Final Project

Data Science II (STAT 301-2)

Austin Shinn

I ended up not using the original data set that I found on Kaggle, as closer inspection on the Happiness report website showed that some numbers didn't make sense for what they were supposed to be; for example, for the 2019 data, social support is supposed to be the average of a binary variable, but the maximum values in the data hover around 1.6. Because of this, I just ended up finding the data again on the official reports website.

```
# 2020 report data (from 2019)
data_pt1 <- read_csv("data/WHR20_DataForFigure2.1.csv") %>%
  clean_names() %>%
  mutate(year = 2019)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `Country name` = col_character(),
##   `Regional indicator` = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
# data from 2019 report, with data from 2005-2018
data_pt2 <- read_csv("data/Chapter2OnlineData-1.csv") %>%
  clean_names()
```

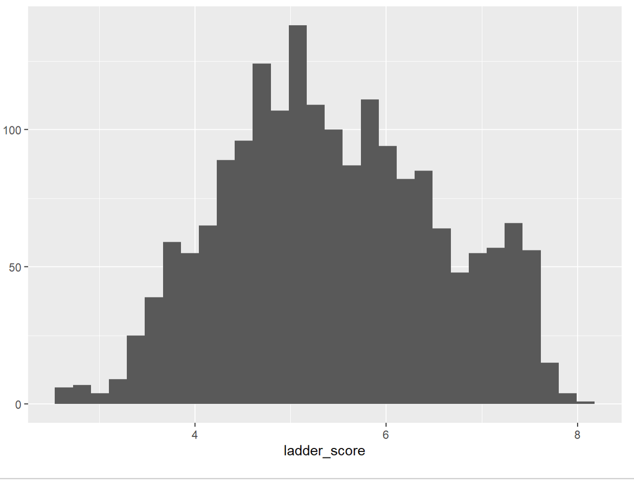
```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `Country name` = col_character()
## )
## See spec(...) for full column specifications.
```

```
# selecting relevant columns for model
data1 <- data_pt1 %>% select(c("country_name", "ladder_score", "logged_gdp_per_capita", "social_suppor

data2 <- data_pt2 %>% select(c("country_name", "life_ladder", "log_gdp_per_capita", "social_support",
  rename("ladder_score" = "life_ladder",
  "logged_gdp_per_capita" = "log_gdp_per_capita",
         "healthy_life_expectancy" = "healthy_life_expectancy_at_birth"
         )

# combining the data sets, adding 2019 data to the existing data, making the final working data set
data <- full_join(data1, data2)
```

```
## Joining, by = c("country_name", "ladder_score", "logged_gdp_per_capita", "social_support", "healt
hy_life_expectancy", "freedom_to_make_life_choices", "perceptions_of_corruption", "year")
```

```
ggplot(data, aes(ladder_score)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
skim(data)
```

Data summary

Name	data
Number of rows	1857
Number of columns	8
Column type frequency:	
character	1
numeric	7
Group variables	None

Variable type: character

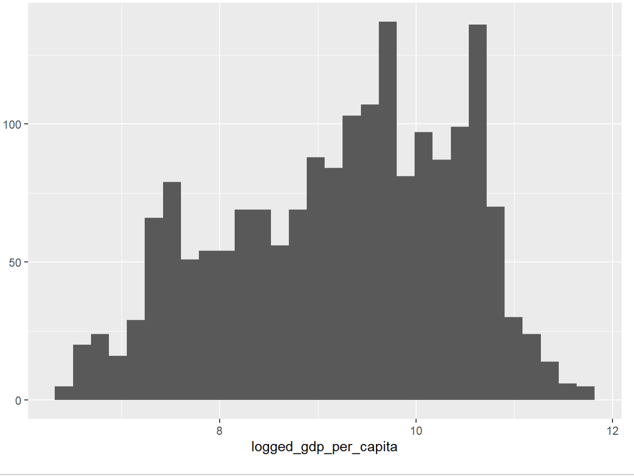
skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
country_name	0	1	4	25	0	166	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ladder_score	0	1.00	5.44	1.12	2.57	4.62	5.35	6.27	8.02	
logged_gdp_per_capita	28	0.98	9.23	1.19	6.46	8.31	9.41	10.21	11.77	
social_support	13	0.99	0.81	0.12	0.29	0.75	0.83	0.90	0.99	
healthy_life_expectancy	28	0.98	63.22	7.55	32.30	58.40	65.10	68.40	76.80	
freedom_to_make_life_choices	29	0.98	0.74	0.14	0.26	0.64	0.76	0.85	0.99	
perceptions_of_corruption	96	0.95	0.75	0.19	0.04	0.69	0.80	0.87	0.98	
year	0	1.00	2012.88	3.98	2005.00	2010.00	2013.00	2016.00	2019.00	

