Homework 7

Brady Lamson

2022-04-09

Appendix E

Problem 5

```
helprtc <-
    mosaicData::HELPrct %>%
    mutate(
        homeless01 =
            case_when(
                homeless == "housed" ~ 0,
                homeless == "homeless" ~ 1
            )
    ) %>%
    select(where(is.numeric))
```

Before we start I want to `skimr::skim()` the data to get a general overview of what I'm working with.

```
helprtc %>%
    skimr::skim()
```

Table 1: Data summary

Name	Piped data
Number of rows	453
Number of columns	22
Column type frequency:	
numeric	22
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
age	0	1.00	35.65	7.71	19.00	30.00	35.00	40.00	60.00	
anysubstatus	207	0.54	0.77	0.42	0.00	1.00	1.00	1.00	1.00	
cesd	0	1.00	32.85	12.51	1.00	25.00	34.00	41.00	60.00	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
d1	0	1.00	3.06	6.19	0.00	1.00	2.00	3.00	100.00	
daysanysub	209	0.54	75.31	79.24	0.00	5.00	33.00	164.25	268.00	
dayslink	22	0.95	255.61	151.02	2.00	74.00	361.00	365.00	456.00	
drugrisk	1	1.00	1.89	4.34	0.00	0.00	0.00	1.00	21.00	
e2b	239	0.47	2.50	2.52	1.00	1.00	2.00	3.00	21.00	
female	0	1.00	0.24	0.43	0.00	0.00	0.00	0.00	1.00	
i1	0	1.00	17.91	20.02	0.00	3.00	13.00	26.00	142.00	
i2	0	1.00	24.55	28.02	0.00	4.00	18.00	33.00	184.00	
id	0	1.00	233.40	134.75	1.00	119.00	233.00	348.00	470.00	
indtot	0	1.00	35.73	7.15	4.00	32.00	38.00	41.00	45.00	
linkstatus	22	0.95	0.38	0.49	0.00	0.00	0.00	1.00	1.00	
mcs	0	1.00	31.68	12.84	6.76	21.68	28.60	40.94	62.18	
pcs	0	1.00	48.05	10.78	14.07	40.38	48.88	56.95	74.81	
pss_fr	0	1.00	6.71	4.00	0.00	3.00	7.00	10.00	14.00	
sexrisk	0	1.00	4.64	2.80	0.00	3.00	4.00	6.00	14.00	
avg_drinks	0	1.00	17.91	20.02	0.00	3.00	13.00	26.00	142.00	
max_drinks	0	1.00	24.55	28.02	0.00	4.00	18.00	33.00	184.00	
hospitalizations	0	1.00	3.06	6.19	0.00	1.00	2.00	3.00	100.00	
homeless01	0	1.00	0.46	0.50	0.00	0.00	0.00	1.00	1.00	

There are a few big takeaways from this skimming.

- First is a large number of NAs. I'm going to outright remove the variables with 200+ missing values as that's nearly half the data set. For the other NAs I'll do median imputation, that is replacing the NA with the median value of that variable.
- Second are many variables that do not appear to follow a normal distribution. Sadly the `skim()` functions console histograms don't show up on pdf, but trust me here! A few of these numeric variables seem log-linear though, I can log transform those to help the model work better.
- Finally is the scale of our numeric variables is all over the place. I'll need to normalize all of these variables if I want my model to be able to glean any important information from the data. There is also have an **id** column tucked away in there, I'll need to handle that.
- There's another problem, Looking at some variables there are a few that are identical. I'm not sure what the best way to handle this is outside of manually selecting out the ones I catch. Below are all the variables I caught.

```
helprtc %>%
    select(avg_drinks, i1, max_drinks, i2, hospitalizations, d1) %>%
    head()
```

```
##   avg_drinks i1 max_drinks i2 hospitalizations d1
## 1         13 13         26 26                3  3
## 2         56 56         62 62               22 22
## 3          0  0          0  0                0  0
## 4          5  5          5  5                2  2
## 5         10 10         13 13               12 12
## 6          4  4          4  4                1  1
```

