Ted Li – tedli
Adhitya Venkatesh - Avenkate

Problem 1:

The code for this problem is found in question_1.R
Here, I will explain what I did and why, with the code added in to help explain.

I begin by reading in and setting up all of the headers/categorical variables:

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

names(income_data) <- 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")

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

I then split the data into a training set and testing set. I did this by randomly sampling 2/3 of the rows into the training set and the remaining into the testing set.

```
training_set_indices = sample(nrow(income_data),
floor(nrow(income_data)*2/3), replace = FALSE)

training_set = income_data[training_set_indices,]
testing_set = income_data[-training_set_indices,]
```

I then made the full, unpruned tree using rpart.

```
unpruned_tree = rpart(income~., data=training_set, control =
rpart.control(cp=0))
```

I then found the optimal CP in two different methods.

The first method was using the built in cross-validation error from RPART. The CP associated with the lowest xerror was CP = 0.0009523579

```
cp_frame = as.data.frame(unpruned_tree$cptable)
min_cp = with(cp_frame, CP[xerror == min(xerror)])
```

I also repeated this again but this time, I used the entire income_data set to make my tree, instead of just the training set. This gave me a CP = 0.0008745591

My other method was to validate by using a disjoint testing set. I did this by using each of the CP values in the table and pruning the tree with that CP. Afterwards, I predicted the incomes for each member of the test set and took the error using the square of the differences, and found the minimum error. This gave me a CP = 0.0009936559.

```
lowest_error = NULL
min_cp2 = 0

for (i in 1:nrow(cp_frame)){
  pruned_tree = prune.rpart(unpruned_tree, cp_frame$CP[i])
  predicted_values = predict(pruned_tree, testing_set)
  error = sum((testing_set$income-predicted_values)^2)
    if (is.null(lowest_error)){
    lowest_error = error
    min_cp2 = cp_frame$CP[i]
  }

  if (error < lowest_error){
    lowest_error = error
    min_cp2 = cp_frame$CP[i]
  }
}
```

Between these methods, we see that using a separate test set will give us the highest CP and thus most pruned tree.

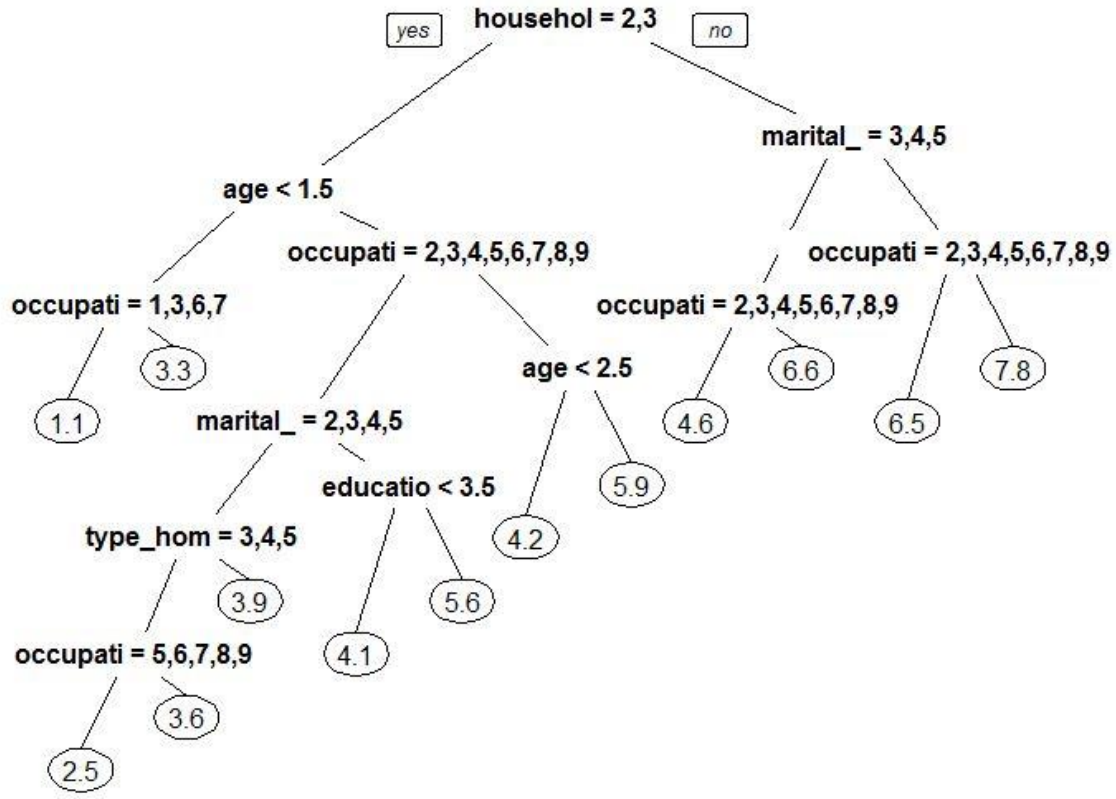
Using printcp, I was able to take a look at the various CP values and errors for my training tree.

