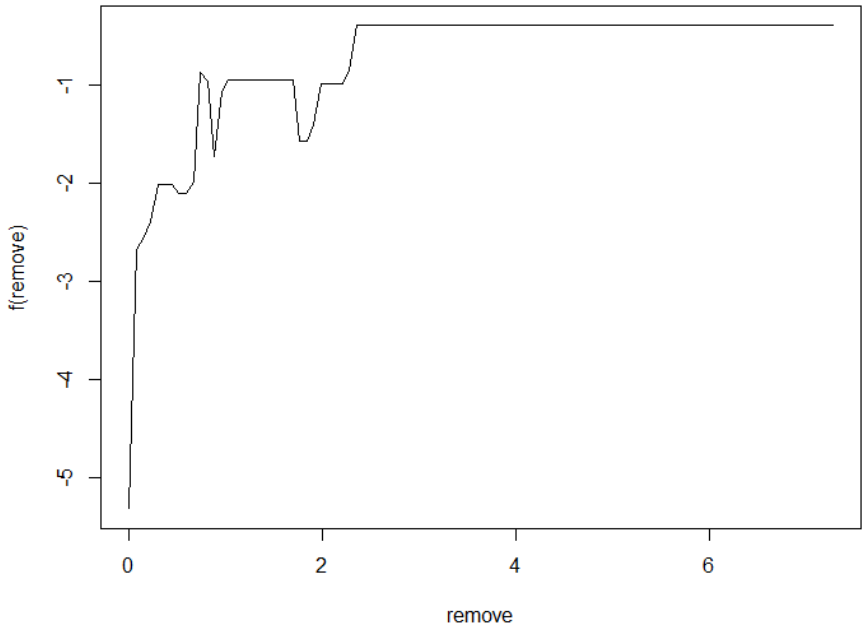
ii. I checked this using the summary function:

```
summary(gbm1)
```

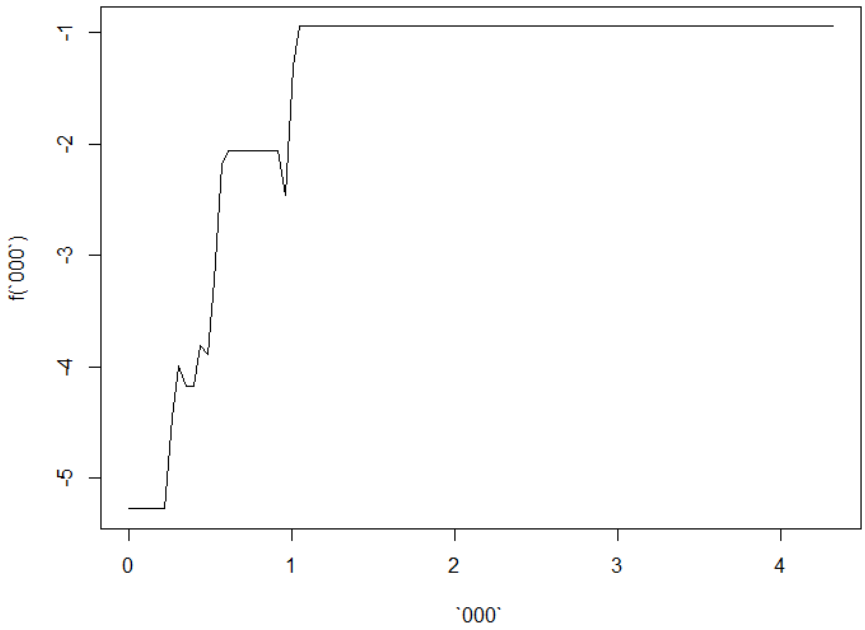


The most important variable was the presence of the word remove. Aside from that, the presence of 000 or ! were good indicators. The next tier of variables included $, money, your, CAPAVE, CAPTOT, and CAPMAX.