Here we setup a recipe, this a convenient way to tackle a lot of the data pre-processing I want to do. This makes life a whole lot easier when working with a data set that's a little moody.

```
homeless_recipe <-
    recipe(homeless01 ~., data = helprtc) %>%
    # Make homeless01 a factor ----
    step_mutate(homeless01 = homeless01 %>% as.factor()) %>%

    # Remove duplicate variables and variables w/ over 200+ NAs ----
    step_select(-c(i1, i2, d1, anysubstatus, daysanysub, e2b)) %>%

    # Set id column to be an id, not a predictor ----
    update_role(id, new_role = "id") %>%

    # Do median imputation for variables with missing values ----
    step_impute_median(dayslink, drugrisk, linkstatus) %>%

    # Normalize numeric predictors so they're on the same scale ----
    step_normalize(all_numeric_predictors()) %>%

    # Log transform variables that appear log-normal to help it approach a normal dist ----
    step_log(avg_drinks, max_drinks, indtot, age, signed = TRUE)

homeless_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##         id          1
##    outcome          1
##  predictor         20
##
## Operations:
##
## Variable mutation for homeless01 %>% as.factor()
## Variables selected -c(i1, i2, d1, anysubstatus, daysanysub, e2b)
## Median imputation for dayslink, drugrisk, linkstatus
## Centering and scaling for all_numeric_predictors()
## Signed log transformation on avg_drinks, max_drinks, indtot, age
```

Next we'll create our model and pass both it and our recipe into a workflow!

```
homeless_model <-
    logistic_reg(mode = "classification") %>%
    set_engine("glm")

homeless_workflow <-
    workflow() %>%
    add_model(homeless_model) %>%
    add_recipe(homeless_recipe)

homeless_workflow
```

```
## == Workflow ====================================================================
```

```
## Preprocessor: Recipe
## Model: logistic_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 5 Recipe Steps
##
## * step_mutate()
## * step_select()
## * step_impute_median()
## * step_normalize()
## * step_log()
##
## -- Model -----------------------------------------------------------------------
## Logistic Regression Model Specification (classification)
##
## Computational engine: glm
```

Workflows are fantastic, they help organize the modeling process and encourage good methodology. Essentially you can bind modeling and pre-processing objects together!

```
fit(homeless_workflow, helprtc) %>%
    broom::tidy() %>%
    arrange(p.value)
```

```
## # A tibble: 15 x 5
##    term             estimate std.error statistic p.value
##    <chr>               <dbl>     <dbl>     <dbl>   <dbl>
##  1 pss_fr            -0.292      0.103    -2.83  0.00471
##  2 (Intercept)       -0.236      0.107    -2.22  0.0267
##  3 avg_drinks         1.18       0.604     1.96  0.0503
##  4 female            -0.211      0.110    -1.93  0.0537
##  5 sexrisk            0.173      0.103     1.69  0.0912
##  6 age                0.504      0.399     1.26  0.207
##  7 indtot             0.584      0.476     1.23  0.220
##  8 linkstatus         0.329      0.305     1.08  0.282
##  9 pcs               -0.0988     0.111    -0.892 0.372
## 10 dayslink           0.244      0.305     0.801 0.423
## 11 drugrisk           0.0634     0.104     0.609 0.542
## 12 cesd               0.0582     0.145     0.400 0.689
## 13 max_drinks         0.164      0.541     0.304 0.761
## 14 hospitalizations   0.0167     0.108     0.154 0.878
## 15 mcs                0.0181     0.142     0.127 0.899
```

At the risk of interpreting these results incorrectly, it appears that, according to our p-values, that we only really have 3 predictors that we have significant evidence for. It is important to note, that with our `homeless0` column, 0 is for housed individuals and 1 is for unhoused individuals. Our first predictor, `pss_fr` is a quantifier for an individuals perceived social support by friends with higher scores indicating more support. The negative coefficient indicates that higher support from friends is a predictor of not being homeless. We see this negative coefficient with `female` as well, which indicates that being female may make someone less likely to be homeless. `avg_drinks` is the last of the predictors with a p.value less than or close to 0.05, and it shows a pretty strong relationship between a larger number of drinks consumed per day and homelessness.