Training Tree:

```
Regression tree:
rpart(formula = income ~ ., data = training_set, control = rpart.control(cp =
0))

Variables actually used in tree construction:
 [1] age                       bay_area_residence_time  dual_income
education                 ethnicity
 [6] household_count           household_count_under_18 householder_status
language                  marital_status
[11] occupation                sex                      type_home

Root node error: 45938/5995 = 7.6627

n= 5995

            CP nsplit rel error  xerror     xstd
1   2.2902e-01      0   1.00000 1.00044 0.010148
2   8.1228e-02      1   0.77098 0.78570 0.011132
3   4.1746e-02      2   0.68975 0.69759 0.011364
4   2.9527e-02      3   0.64800 0.65810 0.011583
5   1.6238e-02      4   0.61848 0.62846 0.011092
6   1.3708e-02      5   0.60224 0.61050 0.011292
...
36  1.0171e-03     48   0.47228 0.53292 0.010839
37  1.0114e-03     49   0.47126 0.53291 0.010842
```

```
38  1.0024e-03     50   0.47025 0.53305 0.010823
39  9.9366e-04     51   0.46925 0.53303 0.010823 # Using test set
40  9.7833e-04     52   0.46826 0.53326 0.010849
41  9.5236e-04     53   0.46728 0.53254 0.010840 # Using RPART xerror
42  9.4359e-04     54   0.46632 0.53290 0.010859
...
346 9.8948e-07    465   0.36480 0.58667 0.012458
347 1.1132e-07    466   0.36479 0.58668 0.012458
348 0.0000e+00    467   0.36479 0.58668 0.012458
```

The tree that we get is shown below, pruned to 10 terminal nodes for ease of view.



Looking at the relationship between annual income and our demographic predictors, it makes sense for the most part. For example, those who are in the lowest age group (and thus would be sorted to the left of the age < 1.5 group) are 14 through 17 years old, and they would make sense to have the lowest income. In addition, those who have a higher income are most likely to own a house, rather than rent or live with others, so we see a correlation there. Another interesting indicator is occupation, where those who have a professional/managerial occupation make more money than all of the other occupations (which are grouped together). We also see that more education leads to a higher prediction in your income.

a.
Surrogate splits were used in the construction in each of the optimal trees we obtained. For

simplicity's sake, from here on out, I will just refer to the tree we got from using a separate test set of data. A surrogate split is a tool that is used whenever a variable that is needed for a split is not present in the current data. It works by finding surrogate variables – in other words, we look for other variables that are representative of the missing variable at hand, and we list all of them in decreasing order of how representative they are. From here, we then start from the best surrogate variable and see if our current data has that variable. If so, we use it as a surrogate to guess which split we would have taken. If that variable is also missing, we then go to the next best representative variable. If we run out of variables, we can randomly guess or pick the majority side of the split. To see an example of this, we can run summary(unpruned_tree), and look at the nodes. Listed below is a node that had a surrogate split

```
Node number 326: 246 observations,    complexity param=0.001076497
  mean=3.898374, MSE=3.2539
  left son=652 (137 obs) right son=653 (109 obs)
  Primary splits:
      marital_status  splits as  -RRLL,    improve=0.07597825, (4 missing)
      ethnicity       splits as  LRR-L-LL, improve=0.05534001, (2 missing)
      household_count < 4.5 to the right,  improve=0.02423658, (14 missing)
      language        splits as  RLL,      improve=0.01818535, (7 missing)
      type_home       splits as  --RRL,    improve=0.01727408, (17 missing)
  Surrogate splits:
      household_count_under_18 < 0.5 to the left,   agree=0.616, adj=0.147, (4 split)
      dual_income              splits as  LRR,      agree=0.612, adj=0.138, (0 split)
      education                < 3.5 to the right,  agree=0.599, adj=0.110, (0 split)
      household_count          < 1.5 to the left,   agree=0.583, adj=0.073, (0 split)
      ethnicity                splits as  LLR-R-LR, agree=0.583, adj=0.073, (0 split)
```

In this node, we see that we had a surrogate split on the variable of marital_status. We see that it's split on the following variables: household_count_under_18, dual_income, education, household_count, and ethnicity.

b.
For "my household," I will be using myself.