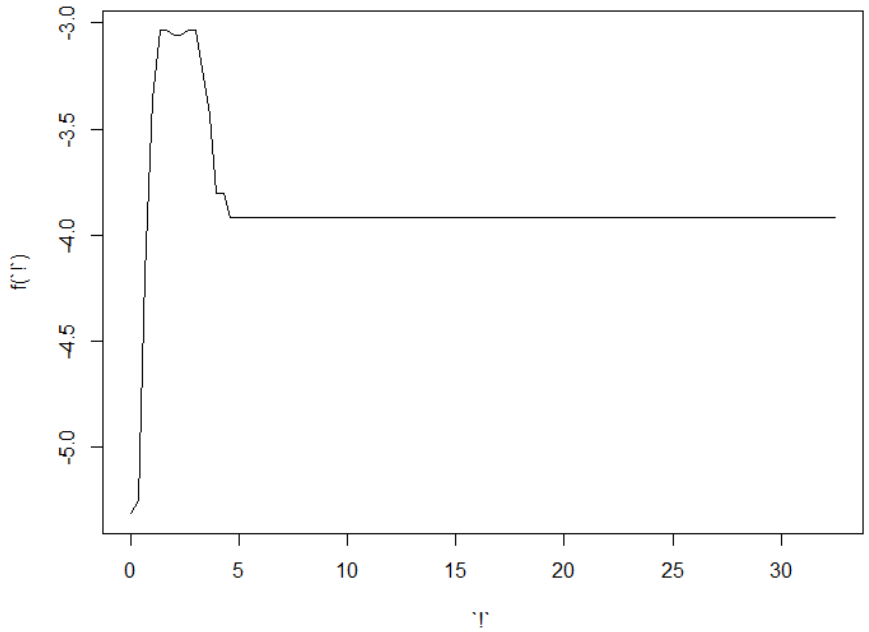
iii. To check dependence, we use the plot function on our relevant variables. For this case, we will look at the top 3: remove, 000, and !.



For remove, we see that having two or more instances of the word remove increases likelihood of it being spam significantly



For 000, we see that having 000 even a single time increases the likelihood of it being spam.



For !, we see that having 2-4 times is extremely indicative of an email being spam, while having 5 or more !s is less indicative (though still a good predictor of spam).