To turn this into a parsimonious model we will only pick the three best variables picked above. Two of them have p-values greater than .05, but they're both incredibly close and the only alternative would be a univariate model.

```
updated_recipe <-
    recipe(homeless01 ~ pss_fr + avg_drinks + female, data = helprtc) %>%

    # Make homeless01 a factor ----
    step_mutate(homeless01 = homeless01 %>% as.factor()) %>%

    # Normalize numeric predictors so they're on the same scale ----
    step_normalize(pss_fr, avg_drinks, female) %>%

    # Log transform variables that appear log-normal to help it approach a normal dist ----
    step_log(avg_drinks, signed = TRUE)

homeless_workflow <-
    workflow() %>%
    add_model(homeless_model) %>%
    add_recipe(updated_recipe)

fit(homeless_workflow, helprtc) %>%
    broom::tidy() %>%
    arrange(p.value)
```

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 pss_fr        -0.331    0.0990     -3.34 0.000827
## 2 avg_drinks     1.49     0.479       3.11 0.00185
## 3 (Intercept)   -0.269    0.101      -2.65 0.00812
## 4 female        -0.192    0.0997     -1.93 0.0540
```

Here our model is far more frugal.

Chapter 10

Problem 3

```
# A

helprtc <-
    helprtc %>%
    mutate(homeless01 = homeless01 %>% as.factor)

my_null_log <- glm(homeless01 ~ 1, data = helprtc, family = "binomial")

pred <-
    helprtc %>%
    select(homeless01) %>%
    bind_cols(
        pred = stats::predict(my_null_log, newdata = helprtc, type = "response")
    ) %>%
    mutate(
        pred = case_when(
            pred > 0.5 ~ 1,
            pred < 0.5 ~ 0
        ),
        pred = factor(pred, levels = c("0", "1"))
    )

pred %>%
    conf_mat(truth = homeless01, estimate = pred)
```

```
##           Truth
## Prediction   0   1
##          0 244 209
##          1   0   0
```

```
# B

log_model <- glm(homeless01 ~ age, data = helprtc, family = "binomial")
log_model
```

```
##
## Call:  glm(formula = homeless01 ~ age, family = "binomial", data = helprtc)
##
## Coefficients:
## (Intercept)          age
##    -0.95724      0.02248
##
## Degrees of Freedom: 452 Total (i.e. Null);  451 Residual
## Null Deviance:       625.3
## Residual Deviance: 621.9     AIC: 625.9
```

The negative coefficient on the intercept here indicates that we're starting with a chance of homelessness less than 50% when age = 0. The coefficient next to age shows what is likely an incredibly poor positive relationship between age and homelessness.

```r
# C

stats::predict(log_model, newdata = data.frame(age = c(20,40)), type = "response")
```

```
##         1         2
## 0.3757450 0.4854891
```

This predicts a 37% chance of homelessness at age 40 and a 48% chance at age 40.

```r
# D

pred <-
    helprtc %>%
    select(homeless01, age) %>%
    bind_cols(
        stats::predict(log_model, newdata = helprtc, type = "response")
    ) %>%
    rename(pred = ...3) %>%
    mutate(
        pred = case_when(
            pred > 0.5 ~ 1,
            pred < 0.5 ~ 0
        ),
        pred = as.factor(pred),
        homeless01 = as.factor(homeless01)
    )
```