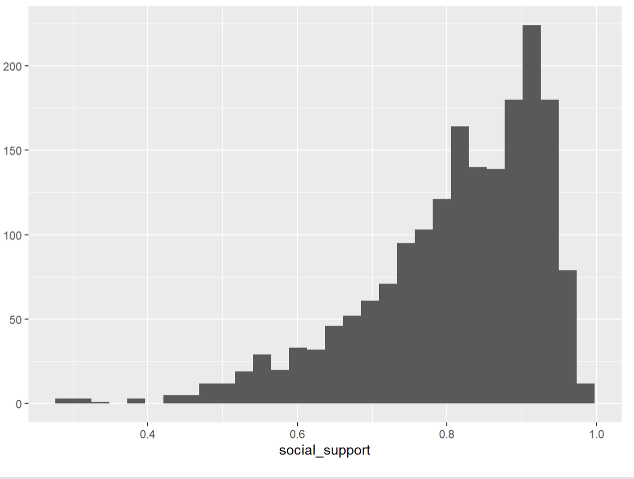
```
# already logged
ggplot(data, aes(logged_gdp_per_capita)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



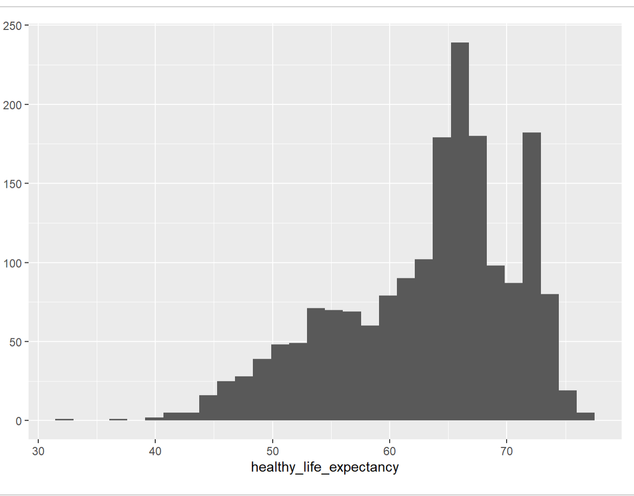

```
# left skew
ggplot(data, aes(social_support)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



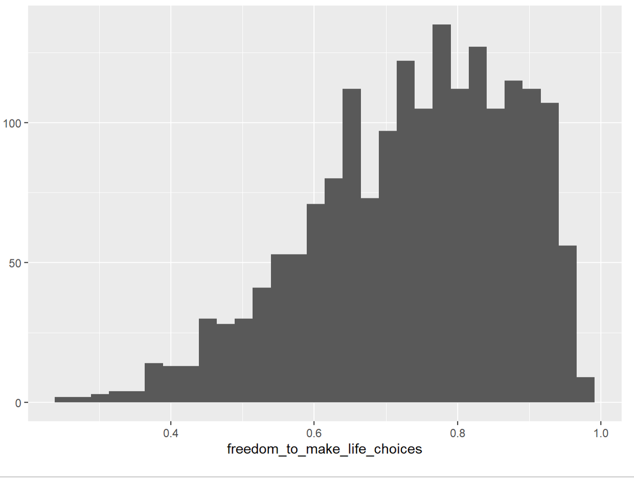

```
# left skew
ggplot(data, aes(healthy_life_expectancy)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



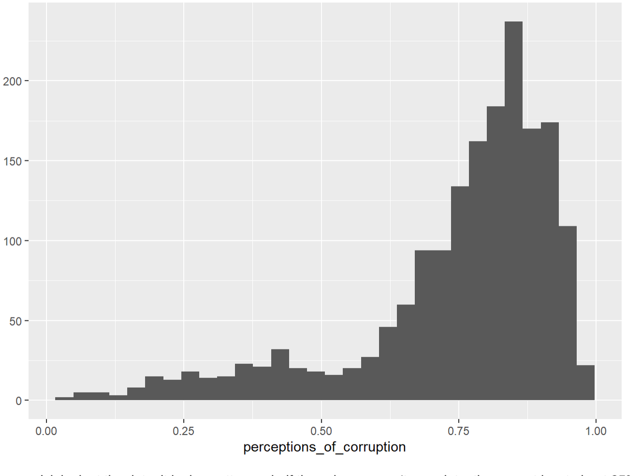