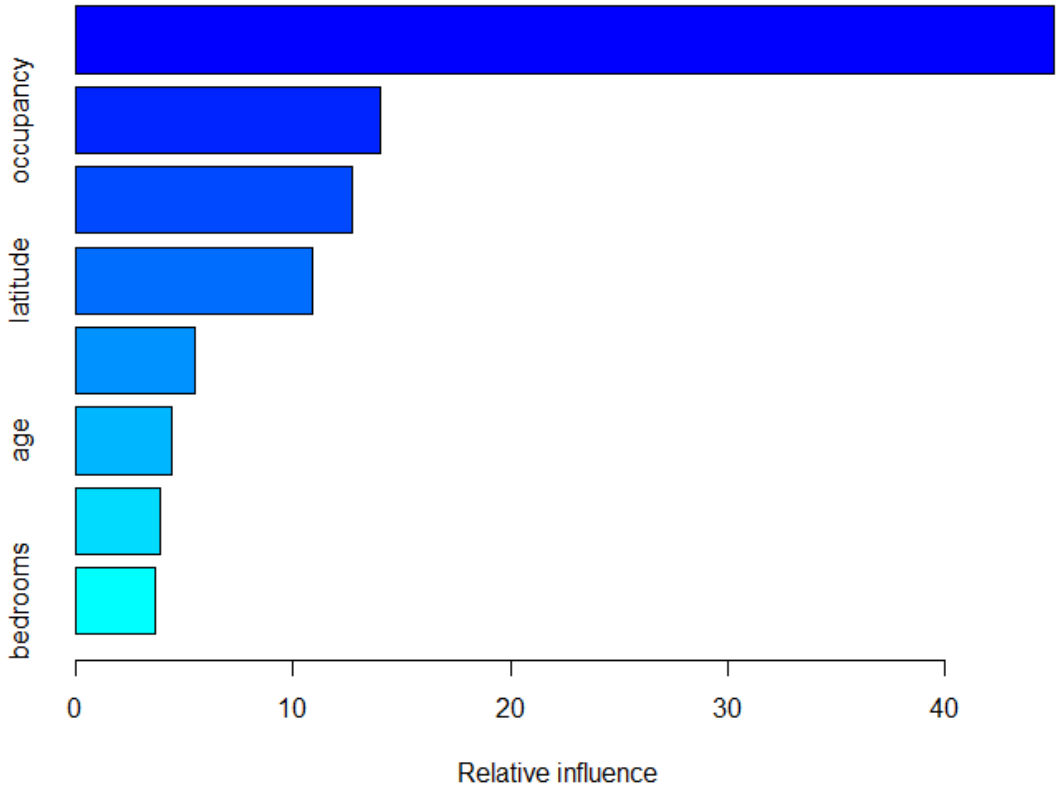
7. Regression: California Housing

Much like with 6, I set up a gbm model. The main difference is that I had to randomly select a training and test set since it was not done for me beforehand. In addition, I used a Gaussian distribution since we're looking at a continuous variable in value.

```
california_all = read.csv("California_Data.txt", header = FALSE)
colnames(california_all) <- c("value", "income", "age", "rooms", "bedrooms",
"population", "occupancy", "latitude", "longitude")

set.seed(131)

# Take out 1/4 for test set, save the other 3/4 for training.
training_set_indices = sample(nrow(california_all),
floor(nrow(california_all)*3/4), replace = FALSE)

california_train = california_all[training_set_indices,]
california_test = california_all[-training_set_indices,]

# Test out different n.trees until we get an iteration value under our max
n.trees. (worked at)
gbm2 <- gbm(value~., data=california_train, train.fraction=1.0,
interaction.depth=4, shrinkage=.05, n.trees=8000, bag.fraction=0.5,
cv.folds=5, distribution="gaussian", verbose=F)
gbm2_iterations = gbm.perf(gbm2, method="cv")

gbm2_predict = predict(gbm2, california_test, type="response",
n.trees=gbm2_iterations)

# We use squared error as our loss function to evaluate our prediction
accuracy

total_error = sum((gbm2_predict - california_test$value)^2)
avg_error = total_error/nrow(california_test)
avg_error
```

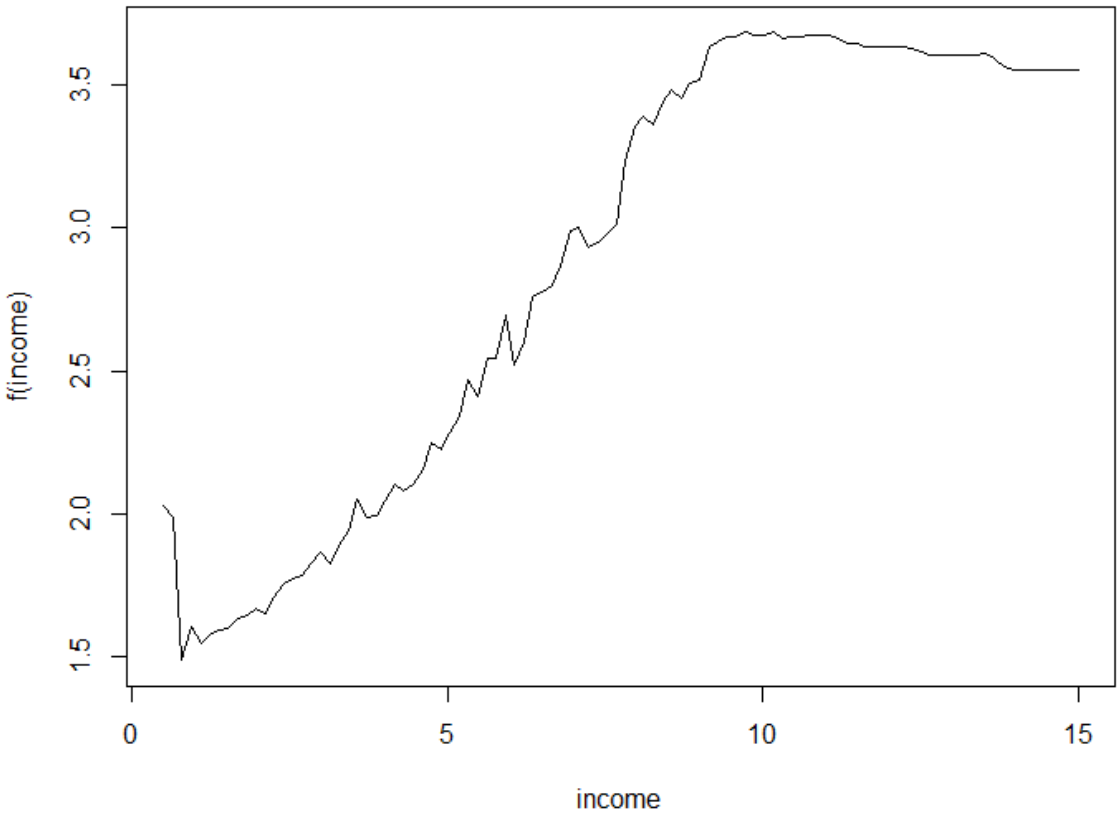
(a) Overall, we see that when we fit our model, using the average squared error as our loss function to evaluate our prediction accuracy, we get a loss of 0.216. Seeing as our range of house values is .1499 to 5, this is a pretty low loss. (b) Calling summary lets us see the most important variables:

```
summary(gbm2)
```