My demographics are as follows:

Sex:	1	
Marital:		5
Age:	2	
Education:	5	
Occupation:	6	
Bay Area length:		3
Dual Income:	1	
Persons:		1
Persons under 18:	0	
Householder status:	3	
Type of home:	3	
Ethnicity:	2	
Language:	1	

Following this, the prediction gives me an income of 2.350365, which is pretty accurate. It gives me an income of around $12,000/year, which is pretty accurate (maybe a bit over-estimated), since I work full time over the summer.

Problem 2:

For this question, we set up our data just as we did in problem 1.

```
housetype_data = read.csv("Housetype_data.txt", header = FALSE)

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

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

From here, we simply constructed a tree with all of our information. In this question, we only used RPART's built in cross validation to select our CP and solve for misclassification error.

From here, we can get our minimum CP and xerror, which are 0.0008121278 and 0.6266919, respectively.

```
house_tree = rpart(type_home~., data=housetype_data, control =
rpart.control(cp=0))
house_cp = as.data.frame(house_tree$cptable)
house_min_cp = with(house_cp, CP[xerror == min(xerror)])
house_min_xerror = with(house_cp, xerror[xerror == min(xerror)])

house_min_cp
house_min_xerror
```

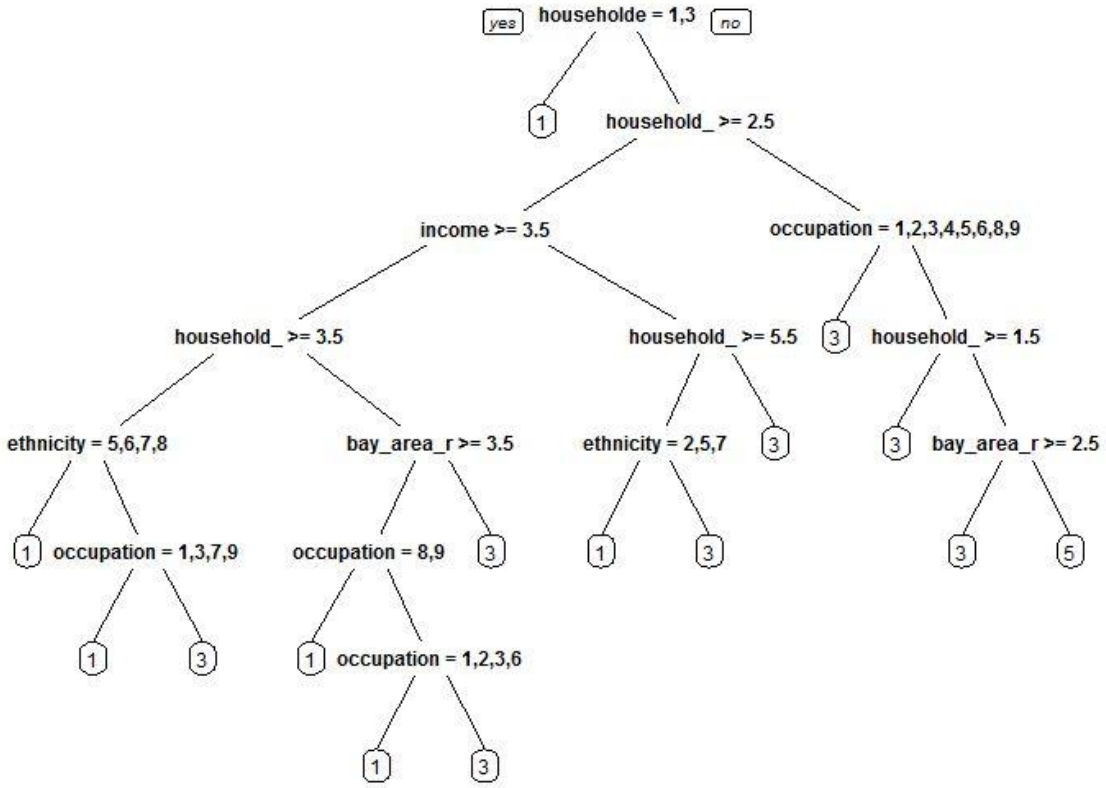
We then check the root node error and find that it's $3694/9013 = 0.40985$.