```
#left skew
ggplot(data, aes(freedom_to_make_life_choices)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# left skew
ggplot(data, aes(perceptions_of_corruption)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



From a quick look at the data, it looks pretty good - if the columns aren't complete, they are at least about 95%+ complete. To fill in the missing values, K-nearest neighbors imputation will be used as recommended by the textbook for all the methods I'll be using (https://www.tmwr.org/pre-proc-table.html). Several of the predictors will also have to be log transformed to account for left or right skew, and will be normalized.

```
# splitting data into 80/20, stratifying by ladder_score to ensure there is data from different happin
split <- initial_split(data, prop = .8, strata = ladder_score)

happy_train <- training(split)
happy_test <- testing(split)

vfold <- vfold_cv(happy_train, v = 10, repeats = 3, strata = ladder_score)
```

From here, I split the data into an 80/20 proportion, since there were enough observations to afford making the test set a smaller proportion. V-fold cross validation was employed, with 10 folds and 3 repetitions.

```
# prepping and baking
recipe <- happy_train %>%
  # all predictors except country name and year of observation
  # using KNN imputation to fill in missing values
  recipe(ladder_score ~ logged_gdp_per_capita + social_support + healthy_life_expectancy + freedom_to_
  step_knnimpute(all_predictors(), neighbors = 3) %>%
  step_log(social_support, healthy_life_expectancy, freedom_to_make_life_choices, perceptions_of_corru
  step_normalize(all_predictors())

recipe %>%
  prep(training = happy_train) %>%
  bake(new_data = NULL) %>%
  view()
```

For the recipe, I first imputed using k-nearest neighbors to fill in the missing values. Then I log transformed the variables (except for gdp per capita, which was already logged). After that, the predictor variables were all normalized.

```
rf_model <- rand_forest(mode = "regression",
                        min_n = tune(),
                        mtry = tune()) %>%
  set_engine("ranger")

bt_model <- boost_tree(mode = "regression",
                       mtry = tune(),
                     min_n = tune(),
                     learn_rate = tune()) %>%
  set_engine("xgboost")

knn_model <- nearest_neighbor(mode = "regression",
                              neighbors = tune()) %>%
  set_engine("kknn")
```

Models were chosen, the same ones as in lab 7.

```
rf_params <- parameters(rf_model) %>%
  update(mtry = mtry(c(2,5)))

rf_grid <- grid_regular(rf_params, levels = 5)

bt_params <- parameters(bt_model) %>%
  update(mtry = mtry(c(2,5)),
         learn_rate = learn_rate(c(-5, -.2)))

bt_grid <- grid_regular(bt_params, levels = 5)

knn_params <- parameters(knn_model)

knn_grid <- grid_regular(knn_params, levels = 5)
```

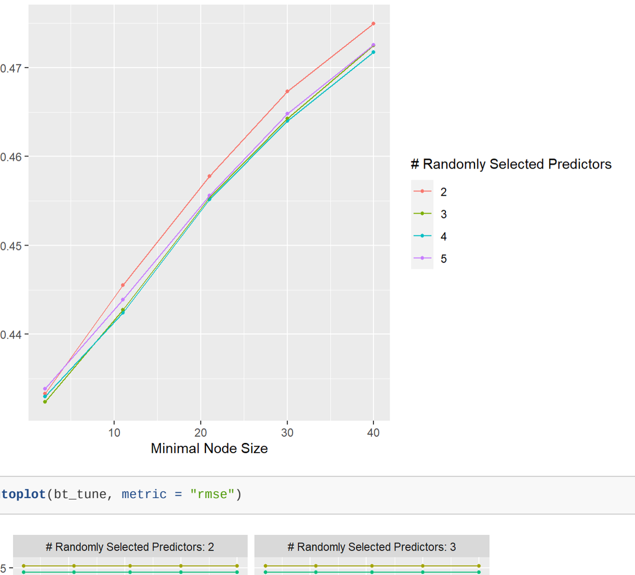
Parameters were set based on the number of predictors, with the tree models spanning from 2 to all of the predictors in a tree.

```
rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(recipe)

 bt_workflow <- workflow() %>%
  add_model(bt_model) %>%
  add_recipe(recipe)

knn_workflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(recipe)
```