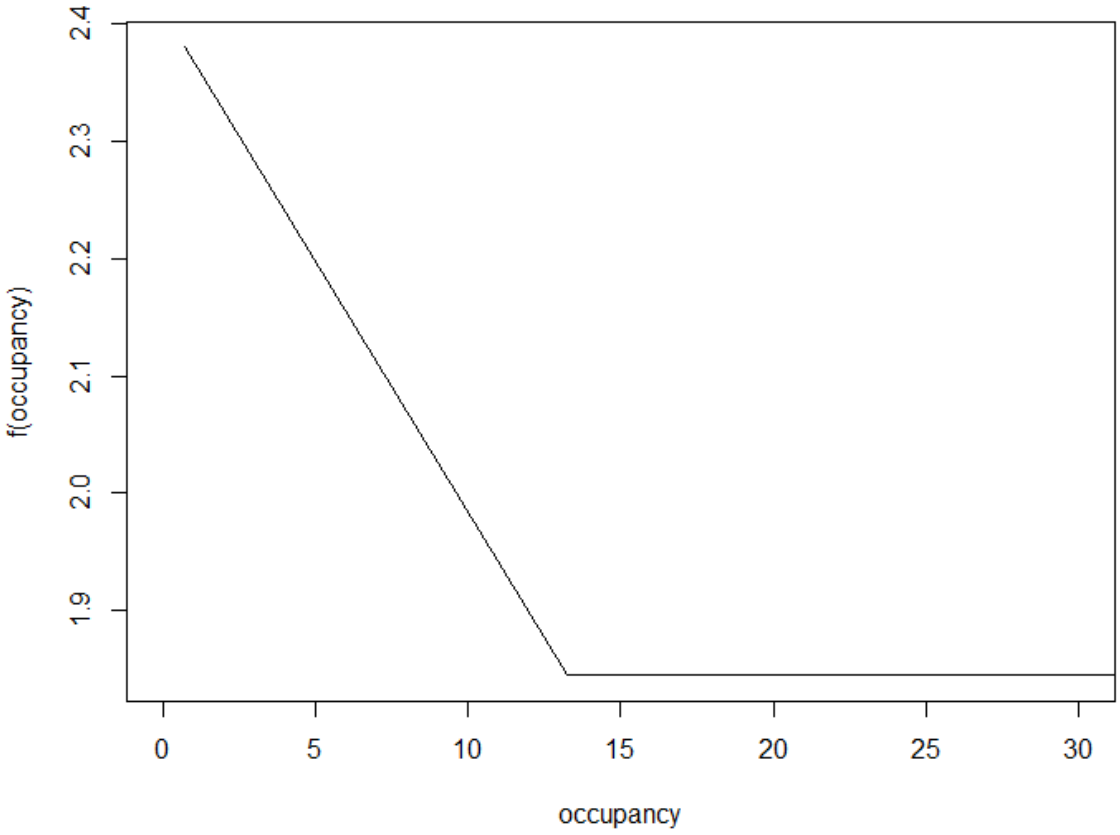


From here, we see that the highest influence by far is income. After that, the next tier of influential variables are occupancy, longitude, and latitude. (c) To take a closer look on the dependence of these variables, we make partial dependence plots.

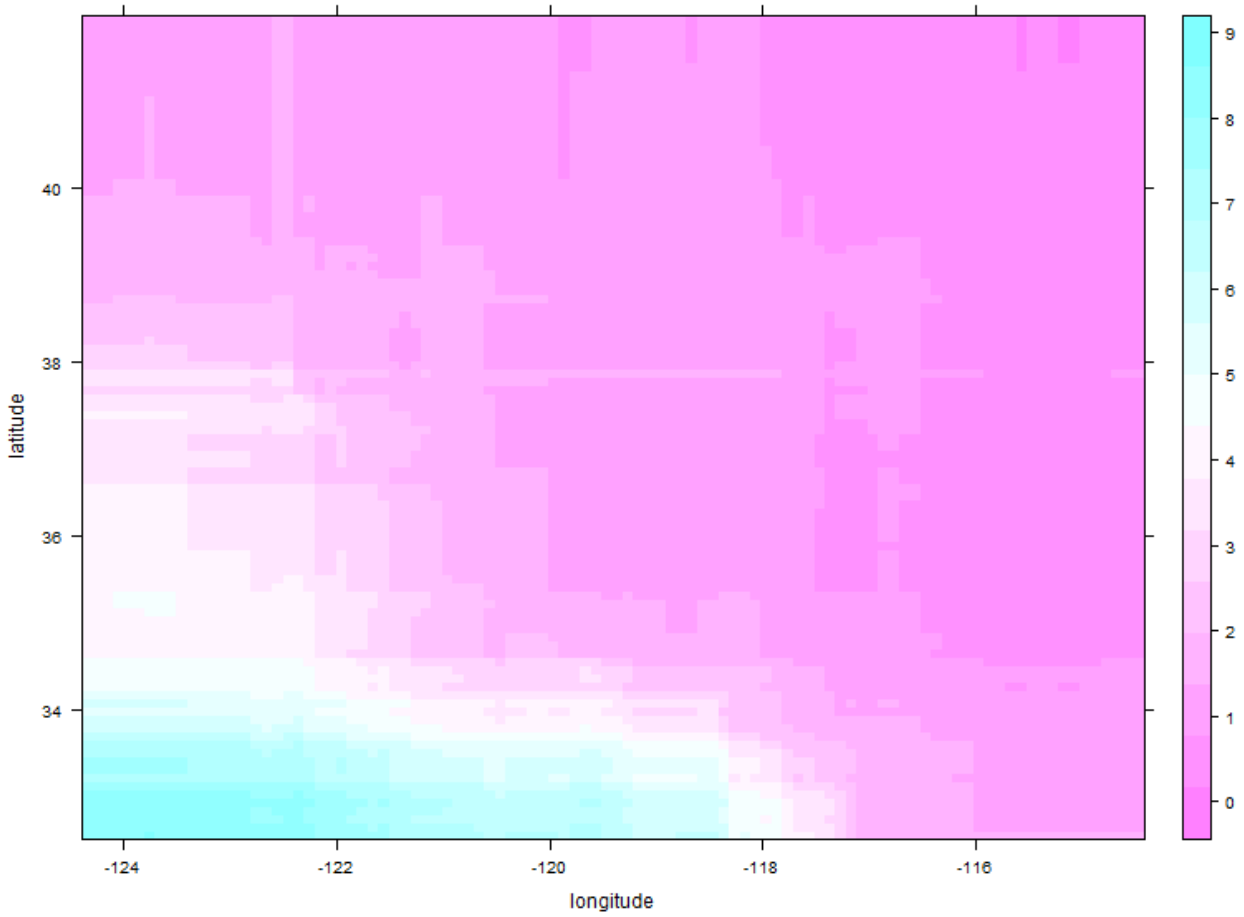
```
plot(gbm2, i.var=1, n.trees=gbm2_iterations)
# Some reason, in row 19007 of california_all, we see that there is an
occupancy of 1243.33. This skews the plot, so I changed the axes
plot(gbm2, i.var=6, n.trees=gbm2_iterations, xlim=c(0, 30))
plot(gbm2, i.var=7, n.trees=gbm2_iterations)
plot(gbm2, i.var=8, n.trees=gbm2_iterations)
plot(gbm2, i.var=c(8,7), n.trees=gbm2_iterations)
```