```
## New names:
## * `` -> ...3
```

```r
pred %>%
    conf_mat(truth = homeless01, estimate = pred)
```

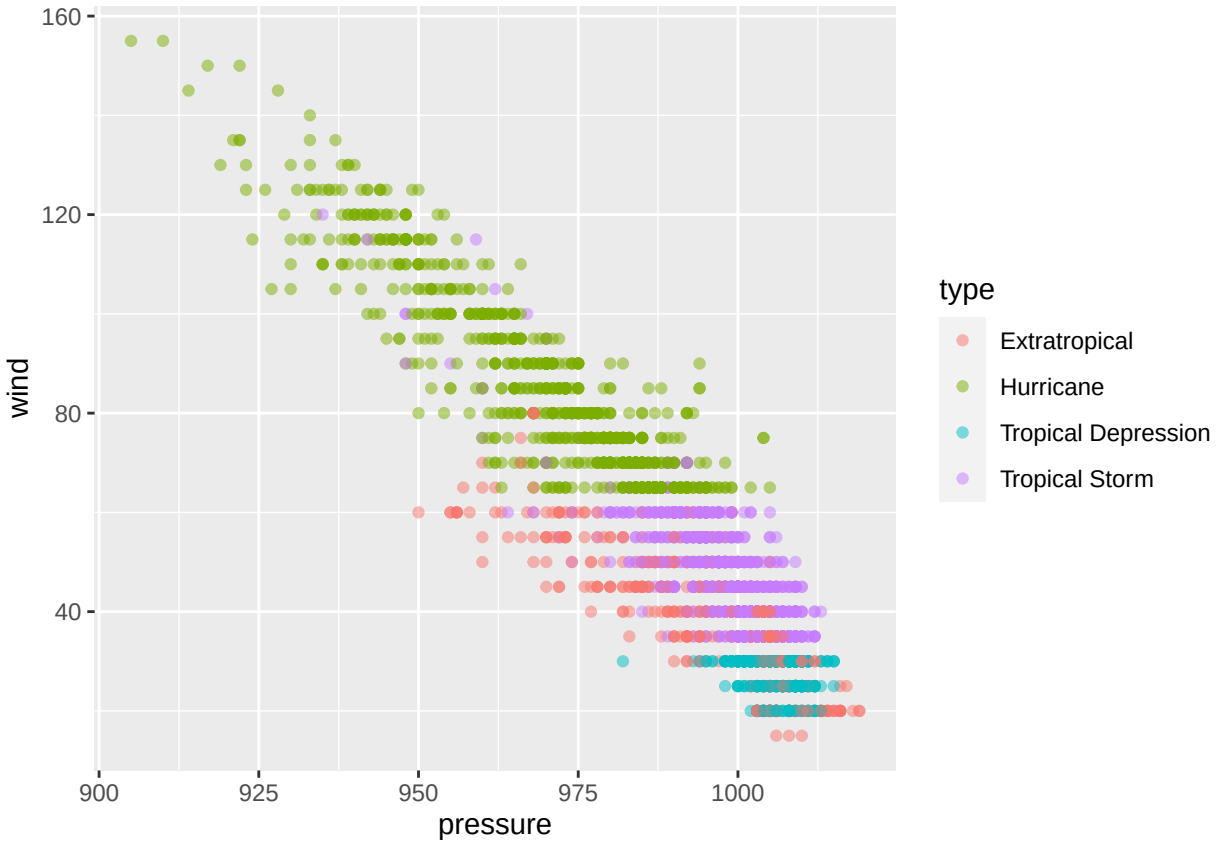
```
##           Truth
## Prediction   0   1
##          0 209 161
##          1  35  48
```

This model incorrectly assigns many people. It makes 161 incorrect "housed" classifications for individuals that are actually homeless.

Chapter 11

Problem 4

```
nasaweather::storms %>%
    ggplot(aes(x = pressure, y = wind, color = type)) +
    geom_point(alpha = 0.5)
```



A decision tree would be great for this data because the types are very neatly separated by pressure and wind values. It likely wouldn't be perfect, but no model needs to be. There are 4 fairly distinct regions for each of the 4 types.

Exercise 6 (a,c)