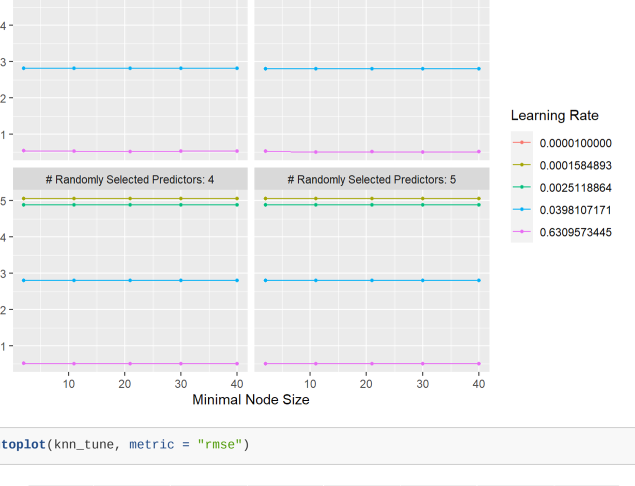
```
rf_tune <- rf_workflow %>%
  tune_grid(resamples = vfold,
          grid = rf_grid)

bt_tune <- bt_workflow %>%
  tune_grid(resamples = vfold,
          grid = bt_grid)

knn_tune <- knn_workflow %>%
  tune_grid(resamples = vfold,
          grid = knn_grid)

save(rf_tune, bt_tune, knn_tune, file = "data/final_tune_grids.rda")
```

Workflows and tuning grids were then created. The tuning grids were run and then saved to avoid rerunning in subsequent knittings.

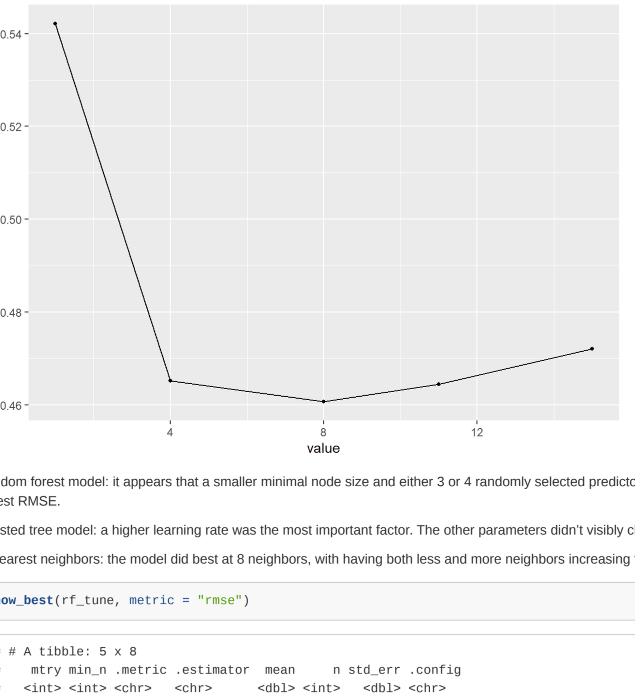
```
load("data/final_tune_grids.rda")

autoplot(rf_tune, metric = "rmse")
```




```
autoplot(bt_tune, metric = "rmse")
```




```
autoplot(knn_tune, metric = "rmse")
```



Random forest model: it appears that a smaller minimal node size and either 3 or 4 randomly selected predictors generated the lowest RMSE.

Boosted tree model: a higher learning rate was the most important factor. The other parameters didn't visibly change the RMSE.