From this, we see that increasing income is an important predictor for increasing value of ones house.



From this plot, we see that as the occupancy increases, the value of the house decreases.



For latitude and longitude, it makes sense to plot them together to get an idea of how location affects value. We see that houses found in the southwest (low lat and long) have the highest value.

8. Regression: Marketing data

a. We set up all of our data the same way as we did in homework 1.

```
income_all = read.csv("Income_data.txt", header = FALSE)

# Add in headers based on information from Income_Info.txt
names(income_all) <- c("income", "sex", "marital_status", "age",
"education", "occupation", "bay_area_residence_time", "dual_income",
"household_count", "household_count_under_18", "householder_status",
"type_home", "ethnicity", "language")

# Mark categorical (unordered in this case) variables
income_all$sex <- as.factor(income_all$sex)
income_all$marital_status <- as.factor(income_all$marital_status)
income_all$occupation <- as.factor(income_all$occupation)
income_all$dual_income <- as.factor(income_all$dual_income)
income_all$householder_status <-
as.factor(income_all$householder_status)
income_all$type_home <- as.factor(income_all$type_home)
income_all$ethnicity <- as.factor(income_all$ethnicity)
income_all$language <- as.factor(income_all$language)
```

From here, we set up our model very similarly to that of question 7, using a Gaussian model again, since we have a somewhat continuous response variable in income.

```
set.seed(131)

# Split into training and testing set. We'll make the split 2:1 with
train:test
training_set_indices = sample(nrow(income_all),
floor(nrow(income_all)*2/3), replace = FALSE)

income_train = income_all[training_set_indices,]
income_test = income_all[-training_set_indices,]

gbm3 <- gbm(income~., data=income_train, train.fraction=1.0,
interaction.depth=4, shrinkage=.05, n.trees=1000, bag.fraction=0.5,
cv.folds=5, distribution="gaussian", verbose=F)
gbm3_iterations = gbm.perf(gbm3, method="cv")

gbm3_predict = predict(gbm3, income_test, type="response",
n.trees=gbm3_iterations)
```

Again, we use squared error as our loss function.

```
total_error = sum((gbm3_predict - income_test$income)^2)
avg_error = total_error/nrow(income_test)
avg_error
```