```
nhanes <- NHANES::NHANES
```

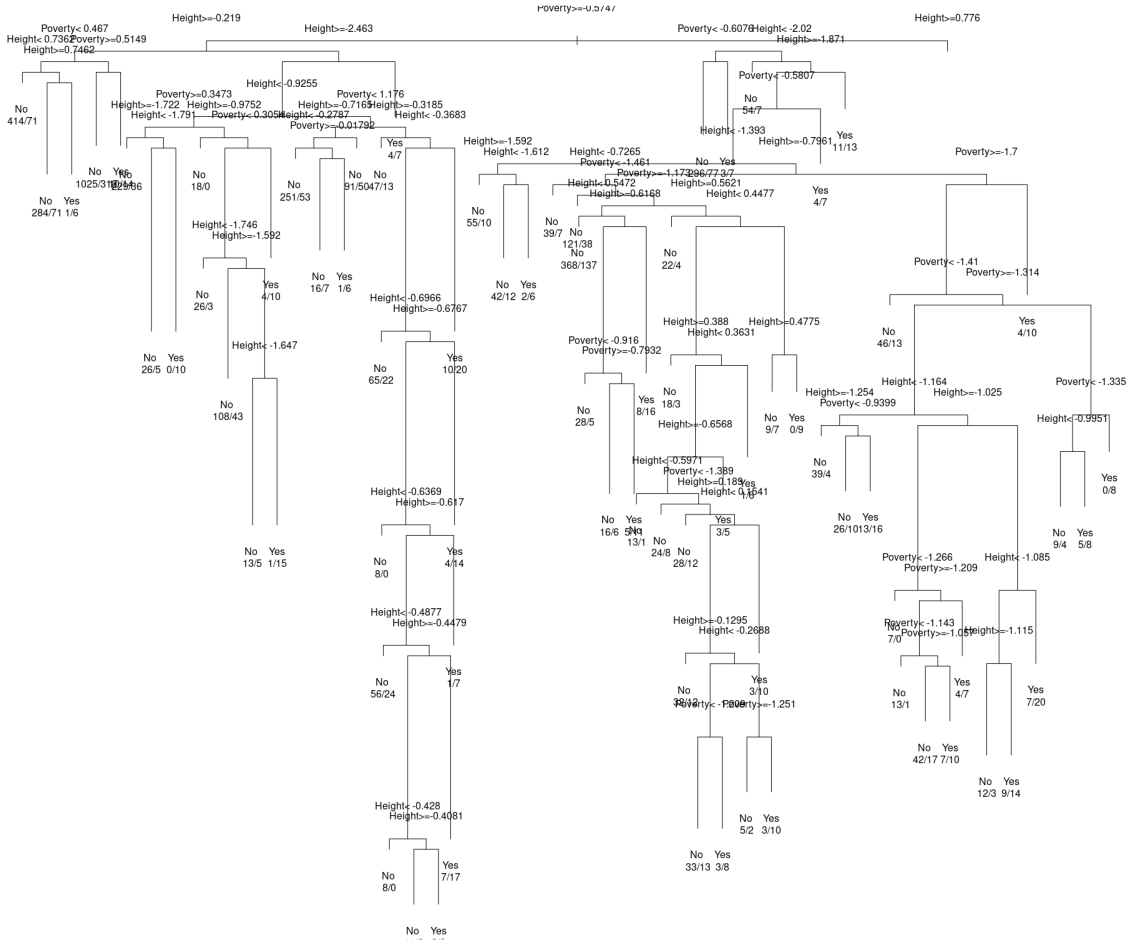
```
# Write my own scale function to circumvent bugs

my_scale <- function(x) {
    (x - mean(x, na.rm=TRUE)) / sd(x, na.rm=TRUE)
}
```

```
sleep <-
    nhanes %>%
    select(SleepTrouble, Poverty, Height) %>%
    na.omit() %>%
    mutate(
        Poverty = my_scale(Poverty),
        Height = my_scale(Height)
    )

sleep_split <-
    sleep %>%
    rsample::initial_split(.8)

sleep_train <-
    sleep_split %>%
    rsample::training()

sleep_test <-
    sleep_split %>%
    rsample::testing()
```

Decision Tree

```
my_tree <-
    rpart::rpart(
        SleepTrouble ~ Poverty + Height,
        data = sleep_train,
        control = rpart::rpart.control(cp = 0.001)
    )
```

Getting this tree to have more than 1 root node and less than 5 million was a task. I decided to simply let it have too much complexity and chalk it up to a flaw of the model. The overly complex tree can be seen completely zoomed out on the page below.



```
predType <-
    predict(my_tree, type = "class")

types <-
    data.frame(Actual = sleep_train$SleepTrouble, Predicted = predType)

types %>% table()
```

```
##       Predicted
## Actual   No  Yes
##    No  4163   79
##    Yes 1202  253
```

```
accuracy(types, truth = Actual, estimate = Predicted)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.775
```

```
sleep_train %>%
    dplyr::group_by(SleepTrouble) %>%
    dplyr::summarise(
        count = n(),
        percent = n() / nrow(sleep_train) * 100
    )
```

```
## # A tibble: 2 x 3
##   SleepTrouble count percent
##   <fct>        <int>   <dbl>
## 1 No            4242    74.5
## 2 Yes           1455    25.5
```

We see a 75/25 split here between no's and yes's. A null model that would assume no 100% of the time would have an accuracy of around 75%, so this decision tree would see a marginal increase against the null model if all one was concerned about was accuracy. Looking at the confusion matrix for our tree we see a lot of mis-classification, but many people with sleep trouble **were** captured, which is great. Overall, this model performs well enough on the training set.

Random Forest