```
printcp(house_tree)
```

From here, we can get our misclassification rate by multiplying our minimum xerror with our root node, since xerror iterates over our left out sets and root node error gives us the initial error. Thus, misclassification rate $= 0.6266919 * 0.40985 = 0.2568497$.

We then plot the tree after pruning it to the CP with the lowest xerror.

```
prp(prune.rpart(house_tree, house_min_cp))
```



There are a few interesting points regarding this tree. First off, we see that after pruning, many variables are no longer necessary for predicting the house type. To elaborate, sex, marital status, age, education, dual incomes, people under 18 in household, and language no longer are necessary for predicting house type. This sort of makes sense, because while they may play a role in income (for question 1), many of the people who would have vastly different incomes may still live in a normal house, because of their parents, spouse, etc. Another interesting point is that our tree only ever classifies to house types of 1, 3, and 5. In other words, our tree will only ever predict for you to live in a house, apartment, or other, and never a condominium or mobile home. This could be caused by the low frequency of these forms of housing or could do with difficulty in their classification. It might also mean that none of these demographics are useful in finding the difference between these types of housing. Ultimately, it shows that our tree values missing the classification on these two types of housing 100% of the time, than risking misclassifying other types of housing as well.

Problem 3:

For a given problem, the target function is our function that minimizes the risk – our best possible predicting function. However, just because it is the best, does not mean that it is good – that it is always accurate. The prediction risk for the problem as a whole, could just be really high. As explained in class, there's always the possibility that you get predictor variables that just don't tell you anything valuable about the response variable, and you can't make accurate predictions, even if you had the target function.

Problem 4:

The empirical risk on the training data is not always the best surrogate for the actual prediction risk. This is because if we only want to minimize empirical risk on the training function, we'll end up overfitting and end up increasing our variance, making our model possibly worse for future uses on testing data. It is expected to be good when the training data is very similar to the testing data.

Problem 5:

The prediction function cannot be chosen from the class of all possible functions because there are an infinite number of functions, and not only that, but there are an infinite number of solutions that would fit our model.

Problem 6:

Bias variance tradeoff is a problem that we have when we are splitting our trees – it dictates when we will stop splitting in this case. As we train more and allow the tree to be larger (with a lower CP), then we will more closely fit the training data and thus monotonically decrease our bias. However, we have a tradeoff with variance, where as we grow our tree (and have lower CP values), at first, variance will decrease. However, in many models, at some point, if we keep increasing the size of our tree (and decreasing CP values), then we will start to increase our variance due to problems such as overfitting. Thus, in many cases, we have to find a balance for our tree size to get the lowest total error from bias and variance.

Problem 7:

Withdrawn

Problem 8:

We use the primary split instead because due to how our algorithm works, the primary split was only picked because it was greedily chosen to give the largest improvement to our risk (decreasing it). Using any other split would only dilute the power of our split, since the surrogates are only surrogates for the primary split – this means that the primary split was the best predictor variable at that point, and the surrogates were strictly worse. Using a surrogate split in this case would be less effective than the primary split. Especially in a case where there are no good surrogates, using surrogates instead of primary splits is extremely detrimental to our greedy algorithm.