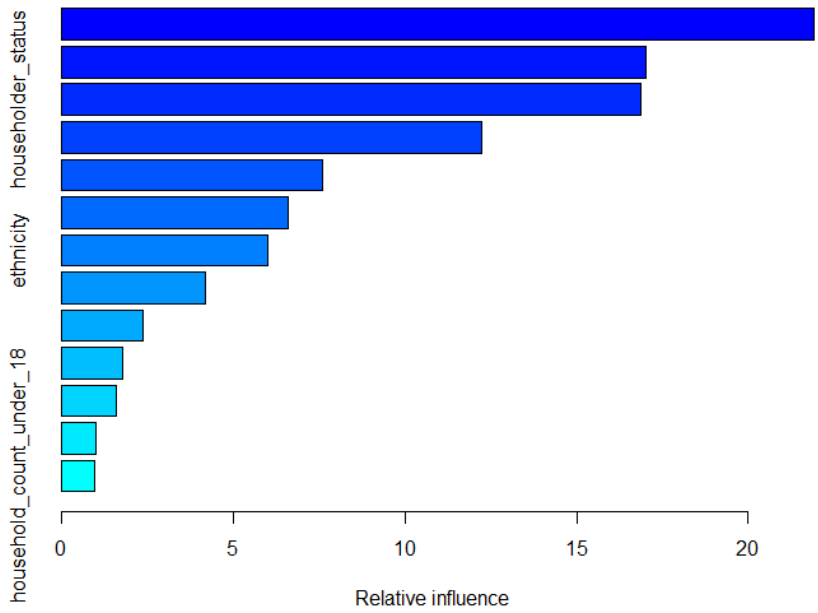
This gave us an error of 3.6 using squared error. Our trees in homework 1 had squared errors of 3.8 to 4.1 through a few iterations, meaning that our new model is

a bit better.

b. To find the most important variables, we once again run the summary function.

```
summary(gbm3)
```

This tells us that occupation, age, householder status, and marital status are the strongest indicators of ones income.



As for why sex seems to be a very weak indicator, it's possible that the sample/data is not representative of the populations where this disparity exists. The largest gaps in pay are found in very highly paid jobs, and this income scoring method in the data collection groups all incomes of $75,000 or more together. As an example, that means that if most men are making $150k while the women are making $100k, this model will not show the difference and sex will not be a good indicator, even though they may have a pay-gap.

9. Multiclass classification: marketing data.

For this question, we once again set up our model very similarly to the others. The main difference here is that we now use a multinomial distribution, since our response variable is categorical. This also means that setting up our matrix of predicted vs. test is a bit more difficult, but what we do here is pick the class that has the highest probability.

```
occupation_all = read.csv("Occupation_data.txt", header = FALSE)

# Add in headers based on information from Income_Info.txt
names(occupation_all) <- c("occupation", "type_home", "sex",
"marital_status", "age", "education", "income", "bay_area_residence_time",
"dual_income", "household_count", "household_count_under_18",
"householder_status", "ethnicity", "language")

# Mark categorical (unordered in this case) variables

occupation_all$sex <- as.factor(occupation_all$sex)
occupation_all$marital_status <- as.factor(occupation_all$marital_status)
occupation_all$occupation <- as.factor(occupation_all$occupation)
occupation_all$dual_income <- as.factor(occupation_all$dual_income)
occupation_all$householder_status <-
as.factor(occupation_all$householder_status)
occupation_all$type_home <- as.factor(occupation_all$type_home)
occupation_all$ethnicity <- as.factor(occupation_all$ethnicity)
occupation_all$language <- as.factor(occupation_all$language)

set.seed(131)

# Split into training and testing set. We'll make the split 2:1 with
train:test
training_set_indices = sample(nrow(occupation_all),
floor(nrow(occupation_all)*2/3), replace = FALSE)

occupation_train = occupation_all[training_set_indices,]
occupation_test = occupation_all[-training_set_indices,]

gbm4 <- gbm(occupation~., data=occupation_train, train.fraction=1.0,
interaction.depth=4, shrinkage=.05, n.trees=1000, bag.fraction=0.5,
cv.folds=5, distribution="multinomial", verbose=F)
gbm4_iterations = gbm.perf(gbm4, method="cv")

gbm4_predict = predict(gbm4, occupation_test, type="response",
n.trees=gbm4_iterations)
gbm4_predicted = matrix(nrow=dim(gbm4_predict)[1], ncol=1)
for(i in 1:dim(gbm4_predict)[1]){
  gbm4_predicted[i] = which.max(gbm4_predict[i,,])
}

gbm4_pred_actual = table(gbm4_predicted, occupation_test$occupation)
gbm4_pred_actual
```