```
my_forest <-
    randomForest::randomForest(
        SleepTrouble ~ Height + Poverty,
        data = sleep_train,
        ntree = 500,
        mtry = 2
    )
```

```
my_forest$confusion
```

```
##       No Yes class.error
## No  3854 388  0.09146629
## Yes  659 796  0.45292096
```

```
correct_class_rate <- (sum(diag(my_forest$confusion)) / nrow(sleep_train)) %>% round(digits = 3)
```

```
## The correct classification rate is 0.816
```

```
my_forest$importance
```

```
##         MeanDecreaseGini
## Height         1120.0452
## Poverty         794.2095
```

We see height being the more important of the two variables. Overall this model performs better than the decision tree.

K-Nearest Neighbors

```
my_knn <-
    class::knn(
        train = sleep_train %>% select(Poverty, Height),
        test = sleep_test %>% select(Poverty, Height),
        cl = sleep_train$SleepTrouble,
        k = 3
    )

sleep_knn_results <- data.frame(Actual = sleep_test$SleepTrouble, Predicted = my_knn)
```

```
knn_conf_mat <-
    sleep_knn_results %>%
    table()

knn_conf_mat
```

```
##       Predicted
## Actual  No Yes
##    No  875 183
##    Yes 225 142
```

```
sum(diag(knn_conf_mat)) / nrow(sleep_knn_results)
```

```
## [1] 0.7136842
```

Here we have a correct classification rate of 72%, which unperforms even the null model.

All of those but with a 75/25 split instead of an 80/20 one.

```
sleep_split <-
    sleep %>%
    rsample::initial_split(.75)

sleep_train <-
    sleep_split %>%
    rsample::training()

sleep_test <-
    sleep_split %>%
    rsample::testing()
```

Decision Tree

```
my_tree <-
    rpart::rpart(
        SleepTrouble ~ Poverty + Height,
        data = sleep_train,
        control = rpart::rpart.control(cp = 0.001)
    )
```

```
predType <-
    predict(my_tree, type = "class")

types <-
    data.frame(Actual = sleep_train$SleepTrouble, Predicted = predType)

types %>% table()
```

```
##       Predicted
## Actual   No  Yes
##    No  3827  141
##    Yes 1015  358
```

```
accuracy(types, truth = Actual, estimate = Predicted)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.784
```

```
sleep_train %>%
    dplyr::group_by(SleepTrouble) %>%
    dplyr::summarise(
        count = n(),
        percent = n() / nrow(sleep_train) * 100
    )
```

```
## # A tibble: 2 x 3
##   SleepTrouble count percent
##   <fct>        <int>   <dbl>
## 1 No            3968    74.3
## 2 Yes           1373    25.7
```

Random Forest

```
my_forest <-
    randomForest::randomForest(
        SleepTrouble ~ Height + Poverty,
        data = sleep_train,
        ntree = 500,
        mtry = 2
    )
```

```
my_forest$confusion
```

```
##       No Yes class.error
## No  3582 386  0.09727823
## Yes  645 728  0.46977422
```

```
correct_class_rate <- (sum(diag(my_forest$confusion)) / nrow(sleep_train)) %>% round(digits = 3)
```

```
## The correct classification rate is 0.807
```

```
my_forest$importance
```

```
##          MeanDecreaseGini
## Height         1075.1998
## Poverty         722.5936
```

K-Nearest Neighbors

```
my_knn <-
    class::knn(
        train = sleep_train %>% select(Poverty, Height),
        test = sleep_test %>% select(Poverty, Height),
        cl = sleep_train$SleepTrouble,
        k = 3
    )

sleep_knn_results <- data.frame(Actual = sleep_test$SleepTrouble, Predicted = my_knn)
```

```
knn_conf_mat <-
    sleep_knn_results %>%
    table()

knn_conf_mat
```

```
##       Predicted
## Actual   No  Yes
##    No  1135  197
##    Yes  269  180
```

```
sum(diag(knn_conf_mat)) / nrow(sleep_knn_results)
```

```
## [1] 0.7383492
```