K-Nearest neighbors: the model did best at 8 neighbors, with having both less and more neighbors increasing the RMSE.

```
show_best(rf_tune, metric = "rmse")
```

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     3     2 rmse    standard   0.432    30 0.00691 Preprocessor1_Model02
## 2     4     2 rmse    standard   0.433    30 0.00700 Preprocessor1_Model03
## 3     2     2 rmse    standard   0.433    30 0.00679 Preprocessor1_Model01
## 4     5     2 rmse    standard   0.434    30 0.00717 Preprocessor1_Model04
## 5     4    11 rmse    standard   0.442    30 0.00694 Preprocessor1_Model07
```

```
show_best(rf_tune, metric = "rsq")
```

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     3     2 rsq     standard   0.853    30 0.00508 Preprocessor1_Model02
## 2     2     2 rsq     standard   0.853    30 0.00503 Preprocessor1_Model01
## 3     4     2 rsq     standard   0.853    30 0.00518 Preprocessor1_Model03
## 4     5     2 rsq     standard   0.852    30 0.00536 Preprocessor1_Model04
## 5     4    11 rsq     standard   0.846    30 0.00529 Preprocessor1_Model07
```

```
show_best(bt_tune, metric = "rmse")
```

```
## # A tibble: 5 x 9
##    mtry min_n learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5    40      0.631 rmse    standard   0.501    30 0.00638 Preprocessor1_M~
## 2     4    30      0.631 rmse    standard   0.504    30 0.00671 Preprocessor1_M~
## 3     5    21      0.631 rmse    standard   0.504    30 0.00660 Preprocessor1_M~
## 4     5    11      0.631 rmse    standard   0.506    30 0.00684 Preprocessor1_M~
## 5     5    30      0.631 rmse    standard   0.507    30 0.00623 Preprocessor1_M~
```

```
show_best(bt_tune, metric = "rsq")
```

```
## # A tibble: 5 x 9
##    mtry min_n learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5    40      0.631 rsq     standard   0.803    30 0.00538 Preprocessor1_M~
## 2     4    30      0.631 rsq     standard   0.801    30 0.00551 Preprocessor1_M~
## 3     5    21      0.631 rsq     standard   0.801    30 0.00580 Preprocessor1_M~
## 4     5    11      0.631 rsq     standard   0.800    30 0.00579 Preprocessor1_M~
## 5     5     2      0.631 rsq     standard   0.799    30 0.00545 Preprocessor1_M~
```

```
show_best(knn_tune, metric = "rmse")
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         8 rmse    standard   0.461    30 0.00634 Preprocessor1_Model3
## 2        11 rmse    standard   0.465    30 0.00625 Preprocessor1_Model4
## 3         4 rmse    standard   0.465    30 0.00671 Preprocessor1_Model2
## 4        15 rmse    standard   0.472    30 0.00632 Preprocessor1_Model5
## 5         1 rmse    standard   0.542    30 0.00711 Preprocessor1_Model1
```

```
show_best(knn_tune, metric = "rsq")
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         8 rsq     standard   0.834    30 0.00487 Preprocessor1_Model3
## 2         4 rsq     standard   0.832    30 0.00528 Preprocessor1_Model2
## 3        11 rsq     standard   0.831    30 0.00478 Preprocessor1_Model4
## 4        15 rsq     standard   0.826    30 0.00489 Preprocessor1_Model5
## 5         1 rsq     standard   0.784    30 0.00625 Preprocessor1_Model1
```

From the RMSE of the models, it appears that the random forest model has the best metrics. With R^2, the difference is smaller, but it still is the best out of the other methods. For a score that is out of 10, an RMSE of around .4 is not amazing, but pretty accurate.

```
rf_workflow_tuned <- rf_workflow %>%
  finalize_workflow(select_best(rf_tune, metric = "rmse"))

rf_results <- fit(rf_workflow_tuned, happy_train)

final_metric <- metric_set(rmse, rsq)

predict(rf_results, new_data = happy_test) %>%
  bind_cols(happy_test %>% select(ladder_score)) %>%
  final_metric(truth = ladder_score, estimate = .pred)
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.435
## 2 rsq     standard       0.848
```

When we run the final model on the test set, we find that the model performs comparably to the test set, showing that it is not overfitted to the training data, and the model is reasonably powerful for predicting happiness score of a nation given the predictors.