Problem 9:

9. We define the error function as $S(\vec{C}_m)$:

$$S(\vec{C}_m) = \sum_{i=1}^{N} \left(y_i - \sum_{m=1}^{M} C_m I(x_i \in R_m) \right)^2$$

where $\vec{C}_m = [C_1, C_2, \ldots, C_m, \ldots C_M]$

$$S(\vec{C}_m) = \sum_{i=1}^{N} y_i^2 - 2y_i \sum_{m=1}^{M} C_m I(x_i \in R_m) \left(\sum_{m=1}^{M} C_m I(x_i \in R_m) \right)^2$$

$$\therefore \frac{\partial S}{\partial C_m} = \sum_{i=1}^{N} \left(-2y_i I(x_i \in R_m) + 2C_m I(x_i \in R_m) \right)$$

Setting the partial derivative to zero and solving for C_m yields

$$C_m = \hat{C}_m = \frac{\sum_{i=1}^{N} y_i I(x_i \in R_m)}{\sum_{i=1}^{N} I(x_i \in R_m)} = \hat{y}_m$$

Problem 10:

10. We define improvement as ΔS or σ, which is given by

$$+\sum_{i=1}^{N}\left((y_i-\bar{y}_m I(x_i\in R_m))^2-(y_i-\bar{y}_l I(x_i\in R_l)-\bar{y}_r I(x_i\in R_r))^2\right)$$

$$=-\sum_{x_i\in R_m}\left(y_i^2-2y_i(\bar{y}_l I(x_i\in R_l)+\bar{y}_r I(x_i\in R_r))+\left(\bar{y}_l I(x_i\in R_l)+\bar{y}_r I(x_i\in R_r)\right)^2-y_i^2+2y_i\bar{y}_m+\bar{y}_m^2\right)$$

$$=-\sum_{x_i\in R_m}\left(2y_i(\bar{y}_m-\bar{y}_l I(x_i\in R_l)-\bar{y}_r I(x_i\in R_r))+\left(\bar{y}_l I(x_i\in R_l)\right)^2+\left(\bar{y}_r I(x_i\in R_r)\right)^2-\bar{y}_m^2\right)$$

As $\sum_{x_i\in R_m} I(x_i\in R_l)=n_l$, $\sum_{x_i\in R_m} I(x_i\in R_r)=n_r$,

and $\dfrac{\sum_{x_i\in R_m} y_i I(x_i\in R_{r,l,m})}{\sum_{x_i\in R_m} I(x_i\in R_{r,l,m})}=\bar{y}_{r,l,m}$, we have

$$=-\left(n\bar{y}_m^2-n_l\bar{y}_l^2-n_r\bar{y}_r^2+n_l\bar{y}_l^2+n_r\bar{y}_r^2-n\bar{y}_m^2\right)$$

$$=-n\bar{y}_m^2+n_l\bar{y}_l^2+n_r\bar{y}_r^2$$

Using the fact that $\bar{y}_m = \frac{n_l \bar{y}_l + n_r \bar{y}_r}{n_l + n_r}$

and $n = n_l + n_r$, we have

$$= \frac{(n_l \bar{y}_l + n_r \bar{y}_r)^2 - n n_l \bar{y}_l^2 - n n_r \bar{y}_r^2}{n}$$

$$= \frac{-n_l^2 \bar{y}_l^2 + n_r^2 \bar{y}_r^2 + n n_l \bar{y}_l^2 + n n_r \bar{y}_r^2 - 2 n_l n_r \bar{y}_l}{n}$$

$$= \frac{n_l (n - n_l) \bar{y}_l^2 + n_r (n - n_r) \bar{y}_r^2 - 2 n_l n_r \bar{y}_l \bar{y}_r}{n}$$

$$= \frac{n_l n_r (\bar{y}_l^2 - 2 \bar{y}_l \bar{y}_r + \bar{y}_r^2)}{n}$$

$$= \boxed{\frac{n_l n_r (\bar{y}_l - \bar{y}_r)^2}{n}} = \sigma(n_l, n_r, \bar{y}_l, \bar{y}_r)$$

Problem 11:

11. We have two cases if observation i is moved from...

i) $\boxed{r \to l}$:

$$\sigma = \frac{(n_l+1)(n_r-1)}{n}(\bar{y}_l^{\,1} - \bar{y}_r^{\,1})^2$$

where $\bar{y}_l^{\,1} = \dfrac{n_l \bar{y}_l + y_i I(x_i \in R_l)}{n_l + 1}$

$$\bar{y}_r^{\,1} = \frac{n_r \bar{y}_r - y_i I(x_i \in R_r)}{n_r - 1}$$

ii) $\boxed{l \to r}$:

$$\sigma = \frac{(n_l-1)(n_r+1)}{n}(\hat{\bar{y}}_l - \hat{\bar{y}}_r)^2 \quad \text{where}$$

$$\hat{\bar{y}}_l = \frac{n_l \bar{y}_l - y_i I(x_i \in R_l)}{n_l - 1}$$

$$\hat{\bar{y}}_r = \frac{n_r \bar{y}_r + y_i I(x_i \in R_r)}{n_r + 1}$$

Now, change in improvement is given by difference in improvements for both splits:

$$= \left(\hat{\bar{y}}_r^2 (n_r+1) + \hat{\bar{y}}_l^2 (n_l-1) - \bar{y}_m^2 n\right)$$

$$+ \left(\bar{y}_m^2 n - \bar{y}_l^2 n_l - \bar{y}_r^2 n_r\right)$$

$$= \frac{(n_l \bar{y}_l - y_i I(x_i \in R_l))^2}{n_l - 1} + \frac{(n_r \bar{y}_r + y_i I(x_i \in R_r))^2}{n_r + 1}$$

$$- n_l \bar{y}_l^2 - n_r \bar{y}_r^2$$

Suppose $y_i I(x_i \in R_l) = y_0$, then

$$= \frac{n_l \bar{y}_l^2 + n_l y_0^2 - 2 n_l \bar{y}_l y_0 - (n_l - 1) y_0^2}{n_l - 1}$$

$$+ \frac{n_r \bar{y}_r^2 + n_r y_0^2 + 2 n_r \bar{y}_r y_0 + (n_r - 1) y_0^2}{n_r + 1}$$

$$= \left| \frac{n_l (\bar{y}_l - y_0)^2}{n_l - 1} + \frac{n_r (\bar{y}_r + y_0)^2 - 2 n_r y_0^2}{n_r + 1} \right|$$

Problem 12:

The first strategy is very efficient computationally because we do not have to do a surrogate split or continue down the tree. In addition, if the children of the node are not as important (are pruned at low CP values), then we will still get a relatively accurate prediction. On the other hand, the second method simply causes the observations to be selected based on the weighted probability of the training data variables. As such, this would be more useful when our training data is very similar to the test data, as well as when the missing variables are extremely skewed towards one side.

Problem 13:

First, I'll discuss the disadvantages of this strategy. The first one is very obvious – in the case that we have data where there are relatively few occurrences of missing data, the splits that are associated with missing data will not have much data or conclusive predictions based on the lack of sufficient training data. To elaborate on that point, if for example, we have a large dataset for predicting income, and only 2 people did not fill in their gender, and we create a split in the tree that is associated with N/A for gender. Then this split has only been trained on at most, 2 sets of data, and if we get to this node in our testing data, we cannot confidently say that we are making accurate predictions based on our tree. Another way to look at this is that doing this basically splits our data up into too many smaller subtrees, not all of which are useful. As mentioned earlier, this is detrimental when we don't have much missing data, but also is detrimental when we have a lot of missing data and a smaller training set. Surrogate splits do not have this problem since they will still split based on the current variable, and thus do not create new splits based on missing data.

However, there are a few advantages of this method. The biggest advantage is that missing data could actually have information in it. For example, suppose people who have heart problems do not go to the doctor's office and end up submitting no answer for length of time since last doctor's visit (and maybe that is why they have heart problems). If this is a common trend, then having an NA there in their training data could be useful information in predicting what a person's health or heart health is. However, if the missing data is random, then this does not help out at all and creates problems with more spread out trees that will have excessive splitting, as explained in the previous paragraph.

This strategy does allow for a sort of surrogate effect. This is because in most cases, when we would want to use a surrogate split, it is because that surrogate split is a good predictor for our current split and also may be a good predictor for the overall output at that point in the tree. As such, when we split to an N/A because of missing data, it's very likely that our next primary split on that side of the tree will be the same variable that the surrogate split would have been on, since that variable was pretty much the second best variable to split on before the split, and after we see that the best variable is missing and we go down that N/A split, we are left with only the second best variable, which would have been the surrogate.

If there are no missing values in the training data, there are a few solutions. The most obvious (and worst) would be to artificially remove data from the training set. While this may sound like it could prepare you for testing data with missing information, all this really does is remove valuable information from your training set and also introduce more random missing data, which would lead to sparseness of your tree data. An easy way to fix this would be to just use surrogate splits instead. However, if we cannot do that, we can just pick a split to go down when we're missing data. This choice can either be random (50/50), weighted (based on % of training data on each side), or simply just go in the direction that the majority of training data was (pick side with higher % of training data always).