From here, we get a distribution of (with rows being predictions and columns being test):

```
gbm4_predicted   1   2   3   4   5   6   7   8   9
             1 734 101  80 147  22  30  20  28  20
             2   9  11   7  11   2   8   2   0   5
             3  36  26  83  22   2  13  11   3  12
             4  76  39  23  86  25  23   6   1  10
             5  16   4   6  10 136   9   0  14   5
             6  30  70  33  36   3 399  14   0  40
             7   3   2   4   1   0   6  23   2   2
             8  36   5   5  12  26   1   2 204   2
             9   1  10   5   2   4   5   3   3  25
```

a. From here, we can calculate our misclassification errors overall and for each class.

We do this with the following code:

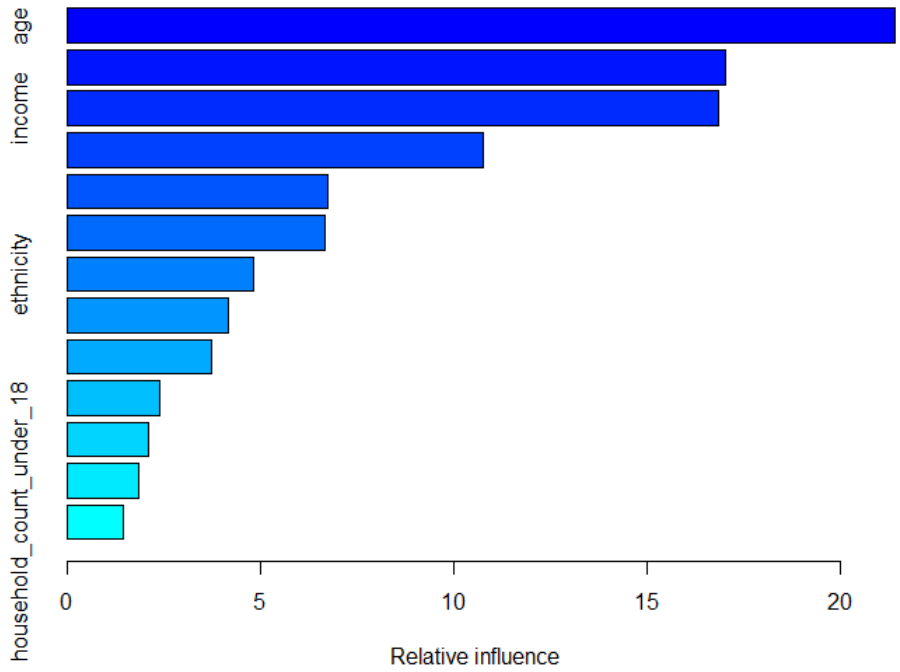
```
misclass = rep(0, 9)
overall_total = 0
overall_correct = 0
for(i in 1:9){
  correct = gbm4_pred_actual[i,i]
  total = sum(gbm4_pred_actual[,i])
  misclass[i] = (total-correct)/total
  overall_total = overall_total + total
  overall_correct = overall_correct + correct
}

misclass
overall_misclass_rate = (overall_total - overall_correct)/overall_total
overall_misclass_rate
```

We find that our overall misclassification rate is 42.4%
Class 1 misclassification rate is: 22.0%
Class 2 misclassification rate is: 95.9%
Class 3 misclassification rate is: 66.2%
Class 4 misclassification rate is: 73.7%
Class 5 misclassification rate is: 38.1%
Class 6 misclassification rate is: 19.2%
Class 7 misclassification rate is: 71.6%
Class 8 misclassification rate is: 20.0%
Class 9 misclassification rate is: 79.3%

b. The most important variables are given by the summary function:

```
summary(gbm4)
```

This shows us that age, education, and income are the most important indicators of occupation. In addition, householder status is a strong indicator of occupation.


age
income
ethnicity
household_count_under_18
0
5
10
15
20
Relative influence